# Оглавление

# Глава 1

# Алгоритм Apriori

**Алгоритм Apriori** — алгоритм поиска ассоциативных правил.

## 1.1. Пример

Предположим необходимо облегчить покупателям поиск товаров в супермаркете и разместить «ассоциированные» товары ближе друг к другу.

Например, покупатель берет томаты и идёт к кассе, стоит ли размещать на его пути соль? Часто ли покупатель берет помидоры и соль за раз (за одну транзакцию)? Ассоциированы ли эти товары?

Необходимо проанализировать рыночную корзину (market basket analysis), не делая никаких предположений (о соленых помидорах) — a priori (лат.).

## 1.2. См. также

- Support (Association Rule Support)
- Confidence (Association Rule Confidence)

## 1.3. Ссылки

- Ассоциативные правила. Data Mining — добыча данных
- Лекция: Методы поиска ассоциативных правил

# Глава 2

# Алгоритм Бога

**Алгори́тм Бо́га** — понятие, возникшее в ходе обсуждения способов решения кубика Рубика. Термин может также быть использован в отношении других перестановочных головоломок. Под алгоритмом Бога головоломки подразумевается любой алгоритм, который позволяет получить решение головоломки, содержащее минимально возможное число ходов (оптимальное решение), начиная с любой заданной конфигурации.

Один из пионеров математической теории кубика Рубика Дэвид Сингмастер[1] так описывает появление термина:

> Джон Конвей, один из крупнейших специалистов по теории групп в мире, отметил, что Кубик подчиняется так называемым законам сохранения (или чётности), а это означает, что некоторые движения просто невозможны. Либо Конвей, либо один из его коллег в Кембридже определил кратчайший путь из любого данного состояния назад к начальному состоянию как «Алгоритм Бога».
>
> *Оригинальный текст* (англ.)

> *John Conway, one of the world's greatest group theorists, observed that the Cube obeys what are known as conservation (or parity) laws, meaning that some moves are simply not possible. Either Conway or one of his colleagues at Cambridge defined the shortest route from any given position back to the starting position as «God's Algorithm».*
>
> — Дэвид Сингмастер[2]

## 2.1. Определение

Алгоритм Бога может существовать для головоломок с конечным числом возможных конфигураций и с конечным набором «ходов», допустимых в каждой конфигурации и переводящих текущую конфигурацию в другую. Термин «решить головоломку» означает — указать последовательность ходов, переводящих некоторую начальную конфигурацию в некоторую конечную конфигурацию. Оптимально решить головоломку — указать самую короткую последовательность ходов для решения головоломки. Оптимальных решений может быть несколько.

К известным головоломкам, подпадающим под это определение, относятся кубик Рубика, Ханойская башня, Пятнашки, Солитер с фишками (*англ.*), различные задачи о переливании и перевозке («Волк, коза (овца) и капуста»). Общим для всех этих головоломок является то, что они могут быть описаны в виде графа, вершинами которого являются всевозможные конфигурации головоломки, а рёбрами — допустимые переходы между ними («ходы»).

Во многих подобных головоломках конечная конфигурация негласно предполагается, например, в «пятнашках» — упорядоченное расположение косточек, для кубика Рубика — одноцветность граней. В этих случаях «собрать головоломку» означает, что требуется для произвольной начальной конфигурации указать последовательность ходов, приводящих в *фиксированную* конечную конфигурацию.

Алгоритм *решает* головоломку, если он принимает в качестве исходных данных произвольную пару начальной и конечной конфигураций (или только начальную конфигурацию, если конечная конфигурация зафиксирована) и возвращает в качестве результата последовательность ходов, переводящих начальную конфигурацию в конечную (если такая последовательность существует, в противном случае, алгоритм сообщает о невозможности решения). *Оптимальное решение* содержит минимально возможное количество ходов.

Тогда алгоритм Бога (для данной головоломки) — это алгоритм, который решает головоломку и находит для произвольной пары конфигураций хотя бы одно оптимальное решение.

Некоторые авторы считают, что алгоритм Бога должен также быть *практичным*, то есть использовать разумный объём памяти и завершаться в разумное время.

Пусть $G$ — группа перестановочной

головоломки (с заданным порождающим множеством), $v$ — вершина графа Кэли группы $G$. Найти эффективный, практичный алгоритм для определения пути из $v$ в вершину $v_0$, связанную с нейтральным элементом, длина которого равна расстоянию от $v$ до $v_0$. [...] Этот алгоритм называется **алгоритмом Бога**.

*Оригинальный текст* (англ.)

*Let G be the group of a permutation puzzle (with a fixed generating set) and let v be a vertex in the Cayley graph of G. Find an effective, practical algorithm for determining a path from v to the vertex $v_0$ associated to the identity having a length equal to the distance from v to $v_0$. [...] This algorithm is called **God's algorithm**.*

— Дэвид Джойнер[3]

Практичность можно понимать по-разному. Так, существуют компьютерные программы, позволяющие за приемлемое время найти оптимальное решение для произвольной конфигурации кубика Рубика[4]. В то же время аналогичная задача для кубика 4×4×4 на данный момент остаётся практически неосуществимой[5][6][7]. Для некоторых головоломок существует стратегия, позволяющая в соответствии с простыми правилами определить оптимальное решение вручную, без помощи компьютера.

Альтернативное определение алгоритма Бога: от алгоритма не требуется нахождения всей последовательности ходов; вместо этого достаточно найти первый ход оптимального решения, приближающий к цели и переводящий в новую конфигурацию. Два определения являются эквивалентными: повторное применение алгоритма к новой паре конфигураций снова находит ход оптимального решения, что позволяет получить всю последовательность ходов оптимального решения.

## 2.2. Число Бога

Числом Бога данной головоломки называется число $n$, такое, что *существует хотя бы одна* конфигурация головоломки, оптимальное решение которой состоит из $n$ ходов, и *не существует ни одной* конфигурации, длина оптимального решения которой превышает $n$. Другими словами, число Бога — это точная верхняя грань множества длин оптимальных решений конфигураций головоломки.

Число Бога для кубика Рубика равно 20 — это диаметр графа Кэли группы кубика Рубика[8].

В общем случае (для произвольной перестановочной головоломки), число Бога равно не диаметру графа Кэли группы головоломки, а эксцентриситету вершины, соответствующей «собранному» состоянию головоломки.

## 2.3. Примеры

- Кубик Рубика 3×3×3 всегда может быть решён не более чем в 20 ходов[9]. Известны конфигурации (англ.), требующие для сборки не менее 20 ходов. Таким образом, «число Бога» кубика Рубика равно 20.

- Число Бога кубика Рубика 2×2×2 равно 11 ходам, если поворот грани на 180° считается за 1 ход, или 14 ходам, если поворот грани на 180° считается за 2 хода. Небольшое (3674160) количество конфигураций кубика Рубика 2×2×2 позволило вычислить алгоритм Бога (в виде оптимального решения для каждой конфигурации) ещё в 80-х годах[10].

- Число Бога пирамидки Мефферта равно 11[11].

- Трёхцветный кубик — кубик Рубика, противоположные грани которого окрашены одинаково. Число конфигураций трёхцветного кубика равно

$$2^{11} \cdot 3^7 \cdot C_{12}^4 \cdot (C_8^4)^2 = 10\,863\,756\,288\,000$$

  В марте-апреле 2012 года было установлено, что число Бога трёхцветного кубика равно 15 FTM, 17 QTM или 14 STM (согласно метрике STM, поворот любого среднего слоя также считается за 1 ход)[12].

- Пятнашки могут быть решены в 80 «коротких»[13] или 43 «длинных»[14] ходов в худшем случае (под «короткими» ходами подразумеваются перемещения отдельных костяшек, а под «длинными» — перемещения целых рядов из 1, 2 или 3 костяшек). Для обобщённых пятнашек (с бóльшим, чем 15, количеством костяшек) задача поиска *кратчайшего* решения является NP-полной[15].

- Для Ханойской башни алгоритм Бога существует при любом количестве дисков, но с добавлением дисков число ходов растёт экспоненциально[16].

## 2.4. См. также

- Алгоритм Судного дня
- Диаметр графа

- Математика кубика Рубика
- Пирамидка Мефферта

## 2.5. Примечания

[1] История кубика Рубика. Проверено 20 июля 2013. Архивировано из первоисточника 4 сентября 2013.

[2] *Jerry Slocum, David Singmaster, Wei-Hwa Huang, Dieter Gebhardt, Geert Hellings* The Cube: The Ultimate Guide to the World's Bestselling Puzzle — Secrets, Stories, Solutions. — 2009. — С. 26. — 142 с.

[3] *David Joyner* Adventures in Group Theory: Rubik's Cube, Merlin's Machine, and Other Mathematical Toys. — 2008. — С. 149. — 328 с.

[4] *Herbert Kociemba.* Cube Explorer. Проверено 27 июля 2013. Архивировано из первоисточника 4 сентября 2013.

[5] Bigger rubik cube bound

[6] 4x4x4 algorithm generator? (like cube explorer)

[7] 4x4 Algorithms

[8] *Weisstein, Eric W.* God's Number.

[9] *Rokicki, T.; Kociemba, H.; Davidson, M.; and Dethridge, J.* God's Number is 20 (англ.). Проверено 19 июля 2013. Архивировано из первоисточника 27 июля 2013.

[10] *Jaap Scherphuis.* Mini Cube, the 2×2×2 Rubik's Cube (англ.). Проверено 21 июля 2013. Архивировано из первоисточника 4 сентября 2013.

[11] *Jaap Scherphuis.* Pyraminx (англ.). Проверено 21 июля 2013. Архивировано из первоисточника 29 августа 2013.

[12] Some 3-color cube results. Domain of the Cube Forum. Проверено 29 июля 2013. Архивировано из первоисточника 4 сентября 2013.

[13] A. Brüngger, A. Marzetta, K. Fukuda and J. Nievergelt, The parallel search bench ZRAM and its applications, *Annals of Operations Research* **90** (1999), pp. 45-63.

[14] *Bruce Norskog.* The Fifteen Puzzle can be solved in 43 "moves" Domain of the Cube Forum (англ.) (Wed, 12/08/2010 - 16:43). Проверено 20 июля 2013. Архивировано из первоисточника 4 сентября 2013.

[15] Daniel Ratner, Manfred K. Warmuth (1986). «Finding a shortest solution for the N × N extension of the 15-puzzle is intractable». in *Proceedings AAAI-86*. National Conference on Artificial Intelligence, 1986. pp. 168—172.

[16] Carlos Rueda, «An optimal solution to the Towers of Hanoi Puzzle».

## 2.6. Ссылки

- Кубик Рубика: штурм твердыни (Константин Кноп)
- Алгоритм Бога и число Бога (Sliding Tile Puzzle Corner)

# Глава 3

# Алгоритм выбора

В информатике **алгоритм выбора** - это алгоритм для нахождения *k*-го по величине элемента в массиве (такой элемент называется *k*-й порядковой статистикой). Частными случаями этого алгоритма являются нахождение минимального элемента, максимального элемента и медианы. Существует алгоритм, который гарантированно решает задачу выбора k-го по величине элемента за O($n$).

## 3.1. Выбор с помощью сортировки

Задачу выбора можно свести к сортировке. В самом деле, можно упорядочить массив, а затем взять нужный по счету элемент. Это эффективно в том случае, когда выбор нужно делать многократно: тогда можно отсортировать массив за O($n$ log $n$) и затем выбирать из него элементы. Однако если выбор нужно произвести однократно, данный алгоритм может оказаться неоправданно долгим.

## 3.2. Линейный алгоритм для нахождения минимума (максимума)

Очевидно, как за линейное время найти минимум (максимум) в данном массиве:

- Изначально присвоить $min = a[0]$;
- Для каждого элемента $a[i]$ выполнить: если $min > a[i]$ , присвоить $min = a[i]$ .

## 3.3. Алгоритм BFPRT

**BFPRT-Алгоритм** позволяет найти *k*-ю порядковую статистику гарантированно за O($n$). Назван в честь своих изобретателей: Manual **B**lum, Robert W. **F**loyd, Vaughan R. **P**ratt, Ronald L. **R**ivest и Robert Endre **T**arjan. Используется при достаточно длинном списке элементов, свыше 800 элементов.

### 3.3.1. Принцип действия

Ввод: число $i$ , обозначающее $i$ -тый элемент

1. Список делится на подмножества элементов, по 5 элементов в каждом (кроме последнего подмножества). Число элементов в подмножествах может варьировать от 5 до 21 и должно быть в любом случае нечётным.
2. Каждое подмножество сортируется с помощью подходящего алгоритма сортировки.
3. Пусть $S$ - множество медиан, образовавшихся в подмножествах после сортировки. Рекурсивно находится медиана в $S$ - медиана медиан. Назовем его $s$ .
   - Результирующая после 3 шага структура, имеет следующую особенность:
     --- Четверть всех элементов в любом случае имеет ключ $< s$ .
     --- Четверть всех элементов в любом случае имеет ключ $> s$ .
4. Теперь список сортируется относительно медианы s на 2 подмножества $S_1$ и $S_2$ . При этом нужно сравнить с s только половину всех элементов, так как две четверти элементов уже отсортированы относительно s. В результате каждое из подмножеств $S_1$ и $S_2$ содержит максимально 3/4 всех элементов (минимально - 1/4 всех элементов).
5. Если:
   - $i = |S_1| + 1$ , то искомый элемент найден - это медиана медиан $s$
   - $i \leq |S_1|$ , то алгоритм вызывается рекурсивно на множестве $S_1$
   - в любом другом случае, алгоритм вызывается рекурсивно на множестве $S_2$

### 3.3.2. Особенности

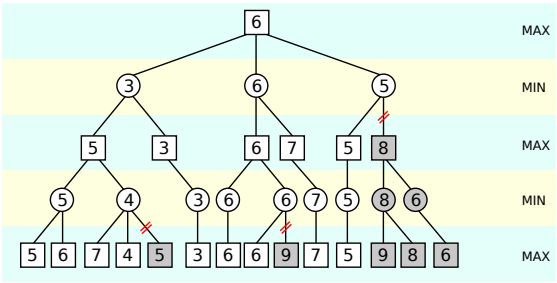
При каждом рекурсивном вызове, алгоритм позволяет отбросить минимум четверть всех элементов.

## 3.4. Литература


- *Volker Heun* Основные Алгоритмы = Grundlegende Algorithmen. — 1-е изд. — Мюнхен: Vieweg Verlag, 2000. — С. 86. — ISBN 3-528-03140-9.
- Time Bounds for Selection by Manual Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Journal of Computer and System Sciences 7,4 August 1973, 448—460 (англ.)

# Глава 4

# Альфа-бета отсечение




**Альфа-бета отсечение** (англ. *Alpha-beta pruning*) — это алгоритм поиска, стремящийся сократить количество узлов, оцениваемых в дереве поиска алгоритмом минимакс. Этот алгоритм предназначен для антагонистических игр и используется для машинной игры (в шахматах, го и других). В основе алгоритма лежит идея, что оценивание ветви дерева поиска может быть досрочно прекращено (без вычисления всех значений оценивающей функции), если было найдено, что для этой ветви значение оценивающей функции в любом случае хуже, чем вычисленное для предыдущей ветви. Альфа-бета отсечение является оптимизацией, так как результаты работы оптимизируемого алгоритма не изменяются.

## 4.1. История

Аллен Ньюэлл и Герберт Саймон, использовавшие то, что Джон Маккарти назвал «аппроксимацией»[1] в 1958 году, написали, что альфа-бета отсечение «кажется, изобреталось неоднократно»[2]. Артур Самуэль, Ричардс, Харт, Левин, Эдвардс независимо предлагали ранние версии этого алгоритма[3]. Маккарти также выдвигал подобные идеи на Дартмутском семинаре в 1956 году, а затем, в 1961 году, предложил для исследования группе своих студентов в MIT, включая Алана Котока[4]. Александр Брудно независимо открыл алгоритм и опубликовал свои результаты в 1963 году[5]. В 1975 году Дональд Кнут и Рональд У. Мур усовершенствовали алгоритм (добавив «бета» отсечения).[6][7]

## 4.2. Оптимизация Минимакс

Преимущество альфа-бета отсечения фактически заключается в том, что некоторые из ветвей подуровней дерева поиска могут быть исключены после того, как хотя бы одна из ветвей уровня рассмотрена полностью. Так как отсечения происходят на каждом уровне вложенности (кроме последнего), эффект может быть весьма значительным. На эффективность метода существенно влияет предварительная сортировка вариантов (без перебора или с перебором на меньшую глубину) — при сортировке чем больше в начале рассмотрено «хороших» вариантов, тем больше «плохих» ветвей может быть отсечено без исчерпывающего анализа.

## 4.3. См. также

- Компьютерные шахматы
- Стратегия слепого поиска

## 4.4. Примечания

[1] *McCarthy, John.* Human Level AI Is Harder Than It Seemed in 1955 (LaTeX2HTML 27 November 2006). Проверено 20 декабря 2006. Архивировано из первоисточника 9 апреля 2012.

[2] Newell, Allen and Herbert A. Simon (March 1976). «Computer Science as Empirical Inquiry: Symbols and Search» (PDF). *Communications of the ACM, Vol. 19, No. 3*. Проверено 2006-12-21.

[3] *Richards, D.J. and Hart, T.P.* The Alpha-Beta Heuristic (AIM-030). Massachusetts Institute of Technology (4 December 1961 to 28 October 1963). Проверено 21 декабря 2006. Архивировано из первоисточника 9 апреля 2012.

[4] *Kotok, Alan* MIT Artificial Intelligence Memo 41 (XHTML 3 December 2004). Проверено 1 июля 2006. Архивировано из первоисточника 9 апреля 2012.

[5] *Marsland, T.A.* Computer Chess Methods (PDF) from Encyclopedia of Artificial Intelligence. S. Shapiro (editor) (PDF) 159-171. J. Wiley & Sons (May 1987). Проверено 21 декабря 2006. Архивировано из первоисточника 9 апреля 2012.

[6]
- Knuth, D. E., and Moore, R. W. (1975). «An Analysis of Alpha-Beta Pruning». *Artificial Intelligence Vol. 6, No. 4*: 293–326.
  - Reprinted as Chapter 9 in *Knuth Donald E.* Selected Papers on Analysis of Algorithms. — Stanford, California: Center for the Study of Language and Information - CSLI Lecture Notes, no. 102, 2000. — ISBN 1-57586-212-3.

[7] Abramson, Bruce (June 1989). «Control Strategies for Two-Player Games». *ACM Computing Surveys, Vol. 21, No. 2* **21**: 137. DOI:10.1145/66443.66444. Проверено 2008-08-20.

## 4.5. Ссылки

- *Knuth D. E., Moor R. W.* Анализ альфа-бета отсечений // Artificial Intelligence, 1975. Пер. с англ. П. Н. Дубер.

# Глава 5

# Двоичный поиск

**Двоичный (бинарный) поиск** (также известен как метод деления пополам и дихотомия) — классический алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива на половины. Используется в информатике, вычислительной математике и математическом программировании.

Частным случаем двоичного поиска является метод бисекции, который применяется для поиска корней заданной непрерывной функции на заданном отрезке.

## 5.1. Поиск элемента в отсортированном массиве

Пример кода на языке программирования Си для поиска элемента x в массиве a[n], отсортированного в возрастающем порядке:

/* Номер первого элемента в массиве */ size_t first = 0; /* Номер элемента в массиве, СЛЕДУЮЩЕГО ЗА последним */ size_t last = n; /* Если просматриваемый участок непустой, first < last */ if (n == 0) { /* массив пуст */ return NOTFOUND(0); } else if (a[0] > x) { /* не найдено; * если вам надо вставить его со сдвигом - то в позицию 0 */ return NOTFOUND(0); } else if (a[n - 1] < x) { /* не найдено; * если вам надо вставить его со сдвигом - то в позицию n */ return NOTFOUND(n); } while (first < last) { /* ВНИМАНИЕ! В отличие от более простого (first + last) / 2, * этот код устойчив к переполнениям. * * Если first и last знаковые, возможен код: * ((unsigned)first + (unsigned)last) >> 1. * Соотвественно в Java: (first + last) >>> 1. */ size_t mid = first + (last - first) / 2; if (x <= a[mid]) last = mid; else first = mid + 1; } /* Если условный оператор if (n == 0) и т.д. в начале опущен - * значит, тут раскомментировать! */ if (/* last < n && */ a[last] == x) { /* Искомый элемент найден. last - искомый индекс */ return FOUND(last); } else { /* Искомый элемент не найден. Но если вам вдруг надо его * вставить со сдвигом, то его место - last. */ return NOTFOUND(last); }

Несмотря на то, что код достаточно прост, в нём есть несколько ловушек.

- Что будет, если first и last по отдельности умещаются в свой тип, а first+last — нет?[1]
- Будет ли работать на пустом массиве (n=0)?
- Способен ли код находить отсутствующие значения? У некоторых программистов написанный «с листа» двоичный поиск в такой ситуации зацикливается — и они этого не осознают, пока тестирование не даст ошибку.
- Способен ли код найти первый и последний элемент? Возможна и обратная ошибка — выход за границы массива, если x задать слишком большим или слишком маленьким.
- Иногда требуется, чтобы, если x в цепочке существует в нескольких экземплярах, находило не любой, а обязательно первый (как вариант: последний; либо вообще не x, а следующий за ним элемент).[2] Данная версия кода в такой ситуации находит первый из равных.

Учёный Йон Бентли утверждает, что 90 % студентов, разрабатывая двоичный поиск, забывают учесть какое-либо из этих требований. И даже в код, написанный самим Йоном и ходивший из книги в книгу, вкралась ошибка: код не стоек к переполнениям[1].

## 5.2. Приложения

Практические приложения метода двоичного поиска разнообразны:

- Широкое распространение в информатике применительно к поиску в структурах данных. Например, поиск в массивах данных осуществляется по ключу, присвоенному каждому из элементов массива (в простейшем случае сам элемент является ключом).

- Также его применяют в качестве численного метода для нахождения приближённого решения уравнений (см. Метод бисекции).
- Метод используется для нахождения экстремума целевой функции и в этом случае является методом условной одномерной оптимизации. Когда функция имеет вещественный аргумент, найти решение с точностью до $\varepsilon$ можно за время $\log_2 1/\varepsilon$ . Когда аргумент дискретен, и изначально лежит на отрезке длины $N$, поиск решения займёт $1 + \log_2 N$ времени. Наконец, для поиска экстремума, скажем для определённости минимума, на очередном шаге отбрасывается тот из концов рассматриваемого отрезка, значение в котором максимально.

## 5.3. См. также

- Линейный поиск
- Метод бисекции
- Метод дихотомии
- Метод Ньютона
- Метод золотого сечения
- Троичный поиск

## 5.4. Ссылки

- Материал на сайте algolist
- Пошаговый разбор алгоритма, коды программ

## 5.5. Примечания

[1] Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken // Joshua Bloch, Google Research; перевод - Почти во всех реализациях двоичного поиска и сортировки слиянием есть ошибка

[2] В C++ std::lower_bound находит первое вхождение x, а std::upper_bound — элемент, следующий за x.

## 5.6. Литература


- *Ананий В. Левитин.* **Глава 4. Метод декомпозиции: Бинарный поиск** // Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Algorithms. — М.: «Вильямс», 2006. — С. 180-183. — ISBN 5-8459-0987-2.
- *Амосов А. А., Дубинский Ю. А., Копченова Н. П.* Вычислительные методы для инженеров. — М.: Мир, 1998.
- *Бахвалов Н. С., Жидков Н. П., Кобельков Г. Г.* Численные методы. — 8-е изд. — М.: Лаборатория Базовых Знаний, 2000.
- *Вирт Н.* Алгоритмы + структуры данных = программы. — М.: «Мир», 1985. — С. 28.
- *Волков Е. А.* Численные методы. — М.: Физматлит, 2003.
- *Гилл Ф., Мюррей У., Райт М.* Практическая оптимизация. Пер. с англ. — М.: Мир, 1985.
- *Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К.* Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с. — ISBN 5-8459-0857-4.
- *Корн Г., Корн Т.* Справочник по математике для научных работников и инженеров. — М.: Наука, 1970. — С. 575-576.
- *Коршунов Ю. М., Коршунов Ю. М.* Математические основы кибернетики. — Энергоатомиздат, 1972.
- *Максимов Ю. А., Филлиповская Е. А.* Алгоритмы решения задач нелинейного программирования. — М.: МИФИ, 1982.
- *Роберт Седжвик.* Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. — СПб.: ДиаСофтЮП, 2003. — С. 672. — ISBN 5-93772-081-4.

# Глава 6

# Двунаправленный поиск

**Двунаправленный поиск пути**[1][2] в ширину (или глубину) — усложнённый алгоритм поиска в ширину (или глубину), идея которого заключается в формировании процесса поиска от начальной (*прямой поиск*) и от конечной вершины (*обратный поиск*) графа.

## 6.1. Идея

Нахождение искомого пути сводится к определению путей от начальной к какой-то промежуточной, а от неё к конечной вершине. Реализуется проверкой в одном или обоих процессах, когда лист одного дерева поиска совпадёт с листом другого, после чего выделяются пути до этого элемента. Соединив пути получаем искомый путь. Если два поиска осуществляются параллельно — это ещё больше экономит время на получение искомого пути по сравнению с однонаправленным поиском.

## 6.2. Преимущества и недостатки

- Повышенное быстродействие;
- Нужна память для хранения дерева поиска для того, чтобы можно было выполнить проверку принадлежности листа другому дереву.

## 6.3. Оценка сложности исполнения

Сложность всего алгоритма оценивается как сумма сложности прямого и обратных поисков, проверки принадлежности, равной одной операции, постоянному времени (O(n)), обращению к хэш-таблице.

### 6.3.1. Подсчёт количества операций

Слишком зависит от конкретной ситуации, если поиск осуществляется не по n-арному дереву.

### 6.3.2. Асимптотическая сложность возрастания количества операций

- Если известны единственные конкретные начальный и целевой элементы, то временная асимптотическая сложность прямого и обратного поисков равна $O(b^{d/2})$ , следовательно общая — $O(b^{d/2})$ + $O(b^{d/2})$ , что гораздо меньше чем $O(b^d)$ . Пространственная асимптотическая сложность $O(b^{d/2})$ , вместо — $O(1)$ у прямого, так как нужно хранить в памяти.
- Если известны конкретный начальный элемент и набор элементов, из которого один — целевой.

### 6.3.3. Статистическая оценка

Двунаправленный поиск, при заданных единственных начального и конечного элементах, может улучшить однонаправленный поиск в ширину, обычно, в 2 раза. Наиболее сложным случаем для двунаправленного поиска является такая задача, в котором для проверки цели дано только неявное описание некоторого (возможно очень большого) множества целевых состояний, например всех состояний, соответствующих проверки цели extquotedblMat extquotedbl в шахматах. При обратном поиске потребовалось бы создать компактные описания всех состояний, которые позволяют поставить мат с помощью ходов $q1, q2, q3...$ и т.д. ; и эти описания нужно было бы сверять с состояниями, формируемыми при прямом поиске. Общего способа эффективного решения такой проблемы не существует.

## 6.4. Алгоритм двунаправленного поиска

Алгоритм состоит:

- прямого поиска, аналогичного одиночному поиску;
- обратного поиска;

- операции определения принадлежности листа другому дереву поиска.

## 6.5. Сложность реализации

Сложность реализации заключается в алгоритме обратного поиска.

## 6.6. Примеры реализации

## 6.7. Практическое применение

## 6.8. См. также

- Поиск в ширину
- Поиск в глубину

## 6.9. Примечания

[1] Другое: двунаправленный поиск элемента — осуществляется в двунаправленных или кольцевых списках от искомого элемента в обе стороны.

[2] Этот алгоритм является полным и оптимальным (при единообразных стоимостях этапов), если оба процесса поиска осуществляются в ширину; другие сочетания методов могут характеризоваться отсутствием полноты, оптимальности или того и другого.

## 6.10. Ссылки

- Алгоритмы поиска пути на pmg.org.ru
- Модели и методы решения задач на Марий Эл.ру
- Реализация на C++ (aisearch.tgz ) на www.cs.cmu.edu

## 6.11. Литература

- *Стюарт Рассел & Питер Норвиг* **Часть 2. Решение проблем** // Искусственный интеллект (Современный подход) = Artificial Intelligence (A Modern Approach). — 2-е изд. — ФГУП. extquotedblПечатный двор extquotedbl им. А.М.Горького: Вильямс, 2006. — С. 135-136. — ISBN 5-8459-0887-6.

# Глава 7

# Дихотомия

**Дихотоми́я** (греч. διχοτομία: δῐχῆ, «надвое» + τομή, «деление») — раздвоенность, последовательное деление на две части, не связанные между собой. Способ логического деления класса на подклассы, который состоит в том, что делимое понятие полностью делится на два взаимоисключающих понятия. Дихотомическое деление в математике, философии, логике и лингвистике является способом образования взаимоисключающих подразделов одного понятия или термина и служит для образования классификации элементов.

## 7.1. Пример

Объём понятия «*человек*» можно разделить на два взаимоисключающих класса: *мужчины* и *не мужчины*. Понятия «*мужчины*» и «*не мужчины*» являются противоречащими друг другу, поэтому их объёмы не пересекаются. От дихотомии следует отличать обычное деление, приводящее к тому же самому результату. Например, объём понятия «*человек*» можно разделить по признаку пола на *мужчин* и *женщин*. Но между понятиями *мужчина* и *женщина* нет логического противоречия, поэтому здесь нельзя говорить о дихотомическом делении.

## 7.2. Преимущества и недостатки

Дихотомическое деление привлекательно своей простотой. Действительно, при дихотомии мы всегда имеем дело лишь с двумя классами, которые исчерпывают объём делимого понятия. Таким образом, дихотомическое деление всегда соразмерно; члены деления исключают друг друга, так как каждый объект делимого множества попадает только в один из классов ***a*** или *не* ***a***; деление проводится по одному основанию — наличие или отсутствие некоторого признака. Обозначив делимое понятие буквой ***a*** и выделив в его объёме некоторый вид, скажем, ***b***, можно разделить объём ***a*** на две части — ***b*** и *не* ***b***.

Дихотомическое деление имеет недостаток: при делении объёма понятия на два противоречащих понятия каждый раз остаётся крайне неопределённой та его часть, к которой относится частица «не». Если разделить учёных на *историков* и *не историков*, то вторая группа оказывается весьма неясной. Кроме того, если в начале дихотомического деления обычно довольно легко установить наличие противоречащего понятия, то по мере удаления от первой пары понятий найти его становится всё труднее.

## 7.3. Применение

Дихотомия обычно используется как вспомогательный приём при установлении классификации.

Она известна также благодаря достаточно широко используемому методу поиска, так называемому ***методу дихотомии***. Он применяется для нахождения значений действительно-значной функции, определяемых по какому-либо критерию (это может быть сравнение на минимум, максимум или конкретное число). Рассмотрим метод дихотомии условной одномерной оптимизации (для определённости минимизации).

### 7.3.1. Метод дихотомии

Метод дихотомии несколько схож с методом бисекции, однако отличается от него критерием отбрасывания концов.

Пусть задана функция $f(x) : \ [a,\ b] \to \mathrm{R},\ f(x) \in \mathrm{C}([a,\ b])$.

Разобьём мысленно заданный отрезок пополам и возьмём две симметричные относительно центра точки $x_1$ и $x_2$ так, что:

$$\begin{aligned} x_1 &= \tfrac{a+b}{2} - \delta, \\ x_2 &= \tfrac{a+b}{2} + \delta, \end{aligned}$$

где $\delta$ — некоторое число в интервале $\left(0,\ \tfrac{b-a}{2}\right)$.

Вычислим два значения функции $f(x)$ в двух новых точках. Сравнением определим в какой из двух новых точек значение функции $f(x)$ максимально. Отбросим тот из концов изначального отрезка, к которому

точка с максимальным значением функции оказалась ближе (напомним, мы ищем минимум), то есть:

- Если $f(x_1) > f(x_2)$, то берётся отрезок $[x_1,\ b]$, а отрезок $[a,\ x_1]$ отбрасывается.
- Иначе берётся зеркальный относительно середины отрезок $[a,\ x_2]$, а отбрасывается $[x_2,\ b]$.

Процедура повторяется, пока не будет достигнута заданная точность, к примеру, пока длина отрезка не достигнет удвоенного значения заданной погрешности.

На каждой итерации приходится вычислять новые точки. Можно добиться того, чтобы на очередной итерации было необходимо высчитывать лишь одну новую точку, что заметно способствовало бы оптимизации процедуры. Это достигается путём зеркального деления отрезка в золотом сечении, в этом смысле метод золотого сечения можно рассматривать, как улучшение метода дихотомии с параметром $\delta = (b - a)\left(\frac{1}{2} - \frac{1}{\phi}\right)$, где $\phi = \frac{\sqrt{5}+1}{2} \approx 1{,}6180339887\ldots$ — золотое сечение.

## 7.4. См. также

- Линейный поиск
- Двоичный поиск
- Метод бисекции
- Метод золотого сечения
- Троичный поиск

## 7.5. Литература


1. *Ананий В. Левитин.* Глава 11. Преодоление ограничений: Метод деления пополам // [= 0-201-74395-7 Алгоритмы: введение в разработку и анализ] = Introduction to The Design and Analysis of Aigorithms. — М.: «Вильямс», 2006. — С. 476-480.
2. *Акулич И. Л.* Математическое программирование в примерах и задачах: Учеб. пособие для студентов эконом. пец. вузов. — М.: Высш. шк., 1986.
3. *Амосов А. А., Дубинский Ю. А., Копченова Н. П.* Вычислительные методы для инженеров. — М.: Мир, 1998.
4. *Бахвалов Н. С., Жидков Н. П., Кобельков Г. Г.* Численные методы. — 8-е изд. — М.: Лаборатория Базовых Знаний, 2000.
5. *Волков Е. А.* Численные методы. — М.: Физматлит, 2003.
6. *Гилл Ф., Мюррей У., Райт М.* Практическая оптимизация. Пер. с англ. — М.: Мир, 1985.
7. *Корн Г., Корн Т.* Справочник по математике для научных работников и инженеров. — М.: Наука, 1970. — С. 575-576.
8. *Коршунов Ю. М., Коршунов Ю. М.* Математические основы кибернетики. — Энергоатомиздат, 1972.
9. *Максимов Ю. А., Филлиповская Е. А.* Алгоритмы решения задач нелинейного программирования. — М.: МИФИ, 1982.
10. *Максимов Ю. А.* Алгоритмы линейного и дискретного программирования. — М.: МИФИ, 1980.

# Глава 8

# Задача поиска ближайшего соседа

*Другие значения этого понятия см. в статье ближайший сосед*

**Задача поиска ближайшего соседа** заключается в отыскании среди множества элементов, расположенных в многомерном метрическом пространстве, элементов близких к заданному, согласно некоторой функции близости.

## 8.1. Приложения

Задача поиска ближайшего соседа встречается во множестве приложений, например в областях:

- распознавание образов;
- классификация текстов;
- Рекомендательные и экспертные системы;
- динамическое размещение рекламы в Интернете.

## 8.2. Модели данных

Перед решением прикладной задачи, необходимо выбрать форму представления объектов и функцию близости. В большинстве случаев, объекты представляются в виде многомерных векторов, а в качестве функции близости используется скалярное произведение векторов, но могут быть и другие формы представления данных, например:

- множества — размер пересечения множеств
- Строки — расстояние Левенштейна.
- Граф — соответствие структур.

## 8.3. Виды целей

Помимо классической задачи отыскания ближайшей к заданной точке, могут быть поставлены задачи:

- найти приблизительных ближайших соседей (не обязательно наиболее близкого);
- найти ближайшего соседа для группы элементов;
- найти несколько ближайших соседей;
- найти все пары элементов, расстояние между которыми меньше некоторого заданного;
- найти ближайших соседей в динамически меняющейся среде.

## 8.4. Алгоритмы

### 8.4.1. Разбиение пространства

- Диаграммы Вороного
- KD-деревья
- BSP-деревья
- Деревья покрытий
- VP-деревья
- R-деревья

### 8.4.2. Обратный индекс

Метод редких точек

## 8.5. См. также

- Метод k ближайших соседей
- Диаграммы Вороного

## 8.6. Ссылки

- Yury Lifshits. Algorithms for Nearest Neighbors: Theoretical Aspects
- Могилко А. А. Параллельный алгорит поиска ближайшей точки в радиусе

# Глава 9

# Интерполирующий поиск

**Интерполирующий поиск** основан на принципе поиска в телефонной книге или, например, в словаре. Вместо сравнения каждого элемента с искомым как при линейном поиске, данный алгоритм производит предсказание местонахождения элемента: поиск происходит подобно двоичному поиску, но вместо деления области поиска на две части, интерполирующий поиск производит оценку новой области поиска по расстоянию между ключом и текущим значением элемента. Другими словами, бинарный поиск учитывает лишь знак разности между ключом и текущим значением, а интерполирующий ещё учитывает и модуль этой разности и по данному значению производит предсказание позиции следующего элемента для проверки. В среднем, интерполирующий поиск производит log(log($N$)) операций, где $N$ есть число элементов, среди которых производится поиск. Число необходимых операций зависит от равномерности распределения значений среди элементов. В плохом случае (например, когда значения элементов экспоненциально возрастают) интерполяционный поиск может потребовать до O($N$) операций.

На практике, интерполяционный поиск часто быстрее бинарного, так как с вычислительной стороны их отличают лишь применяемые арифметические операции: интерполирование — в интерполирующем поиске и деление на два — в двоичном, а скорость их вычисления отличается незначительно, с другой стороны интерполирующий поиск использует такое принципиальное свойство данных, как однородность распределения значений. Ключом может быть не только номер, число, но и, например, текстовая строка, тогда становится понятна аналогия с телефонной книгой: если мы ищем имя в телефонной книге, начинающееся на «А», следовательно нужно искать его в начале, но никак не в середине. В принципе, ключом может быть всё что угодно, так как те же строки, например, запросто кодируются посимвольно, в простейшем случае символ можно закодировать значением от 1 до 33 (только русские символы) или, например, от 1 до 26 (только латинский алфавит) и т. д.

Интерполяция может производиться на основе функции, аппроксимирующей распределение значений, либо набора кривых, выполняющих аппроксимацию на отдельных участках. В этом случае поиск может завершиться за несколько проверок. Преимущества этого метода состоят в уменьшении запросов на чтение медленной памяти (такой как, например, жесткий диск), если запросы происходят часто.

Часто, анализ и построение аппроксимирующих кривых не требуется, показательный случай здесь — когда все элементы отсортированы по возрастанию. В таком списке, минимальное значение будет по индексу 1, а максимальное по индексу $N$. В этом случае аппроксимирующую кривую можно принять за прямую и применять линейную интерполяцию.

Следующий пример на Java показывает простейшую реализацию интерполирующего поиска. На каждой стадии алгоритм рассчитывает позицию для следующей проверки, как при двоичном поиске, переносит верхнюю или нижнюю границу, определяя тем самым новую область поиска, содержащую искомое значение.

public int interpolationSearch(int[] sortedArray, int toFind) { // Возвращает индекс элемента со значением toFind или −1, если такого элемента не существует int mid; int low = 0; int high = sortedArray.length - 1; while (sortedArray[low] < toFind && sortedArray[high] > toFind) { mid = low + ((toFind - sortedArray[low]) * (high - low)) / (sortedArray[high] - sortedArray[low]); if (sortedArray[mid] < toFind) low = mid + 1; else if (sortedArray[mid] > toFind) high = mid - 1; else return mid; } if (sortedArray[low] == toFind) return low; else if (sortedArray[high] == toFind) return high; else return −1; // Not found }

## 9.1. См. также

- Линейный поиск
- Двоичный поиск
- Троичный поиск
- Хеш-таблица

## 9.2. Ссылки

- Interpolation search (англ.)
- National Institute of Standards and Technology (англ.)

## 9.3. Литература


- *Ананий В. Левитин* **Глава 5. Метод уменьшения размера задачи: Интерполяционный поиск** // Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Algorithms. — М.: «Вильямс», 2006. — С. 240-242. — ISBN 0-201-74395-7.
- *Роберт Седжвик* Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск = Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. — СПб.: ДиаСофтЮП, 2003. — С. 672. — ISBN 5-93772-081-4.

# Глава 10

# Ключ для определения

**Ключ для определения** в биологии — это описанный или реализованный в CAE-системе алгоритм, служащий для помощи в идентификации биологических сущностей, таких как растения, животные, фоссилии, микроорганизмы, зёрна пыльцы. Такие ключи, кроме того, нашли широкое применение в различных областях науки и техники для идентификации разного вида сущностей, например заболеваний, почв, минералов, археологических и антропологических артефактов.

Обычно ключ для определения представляет собой форму ключевого пути (англ.)русск., то есть предлагает выполнить фиксированное число идентификационных шагов, каждый из которых предполагает выбор одной из нескольких альтернатив. Сделанный на текущем шаге выбор определяет следующий шаг в последовательности определения. Если в каждом шаге ключа делается выбор из двух вариантов, ключ называют **дихотомическим**, если больше — **политомическим**. Современные интерактивные ключи (англ.)русск. позволяют гибко выбирать необходимые шаги идентификации и их порядок следования.

В каждом шаге идентификации оператор должен ответить на вопрос об одной или нескольких характеристиках исследуемого объекта. Например, в шаге ключа ботаники может содержаться вопрос об окраске цветков растения или о типе расположения листьев на стебле. Ключ для определения насекомого может содержать в себе шаг с вопросом о количестве щетинок на задней ноге исследуемого насекомого.

## 10.1. Принципы разработки качественного ключа

Ошибки определения могут иметь серьёзные последствия как в фундаментальных, так и в прикладных дисциплинах, таких как экология, медицинская диагностика, контроль за вредителями, судебных экспертизах и т. д.[1] Поэтому ключи для определения должны создаваться очень аккуратно, так, чтобы свести вероятность ошибки к минимуму.

Всегда, когда это возможно, характеристика, используемая в шаге идентификации, должна быть *диагностической*, то есть быть применима для всех сущностей в отсекаемой группе и быть уникальной для этой группы. Также характеристика должна быть *дифференцирующей*, то есть позволять однозначно ограничить группу сущностей, отсекаемую от всего числа сущностей. Однако характеристики, не являющиеся ни диагностическими, ни дифференцирующими, всё же могут использоваться для уточнения (например, характеристики, общие для всех сущностей в группе, но не уникальные).

Всегда, когда это возможно, в шаге идентификации должны использоваться избыточные характеристики. Например, если группа делится на две подгруппы, одна из которых характеризуется шестью чёрными точками, а вторая четырьмя коричневыми полосками, должен использоваться запрос всех трех характеристик (количество, цвет, форма отметин), даже в том случае, если теоретически даже одной характеристики вполне хватает. Эта избыточность повышает надёжность идентификации, страхуя от ошибок оператора и позволяя продолжать определение, даже если какие-то из характеристик невозможно по какой-то причине оценить. В таком случае характеристики должны быть упорядочены в соответствии с их влиянием на надёжность определения и лёгкостью их оценки. Дополнительное увеличение точности определения может быть достигнуто применением сетки ключей (англ.)русск..

Термины, используемые при построении ключа, должны соответствовать области знаний, для которой создается ключ, и не должны допускать двоякой интерпретации. Использование различных вариантов обозначения одного и того же для «литературности слога» недопустимо. Утверждения более предпочтительны, чем отрицания. Альтернативные варианты должны иметь одинаковую широту охвата характеристик. К примеру, варианты: «красные цветки размером 10-40 мм» и «жёлтые цветки» включают в себя разное количество характеристик. Такого следует избегать.

Характеристики, определяющие географическое распространение, следует применять с большой осторожностью. Виды, которые до сих пор не встречались

в регионе, вполне могут всё же находиться в нём, либо быть по какой-то причине завезены. Также нельзя исключать изменения ареала обитания в связи с изменениями условий (например, при глобальном потеплении).

Редкость не является значимой характеристикой. Идентификация должна быть корректной даже для очень редких видов.

### 10.1.1. Общие проблемы использования ключей

При использовании ключей для определения можно столкнуться со множеством практических проблем, например:

- *Различие форм*: Ключ может позволять определять только некоторые из форм одного вида, например, только мужские особи (или, реже, только женские). Ключ для определения вида личинок может быть ограничен только одной из возрастных стадий. (Это не касается, однако, ключей, разработанных для судебной энтомологии.)
- *Неполнота охвата*: Виды и группы, определение которых по каким-либо причинам сильно затруднено, либо которые просто слабо изучены, могут отсутствовать в ключе или быть упомянуты лишь парой слов.
- *Условия наблюдения*: Только очень немногие из ключей содержат информацию об требуемых условиях наблюдения (освещение, увеличение, угол зрения и др.), что может привести к проблеме. Например, требуется указать наличие тонких щетинок или волосков. Но, насколько тонкими должны быть щетинки, чтобы сказать, что они в наличии?
- *Язык*: Большинство идентификационных ключей существуют только на одном языке[2]. Перевод ключа может быть некорректным или неполным. Многие ключи содержат термины, не имеющие точных аналогов в других языках.
- *Устаревание*: В старых ключах может отсутствовать информация о недавно описанных видах. Также подобные ключи могут содержать устаревшие и уже не используемые наименования видов.

## 10.2. Проверка правильности определения

Определение, выполненное с помощью ключа, должно восприниматься только как предположение. Уверенность в правильности определения вида возможна только после сличения образца с описанием из авторитетного источника, например, полного и подробного описания вида, предпочтительно в монографии. Большинство ключей содержат краткое описание, позволяющее в какой-то мере подтвердить правильность идентификации, но недостаточное для принятия окончательного решения.

Подтверждение правильности идентификации с использованием монографий зачастую затруднено, так как монографии могут быть дороги, выпущены крайне малым тиражом, вследствие чего труднодоступны, либо просто сложны для понимания. Наконец, они могут быть выпущены на иностранном языке. Бывает, что монография выпущена десятки лет назад и наименования видов, используемые в ключе, не совпадают с приведёнными в монографии.

Альтернативой сравнению с описанием из монографии является идентификация вида в музеях естествознания или других подобных хранилищах. В последнее время получили широкое распространение заверенные авторитетными источниками фотографии, доступные через Интернет. Подтверждением истинности фотографии является номер паспорта образца, имя учёного, заверившего фотографию, и наименование государственного учреждения, где хранится образец, изображённый на фотографии (это может быть особенно интересно тем, кто хочет самостоятельно повторить идентификацию образца).

## 10.3. Программные реализации

- Lucid — программное обеспечение для интерактивной идентификации и диагностики (англ.)
- Linnaeus II — программное обеспечение для интерактивной идентификации и управления базой описаний (англ.)
- Xper2 — программное обеспечение для интерактивной идентификации и управления базой описаний (англ.)
- Веб-сервис ключей для определения (англ.)
- Онлайн система визуальной идентификации трав Луизианы (англ.)
- Discover Life — онлайн система интерактивной идентификации вида

## 10.4. См. также

Биологическая систематика

## 10.5. Ссылки

- *Dallwitz, M.J., Paine, T.A., Zurcher E.J.* Principles of interactive keys (англ.) (27 февраля 2012). Проверено 30 ноября 2012. Архивировано из первоисточника 30 января 2013.
- *Dallwitz, M.J.* Programs for Interactive Identification and Information Retrieval (англ.) (1 сентября 2011). Проверено 30 ноября 2012. Архивировано из первоисточника 30 января 2013.

## 10.6. Примечания

[1] *Стив Маршалл.* Comments on error rates in insect identifications (англ.) (2000). — Newsletter of the Biological Survey of Canada (Terrestrial Arthropods). Проверено 30 ноября 2012. Архивировано из первоисточника 30 января 2013.

[2] *Richter, H.G., Dallwitz, M.J.* Commercial timbers: descriptions, illustrations, identification, and information retrieval. In English, French, German, Portuguese, and Spanish. (англ.) (25 июня 2009). Проверено 30 ноября 2012. Архивировано из первоисточника 30 января 2013. — пример ключа для определения вида коммерческой древесины с пятью языковыми версиями

# Глава 11

# Линейный поиск

**Линейный, последовательный поиск** — алгоритм нахождения заданного значения произвольной функции на некотором отрезке. Данный алгоритм является простейшим алгоритмом поиска и в отличие, например, от двоичного поиска, не накладывает никаких ограничений на функцию и имеет простейшую реализацию. Поиск значения функции осуществляется простым сравнением очередного рассматриваемого значения (как правило поиск происходит слева направо, то есть от меньших значений аргумента к большим) и, если значения совпадают (с той или иной точностью), то поиск считается завершённым.

Если отрезок имеет длину N, то найти решение с точностью до $\epsilon$ можно за время $\frac{N}{\epsilon}$ . Т.о. асимптотическая сложность алгоритма — $O(n)$ . В связи с малой эффективностью по сравнению с другими алгоритмами линейный поиск обычно используют только если отрезок поиска содержит очень мало элементов, тем не менее линейный поиск не требует дополнительной памяти или обработки/анализа функции, так что может работать в потоковом режиме при непосредственном получении данных из любого источника. Так же, линейный поиск часто используется в виде линейных алгоритмов поиска максимума/минимума.

В качестве примера можно рассмотреть поиск значения функции на множестве целых чисел, представленной таблично.

## 11.1. Пример

Переменные $L$ и $R$ содержат, соответственно, левую и правую границы отрезка массива, где находится нужный нам элемент. Исследования начинаются с первого элемента отрезка. Если искомое значение не равно значению функции в данной точке, то осуществляется переход к следующей точке. Т.е., в результате каждой проверки область поиска уменьшается на один элемент.

int function LinearSearch (Array A, int L, int R, int Key); begin for X = L to R do if A[X] = Key then return X return $-1$; // элемент не найден end;

## 11.2. Анализ

Для списка из $n$ элементов лучшим случаем будет тот, при котором искомое значение равно первому элементу списка и требуется только одно сравнение. Худший случай будет тогда, когда значения в списке нет (или оно находится в самом конце списка), в случае чего необходимо $n$ сравнений.

Если искомое значение входит в список $k$ раз и все вхождения равновероятны, то ожидаемое число сравнений

$$\begin{cases} n & k = 0 \\ \dfrac{n+1}{k+1} & 1 \leq k \leq n. \end{cases}$$

Например, если искомое значение встречается в списке один раз, и все вхождения равновероятны, то среднее число сравнений равно $\frac{n+1}{2}$ . Однако, если *известно*, что оно встречается один раз, то достаточно $n$ - 1 сравнений и среднее число сравнений будет равняться

$$\frac{(n+2)(n-1)}{2n}$$

(для $n$ = 2 это число равняется 1, что соответствует одной конструкции if-then-else).

В любом случае, вычислительная сложность алгоритма O($n$).

## 11.3. Приложения

Обычно линейный поиск очень прост в реализации и применим, если список содержит мало элементов, либо в случае одиночного поиска в неупорядоченном списке.

Если предполагается в одном и том же списке большое число раз, то часто имеет смысл предварительной обработки списка, например сортировки и последующего использования бинарного поиска, либо

построения какой-либо эффективной структуры данных для поиска. Частая модификация списка также может оказывать влияние на выбор дальнейших действий, поскольку делает необходимым процесс перестройки структуры.

## 11.4. Литература


- *Дональд Кнут* Искусство программирования, том 3. Сортировка и поиск = The Art of Computer Programming, vol.3. Sorting and Searching. — 2-е изд. — М.: «Вильямс», 2007. — С. 824. — ISBN 0-201-89685-0.


## 11.5. См. также

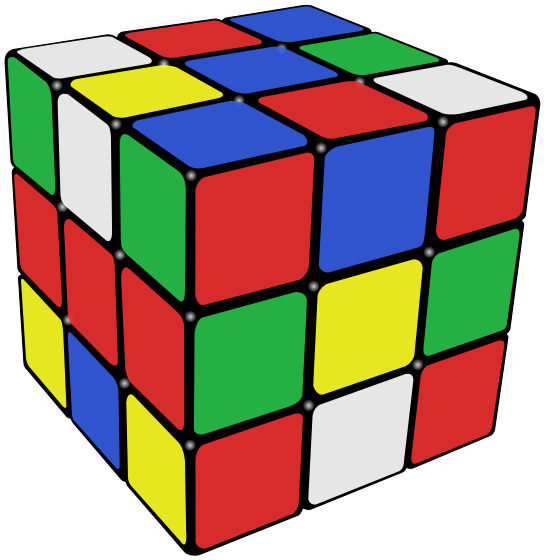
- Двоичный поиск
- Метод дихотомии
- Метод золотого сечения
- Троичный поиск

## 11.6. Ссылки

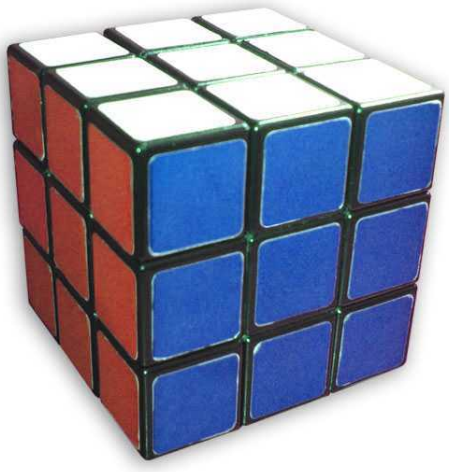
- http://algolist.manual.ru

# Глава 12

# Математика кубика Рубика



*Запутанный кубик Рубика*



*Собранный кубик Рубика*

**Матема́тика ку́бика Ру́бика** — совокупность математических методов для изучения свойств кубика Рубика с абстрактно-математической точки зрения. Изучает алгоритмы сборки кубика, оценки алгоритмов его сборки и др. Основана на теории графов, теории групп, теории вычислимости, комбинаторике.

Существует множество алгоритмов, предназначенных для перевода кубика Рубика из произвольной конфигурации в *конечную конфигурацию* (собранную, все грани одноцветны). В 2010 г. строго доказано, что для перевода кубика Рубика из произвольной конфигурации в собранную конфигурацию (часто этот процесс называют «сборкой» или «решением») достаточно не более чем 20 поворотов граней (ходов)[1]. Это число является диаметром графа Кэли группы кубика Рубика. Алгоритм, который решает головоломку за минимально возможное количество ходов, называют алгоритмом Бога.

## 12.1. Нотация

В этой статье для обозначения последовательности поворотов граней кубика Рубика 3×3×3 используется «нотация Сингмастера»[2], разработанная Дэвидом Сингмастером и опубликованная им в 1981 г.

Буквы $L, R, F, B, U, D$ обозначают поворот на 90° по часовой стрелке левой (*left*), правой (*right*), передней (*front*), задней (*back*), верхней (*up*) и нижней (*down*) граней соответственно. Повороты на 180° обозначаются добавлением справа к букве цифры 2 или добавлением в верхнем индексе цифры 2 справа от буквы. Поворот на 90° против часовой стрелки обозначается добавлением штриха ( ′ ) или добавлением в верхнем индексе $-1$ справа от буквы. Так, например, записи $L^2$ и $L2$ ; $L'$ и $L^{-1}$ эквивалентны.

## 12.2. Метрики графа конфигураций

Существует два наиболее распространённых способа измерения длины решения (метрики). Первый способ — одним шагом (ходом) решения считается поворот грани на 90° (*quarter turn metric*, **QTM**). По второму способу — за 1 ход также считается и полуоборот грани (*face turn metric*, **FTM**, иногда это обозначают

**HTM** — *half-turn metric*). Так, F2 (поворот передней грани на 180°) должен считаться за два хода в метрике QTM или за 1 ход в метрике FTM[3][4].

Для указания в тексте длины последовательности для используемой метрики используется нотация[5][6][7], состоящая из цифр числа ходов и строчной первой буквы обозначения метрики. Так, **14f** обозначает «14 ходов в метрике FTM», а **10q** — «10 ходов в метрике QTM». Чтобы указать, что количество ходов является минимальным в данной метрике, используется звёздочка: **10f*** обозначает оптимальность решения в 10 ходов FTM.

## 12.3. Группа кубика Рубика

Кубик Рубика может рассматриваться как пример математической группы.

Каждый из шести поворотов граней кубика Рубика может рассматриваться как элемент симметрической группы множества 48 этикеток кубика Рубика, не являющихся центрами граней. Более конкретно, можно пометить все 48 этикеток числами от 1 до 48 и сопоставить каждому из ходов $\{F, B, U, D, L, R\}$ элемент симметрической группы $S_{48}$ .

Тогда группа кубика Рубика $G$ определяется как подгруппа $S_{48}$ , порождаемая шестью поворотами граней:

$$G = \langle F, B, U, D, L, R \rangle$$

Порядок группы $G$ равен[8][9]

$$|G| = \frac{8! \cdot 12! \cdot 3^8 \cdot 2^{12}}{3 \cdot 2 \cdot 2} = 43\,252\,003\,274\,489\,856\,000 = 2^{27} \cdot 3^{14} \cdot 5^3 \cdot 7^2 \cdot 11$$

Каждая из $4{,}32 \cdot 10^{19}$ конфигураций может быть решена не более чем за 20 ходов (если считать за ход любой поворот грани)[1].

Наибольший порядок элемента в $G$ равен 1260. Например, последовательность ходов $(R\ U^2\ D'\ B\ D')$ необходимо повторить 1260 раз, прежде чем кубик Рубика вернётся в исходное состояние[2].

## 12.4. Алгоритм Бога

История поиска алгоритма Бога кубика Рубика началась не позже 1980 года, когда открылся список рассылки для любителей кубика Рубика[3]. С тех пор математики, программисты и просто любители стремились найти алгоритм Бога — алгоритм, который бы позволил на практике решать кубик Рубика за минимальное число ходов. С этой проблемой была связана проблема определения числа Бога — числа ходов, всегда достаточного для сборки головоломки.

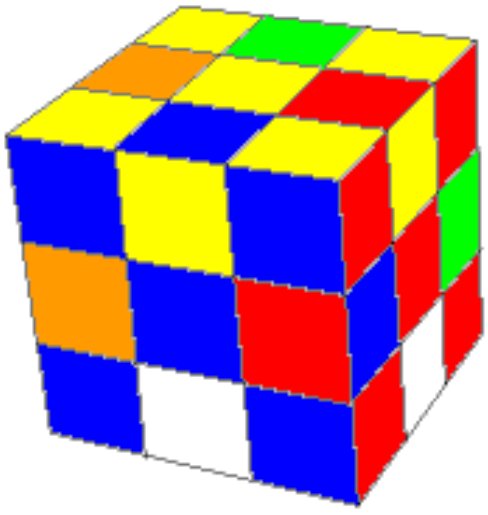
В июле 2010 года программист из Пало-Альто Томас Рокики, учитель математики из Дармштадта Герберт Коцемба, математик из Кентского университета Морли Дэвидсон и инженер компании Google Inc. Джон Детридж доказали, что каждая конфигурация кубика Рубика может быть решена не более чем в 20 ходов. При этом любой поворот грани считался одним ходом. Таким образом, число Бога в метрике FTM оказалось равно 20 ходам. Объём вычислений составил около 35 лет процессорного времени, пожертвованного компанией Google[1][10]. Технические данные о производительности и количестве компьютеров не разглашаются; продолжительность вычислений составляла несколько недель[11][12][13].

### 12.4.1. Нижние оценки числа Бога

Достаточно легко показать, что существуют разрешимые конфигурации[14], которые не могут быть решены менее чем в 17 ходов в метрике FTM или 19 ходов в метрике QTM.

Доказательство для метрики FTM

Эту оценку можно улучшить, принимая во внимание дополнительные тождества, например, коммутативность поворотов двух противоположных граней (L R = R L, L2 R = R L2 и т. д.) Подобный подход позволяет получить нижнюю оценку для числа Бога, равную 18f или 21q[15][16].



*«Суперфлип» — первая обнаруженная конфигурация, находящаяся на расстоянии 20f* от начальной*

Эта оценка в течение многих лет оставалась наилучшей известной. Кроме того, она вытекает из неконструктивного доказательства, так как оно не указы-

вает конкретный пример конфигурации, требующей для сборки 18f или 21q.

Одной из конфигураций, для которой не удавалось найти короткое решение, был так называемый «суперфлип» (англ.), или «12-флип». «Суперфлип» представляет собой конфигурацию, в которой все угловые и рёберные кубики находятся на своих местах, но каждый рёберный кубик ориентирован противоположно[17].

Вершина, отвечающая суперфлипу в графе кубика Рубика, является локальным максимумом: любой ход из этой конфигурации уменьшает расстояние до начальной конфигурации. Это дало основание предположить, что суперфлип находится на максимальном расстоянии от начальной конфигурации, то есть является глобальным максимумом[18][19][20].

В 1992 году Дик Т. Винтер нашёл решение суперфлипа в 20f[21]. В 1995 году Майкл Рид доказал оптимальность этого решения[22], в результате чего нижняя оценка числа Бога стала равной 20 FTM. В том же году Майкл Рид обнаружил решение «суперфлипа» в 24q[23]. Оптимальность этого решения была доказана Джерри Брайаном[24].

В 1998 году Майкл Рид нашёл конфигурацию, оптимальное решение которой составляло 26q*[25]. По состоянию на июль 2013 года, это число является наилучшей известной нижней оценкой числа Бога в метрике QTM.

## 12.4.2. Верхние оценки числа Бога

Чтобы получить оценку сверху для числа Бога, достаточно указать любой алгоритм сборки головоломки, состоящий из конечного числа ходов.

Первые оценки сверху для числа Бога были основаны на «человеческих» алгоритмах, состоящих из нескольких этапов. Сложение оценок сверху для каждого из этапов позволяло получить итоговую оценку порядка нескольких десятков или сотен ходов.

Вероятно, впервые конкретная оценка сверху была указана Дэвидом Сингмастером в 1979 году. Его алгоритм сборки позволял решить кубик Рубика не более чем за 277 ходов[11][26]. Позднее Сингмастер сообщил, что Элвин Берлекэмп (англ.)русск., Джон Конвей и Ричард Гай (англ.)русск. разработали алгоритм сборки, требующий не более 160 ходов. Вскоре после этого группа «Conway's Cambridge Cubists», которая занималась составлением списка комбинаций для одной грани, нашла 94-ходовый алгоритм[11][27].

В 1982 году в журнале «Квант» был опубликован список комбинаций, позволяющих решить кубик Рубика в 79 ходов[28].

**Алгоритм Тистлетуэйта**

В начале 1980-х английский математик Морвин Тистлетуэйт (англ.)русск. разработал алгоритм, позволивший значительно улучшить верхнюю оценку. Детали алгоритма были опубликованы Дугласом Хофштадтером в 1981 году в журнале *Scientific American*. Алгоритм был основан на теории групп и включал в себя четыре этапа.

**Описание** Тистлетуэйт использовал ряд подгрупп длины 4

$$G = G_0 \supseteq G_1 \supseteq G_2 \supseteq G_3 \supseteq G_4 = \{1\}$$ , где:

- $G_0 = \langle L, R, F, B, U, D \rangle$

  Эта группа совпадает с группой кубика Рубика $G$ . Её порядок равен[29][30]

$$\frac{8! \cdot 3^8}{3} \cdot \frac{12! \cdot 2^{12}}{2} \cdot \frac{1}{2} = 43\ 252\ 003\ 274\ 489\ 856\ 000$$

- $G_1 = \langle L^2, R^2, F, B, U, D \rangle$

  Эта подгруппа включает в себя все конфигурации, которые могут быть решены без использования поворотов левой или правой граней на ±90°. Её порядок равен

$$\frac{8! \cdot 3^8}{3} \cdot 12! \cdot \frac{1}{2} = 21\ 119\ 142\ 223\ 872\ 000$$

- $G_2 = \langle L^2, R^2, F^2, B^2, U, D \rangle$

  Эта подгруппа включает в себя все конфигурации, которые могут быть решены при условии, что повороты четырёх вертикальных граней на ±90° запрещены. Её порядок равен

$$8! \cdot (8! \cdot 4!) \cdot \frac{1}{2} = 19\ 508\ 428\ 800$$

- $G_3 = \langle L^2, R^2, F^2, B^2, U^2, D^2 \rangle$

  Эта подгруппа включает в себя все конфигурации, которые могут быть решены с использованием только поворотов на 180° (half-turns). Она получила название «группа квадратов» (squares group). Её порядок равен

$$(4! \cdot 4) \cdot \frac{4! \cdot 4! \cdot 4!}{2} \cdot 1 = 663\ 552$$

- $G_4 = \{1\}$

  Эта подгруппа включает в себя единственную начальную конфигурацию.

На первом этапе произвольно заданная конфигурация из группы $G$ приводится к конфигурации, лежащей в подгруппе $G_1$ . Цель второго этапа состоит в том, чтобы привести кубик к конфигурации, находящейся в подгруппе $G_2$ , не используя поворотов левой и правой граней на ±90°. На третьем этапе кубик Рубика приводится к конфигурации, принадлежащей группе $G_3$ , при этом повороты вертикальных граней на ±90° запрещены. На заключительном, четвёртом этапе, кубик Рубика полностью собирается поворотами граней на 180°.

Индексы подгрупп $[G_i : G_{i+1}]$ при $i = 0, 1, 2, 3$ равны соответственно 2048, 1082565, 29400 и 663552.

Для каждого из четырёх семейств правых смежных классов $G_i/G_{i+1}$ строится таблица поиска $T_i$ , размер которой совпадает с индексом подгруппы $G_{i+1}$ в группе $G_i$ . Для каждого класса смежности по подгруппе $G_i$ таблица поиска $T_i$ содержит последовательность ходов, переводящую любую конфигурацию из этого класса смежности в конфигурацию, лежащую в самой подгруппе $G_i$ .

Чтобы уменьшить количество записей в таблицах поиска, Тистлетуэйт использовал упрощающие предварительные ходы. Первоначально он получил оценку в 85 ходов. В течение 1980 года эта оценка была снижена до 80 ходов, затем до 63 и 52[11][31]. Студенты Тистлетуэйта провели полный анализ каждого из этапов. Максимальные значения в таблицах равны 7, 10, 13 и 15 ходам FTM соответственно. Так как 7 + 10 + 13 + 15 = 45, кубик Рубика всегда может быть решён в 45 ходов FTM[32][33][20].

**Граф Шрайера** Граф Шрайера (англ.)русск. — граф $S(G, X, H)$ , ассоциированный с группой $G$ , порождающим множеством $X = \{x_i | i \in I\}$ и подгруппой $H \subseteq G$ . Каждая вершина графа Шрайера является правым смежным классом $Hg = \{hg | h \in H\}$ для некоторого $g \in G$ . Рёбра графа Шрайера представляют собой пары $(Hg, Hgx_i)$ .

Каждый из четырёх этапов алгоритма Тистлетуэйта представляет собой обход в ширину графа Шрайера $S_i(G_i, X_i, G_{i+1})$ , где $X_i$ — порождающее множество группы $G_i$ .

Таким образом, верхняя оценка в 45 ходов равна

$$\sum_{i=0}^{3}(\epsilon(v_i))$$

где

$\epsilon(v_i)$ — эксцентриситет вершины $v_i \in S_i$ , соответствующей единичному смежному классу $G_{i+1}$ .

Понятие графа Шрайера было использовано в работах Раду[34], Канкла и Купермана[35].

### Модификации алгоритма Тистлетуэйта

В декабре 1990 года Ханс Клоостерман использовал модифицированную версию метода Тистлетуэйта для доказательства достаточности 42 ходов[15][1][36].

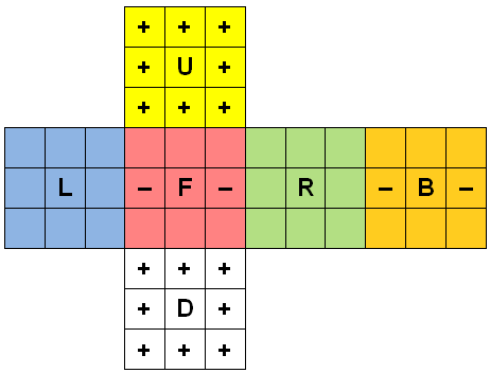
В мае 1992 года Майкл Рид показал достаточность 39f или 56q[37]. В его модификации использовалась следующая цепочка подгрупп:

- $G_0 = \langle U, F, R, L, B, D \rangle$
- $G_1 = \langle U, R, F \rangle$
- $G_2 = \langle U, R^2, F^2 \rangle$
- $G_3 = \{1\}$

Уже через несколько дней Дик Т. Винтер снизил оценку в метрике FTM до 37 ходов[38].

В декабре 2002 года Райан Хайз разработал вариант алгоритма Тистлетуэйта, предназначенный для скоростной сборки кубика Рубика[39].

### Двухфазный алгоритм Коцембы




*Промежуточное состояние кубика Рубика в алгоритме Коцембы. Ходы, разрешённые на втором этапе, сохраняют расположение меток «+» и «—»*

Алгоритм Тистлетуэйта был в 1992 году улучшен учителем математики из Дармштадта Гербертом Коцембой.

**Описание** Коцемба сократил количество этапов алгоритма до двух[15][40][41]:

- $G_0 = \langle U, D, L, R, F, B \rangle$
- $G_1 = \langle U, D, L^2, R^2, F^2, B^2 \rangle$
- $G_2 = \{1\}$

Наглядное описание группы $G_1$ можно получить, если произвести следующую разметку[15][42]:

- Все этикетки **U** и **D** пометить знаком «+».
- Этикетки **F** и **B** на рёберных элементах **FR**, **FL**, **BL** и **BR** пометить знаком «—».
- Все остальные этикетки оставить без меток.

Тогда все конфигурации группы $G_1$ будут иметь одну и ту же разметку (совпадающую с разметкой на собранном кубике).

Решение состоит из двух этапов. На первом этапе заданная конфигурация $x \in G_0$ переводится последовательностью ходов $s_1$ в некоторую конфигурацию $x' \in G_1$ . Иными словами, задача первого этапа — восстановить разметку, соответствующую начальной конфигурации, игнорируя цвета этикеток.

На втором этапе конфигурация $x' \in G_1$ переводится последовательностью ходов $s_2$ в начальную конфигурацию. При этом повороты боковых граней на ±90° не используются, благодаря чему разметка автоматически сохраняется.

Склейка последовательностей ходов $s_1$ и $s_2$ является субоптимальным решением к исходной конфигурации[15][41][43].

**Альтернативные субоптимальные решения** Алгоритм Коцембы не останавливается после обнаружения первого решения. Альтернативные оптимальные решения на первом этапе могут привести к более коротким решениям второго этапа, что сократит общую длину решения. Алгоритм продолжает искать также неоптимальные решения на первом этапе в порядке возрастания их длины[15][41][43].

**Особенности реализации** Для поиска решений на каждом из двух этапов применяется алгоритм IDA* с эвристическими функциями, основанными на стоимостях решения соответствующих подзадач[43].

Задача первого этапа сводится к поиску пути в пространстве с тремя координатами из разметки с координатами ($x_1$, $y_1$, $z_1$) в разметку (0, 0, 0)[44]:

1. Ориентация $x_1$ восьми угловых элементов (2187 значений)
2. Ориентация $y_1$ двенадцати рёберных элементов (2048 значений)
3. Расстановка $z_1$ четырёх рёберных элементов **FR**, **FL**, **BL** и **BR** (495 значений)

Рассмотрим три подзадачи поиска пути из разметки ($x_1$, $y_1$, $z_1$) в разметки ($x_1$', $y_1$', 0), ($x_1$', 0, $z_1$'), (0, $y_1$', $z_1$'). Значение эвристической функции $h_1$, используемой на первом этапе, равно максимальной из стоимостей решения этих подзадач. Стоимости решения подзадач предварительно вычисляются и хранятся в виде баз данных с шаблонами[45][46].

Задача второго этапа сводится к поиску пути в пространстве с тремя координатами из конфигурации ($x_2$, $y_2$, $z_2$) в конфигурацию (0, 0, 0)[47]:

1. Перестановка $x_2$ восьми угловых элементов (40320 значений)
2. Перестановка $y_2$ восьми рёберных элементов верхней и нижней граней (40320 значений)
3. Перестановка $z_2$ четырёх рёберных элементов среднего слоя (24 значения)

Рассмотрим три подзадачи поиска пути из конфигурации ($x_2$, $y_2$, $z_2$) в конфигурации ($x_2$', $y_2$', 0), ($x_2$', 0, $z_2$'), (0, $y_2$', $z_2$'). Значение эвристической функции $h_2$, используемой на втором этапе, равно максимальной из стоимостей решения этих подзадач.

В алгоритме применяются дополнительные оптимизации, направленные на увеличение быстродействия и уменьшение памяти, занимаемой таблицами[15][40][41][48].

**Программные реализации** Cube Explorer — программная реализация алгоритма для ОС Windows. Ссылки для загрузки находятся на сайте Герберта Коцембы[49]. В 1992 году на ПК Atari ST с памятью 1 Мбайт и частотой 8 МГц алгоритм позволял в течение 1-2 минут найти решение не длиннее 22 ходов[15][40]. На современном компьютере Cube Explorer позволяет за несколько секунд найти решение не длиннее 20 ходов для произвольно заданной конфигурации[40].

Существует онлайн-версия алгоритма[50].

**Анализ** В 1995 году Майкл Рид показал, что первая и вторая фазы алгоритма Коцембы могут потребовать не более 12 и 18 ходов (FTM) соответственно. Из этого следует, что кубик Рубика всегда может быть решён в 30 ходов. Дополнительный анализ показал, что всегда достаточно 29f или 42q[51][20].

### 12.4.3. Алгоритм Корфа

Алгоритм Коцембы позволяет быстро находить короткие, субоптимальные решения. Тем не менее, может потребоваться значительное время, чтобы доказать оптимальность найденного решения. Был необходим специализированный алгоритм поиска оптимальных решений.

В 1997 году Ричард Корф опубликовал алгоритм, позволявший оптимально решать произвольные конфигурации кубика Рубика. Десять выбранных случайным образом конфигураций были решены не более чем в 18 ходов FTM[52].

Собственно алгоритм Корфа работает следующим образом. В первую очередь определяются подзадачи, достаточно простые для того, чтобы осуществить полный перебор. Были использованы следующие три подзадачи[20]:

1. Состояние головоломки определяется только восемью угловыми кубиками, положения и ориентации рёбер игнорируются.
2. Состояние головоломки определяется шестью из 12 рёберных кубиков, другие кубики игнорируются.
3. Состояние головоломки определяется *другими* шестью рёберными кубиками.

Количество ходов, необходимое для решения каждой из этих подзадач, является нижней оценкой количества ходов, необходимого для полного решения. Произвольно заданная конфигурация кубика Рубика решается с помощью алгоритма IDA*, использующего эту оценку в качестве эвристики. Стоимости решения подзадач хранятся в виде баз данных с шаблонами[52][45].

Хотя этот алгоритм будет всегда находить оптимальные решения, он не позволяет напрямую определить, как много ходов может потребоваться в худшем случае.

Реализацию алгоритма на языке C можно найти в[53].

### 12.4.4. Дальнейшие улучшения

В 2005 году результат Майкла Рида в метрике QTM был улучшен Силвиу Раду до 40q[54].

В 2006 году, Силвиу Раду удалось доказать, что кубик Рубика может быть решён в 27f[34] или 35q[55].

В 2007 году Дэниел Канкл и Джин Куперман использовали суперкомпьютер для доказательства того, что все нерешённые конфигурации могут быть решены не более чем в 26 ходов FTM. Идея доказательства состояла в том, чтобы привести кубик Рубика в одну из 15 752 конфигураций подгруппы квадратов, каждая из которых может быть окончательно решена с помощью нескольких дополнительных ходов. Большинство конфигураций удалось решить таким образом не более чем в 26 ходов; оставшиеся конфигурации были подвергнуты дополнительному анализу, который показал, что они также решаются в 26 ходов[56][35].

В 2008 году Томас Рокики опубликовал вычислительное доказательство разрешимости головоломки в 25 ходов FTM[57].

В августе 2008 года Рокики объявил о доказательстве разрешимости в 23f[58]. Позже эта оценка была улучшена до 22f[59]

В 2009 году ему же удалось показать разрешимость в 29 ходов QTM[60].

В 2010 году Томас Рокики, Герберт Коцемба, Морли Дэвидсон и Джон Детридж объявили о доказательстве того, что любая конфигурация кубика Рубика может быть решена не более чем в 20 ходов в метрике FTM[1][10].

Число Бога для кубика Рубика в метрике QTM по состоянию на июль 2013 года остаётся неизвестным. Оно находится в промежутке между 26q и 29q[16].

В августе 2014 года объявлено, что в метрике QTM число Бога равно 26q[1].

### 12.4.5. Асимптотические оценки

В 2011 году было показано, что число Бога кубика Рубика $n \times n \times n$ является $\Theta(n^2 / \log(n))$[61][62].

## 12.5. Другие головоломки

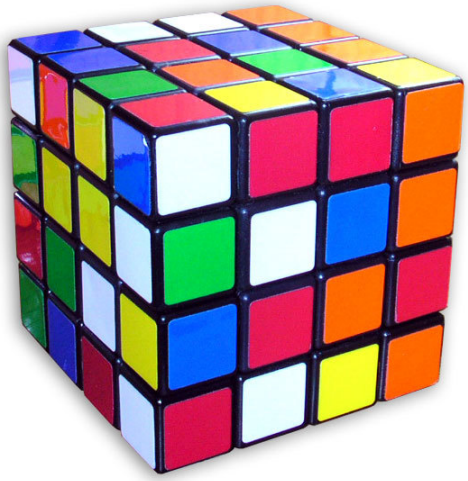
### 12.5.1. Void Cube

Void Cube — кубик Рубика без этикеток в центрах граней.

Длина оптимального решения «void-суперфлипа» в метрике FTM равна 20[63][64], что является доказательством необходимости 20 ходов. Так как Void Cube является *ослабленной задачей*[45] кубика Рубика, существующее доказательство достаточности 20 ходов для кубика Рубика[1] применимо и к Void Cube. Следовательно, число Бога головоломки Void Cube в метрике FTM равно 20.

### 12.5.2. Кубик 4×4×4

Количество конфигураций головоломки 4×4×4 (англ. *Rubik's Revenge*) равно[65]



*Rubik's Revenge — «Месть Рубика»*

$$\frac{8! \times 3^7 \times 24!^2}{4!^6 \times 24} \approx 7.40 \times 10^{45}$$

### Метрики

Существует несколько способов определения метрики для кубика 4×4×4. В этом разделе используются следующие метрики:

- **q-s** (quarter slice): одним ходом считается поворот любого из 12 слоёв головоломки на ±90°;
- **q-w** (quarter twist): одним ходом считается поворот любого внешнего блока (т.е. *только грани* или *грани с несколькими идущими подряд прилегающими к ней слоями*) на ±90°;
- **q-b** (quarter block): одним ходом считается поворот любого внешнего или внутреннего блока (т.е. любой последовательности идущих *подряд* параллельных слоёв) на ±90° относительно остальной части головоломки;
- **h-s**, **h-w**, **h-b**: аналогичны трём предыдущим метрикам, за исключением того, что разрешены также повороты на 180°.

Длина оптимального решения произвольной конфигурации в метрике **q-b** не больше, чем в метриках **q-w** или **q-s**, так как в метрике **q-b** разрешены все ходы двух других q-метрик. То же относится к h-метрикам.

### Оценки числа Бога кубика 4×4×4

В 2006—2007 гг. Брюс Норског (Bruce Norskog) провёл 5-этапный анализ кубика 4×4×4, аналогичный 4-этапному анализу, проведённому Тистлетуэйтом для кубика Рубика 3×3×3. В результате анализа были получены верхние оценки в 82 хода в метрике **h-w**[66], 77 ходов в метрике **h-s**[67][68] и 67 ходов в метрике **h-b**[66].

В 2013 году Shuang Chen (□□) снизил оценку в метрике **h-w** с 82 до 57 поворотов[69].

В 2011 году Томас Рокики на основании нескольких существующих идей определил нижние оценки числа Бога в шести метриках для кубических головоломок с размерами от 2×2×2 до 20×20×20[70]. Результаты для кубика 4×4×4 приведены в таблице (вместе с известными верхними оценками).

*12-цветный мегаминкс*

## 12.5.3. Мегаминкс

Мегаминкс — простейший аналог кубика Рубика в форме додекаэдра. Количество конфигураций 12-цветного Мегаминкса равно $1.01 \cdot 10^{68}$.

В 2012 году Герберт Коцемба определил нижнюю оценку числа Бога для Мегаминкса, равную 45 поворотам граней на любой угол или 55 поворотам на угол ±72°[71][72].

По непроверенным данным, существует алгоритм (предназначенный для робота), позволяющий собрать Мегаминкс не более чем за 194 хода. Детали алгоритма неизвестны[73].

# 12.6. См. также

- Алгоритм Бога
- Группа кубика Рубика
- Доказательные вычисления

- Кубик Рубика
- Теория групп

## 12.7. Примечания


[1] *Rokicki, T.; Kociemba, H.; Davidson, M.; and Dethridge, J.* God's Number is 20 (англ.). Проверено 19 июля 2013.

[2] *Joyner, David* Adventures in group theory: Rubik's Cube, Merlin's machine, and Other Mathematical Toys. — Baltimore: Johns Hopkins University Press, 2002. — P. 7. — ISBN 0-8018-6947-1.

[3] *Jerry Slocum, David Singmaster, Wei-Hwa Huang, Dieter Gebhardt, Geert Hellings* The Cube: The Ultimate Guide to the World's Bestselling Puzzle — Secrets, Stories, Solutions. — 2009. — C. 26. — 142 с.

[4] *Jaap Scherphuis.* Useful Mathematics Metrics (англ.). Проверено 17 июля 2013.

[5] *Jerry Bryan.* Notational convention (05/27/2006). Проверено 28 июля 2013.

[6] *David Singmaster* Cubic Circular (англ.). — 1982. — В. 5 & 6. — С. 26.

[7] *Jaap Scherphuis.* Puzzle Statistics (англ.). Проверено 17 июля 2013.

[8] *Schönert, Martin* Analyzing Rubik's Cube with GAP (англ.). Проверено 19 июля 2013.

[9] *Jaap Scherphuis.* Rubik's Cube 3x3x3 (англ.). Проверено 19 июля 2013.

[10] *Herbert Kociemba.* Two-Phase-Algorithm and God's Algorithm: God's number is 20 (англ.). Проверено 23 июля 2013.

[11] *Rik van Grol.* The Quest For God's Number (англ.). Math Horizons (November 2010). Проверено 26 июля 2013.

[12] *Stefan Edelkamp, Stefan Schrōdl* Heuristic search: theory and applications. — Morgan Kaufmann Publishers, 2012. — 712 с. — ISBN 978-0-12-372512-7.

[13] SIAM J. Discrete Math., 27(2), 1082–1105

[14] «Разрешимой» конфигурацией называется конфигурация, которая может быть переведена в собранную. Так как граф кубика Рубика ненаправленный, из этого следует, что любая конфигурация, полученная из исходной произвольной последовательностью ходов разрешима. Но существуют неразрешимые конфигурации. Например, если в собранном состоянии повернуть одну из вершин кубика на 120°, то получится неразрешимая конфигурация.

[15] *В. Дубровский, А. Калинин* Новости кубологии // *Квант*. — 1992. — № 11. — С. 52-56.

[16] *Jaap Scherphuis.* Useful Mathematics God's Algorithm (англ.). Проверено 17 июля 2013.

[17] *Michael Reid.* Michael Reid's Rubik's cube page M-symmetric positions (англ.) (May 24, 2005). Проверено 17 июля 2013.

[18] *David Singmaster* Cubic Circular (англ.). — 1982. — В. 5 & 6. — С. 24.

[19] *Joyner, David* Adventures in group theory: Rubik's Cube, Merlin's machine, and Other Mathematical Toys. — P. 149.

[20] *Stefan Pochmann* Analyzing Human Solving Methods for Rubik's Cube and similar Puzzles (Diploma Thesis) (англ.).

[21] *Dik T. Winter.* Kociemba's algorithm (англ.) (Mon, 18 May 92 00:49:48 +0200). Проверено 17 июля 2013.

[22] *Michael Reid.* superflip requires 20 face turns (англ.) (Wed, 18 Jan 95 10:13:45 −0500). Проверено 17 июля 2013.

[23] *Michael Reid.* superflip (англ.) (Tue, 10 Jan 95 18:57:20 −0500). Проверено 17 июля 2013.

[24] *Jerry Bryan.* Qturn Lengths of M-Symmetric Positions (англ.) (Sun, 19 Feb 95 23:20:22 −0500 (EST)). Проверено 17 июля 2013.

[25] *Michael Reid.* Temporary Cube-Lovers mailing list archive. Mar 8, 1998 to Sep 14, 1998 superflip composed with four spot (англ.) (Sun, 2 Aug 1998 08:47:44 −0400). Проверено 17 июля 2013.

[26] *Л. А. Калужнин, В. И. Сущанский* Преобразования и перестановки. — М, 1985. — С. 143. — 160 с.

[27] *David Singmaster* Notes on Rubik's Magic Cube. — Enslow Publishers, 1981. — P. 30.

[28] *В. Дубровский* Алгоритм волшебного кубика // *Квант*. — 1982. — № 7. — С. 22-25.

[29] Порядок этой и следующих трёх групп вычисляется как произведение трёх множителей, указывающих соответственно количество разрешимых конфигураций углов, количество разрешимых конфигураций рёбер и общее ограничение «чётности» разрешимой конфигурации.

[30] *Jaap Scherphuis.* Cube subgroups Subgroups generated by face moves (англ.). Проверено 19 июля 2013.

[31] *Mark Longridge.* Progressive Improvements in Solving Algorithms (англ.). Проверено 28 июля 2013.

[32] *В. Дубровский* Математика волшебного кубика // *Квант*. — 1982. — № 8. — С. 22-27, 48.

[33] *Jaap Scherphuis.* Thistlethwaite's 52-move algorithm (англ.). Проверено 17 июля 2013.

[34] *silviu.* Rubik can be solved in 27f. Domain of the Cube Forum (04/01/2006). Проверено 20 июля 2013.

[35] Kunkle, D.; Cooperman, C. (2007). extquotedblTwenty-Six Moves Suffice for Rubik's Cube extquotedbl (PDF). *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC '07)*, ACM Press.

[36] *Michael Reid.* an upper bound on god's number (англ.) (Wed, 29 Apr 92 01:37:26 −0700 (PDT)). Проверено 17 июля 2013.

[37] *Michael Reid.* new upper bound (англ.) (Fri, 22 May 92 22:56:34 −0700 (PDT)). Проверено 19 июля 2013.

[38] *Dik T. Winter.* Corrected calculations are now done. (англ.) (Thu, 28 May 92 15:00:49 +0200). Проверено 19 июля 2013.

[39] *Ryan Heise.* The Human Thistlethwaite Algorithm (англ.). Проверено 19 июля 2013.

[40] *Herbert Kociemba.* Two-Phase Algorithm Details (англ.). Проверено 20 июля 2013.

[41] *Jaap Scherphuis.* Computer Puzzling Kociemba's Algorithm (англ.). Проверено 23 июля 2013.

[42] *Herbert Kociemba.* The Subgroup H and its cosets (англ.). Проверено 23 июля 2013.

[43] *Herbert Kociemba.* The Two-Phase-Algorithm (англ.). Проверено 23 июля 2013.

[44] Биекция между конфигурациями и тройками координат установлена таким образом, что целевой разметке первого этапа (т.е. любой конфигурации из группы $G_1$) соответствует тройка (0, 0, 0).

[45] *Стюарт Рассел, Питер Норвиг* Составление допустимых эвристических функций // Искусственный интеллект: современный подход = Artificial Intelligence: A Modern Approach. — 2-е изд.. — М.: Вильямс, 2006. — С. 170 — 173. — 1408 с. — ISBN 5-8459-0887-6.

[46] Англ. *pattern databases*. В изложении автора алгоритма — «pruning tables».

[47] Из-за ограничения, связанного с чётностью перестановки элементов, встречается только половина всех троек координат.

[48] *Herbert Kociemba.* Pruning Tables (англ.). Проверено 23 июля 2013.

[49] *Herbert Kociemba.* Download (англ.). Проверено 20 июля 2013.

[50] *Eric Dietz.* Solve Your Cube In Less Than 25 Moves (англ.). Проверено 20 июля 2013.

[51] *Michael Reid.* new upper bounds (англ.) (Sat, 07 Jan 95 20:24:35 −0500). Проверено 19 июля 2013.

[52] *Richard E. Korf* Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. — 1997.

[53] *Michael Reid.* Rubik's cube page Optimal Rubik's cube solver (October 28, 2006). Проверено 20 июля 2013.

[54] Silviu Radu, "A New Upper Bound on Rubik's Cube Group", *arXiv:math/0512485*

[55] *silviu.* Rubik can be solved in 35q. Domain of the Cube Forum (03/22/2006). Проверено 20 июля 2013.

[56] Northeastern University Researchers Solve Rubik's Cube in 26 Moves. Проверено 20 июля 2013.

[57] Tom Rokicki, "Twenty-Five Moves Suffice for Rubik's Cube", *arXiv:0803.3435*

[58] *rokicki.* Twenty-Three Moves Suffice. Domain of the Cube Forum (04/29/2008). Проверено 20 июля 2013.

[59] *rokicki.* Twenty-Two Moves Suffice. Domain of the Cube Forum (08/12/2008). Проверено 20 июля 2013.

[60] *rokicki.* Twenty-Nine QTM Moves Suffice. Domain of the Cube Forum (06/15/2009). Проверено 20 июля 2013.

[61] Demaine, Erik D.; Demaine, Martin L.; Eisenstat, Sarah; Lubiw, Anna & Winslow, Andrew (2011), "Algorithms for Solving Rubik's Cubes", *arXiv:1106.5736v1* [cs.DS]

[62] *Larry Hardesty.* The math of the Rubik's cube New research establishes the relationship between the number of squares in a Rubik's-cube-type puzzle and the maximum number of moves required to solve it (англ.). MIT News Office (June 29, 2011). Проверено 23 июля 2013.

[63] *rokicki.* Void cube diameter at least 20 (face turn metric). Domain of the Cube Forum (01/19/2010). Проверено 28 июля 2013.

[64] *rokicki.* Void cube diameter at least 20 (probably 20). Speedsolving.com (01/19/2010). Проверено 28 июля 2013.

[65] *Jaap Scherphuis.* Rubik's Revenge (англ.). Проверено 28 июля 2013.

[66] *Bruce Norskog.* New upper bounds: 82 twist turns, 67 block turns. Domain of the Cube Forum (08/13/2007). Проверено 28 июля 2013.

[67] *Bruce Norskog.* The 4x4x4 can be solved in 77 single-slice turns. Domain of the Cube Forum (07/26/2007).

[68] Bigger rubik cube bound. Project Euler. Проверено 28 июля 2013.

[69] *CS.* Solving the 4x4x4 in 57 moves(OBTM). Domain of the Cube Forum (09/30/2013). Проверено 19 ноября 2013.

[70] *rokicki.* Lower Bounds for n x n x n Rubik's Cubes (through n=20) in Six Metrics. Domain of the Cube Forum (07/18/2011). Проверено 28 июля 2013.

[71] *Herbert Kociemba.* Megaminx needs at least 45 moves. Domain of the Cube Forum (02/28/2012). Проверено 28 июля 2013.

[72] *Herbert Kociemba.* Lower bound for Megaminx in htm and qtm. Speedsolving.com (03-01-2012). Проверено 28 июля 2013.

[73] Upper bound? - 03/12/2012. Domain of the Cube Forum

## 12.8. Рекомендуемая литература


- *В. Залгаллер, С. Залгаллер* Венгерский шарнирный кубик (рус.) // *Квант*. — 1980. — № 12. — С. 17 — 21.
- *В. Дубровский* Алгоритм волшебного кубика (рус.) // *Квант*. — 1982. — № 7. — С. 22-25.
- *В. Дубровский* Математика волшебного кубика (рус.) // *Квант*. — 1982. — № 8. — С. 22 — 27, 48.
- *В. Дубровский, А. Калинин* Новости кубологии (рус.) // *Квант*. — 1992. — № 11. — С. 52 — 56.
- *Л. А. Калужнин, В. И. Сущанский* Преобразования и перестановки. — М, 1985. — С. 143. — 160 с.
- *David Singmaster* Cubic Circular. — 1981 — 1985.
- *David Singmaster, Alexander Frey* Handbook of Cubik Math. — 1987.
- *E. D. Demaine, M. L. Demaine, S. Eisenstat, A. Lubiw, A. Winslow* Algorithms for Solving Rubik's Cubes // *Lecture Notes in Computer Science*. — 2011. — Т. 6942. — С. 689-700. — DOI:10.1007/978-3-642-23719-5_58


## 12.9. Ссылки


- *Константин Кноп.* Кубик Рубика: штурм твердыни (рус.). Проверено 20 июля 2013.
- *Rik van Grol.* The Quest For God's Number (англ.). Math Horizons (November 2010). Проверено 26 июля 2013.
- *Jaap Scherphuis.* Jaap's Puzzle Page (англ.). — Подборка информации по множеству головоломок и методов их решения. Проверено 20 июля 2013.
- *Herbert Kociemba.* Cube Explorer 5.01 (англ.). — Домашняя страница Герберта Коцембы. Подробное описание алгоритмов, используемых в программе Cube Explorer. Проверено 20 июля 2013.
- *Tomas Rokicki, Herbert Kociemba, Morley Davidson, John Dethridge.* God's Number is 20 (англ.). Проверено 20 июля 2013.
- *Martin Schönert.* Cube Lovers archives converted to HTML (англ.). — 1947 articles between July 1980 and June 1996. Проверено 20 июля 2013.
- *Mark Longridge.* Domain of the Cube Forum (англ.). — Discussions on the mathematics of the cube. Проверено 20 июля 2013.
- *W. D. Joyner.* Lecture notes on the mathematics of the Rubik's cube (англ.). Проверено 22 июля 2013.
- *Tomas Rokicki.* Computers and the Cube (slides) (англ.) (3 November 2009). — **MATH 78SI**: Speedcubing: History, Theory, and Practice. Student-initiated course at Stanford (Autumn 2009). Проверено 30 июля 2013.

# Глава 13

# Метод золотого сечения

**Метод золотого сечения** — метод поиска экстремума действительной функции одной переменной на заданном отрезке. В основе метода лежит принцип деления отрезка в пропорциях золотого сечения. Является одним из простейших вычислительных методов решения задач оптимизации. Впервые представлен Джеком Кифером в 1953 году.

## 13.1. Описание метода

Пусть задана функция $f(x) : [a,\ b] \to \mathbb{R},\ f(x) \in C([a,\ b])$. Тогда для того, чтобы найти определённое значение этой функции на заданном отрезке, отвечающее критерию поиска (пусть это будет минимум), рассматриваемый отрезок делится в пропорции золотого сечения в обоих направлениях, то есть выбираются две точки $x_1$ и $x_2$ такие, что:


*Иллюстрация выбора промежуточных точек метода золотого сечения.*

$\frac{b-a}{b-x_1} = \frac{b-a}{x_2-a} = \varphi = \frac{1+\sqrt{5}}{2} = 1.618\ldots$, где $\varphi$ — пропорция золотого сечения.

Таким образом:

$$\begin{aligned} x_1 &= b - \frac{(b-a)}{\varphi} \\ x_2 &= a + \frac{(b-a)}{\varphi} \end{aligned}$$

То есть точка $x_1$ делит отрезок $[a,\ x_2]$ в отношении золотого сечения. Аналогично $x_2$ делит отрезок $[x_1,\ b]$ в той же пропорции. Это свойство и используется для построения итеративного процесса.

### 13.1.1. Алгоритм

1. На первой итерации заданный отрезок делится двумя симметричными относительно его центра точками и рассчитываются значения в этих точках.
2. После чего тот из концов отрезка, к которому среди двух вновь поставленных точек ближе оказалась та, значение в которой максимально (для случая поиска минимума), отбрасывают.
3. На следующей итерации в силу показанного выше свойства золотого сечения уже надо искать всего одну новую точку.
4. Процедура продолжается до тех пор, пока не будет достигнута заданная точность.

### 13.1.2. Формализация

1. **Шаг 1.** Задаются начальные границы отрезка $a,\ b$ и точность $\varepsilon$.
2. **Шаг 2.** Рассчитывают начальные точки деления: $x_1 = b - \frac{(b-a)}{\varphi},\quad x_2 = a + \frac{(b-a)}{\varphi}$ и значения в них целевой функции: $y_1 = f(x_1),\ y_2 = f(x_2)$.
   - Если $y_1 \geq y_2$ (для поиска max изменить неравенство на $y_1 \leq y_2$), то $a = x_1$
   - Иначе $b = x_2$.
3. **Шаг 3.**
   - Если $|b-a| < \varepsilon$, то $x = \frac{a+b}{2}$ и останов.
   - Иначе возврат к шагу 2.

Алгоритм взят из источника: *Джон Г.Мэтьюз, Куртис Д.Финк* extquotedblЧисленные методы. Использование MATLAB". — М, СПб: extquotedblВильямс extquotedbl, 2001. — 716 с.

## 13.2. Метод чисел Фибоначчи

В силу того, что в асимптотике $\varphi = \lim_{n\to\infty} \frac{F_{n+1}}{F_n}$ , метод золотого сечения может быть трансформирован в так называемый метод чисел Фибоначчи. Однако при этом в силу свойств чисел Фибоначчи количество итераций строго ограничено. Это удобно, если сразу задано количество возможных обращений к функции.

### 13.2.1. Алгоритм

1. **Шаг 1.** Задаются начальные границы отрезка $a,\ b$ и число итераций $n$, рассчитывают начальные точки деления: $x_1 = a+(b-a)\frac{F_{n-2}}{F_n},\quad x_2 = a+(b-a)\frac{F_{n-1}}{F_n}$ и значения в них целевой функции: $y_1 = f(x_1),\ y_2 = f(x_2)$.

2. **Шаг 2.** $n = n-1$.

   - Если $y_1 > y_2$, то $a = x_1,\ x_1 = x_2,\ x_2 = b-(x_1-a),\ y_1 = y_2,\ y_2 = f(x_2)$.
   - Иначе $b = x_2,\ x_2 = x_1,\ x_1 = a+(b-x_2),\ y_2 = y_1,\ y_1 = f(x_1)$.

3. **Шаг 3.**

   - Если $n = 1$, то $x = x_1 = x_2$ и останов.
   - Иначе возврат к шагу 2.

## 13.3. Литература


1. *Акулич И.Л.* Математическое программирование в примерах и задачах: Учеб. пособие для студентов эконом. спец. вузов. — М.: Высш. шк., 1986.

2. *Гилл Ф., Мюррей У., Райт М.* Практическая оптимизация. Пер. с англ. — М.: Мир, 1985.

3. *Коршунов Ю.М.* Математические основы кибернетики. — М.: Энергоатомиздат, 1972.

4. *Максимов Ю.А., Филлиповская Е.А.* Алгоритмы решения задач нелинейного программирования. — М.: МИФИ, 1982.

5. *Максимов Ю.А.* Алгоритмы линейного и дискретного программирования. — М.: МИФИ, 1980.

6. *Корн Г., Корн Т.* Справочник по математике для научных работников и инженеров. — М.: Наука, 1970. — С. 575-576.

7. *Корн Г., Корн Т.* Справочник по математике для научных работников и инженеров. — М.: Наука, 1973. — С. 832 с илл..

8. *Джон Г.Мэтьюз, Куртис Д.Финк* . Численные методы. Использование MATLAB. — 3-е издание. — М., СПб.: Вильямс, 2001. — С. 716.


## 13.4. Реализация

import math phi = (1 + math.sqrt(5)) / 2 eps = 0.001 def find_min_gold(f, a, b): while math.fabs(b - a) > eps: x1, x2 = b - (b - a) / phi, a + (b - a) / phi y1, y2 = f(x1), f(x2) if y1 >= y2: a = x1 else: b = x2 return (a + b) / 2 def find_min_half(f, a, b): while (math.fabs(b - a)) > eps: half_x1, half_x2 = ((a + b)/2 - eps/4), ((a + b)/2 + eps/4) if f(half_x1) <= f(half_x2): b = half_x2 else: a = half_x1 return (a + b) / 2 func = lambda x: x**2 - 1 f_a, f_b = −3, 3 minimum1 = find_min_gold(func, f_a, f_b) minimum2 = find_min_half(func, f_a, f_b) print 'GOLD Minimum of function on [%s, %s] is %s in dot %s' % (f_a, f_b, func(minimum1), minimum1) print 'HALF Minimum of function on [%s, %s] is %s in dot %s' % (f_a, f_b, func(minimum2), minimum2)

## 13.5. См. также

- Золотое сечение
- Числа Фибоначчи

# Глава 14

# Метод перебора

**Метод перебора (метод равномерного поиска)** — простейший из методов поиска значений действительно-значных функций по какому-либо из критериев сравнения (на максимум, на минимум, на определённую константу). Применительно к экстремальным задачам является примером прямого метода условной одномерной пассивной оптимизации.

## 14.1. Описание

Проиллюстрируем суть метода равномерного поиска посредством рассмотрения задачи нахождения минимума.

Пусть задана функция $f(x) : [a,\ b] \to \mathbb{R}$ .
И задача оптимизации выглядит так: $f(x) \to \min_{x \in [a,\ b]}$
Пусть также задано число наблюдений $n$ .

Тогда отрезок $[a, b]$ разбивают на $(n + 1)$ равных частей точками деления:

$$x_i = a + i\frac{(b-a)}{(n+1)}, \quad i = 1, \ldots, n$$

Вычислив значения $F(x)$ в точках $x_i,\ i = 1, \ldots, n$ , найдем путем сравнения точку $x_m$ , где $m$ — это число от 1 до $n$ такую, что

$$F(x_m) = \min F(x_i) \text{ для всех } i \text{ от } 1 \text{ до } n\ .$$

Тогда интервал неопределённости составляет величину $2\frac{(b-a)}{(n+1)}$ , а погрешность определения точки минимума $x_m$ функции $F(x)$ соответственно составляет : $\varepsilon = \frac{(b-a)}{n+1}$ .

## 14.2. Модификация

Если заданное количество измерений чётно ( $n = 2k$ ), то разбиение можно проводить другим, более изощрённым способом:

$$x_{2i} = a + \frac{(b-a)}{(n/2+1)}2i, \quad i = 1, \ldots, n/2$$

$x_{2i-1} = x_{2i} - \delta$ , где $\delta$ — некая константа из интервала $\left(0,\ \frac{(b-a)}{(n/2+1)}\right)$ .

Тогда в худшем случае интервал неопределённости имеет длину $\frac{(b-a)}{(n/2+1)} + \delta$ .

## 14.3. Литература


1. *Акулич И.Л.* Математическое программирование в примерах и задачах: Учеб. пособие для студентов эконом. спец. вузов. — М.: Высш. шк., 1986.

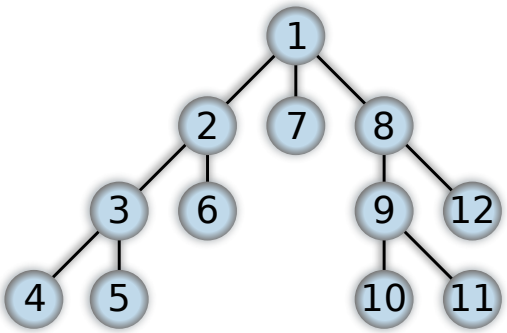
2. *Гилл Ф., Мюррей У., Райт М.* Практическая оптимизация. Пер. с англ. — М.: Мир, 1985.

3. *Максимов Ю.А.,Филлиповская Е.А.* Алгоритмы решения задач нелинейного программирования. — М.: МИФИ, 1982.

4. *Корн Г., Корн Т.* Справочник по математике для научных работников и инженеров. — М.: Наука, 1970. — С. 575-576.

# Глава 15

# Поиск в глубину




*Порядок обхода дерева в глубину*

**Поиск в глубину** (англ. *Depth-first search,* ***DFS***) — один из методов обхода графа. Стратегия поиска в глубину, как и следует из названия, состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину. Если после завершения алгоритма не все вершины были рассмотрены, то необходимо запустить алгоритм от одной из нерассмотренных вершин[1].

## 15.1. Применение

- В качестве подпрограммы в алгоритмах поиска одно- и двусвязных компонент.
- В топологической сортировке.
- В различных расчётах на графах. Например, как часть алгоритма Диница поиска максимального потока.
- В поиске по древовидным структурам.

## 15.2. Алгоритм поиска в глубину

Пусть задан граф $G = (V, E)$ , где $V$ — множество вершин графа, $E$ — множество ребер графа. Предположим, что в начальный момент времени все вершины графа окрашены в *белый* цвет. Выполним следующие действия:

1. Пройдём по всем вершинам $v \in V$ .
   - Если вершина $v$ белая, выполним для неё DFS(v).

Процедура DFS (параметр — вершина $u \in V$ )

1. Перекрашиваем вершину $u$ в *серый* цвет.
2. Для всякой вершины $w$ , смежной с вершиной $u$ и окрашенной в белый цвет, рекурсивно выполняем процедуру DFS(w)[2].
3. Перекрашиваем вершину $u$ в *чёрный* цвет.

Часто используют двухцветные метки — без серого, на 1-м шаге красят сразу в чёрный цвет.

### 15.2.1. Нерекурсивные варианты

На больших графах поиск в глубину серьёзно нагружает стек вызовов. Если есть риск переполнения стека, используют нерекурсивные варианты поиска.

**Первый вариант**: можно сэмулировать стек вызова программно: для каждой из серых вершин в стеке будет храниться её номер $u$ и номер текущей смежной вершины $w$ .

Процедура DFS (параметр — вершина $u \in V$ )

1. Кладём на стек пару $(u, \varnothing)$ . Перекрашиваем вершину $u$ в *серый* цвет.
2. Пока стек не пуст…
   (a) Берём верхнюю пару $(v, w)$ , не извлекая её из стека.

(b) Находим вершину $w'$ , смежную с $v$ и следующую за $w$ .

i. Если таковой нет, извлекаем $(v, w)$ из стека, перекрашиваем вершину $v$ в *чёрный* цвет.

ii. В противном случае присваиваем $w := w'$ .

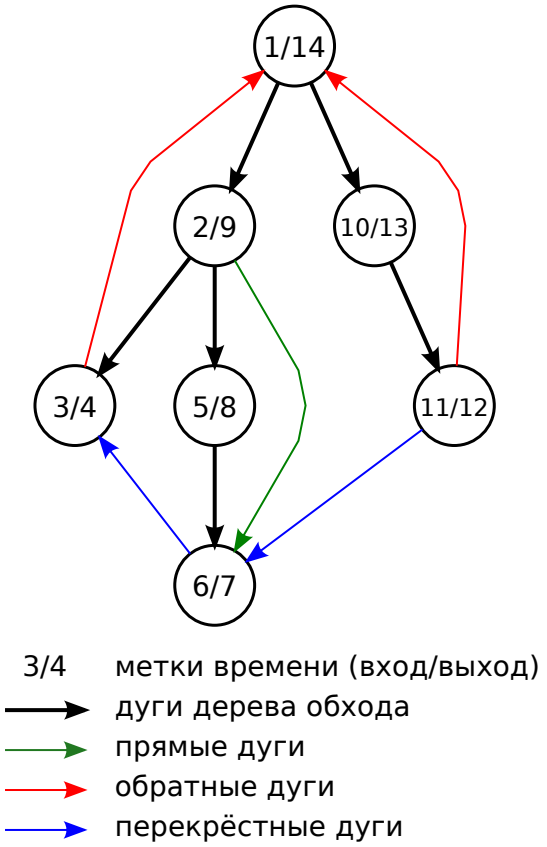
- Если к тому же вершина $w'$ белая, кладём на стек пару $(w', \varnothing)$ , перекрашиваем $w'$ в *серый* цвет.

**Второй вариант**: можно в каждой из «серых» вершин держать текущее $w$ и указатель на предыдущую (ту, из которой пришли).

**Третий вариант** работает, если хватает двухцветных меток.

Процедура DFS (параметр — вершина $u \in V$ )

1. Кладём на стек вершину $u$ .
2. Пока стек не пуст…
   (a) Берём верхнюю вершину $v$ .
   (b) Если она белая…
      i. Перекрашиваем её в *чёрный* цвет.
      ii. Кладём в стек все смежные с $v$ белые вершины.




*Поиск в глубину с метками времени. Порядок выбора рёбер — слева направо.*

## 15.3. Поиск в глубину с метками времени. Классификация рёбер

Для каждой из вершин установим два числа — «время» входа $entry[u]$ и «время» выхода $leave[u]$ .

Модифицируем процедуру DFS так.

1. Увеличиваем «текущее время» на 1. $entry[u] = t$ .
2. Перекрашиваем вершину $u$ в серый цвет.
3. Для всякой вершины $w$ , смежной с вершиной u и окрашенной в белый цвет, выполняем процедуру DFS(w).
4. Перекрашиваем вершину $u$ в чёрный цвет.
5. Увеличиваем «текущее время» на 1. $leave[u] = t$ .

Считаем, что граф ориентированный. Очевидно, для любой вершины $entry[u] < leave[u]$ . Просматриваемые на шаге 3 дуги *u*→*v* могут быть:

- $entry[u] \leqslant t < entry[v] < leave[v] < leave[u]$ . В момент выполнения шага 3 (обозначенный как *t*) вершина *v* белая. В таком случае мы для вершины *v* исполняем DFS, а дуга называется **дугой дерева поиска**.
- $entry[u] < entry[v] < leave[v] \leqslant t < leave[u]$ . В момент *t* вершина *v* чёрная, сравнение *entry* говорит, что в *v* попали из *u*. Такая дуга называется **прямой**.
- $entry[v] < leave[v] < entry[u] \leqslant t < leave[u]$ . В момент *t* вершина *v* также чёрная, но сравнение *entry* говорит, что в *v* попали в обход *u*. Такая дуга называется **перекрёстной**.
- $entry[v] < entry[u] \leqslant t < leave[u] < leave[v]$ . В момент *t* вершина *v* серая. Имеем дело с **обратной** дугой.

Рёбра неориентированного графа могут быть рёбрами дерева и обратными, но не прямыми и перекрёстными.[3] Чтобы различать рёбра неориентированного графа, достаточно указанных выше трёх- или двухцветных отметок. Ребро, идущее в белую вершину,— ребро дерева. В серую (чёрную в двухцветном варианте) — обратное. В чёрную — такого не бывает.[4]

Алгоритм Косарайю требует сортировки вершин в обратном порядке по времени выхода. Метка входа и типы рёбер нужны в алгоритмах поиска точек сочленения и мостов.

## 15.4. Примеры реализации

### 15.4.1. Java

List<Integer>[] graph = readGraph(); boolean[] used = new boolean[graph.length]; public static void dfs(int pos) { used[pos] = true; System.out.println(pos); for (int next : graph[pos]) if (!used[next]) dfs(next); }

### 15.4.2. C++

vector < vector<int> > graph; vector<bool> used; void dfs(int node_index) { used[node_index] = true; for (auto i : graph[node_index]) { if (!used[i]) dfs(i); } }

### 15.4.3. Pascal

const MAX_N = 10; var graph: array [1..MAX_N, 1..MAX_N] of boolean; // массив для определения графа visited: array [1..MAX_N] of boolean; procedure dfs(v: integer); var i: integer; begin visited[v] := true; for i := 1 to MAX_N do if graph[v, i] and not visited[i] then dfs(i); end;

### 15.4.4. Python

g = [[0,1,0], # матрица связности [1,1,0], [0,0,1]] ex = set() # множество посещенных вершин def dfs(node): # start - начальная вершина ex.add(node) for i in range(len(g)): if g[node][i] == 1 and i not in ex: print(i) dfs(i)

### 15.4.5. PHP

$graph = array( 'A' => array('B','C','D','Z'), 'B' => array('A','E'), 'C' => array('A','F','G'), 'D' => array('A','I'), 'E' => array('B','H'), 'F' => array('C','J'), 'G' => array('C'), 'H' => array('E','Z'), 'I' => array('D'), 'J' => array('J'), 'Z' => array('A','H')); function dfs( $graph , $startNode = 'A' ) { global $visited; $visited[] = $startNode; foreach($graph[$startNode] as $index => $vertex) { if( !in_array( $vertex , $visited ) ) dfs( $graph , $vertex ); } }

## 15.5. См. также

- Поиск в ширину

## 15.6. Примечания

[1] Cormen, p. 622

[2] Обход в глубину, цвета вершин — Викиконспекты

[3] Если в сторону u→v оно прямое, то ранее его прошли в сторону v→u как обратное. Если в сторону u→v оно перекрёстное, его должны были пройти v→u как ребро дерева.

[4] Cormen, с. 628-629

## 15.7. Литература


- *Ананий В. Левитин* **Глава 5. Метод уменьшения размера задачи: Поиск в глубину** // Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Aigorithms. — М.: «Вильямс», 2006. — С. 212-215. — ISBN 0-201-74395-7.
- *Кормен Т., Лейзерсон Ч., Ривест Р.* **Глава 22. Элементарные алгоритмы для работы с графами** // Алгоритмы: построение и анализ(второе издание). — М.: «Вильямс», 2005. — С. 622-632.


## 15.8. Ссылки

- ВКИ НГУ: Методы программирования. Обходы графа.
- СпбГУ ИТМО, Факультет информационных технологий и программирования: Дискретная математика. Алгоритмы. Обход графа в глубину.
- Реализация поиска в глубину и различные задачи, решаемые с его помощью (сайт e-maxx.ru)
- Поиск в глубину
- Обход в глубину, цвета вершин — Викиконспекты ИТМО

# Глава 16

# Поиск в пространстве состояний

**Поиск в пространстве состояний** (англ. *state space search*) — группа методов, предназначенных для решения задач искусственного интеллекта. Методы поиска в пространстве состояний осуществляют последовательный просмотр *конфигураций* или *состояний* задачи с целью обнаружения *целевого состояния*, имеющего заданные характеристики.

## 16.1. Формальное определение задачи

### 16.1.1. Компоненты задачи

Во многих задачах присутствует множество состояний, в которых может находиться определённый объект или система, с правилами перехода из одного состояния в другое (например, в играх). Подобные задачи могут быть формально определены с помощью четырёх компонентов:

- **Начальное состояние** — состояние, в котором система находится в начальный момент;
- **Функция определения преемника** — описание возможных переходов из одного состояния в другое;
- **Проверка цели**, позволяющая определить, является ли данное состояние целевым;
- **Функция стоимости пути** — функция, назначающая каждому пути определённую стоимость.

### 16.1.2. Граф пространства состояний

Большинство алгоритмических формулировок поиска на графах использует понятие **явного графа** (англ. *explicit graph*). Граф $G = \{V, E\}$ может быть представлен в виде *матрицы смежности* или *списка смежности*.

В алгоритмах поиска в пространстве состояний применяется понятие **неявного графа** (англ. *implicit graph*). Отличие состоит в том, что рёбра графа не хранятся в памяти явно, а порождаются «на лету» (англ. *on-the-fly*) набором правил перехода. Определение графа пространства состояний включает в себя *начальную вершину*, *множество целевых вершин* и *процедуру развёртывания вершины*[1].

### 16.1.3. Решение задачи

**Решением задачи** называется путь от начального состояния до целевого состояния.

**Оптимальным решением** называется решение, имеющее наименьшую стоимость среди всех прочих решений.

## 16.2. Оценка алгоритма

Производительность алгоритма оценивается с помощью четырёх основных показателей:

- **Полнота** — свойство алгоритма всегда находить решение, если оно существует;
- **Оптимальность** — свойство алгоритма всегда находить решение с наименьшей стоимостью;
- **Временная сложность** — оценка времени работы алгоритма;
- **Пространственная сложность** — оценка объёма памяти, необходимого алгоритму.

## 16.3. Методы поиска в пространстве состояний

Методы поиска в пространстве состояний делятся на информированные и неинформированные.

**Неинформированные методы** (*методы слепого поиска*, *методы грубой силы*) не используют никакой информации о конкретной задаче. Они последовательно просматривают все состояния до тех пор, пока

не будет найдено решение. Различия между методами неинформированного поиска сводятся к порядку просмотра состояний.

**Информированные методы поиска** (*эвристические методы*) пользуются дополнительной информацией о конкретной задаче. Дополнительная информация (**эвристика**) позволяет сократить перебор путём исключения заведомо бесперспективных вариантов. Такой подход ускоряет работу алгоритма по сравнению с полным перебором. Платой за это является отсутствие гарантии того, что выбрано правильное или наилучшее из всех возможных решение.

## 16.4. См. также

- Информированный метод поиска
- Неинформированный метод поиска

## 16.5. Примечания

[1] *Stefan Edelkamp, Stefan Schrödl* Heuristic search: theory and applications. — Morgan Kaufmann Publishers, 2012. — 712 с. — ISBN 978-0-12-372512-7.

## 16.6. Литература

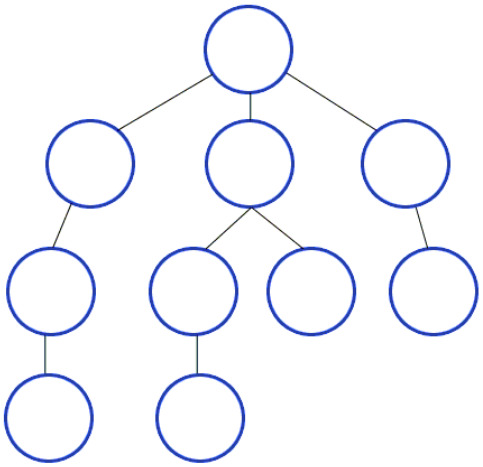
- *Стюарт Рассел, Питер Норвиг* Искусственный интеллект: современный подход = Artificial Intelligence: A Modern Approach. — 2-е изд. — М: Вильямс, 2006. — 1408 с. — ISBN 5-8459-0887-6.

## 16.7. Ссылки

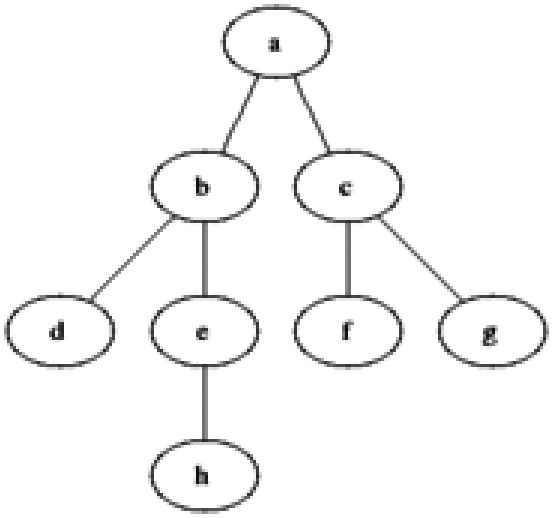
- Толковый словарь по искусственному интеллекту

# Глава 17

# Поиск в ширину



*Поиск в ширину*




*Белый — вершина, которая еще не обнаружена. Серый — вершина, уже обнаруженная и добавленная в очередь. Черный — вершина, извлечённая из очереди*[2]

**Поиск в ширину** (англ. *breadth-first search*, **BFS**) — метод обхода графа и поиска пути в графе. Поиск в ширину является одним из неинформированных алгоритмов поиска[1].

## 17.1. Работа алгоритма

Поиск в ширину работает путём последовательного просмотра отдельных *уровней* графа, начиная с узла-источника $u$ .

Рассмотрим все рёбра $(u, v)$ , выходящие из узла $u$ . Если очередной узел $v$ является целевым узлом, то поиск завершается; в противном случае узел $v$ добавляется в очередь. После того, как будут проверены все рёбра, выходящие из узла $u$ , из очереди извлекается следующий узел $u$ , и процесс повторяется.

## 17.2. Неформальное описание

1. Поместить узел, с которого начинается поиск, в изначально пустую очередь.
2. Извлечь из начала очереди узел $u$ и пометить его как развёрнутый.
   - Если узел $u$ является целевым узлом, то завершить поиск с результатом «успех».
   - В противном случае, в конец очереди добавляются все преемники узла $u$ , которые ещё не развёрнуты и не находятся в очереди.
3. Если очередь пуста, то все узлы связного графа были просмотрены, следовательно, целевой узел недостижим из начального; завершить поиск с результатом «неудача».
4. Вернуться к п. 2.

## 17.3. Формальное описание

Ниже приведён псевдокод алгоритма для случая, когда необходимо лишь найти целевой узел. В зависимости от конкретного применения алгоритма, мо-

жет потребоваться дополнительный код, обеспечивающий сохранение нужной информации (расстояние от начального узла, узел-родитель и т. п.)

Рекурсивная формулировка:

BFS(*start_node*, *goal_node*) { **return** BFS'({*start_node*}, ∅, *goal_node*); } BFS'(*fringe*, *visited*, *goal_node*) { **if**(*fringe* == ∅) { // Целевой узел не найден **return** false; } **if** (*goal_node* ∈ *fringe*) { **return** true; } **return** BFS'({*child* | *x* ∈ *fringe*, *child* ∈ expand(*x*)} \ *visited*, *visited* ∪ *fringe*, *goal_node*); }

Итеративная формулировка:

BFS(*start_node*, *goal_node*) { **for**(all nodes i) *visited*[*i*] = false; // изначально список посещённых узлов пуст *queue*.push(*start_node*); // начиная с узла-источника *visited*[*start_node*] = true; **while**(! *queue*.empty() ) { // пока очередь не пуста *node* = *queue*.pop(); // извлечь первый элемент в очереди **if**(*node* == *goal_node*) { **return** true; // проверить, не является ли текущий узел целевым } **foreach**(*child* **in** expand(*node*)) { // все преемники текущего узла, ... **if**(visited[*child*] == false) { // ... которые ещё не были посещены ... *queue*.push(*child*); // ... добавить в конец очереди... *visited*[*child*] = true; // ... и пометить как посещённые } } } **return** false; // Целевой узел недостижим }

## 17.4. Свойства

Обозначим число вершин и рёбер в графе как $|V|$ и $|E|$ соответственно.

### 17.4.1. Пространственная сложность

Так как в памяти хранятся все развёрнутые узлы, пространственная сложность алгоритма составляет $O(|V| + |E|)$ [1].

Алгоритм поиска с итеративным углублением похож на поиск в ширину тем, что при каждой итерации перед переходом на следующий уровень исследуется полный уровень новых узлов, но требует значительно меньше памяти.

### 17.4.2. Временная сложность

Так как в худшем случае алгоритм посещает все узлы графа, при хранении графа в виде списков смежности, временная сложность алгоритма составляет $O(|V| + |E|)$ [1][2].

### 17.4.3. Полнота

Если у каждого узла имеется конечное число преемников, алгоритм является полным: если решение существует, алгоритм поиска в ширину его находит, независимо от того, является ли граф конечным. Однако если решения не существует, на бесконечном графе поиск не завершается.

### 17.4.4. Оптимальность

Если длины рёбер графа равны между собой, поиск в ширину является оптимальным, то есть всегда находит кратчайший путь. В случае взвешенного графа поиск в ширину находит путь, содержащий минимальное количество рёбер, но не обязательно кратчайший.

Поиск по критерию стоимости является обобщением поиска в ширину и оптимален на взвешенном графе с неотрицательными весами рёбер. Алгоритм посещает узлы графа в порядке возрастания стоимости пути из начального узла и обычно использует очередь с приоритетами.

## 17.5. История и применения

Поиск в ширину был формально предложен Э. Ф. Муром в контексте поиска пути в лабиринте[3]. Ли независимо открыл тот же алгоритм в контексте разводки проводников на печатных платах[4][5][6].

Поиск в ширину может применяться для решения задач, связанных с теорией графов:

- Волновой алгоритм поиска пути в лабиринте[3]
- Волновая трассировка печатных плат[4]
- Поиск компонент связности в графе
- Поиск кратчайшего пути между двумя узлами невзвешенного графа
- Поиск в пространстве состояний: нахождение решения задачи с наименьшим числом ходов, если каждое состояние системы можно представить вершиной графа, а переходы из одного состояния в другое — рёбрами графа[1]
- Нахождение кратчайшего цикла в ориентированном невзвешенном графе[1]
- Нахождение всех вершин и рёбер, лежащих на каком-либо кратчайшем пути между двумя вершинами $a$ и $b$ [1]
- Поиск увеличивающего пути в алгоритме Форда-Фалкерсона (алгоритм Эдмондса-Карпа)

## 17.6. Примеры реализации

### 17.6.1. Python

```
def BFS(s,Adj): level = { s: 0 } parent = { s: None } i = 1 frontier = [s] while frontier: next = [] for u in frontier: for v in Adj[u]: if v not in level: level[v] = i parent[v] = u next.append(v) frontier = next i += 1
```

### 17.6.2. PHP

```
$graph = array( 'A' => array('B','C','D','Z'), 'B' => array('A','E'), 'C' => array('A','F','G'), 'D' => array('A','I'), 'E' => array('B','H'), 'F' => array('C','J'), 'G' => array('C'), 'H' => array('E','Z'), 'I' => array('D'), 'J' => array('J'), 'Z' => array('A','H')); function bfs($graph , $startNode = 'A') { $visited = array(); $queue = array(); $queue[] = $graph[$startNode]; $visited[$startNode] = true; while( count($queue) > 0 ) { $currentNodeAdj = array_pop($queue); foreach($currentNodeAdj as $vertex) { if(!array_key_exists($vertex,$visited)) { array_unshift( $queue , $graph[$vertex] ) ; } $visited[$vertex] = true; } } }
```

## 17.7. См. также

- Поиск с ограничением глубины

## 17.8. Примечания


[1] MAXimal :: algo :: Поиск в ширину в графе и его приложения

[2] НГТУ Структуры и алгоритмы обработки данных Обход графа в ширину

[3] E. F. Moore (1959), The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*, Harvard University Press, pp. 285–292.

[4] C. Y. Lee (1961), An algorithm for path connection and its applications. *IRE Transactions on Electronic Computers*, EC-10(3), pp. 346–365

[5] Cormen *et al*, Introduction to Algorithms, 3rd edition, p. 623

[6] *Mathematics Stack Exchange* Origin of the Breadth-First Search algorithm


## 17.9. Литература


- *T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein* Introduction to Algorithms. — 3rd edition. — The MIT Press, 2009. — ISBN 978-0-262-03384-8.. Перевод 2-го издания: *Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн* **Алгоритмы: построение и анализ** = Introduction to Algorithms. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 0-07-013151-1.

- *Ананий В. Левитин* **Глава 5. Метод уменьшения размера задачи: Поиск в ширину** // Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Aigorithms. — М.: Вильямс, 2006. — С. 215-218. — ISBN 0-201-74395-7.


## 17.10. Ссылки

- *Steven M. Rubin* Computer Aids for VLSI Design. Chapter 4: Synthesis Tools

- Волновой алгоритм поиска пути

- Поиск в ширину на Pascal и C++

# Глава 18

# Пчелиный алгоритм

**Пчелиный алгоритм** (в англоязычных статьях также встречаются названия **Artificial Bee Colony** (ABC) **Algorithm** и **Bees Algorithm**) — довольно новый алгоритм для нахождения глобальных экстремумов (максимумов или минимумов) сложных многомерных функций.

В информатике и исследовании операций, пчелиный алгоритм на основе алгоритма поиска впервые разработан в 2005 году. Он имитирует поведение питания стаи пчел. В базовой версии, алгоритм выполняет своего рода соседней поиск в сочетании со случайным поиском и может использоваться для функциональной и комбинаторной оптимизации.

## 18.1. Ссылки

- Природные алгоритмы. Алгоритм поведения роя пчел
- Пчелиный алгоритм для оптимизации функции

# Глава 19

# Радужная таблица

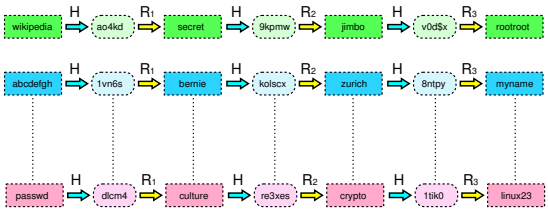



*Схема упрощенной радужной таблицы с длиной цепочек, равной трем. R1 R2 R3 — функции редукции, H — функция хеширования.*

**Радужная таблица** (англ. *rainbow table*) — специальный вариант таблиц поиска (англ. *lookup table*), использующий механизм разумного компромисса между временем поиска по таблице и занимаемой памятью (time-memory tradeoff). Радужные таблицы используются для вскрытия паролей, преобразованных при помощи сложнообратимой хеш-функции, а также для атак на симметричные шифры на основе известного открытого текста.

## 19.1. Предпосылки к появлению

Компьютерные системы, которые используют пароли для аутентификации, должны каким-то образом определять правильность введенного пароля. Самым простым способом решения данной проблемы является хранение списка всех допустимых паролей для каждого пользователя. Минусом данного метода является то, что в случае несанкционированного доступа к списку злоумышленник узнает все пользовательские пароли. Более распространенный подход заключается в хранении значений криптографической хеш-функции от парольной фразы. Однако большинство хешей быстро вычисляются, поэтому злоумышленник, получивший доступ к хешам, может быстро проверить список возможных паролей на валидность. Чтобы избежать этого, нужно использовать более длинные пароли, тем самым увеличивая список паролей, которые должен проверить злоумышленник. Для простых паролей, не содержащих соль, взломщик может заранее подсчитать значения хешей для всех распространенных и коротких паролей и сохранить их в таблице. Теперь можно быстро найти совпадение в заранее полученной таблице. Но чем длиннее пароль, тем больше таблица, и необходимо больше памяти для её хранения. Альтернативным вариантом является хранение только первых элементов цепочек хэшей. Это потребует больше вычислений для поиска пароля, но значительно уменьшит количество требуемой памяти. А радужные таблицы являются улучшенным вариантом данного метода, которые позволяют избежать коллизий.

## 19.2. Теория

Радужная таблица создается построением цепочек возможных паролей. Каждая цепочка начинается со случайного возможного пароля, затем подвергается действию хеш-функции и функции редукции. Данная функция преобразует результат хеш-функции в некоторый возможный пароль (например, если мы предполагаем, что пароль имеет длину 64 бита, то функцией редукции может быть взятие первых 64 бит хеша, побитовое сложение всех 64-битных блоков хеша и т. п.). Промежуточные пароли в цепочке отбрасываются и в таблицу записываются только первый и последний элементы цепочек. Создание таких таблиц требует больше времени, чем нужно для создания обычных таблиц поиска, но значительно меньше памяти (вплоть до сотен гигабайт, при объёме для обычных таблиц в N слов для радужных нужно всего порядка $N^{2/3}$). При этом они требуют хоть и больше времени (по сравнению с обычными методами) на восстановление исходного пароля, но на практике более реализуемы (для построения обычной таблицы для 6-символьного пароля с байтовыми символами потребуется $256^6$ = 281 474 976 710 656 блоков памяти, в то время как для радужной — всего $256^{6 \cdot 2/3}$ = 4 294 967 296 блоков).

Для восстановления пароля данное значение хеш-функции подвергается функции редукции и ищется в таблице. Если не было найдено совпадения, то снова применяется хеш-функция и функция редукции. Данная операция продолжается, пока не будет найдено совпадение. После нахождения совпадения цепоч-

ка, содержащая его, восстанавливается для нахождения отброшенного значения, которое и будет искомым паролем.

Следует отметить, что в радужных таблицах применяется упорядоченный набор различных функций редукции и перед i-тым хешированием в цепочке применяется функция редукции под номером i. Этим они отличаются от обычных цепочек хэшей, в которых используется одна и та же функция редукции. Благодаря набору функций редукции, радужные таблицы избегают слияния цепочек и имеют более высокую эффективность.

В итоге получается таблица, которая может с высокой вероятностью восстановить пароль за небольшое время.

Таблицы могут взламывать только ту хеш-функцию, для которой они создавались, то есть таблицы для MD5 могут взломать только хеш MD5. Теория данной технологии была разработана Philippe Oechslin[1] как быстрый вариант *time-memory tradeoff* [2]. Впервые технология использована в программе Ophcrack для взлома хешей LanMan (LM-хеш), используемых в Microsoft Windows. Позже была разработана более совершенная программа RainbowCrack, которая может работать с большим количеством хешей, например LanMan, MD5, SHA1 и др.

Следующим шагом было создание программы The UDC, которая позволяет строить Hybrid Rainbow таблицы не по набору символов, а по набору словарей, что позволяет восстанавливать более длинные пароли (фактически неограниченной длины).

## 19.3. Применение

При генерации таблиц важно найти наилучшее соотношения взаимосвязанных параметров:

- вероятность нахождения пароля по полученным таблицам;
- времени генерации таблиц;
- время подбора пароля по таблицам;
- занимаемое место.

Вышеназванные параметры зависят от настроек заданных при генерации таблиц:

- допустимый набор символов;
- длина пароля;
- длина цепочки;
- количество таблиц.

При этом время генерации зависит почти исключительно от желаемой вероятности подбора, используемого набора символов и длины пароля. Занимаемое таблицами место зависит от желаемой скорости подбора 1 пароля по готовым таблицам.

Хотя применение радужных таблиц облегчает использование метода грубой силы (bruteforce) для подбора паролей, в некоторых случаях необходимые для их генерации/использования вычислительные мощности не позволяют одиночному пользователю достичь желаемых результатов за приемлемое время. К примеру, для паролей длиной не более 8 символов, состоящих из букв, цифр и специальных символов **!@#$%^&*()-_+=**, захешированных алгоритмом MD5, могут быть сгенерированы таблицы со следующими параметрами:

- длина цепочки — 1400
- количество цепочек — 50 000 000
- количество таблиц — 800

При этом вероятность нахождения пароля с помощью данных таблиц составит 0,7542 (75,42 %), сами таблицы займут 596 Гб, генерация их на компьютере уровня Пентиум−3 1ГГц займёт 3 года, а поиск 1 пароля по готовым таблицам — не более 22 минут.

Однако процесс генерации таблиц возможно распараллелить, например, расчёт одной таблицы с вышеприведёнными параметрами занимает примерно 33 часа. В таком случае, если в нашем распоряжении есть 100 компьютеров, все таблицы можно сгенерировать через 11 суток.

## 19.4. Защита от радужных таблиц

Один из распространённых методов защиты от взлома с помощью радужных таблиц — использование необратимых хеш-функций, которые включают *salt* («соль», или «затравка», «модификатор»). Существует множество возможных схем смешения затравки и пароля. Например, рассмотрим следующую функцию для создания хеша от пароля:

хеш = MD5( пароль + соль ), или хеш = MD5( MD5(пароль) + соль ).

Для восстановления такого пароля взломщику необходимы таблицы для всех возможных значений соли. При использовании такой схемы, соль должна быть достаточно длинной (6-8 символов), иначе злоумышленник может вычислить таблицы для каждого значения соли, случайной и различной для каждого пароля. Таким образом два одинаковых пароля будут

иметь разные значения хешей, если только не будет использоваться одинаковая соль.

По сути, соль увеличивает длину и, возможно, сложность пароля. Если таблица рассчитана на некоторую длину или на некоторый ограниченный набор символов, то соль может предотвратить восстановление пароля. Например, в старых Unix-паролях использовалась соль, размер которой составлял всего лишь 12 бит. Для взлома таких паролей злоумышленнику нужно было посчитать всего 4096 таблиц, которые можно свободно хранить на терабайтных жестких дисках. Поэтому в современных приложениях стараются использовать более длинную соль. Например, в алгоритме хеширования bcrypt используется соль размером 128 бит.[3] Подобная длина соли делает предварительные вычисления просто бессмысленными. Другим возможным способом борьбы против атак, использующих предварительные вычисления, является растяжение ключа(англ. *key stretching*).Например:

ключ = хеш(пароль + соль) for 1 to 65536 do ключ = хеш(ключ + пароль + соль)

Этот способ снижает эффективность применения предварительных вычислений, так как использование промежуточных значений увеличивает время, которое необходимо для вычисления одного пароля, и тем самым уменьшает количество вычислений, которые злоумышленник может произвести в установленные временные рамки.[4] Данный метод применяется в следующих алгоритмах хеширования: MD5, в котором используется 1000 повторений, и bcrypt.[5] Альтернативным вариантом является использование усиления ключа(англ. *key strengthening*), который часто принимают за растяжение ключа. Применяя данный метод, мы увеличиваем размер ключа за счет добавки случайной соли, которая затем спокойно удаляется, в отличие от растяжения ключа, когда соль сохраняется и используется в следующих итерациях.[6] Также рассмотрим LM-хеш – это старый алгоритм хеширования, который используется в Microsoft. Он чрезвычайно уязвим, так пароль, состоящий больше, чем из 7 символов, но меньше, чем из 15 символов, разбивается на две части, которые хешируются независимо друг от друга. Поэтому, чтобы избежать создания LM-хеша, необходимо использовать пароли из 15 символов и более.[7]

## 19.5. Использование

Практически все дистрибутивы ОС Unix, GNU/Linux и BSD используют хеши с солью для хранения системных паролей, хотя многие приложения, например, интернет-скрипты, используют простой хеш (обычно MD5) без «соли». ОС Microsoft Windows и Windows NT используют хеши LM-хеш и NTLM, которые также не используют «соль», что делает их самыми популярными для создания радужных таблиц.

## 19.6. Примечания

[1] Philippe Oechslin

[2] Доклад Philippe Oechslin на конференции CRYPTO'03 (PDF)

[3] Alexander, Steven (June 2004). «Password Protection for Modern Operating Systems». *;login:* (USENIX Association) **29** (3).

[4] *Ferguson Neils* Practical Cryptography. — Indianapolis: John Wiley & Sons. — ISBN 0-471-22357-3.

[5] (June 6, 1999) «A Future-Adaptable Password Scheme». *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference* (USENIX Association).

[6] U. Manber, "A Simple Scheme to Make Passwords Based on One-Way Functions Much Harder to Crack," Computers & Security, v.15, n.2, 1996, pp.171-176.

[7] How to prevent Windows from storing a LAN manager hash of your password in Active Directory and local SAM databases, Microsoft

## 19.7. Внешние ссылки

- Страница Ophcrack Оригинальная исследовательская работа с демонстрацией
- Проект RainbowCrack
- oxid.it Содержит *winrtgen* — графическую оболочку для *rtgen* (утилиты создания таблиц)
- Создание радужных таблиц для A5/1 (используется в GSM сетях, пока только для видеокарт с CUDA)
- Большие радужные таблицы от группы Shmoo Group, создателей AirSnort

# Глава 20

# Троичный поиск

**Трои́чный по́иск** *(Тернарный поиск)* — это метод в информатике для поиска максимумов и минимумов функции, которая либо сначала строго возрастает, затем строго убывает, либо наоборот. Троичный поиск определяет, что минимум или максимум не может лежать либо в первой, либо в последней трети области, и затем повторяет поиск на оставшихся двух третях. Троичный поиск демонстрирует парадигму программирования «разделяй и властвуй».

## 20.1. Функция

Предположим, что мы ищем максимум функции $f(x)$, и что нам известно, что максимум лежит между $A$ и $B$. Чтобы алгоритм был применим, должно существовать некоторое значение $x$, такое что

- для всех $a$,$b$, для которых $A \le a < b \le x$, выполняется $f(a) < f(b)$, и
- для всех $a$,$b$, для которых $x \le a < b \le B$, выполняется $f(a) > f(b)$.

## 20.2. Алгоритм

double l = ..., r = ..., EPS = ...; // входные данные while (r - l > EPS) { double m1 = l + (r - l) / 3, m2 = r - (r - l) / 3; if (f (m1) < f (m2)) l = m1; else r = m2; }

## 20.3. См. также

- Алгоритмы поиска
- Двоичный поиск (используется для поиска точки, где производная меняет знак)
- Метод Ньютона (используется для поиска точки, где производная обращается в ноль)
- Метод золотого сечения (схож с троичным поиском, полезен, если за одну итерацию вычисление f занимает больше всего времени)
- Интерполирующий поиск
- Линейный поиск
- Троичные алгоритмы

# Глава 21

# Компьютерные шахматы

**Компьютерные шахматы** — популярный термин из области исследования искусственного интеллекта, означающий создание программного обеспечения и специальных компьютеров для игры в шахматы. Также термин «компьютерные шахматы» употребляется для обозначения игры против компьютерной шахматной программы, игры программ между собой.

## 21.1. История

*Шахматный компьютер*

История шахматных машин старше, чем история компьютеров. Идея создать машину, играющую в шахматы, датируется ещё восемнадцатым веком. Около 1769 года появился шахматный автомат «Механический турок». Он был предназначен для развлечения королевы Марии-Терезии. Машина действительно неплохо играла — внутри неё находился сильный шахматист, который и делал ходы.

Создание механических шахматных автоматов прекратилось с появлением цифровых компьютеров в середине XX века. В 1951 году Алан Тьюринг написал алгоритм, с помощью которого машина могла бы играть в шахматы, только в роли машины выступал сам изобретатель. Этот нонсенс даже получил название — «бумажная машина Тьюринга». Человеку требовалось более получаса, чтобы сделать один ход. Алгоритм был довольно условный, и сохранилась даже запись партии, где «бумажная машина» Тьюринга проиграла одному из его коллег[1]. За отсутствием доступа к компьютеру, программа ни разу не проверялась в работе.

Примерно в это же время, в 1951 году, математик Клод Шеннон написал свою первую статью о шахматном программировании. Он писал: «*Хотя, возможно, это и не имеет никакого практического значения, сам вопрос представляется теоретически интересным, и будем надеяться, что решение этой задачи послужит толчком для решения других задач аналогичной природы и большего значения*». Шеннон также отметил теоретическое существование лучшего хода в шахматах и практическую невозможность его найти.

Следующим шагом в развитии шахматного программирования стала разработка в ядерной лаборатории Лос-Аламоса в 1952 году на компьютере Maniac 1 (тактовая частота 11 кГц) шахматной программы для игры на доске 6x6, без участия слонов. Известно, что этот компьютер сыграл одну партию против сильного шахматиста, она продолжалась 10 часов и закончилась победой шахматиста. Ещё одна партия была сыграна против девушки, которая недавно научилась играть в шахматы. Машина победила на 23-м ходу. Сейчас это выглядит смешно, но для своего времени это было большое достижение.

В 1957 году Алексом Бернстейном была создана первая программа для игры на стандартной шахматной доске и при участии всех фигур[2].

Важное событие для компьютерных шахмат произошло в 1958 году, когда Аллен Ньюэлл, Клифф Шоу и Герберт Саймон разработали алгоритм уменьшения дерева поиска, названный Альфа-бета отсечение[2][3], на основе которого построены функции поиска всех сильных современных программ.

Первой же машиной, которая достигла уровня шахматного мастера, была Belle, законченная в 1983 г. Джо Кондоном и Кеном Томпсоном. «Belle» был первым компьютером, спроектированным только для игры в шахматы. Его официальный рейтинг Эло был 2250, таким образом, это была самая сильная шахматная машина своего времени.

В 1994 Гарри Каспаров проиграл программе Fritz 3 турнирную блиц-партию в Мюнхене. Программа также выиграла у Вишванатана Ананда, Бориса Гельфанда и Владимира Крамника. Гроссмейстер Роберт Хюбнер отказывался играть против программы и автоматически проиграл. Каспаров сыграл второй матч с *Fritz* и победил с 4 выигрышами и 2 ничьими.

В феврале 1996 года Гарри Каспаров победил шахматный суперкомпьютер Deep Blue со счетом 4-2. Этот матч выдающийся тем, что первую партию выиграл *Deep Blue*, автоматически став первым компьютером, победившим чемпиона мира по шахматам в турнирных условиях. *Deep Blue* вычислял 50 миллиардов позиций каждые три минуты, в то время как Каспаров 10 позиций за это же время. В *Deep Blue* было 200 процессоров. С тех пор шахматные энтузиасты и компьютерные инженеры создали много шахматных машин и компьютерных программ.

Шахматные компьютеры сейчас доступны по очень низкой цене. Появилось много программ с открытыми кодами, в частности *Crafty*, *Fruit* и *GNU Chess*, которые можно свободно загрузить из сети Интернет и которые могут победить многих профессиональных шахматистов. А лучшие коммерческие программы, например, *Shredder* или *Fritz* уже превысили уровень людей-чемпионов. Между тем именно движок с открытым кодом Stockfish 5 находится на первом месте в таких компьютерных рейтинг-листах, как CEGT, CCRL,SCCT и CSS., благодаря совместной разработке и тестированию многих людей. Stockfish Testing Queue

## 21.2. Мотивация

Первыми мотивами для компьютеризации шахмат было желание развлечься, создать программы для компьютерных шахматных турниров и провести научное исследование, которое позволило бы глубже понять познавательную способность человека. Для первых двух целей компьютерные шахматы имели феноменальный успех: от первых попыток к созданию шахматной программы, которая могла на равных соперничать с лучшими шахматистами, прошло менее пятидесяти лет.

А. С. Кронрод определил роль компьютерных шахмат известной фразой: «шахматы — это дрозофила искусственного интеллекта». Аналогия лежит на поверхности: шахматы представляют собой безусловно интеллектуальную, но при этом чётко формализованную, простую по структуре и компактную задачу, то есть являются удобным объектом лабораторных исследований в искусственном интеллекте, так же как мушка-дрозофила, благодаря малым размерам, плодовитости и быстрой смене поколений является удобным лабораторным объектом для изучения наследственности. Действительно, на шахматах были апробированы многие известные методы и направления искусственного интеллекта, в том числе методики оптимизации перебора (уход от «комбинаторного взрыва» при просчёте вариантов вперёд на несколько ходов), распознавание образов, экспертные системы, логическое программирование.

Однако, к удивлению и огорчению многих, шахматы мало приблизили людей к созданию машин с человекоподобным интеллектом. Современные шахматные программы, по сути, остановились на наиболее примитивном этапе интеллектуальной деятельности: они исследуют огромное число возможных ходов обоих игроков, применяя различные методы усечения дерева перебора, в том числе относительно простую функцию оценки. В сочетании с базами данных, хранящими заранее рассчитанные готовые варианты дебютов и эндшпилей, благодаря быстродействию и объёмам памяти современных компьютеров эти методы уже обеспечивают игру компьютера в шахматы на гроссмейстерском уровне. По этим причинам компьютерные шахматы больше не имеют такого большого академического интереса. Роль «дрозофилы искусственного интеллекта» перешла к другим интеллектуальным играм, таким как, например, го. Гораздо больший, чем в шахматах, объём перебора вариантов в таких играх ограничивает возможности использования простых методов и требует от ученых применять более умозрительные подходы к игре.

## 21.3. Проблемы реализации

Разработчики шахматных программ должны сделать ряд решений при их написании. Они включают:

- Способ изображения шахматной доски — представление цельной позиции как структуры данных.
- Методы поиска — поиск возможных лучших ходов.
- Листовая оценка — оценка позиции без учета дальнейших ходов.

См. также:

- Минимакс
- Альфа-бета отсечение
- Killer эвристика
- Итерационное заглубление
- Эвристика нулевого хода

## 21.4. Структура шахматной программы

Первое исследование на тему шахматного программирования сделал в 1950 году американский математик Клод Шеннон, успешно предусмотревший два основных возможных метода поиска, которые можно использовать, и назвал их «Тип А» и «Тип В».

Программы типа А используют так называемый подход «грубой силы» (*brute force*), изучая каждую возможную позицию на фиксированную глубину с помощью алгоритма Минимакс. Шеннон утверждал, что этот метод будет непрактичным по двум причинам.

Во-первых, с примерно тридцатью ходами, возможными в типичной позиции, на изучение около 10 млрд узловых позиций (просчет примерно на три хода вперед для обеих сторон), надо примерно 16 минут, даже в «очень оптимистичном» случае, когда компьютер сможет оценивать миллион позиций в секунду. (Чтобы достичь этого понадобилось сорок лет.)

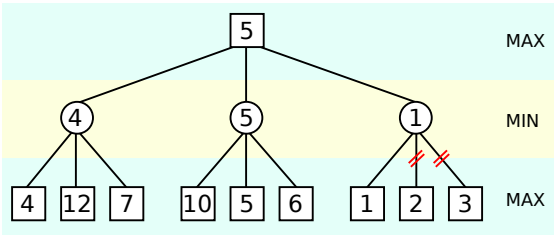
Во-вторых, программы Типа А пренебрегали так называемой проблемой статического состояния, пытаясь оценить позицию в начале обмена фигур или другой важной последовательности ходов (например тактических комбинаций). Поэтому Шеннон предполагал, что с применением алгоритма Типа А число позиций, которые надо исследовать чрезвычайно возрастет, что значительно замедлит программу. Вместо бесполезной траты вычислительной мощности компьютера для исследования плохих или незначительных ходов Шеннон предложил использовать программы Типа В. Этот метод имеет два усовершенствования:

1. Применяется поиск «по спокойствию» (*quietness*).
2. Исследует не все, а только некоторые пригодные ходы для каждой позиции.

Это давало программам возможность просчитывать важные ходы на бóльшую глубину и делать это за приемлемое время. Первый подход выдержал испытание временем: все современные программы применяют конечный поиск «по спокойствию» перед оценкой позиции.

### 21.4.1. Основные алгоритмы современных программ

Компьютерные шахматные программы рассматривают шахматные ходы как игровое дерево. Теоретически, они должны оценивать все позиции, которые возникнут после всех возможных ходов, затем все возможные ходы после этих ходов и т. д. Каждый ход




*Примерная схема осуществления альфа-бета отсечения слабых ходов*

одного игрока называется «*узел*». Перебор ходов продолжается, пока программа не достигает максимальной глубины поиска или определяет, что достигнута конечная позиция (например мат или пат). Уже на основании оценки позиции выбирает оптимальную стратегию. В каждой позиции количество возможных ходов игрока примерно равно 35. Для полного анализа четырёх ходов (по два хода каждого игрока) нужно исследовать около полутора миллиона возможностей, для шести — почти два миллиарда. Анализ на 3 хода вперед — очень мало для хорошей игры.

Программисты пытаются по-разному ограничить массу ходов, которые надо перебрать (обрезание дерева поиска — *game tree pruning*). Самым популярным является *альфа-бета отсечение*, в котором не рассматриваются позиции, имеющие меньшую оценку, чем уже оценённые.

Приблизительная программная реализация:

```
private int AlphaBeta(int color, int Depth, int alpha, int beta) { if (Depth == 0) return Evaluate(color); int bestmove; Vector moves = GenerateMoves(); for(int i = 0; i < moves.size(); i++) { makeMove(moves.get(i)); eval = -AlphaBeta(-color, Depth-1, -beta, -alpha); unmakeMove(moves.get(i)); if(eval >= beta) return beta; if(eval > alpha) { alpha = eval; if (Depth == defaultDepth) { bestmove = moves.get(i); } } } return alpha; }
```

Пример первого вызова:

```
AlphaBeta(1, 6, Integer.MIN_VALUE, Integer.MAX_VALUE);
```

При первом вызове метод (функция) вызывается с максимальным окном. При рекурсивных вызовах переменные alpha и beta меняются местами с инверсией знака и «сужают» массу поиска.

Вторым распространенным методом является итерационное заглубление. Сначала перебирается дерево игры до определенной глубины, после чего выделяется несколько лучших ходов. Затем программа оценивает эти ходы применительно к большей глубине, чтобы узнать больше об их последствиях. Эта операция повторяется до наилучшего с точки

зрения программы хода. Такой подход позволяет быстро отбросить немалый процент неперспективных вариантов игры. Например, не имеет смысла исследовать, что произойдет, когда обменять ферзя на пешку, если в позиции есть лучшие ходы.

Важный элемент шахматных алгоритмов — это система оценки позиции. Нельзя абсолютно точно оценить позицию, ибо для этого нужно было бы проанализировать триллионы последовательностей ходов от начала и до завершения партии. Если бы существовала функция, которая давала бы возможность достоверно оценить позицию, задача игры в шахматы упростилась бы к оценке каждого из нескольких десятков доступных в данный момент ходов, и не надо было бы вычислять дальнейшие ходы.

Следовательно, оценка программой позиции очень приблизительная, хотя оценочные функции программ постоянно совершенствуются. Функции оценки обычно оценивают позиции в сотых частях пешки. Эти функции оценивают только несколько простых параметров:

1. Во-первых, это оценка материала: каждая пешка — это 1 пункт, слон и конь — по 3, ладья — 5, ферзь — 9. Король иногда ценится в 200 пешек (Статья Шеннона) или 1 000 000 000 пешек (программа разработана в СССР в 1961 г.), чтобы гарантировать, что мат перевесит все другие факторы. Более развитые функции имеют точнее установленные коэффициенты ценности фигур, которые зависят от стадии партии и позиции на шахматной доске.

2. Во-вторых, позиционное преимущество, которое зависит от положения фигур на доске; например, заблокированная фигура ценится меньше, чем свободная; оценивается также безопасность короля, господство над центром доски и т. д.; существуют также более сложные системы оценки (некоторые даже используют знания о нейронных сетях), однако даже такая простая функция позволяет программе играть очень сильно; в шахматах главная проблема заключается не в оценке позиции, а в переборе дерева возможных ходов.

Функции оценки позиции бывают неэффективны, когда ситуация на доске резко меняется с каждым ходом, когда, например, идёт обмен фигур или реализуется какая-нибудь шахматная комбинация. Отсюда возникло понятие статического состояния (*quiescent*) и горизонта вычисления. В статическом состоянии на шахматной доске идёт медленная позиционная борьба, а достойный внимания горизонт вычисления очень широк. Это означает, что решающая перемена не наступит в том будущем, которое удается легко предвидеть. В такой ситуации большую роль играют функции оценки позиции, нежели попытки вычисления возможных вариантов.

В динамичной ситуации игра, опирающаяся на функцию оценки позиции, может привести к совершенно ошибочным решениям. В крайнем случае, если программа имеет коротко настроенный горизонт вычисления и в ней учитывается только кратковременная оценка позиции, то конец может прийтись как раз на момент, когда идёт обмен ферзей, и один из них может быть уже побит, а второй взамен ещё нет. Оценка программой такого состояния ведёт к совершенно ошибочному выводу, что один из игроков имеет огромное преимущество, тогда как оно исчезнет через один ход, которого, однако, программа не видит. Если состояние ещё не статическое, нужно продолжить обмен до конца и оценить ситуацию, когда уже нет возможных радикальных изменений. Люди в целом интуитивно различают эти две ситуации — шахматные же программы должны иметь набор критериев, позволяющих изменять способ функционирования в статических и динамических состояниях.

Труднее дать оценку ходам в дебюте. Большинство программ используют при этом написанные заранее дебютные библиотеки, в которых есть определённое небольшое количество начальных ходов и ответов к определенному числу ходов, которое не является постоянным, потому что зависит от типа дебюта.

## 21.5. Компьютер против Человека

Даже в 1970—80-х годах оставался открытым вопрос, когда шахматная программа сможет победить сильнейших шахматистов. В 1968 году международный гроссмейстер Дэвид Леви пошел на пари, что ни один компьютер не сможет обыграть его в течение ближайших десяти лет. Он выиграл пари, победив в 1978 году программу Chess 4.7 (сильнейшую в то время), но сознавал, что осталось не так уж много времени до того, когда компьютеры будут побеждать мировых чемпионов. В 1989 году программа Deep Thought выиграла у Леви.

Но программы все еще были значительно ниже уровня чемпиона мира, который продемонстрировал Гарри Каспаров, победив ту же *Deep Thought* дважды в 1991 году.

Это длилось до 1996 года, когда состоялся матч Каспарова с компьютером Deep Blue фирмы IBM, где чемпион проиграл свою первую партию. Впервые компьютерная шахматная программа обыграла чемпиона мира при стандартном часовом контроле. Однако Каспаров изменил свой стиль игры, выиграв три и сведя вничью две из оставшихся пяти партий. В мае 1997 года усовершенствованная версия *Deep Blue* на-

несла поражение Каспарову со счетом 3,5-2,5. В 2003 году был снят документальный фильм, в котором исследовались упреки Каспарова по поводу возможного использования шахматиста IBM, под названием «Матч окончен: Каспаров и машина» (англ. *Game Over: Kasparov and the machine*). В фильме утверждалось, что сильно раскрученная победа *Deep Blue* подстроена для увеличения рыночной стоимости IBM. Частично эти упреки были оправданными. Правила позволяли разработчикам изменять программу между играми. *Deep Blue* был изменен между партиями для лучшего понимания машиной стиля игры Каспарова, помогая избежать ловушки в эндшпиле, в которую дважды попадал искусственный интеллект.

Матч *Deep Blue* против Каспарова 1996, первая партия.

IBM разобрала Deep Blue после матча, с тех пор этот компьютер не играл ни разу. Впоследствии происходили другие матчи «Человек против Машины».

Имея все большую вычислительную мощность, шахматные программы, запущенные на персональных компьютерах, стали достигать уровня лучших шахматистов. В 1998 году программа Rebel 10 победила Вишванатана Ананда, который тогда занимал второе место в мире. Однако не все партии игрались со стандартным временным контролем. Из восьми партий матча, четыре играли с блиц-контролем (пять минут плюс пять секунд за каждый ход), которые *Rebel* выиграл со счетом 3-1. Еще две игры были с полу-блиц контролем (пятнадцать минут на каждого), которые программа также выиграла (1,5-1). Наконец, две последние партии были сыграны со стандартным турнирным временным контролем (два часа на 40 ходов и час на остальные партии) и тут выиграл уже Ананд со счетом 0,5-1,5. К тому времени в быстрых партиях компьютеры играли лучше людей, но при классическом временном контроле преимущество компьютера над человеком было уже не так велико.

В 2000 году коммерческие шахматные программы Junior и Fritz смогли свести в ничью матчи против предыдущих мировых чемпионов Гарри Каспарова и Владимира Крамника.

В октябре 2002 года Владимир Крамник и Deep Fritz соревновались в матче из восьми партий в Бахрейне. Матч закончился вничью. Крамник выиграл вторую и третью партии, используя традиционную противокомпьютерную тактику — играл осторожно, имея целью долгосрочное преимущество, которое компьютер не может увидеть в своем дереве поиска. И всё же *Fritz* выиграл пятую партию после грубой ошибки Крамника. Шестую партию много турнирных комментаторов назвали очень увлекательной. Крамник, имея лучшую позицию в начале миттельшпиля, попытался пожертвовать фигуру, чтобы создать сильную тактическую атаку (такая стратегия очень рискована против компьютеров). *Fritz* нашел сильную защиту, и эта атака значительно ухудшила позицию Крамника. Крамник сдал игру, веря, что партия проиграна. Однако последующий анализ показал, что *Fritz* вряд ли смог бы довести игру до своего выигрыша. Последние две партии закончились вничью.

В январе 2003 года Гарри Каспаров играл против программы Junior в Нью-Йорке. Матч закончился со счётом 3-3.

В ноябре 2003 года Гарри Каспаров играл с X3D Fritz. Матч закончился со счётом 2-2.

В 2005 году *Hydra*, специальный шахматный программно-аппаратный комплекс с 64 процессорами, победил Майкла Адамса — шахматиста, который в то время был на седьмом месте в мире по рейтингу Эло — в матче из шести партий со счётом 5,5-0,5 (хотя домашняя подготовка Адамса была намного ниже, чем у Каспарова в 2002 году). Некоторые комментаторы верили, что *Hydra* наконец получит несомненное преимущество над лучшими шахматистами.

В ноябре-декабре 2006 года Владимир Крамник играл с программой *Deep Fritz*. Матч закончился со счётом 2-4.

## 21.6. Базы данных эндшпиля

*Подробнее Базы данных эндшпиля*

Компьютеры используются для анализа некоторых эндшпильных позиций. Такие базы данных эндшпиля создаются, используя ретроспективный анализ, начиная с позиций, где конечный результат известен (например, где одной стороне был поставлен мат) и видя какие еще позиции есть на расстоянии хода, затем на один ход от этих и т. д. Кен Томпсон, известный как главный проектировщик операционной системы UNIX, был пионером в этой области.

Игра в эндшпиле долго была заметной слабостью шахматных программ, так как глубина поиска была недостаточной. Таким образом, даже программы, которые играли в силу мастера не в состоянии выиграть в эндшпильних позициях, где даже шахматист средней силы мог форсировать выигрыш.

Но результаты компьютерного анализа иногда удивляли людей. В 1977 г. шахматная машина Томпсона Belle, используя эндшпильные базы данных король + ладья против короля + ферзь, была способна свести вничью теоретически проигрышные эндшпили против титулованных шахматистов.

Большинство гроссмейстеров отказывались играть против компьютера в эндшпиле *ферзь против ладьи*, но Уолтер Браун принял вызов. Позицию расставили так, что теоретически можно было выиграть в 30 хо-

дов с безупречной игрой. Брауну дали два с половиной часа на пятьдесят ходов. После сорока пяти ходов Браун согласился на ничью, будучи не способным выиграть в последние пять ходов. В конечной позиции, Браун мог поставить мат только через семнадцать ходов.

В 2002 г. были опубликованы основные форматы эндшпильных баз данных, включая Edward Tablebases, De Koning Endgame Database и Nalimov Endgame Tablebases, которые теперь поддерживают многие шахматные программы, такие как Rybka, Shredder и Fritz. Эндшпили с пятью или менее фигурами были полностью проанализированы. Эндшпили с шестью фигурами были проанализированы, за исключением позиций с пятью фигурами против одинокого короля. Марк Буржуцкий и Яков Коновал проанализировали некоторые эндшпили с семью фигурами. Во всех этих эндшпильных базах данных считается, что рокировка невозможна.

Базы данных генерируются с помощью хранения в памяти оценок позиций, которые возникали до сих пор, и используют этих результаты для уменьшения дерева поиска, если такие позиции возникнут снова. Простая целесообразность запоминания оценок всех ранее достигнутых позиций означает, что ограничивающим фактором при решении эндшпиля является просто количество памяти, которую имеет компьютер. С ростом ёмкости компьютерной памяти, эндшпили повышенной сложности рано или поздно будут решены.

Компьютер, использующий базу данных эндшпиля будет, при достижении позиции в них, способен играть безупречно и безотлагательно определять, является ли позиция выигрышной, проигрышной или ничейной, а также находить самый быстрый и самый долгий способ достижения результата. Знание точной оценки позиции также полезно при увеличении силы компьютера, так как это позволит программе выбирать пути достижения цели в зависимости от ситуации [то есть упрощая и разменяваясь получить четко исследованную позицию].

Базы эндшпиля Налимова на пять фигур, которые используют методы современной компрессии, занимают 7.05 Гб на жестком диске. Для хранения баз данных на шесть фигур надо примерно 1.2 терабайт. Оценено, что полная семифигурная база данных потребует больше места, чем будет доступно рядовым пользователям в ближайшем будущем.

## 21.7. Игра против программ

Компьютеры ощутимо опережают людей в коротких тактических маневрах, которые находятся в пределах глубины поиска программы. Особенно опасным в таких случаях является ферзь, который прекрасно подходит для кратковременных маневров. Поэтому в игре против компьютера люди часто делают попытку побудить программу к размену ферзей. Это происходит, например, когда человек в начале партии намеренно ухудшает свою позицию, а компьютер расценивает её, как выгодную ему. Если программа устанавливает оценку позиции как преимущественную для себя, то, скорее всего, будет разменивать фигуры, а это выгодно человеку. Конечно, программисты узнали о таких «трюках», и это учитывается в последних версиях их программ.

Вместо этого шахматисты должны играть против компьютера долгосрочными маневрами, которые программа не может увидеть в рамках своей глубины поиска. Например, Крамник в партии с *Deep Fritz* выиграл с помощью долгосрочного продвижения проходной пешки, которую *Fritz* обнаружил слишком поздно.

## 21.8. Шахматы и другие игры

Успех шахматных программ внушает мысль, что можно написать программы, которые играли бы так же хорошо и в другие игры, например сёги или го.

Похожие алгоритмы, пожалуй, можно бы использовать и во время игры в других разновидностях шахмат. В *сёги* больше возможных ходов, материальное преимущество значит гораздо меньше, зато намного существеннее позиционное преимущество. Строятся сложные системы, имеющие целью гарантировать королю безопасность, но оценка этих систем для компьютера нелегка. Количество фигур в этой игре постоянно, а потому игра не упрощается со временем, что делает невозможным создать базу эндшпилей. Нет здесь также вполне статических состояний, ведь игра на протяжении всего времени сводится к позиционной борьбе. Поэтому написать хорошую программу для игры в сёги значительно тяжелее, чем шахматную программу, хотя огромный опыт в шахматных играх можно приложить и к этой игре.

Настоящим вызовом для программистов стало го. Сложность вычисления го на несколько порядков больше, чем в шахматах. На каждом шаге возможны около 200—300 ходов, статическая же оценка жизни групп пешек фактически невозможна. Одним ходом здесь можно вполне испортить всю игру, даже если остальные ходы были успешны. Поэтому программы для игры в го не используют таких алгоритмов, как шахматные программы, и обычно имеют несколько десятков модулей для оценки различных аспектов игры и при анализе пытаются пользоваться теми же понятиями, что и люди. Несмотря на это, компьютеры в го играют еще очень слабо и проигрывают даже не очень сильным любителям.

## 21.9. Хронология компьютерных шахмат

- 1769, Вольфганг Кемпелен, построил шахматиста-автомата, который стал одной из величайших мистификаций этого периода.
- 1868, Чарльз Хупер представил автомат *Ajeeb* — в котором тоже был спрятан шахматист.
- 1912, Леонардо Торрес Квеведо построил машину, которая могла играть эндшпиль Король + Ладья против короля.
- 1948, книга Норберта Винера «Кибернетика» описывает, как можно создать шахматную программу, используя поиск минимакса с лимитированной глубиной и оценочной функцией.
- 1950, Клод Шеннон опубликовал «Программирование компьютера для игры в шахматы», одну из первых статей о компьютерных шахматах.
- 1951, Алан Тьюринг разработал на бумаге первую программу, способную играть в шахматы.
- 1952, Дитрих Принц разработал программу, которая решала шахматные задачи.
- 1956, г. Лос-Аламос — первая подобная шахматам игра, которую смогли играть программы, разработанная Полом Штейном и Марком Уэллсом для компьютера MANIAC I.
- 1956, Джон Маккарти изобрел альфа-бета алгоритм поиска.
- 1958, NSS стала первой программой, которая использовала альфа-бета алгоритм поиска.
- 1958, первыми шахматными программами, которые могли играть полные шахматные партии, стали одна, созданная Алексом Бернштайн и вторая российскими программистами.
- 1962, первой программой, которая играла правдоподобно стала Kotok-McCarthy.
- 1966—1967, первый матч между программами.
- 1967, Mac Hack Six, разработанная Ричардом Гринблатт стала первой программой, победившей человека при турнирном контроле времени.
- 1970, первый год Североамериканского Компьютерного Шахматного Чемпионата.
- 1974, Каисса выиграла первый Всемирный Компьютерный Шахматный Чемпионат.
- 1977, создана первая шахматная программа для микрокомпьютеров CHESS CHALLENGER.
- 1977, создание Международной Компьютерной Шахматной Ассоциации.
- 1977, Chess 4.6 стал первым шахматным компьютером, который добился успеха в главном шахматном турнире.
- 1980, первый год Всемирного микрокомпьютерного Шахматного Чемпионата.
- 1981, Cray Blitz выиграл Чемпионат Штата Миссисипи с 5-0 счетом и рейтингом производительности 2258.
- 1982, аппаратный шахматный игрок Кена Томпсона Belle зарабатывает титул мастера США.
- 1988, HiTech, разработанная Гансом Берлинером и Карлом Ебелингом, выигрывает матч против гроссмейстера Арнольда Денкер со счетом 3.5 — 0.5.
- 1988, Deep Thought делит первое место с Тони Майлзом в Чемпионате ПО Toolworks, впереди бывшего чемпиона мира Михаила Таля и нескольких гроссмейстеров, в частности Самуэля Решевского, Уолтера Брауна, Эрнста Грюнфельда и Михаила Гуревича. Программа также нанесла поражения гроссмейстер Бенту Ларсену, и стала первым компьютером, который выиграл у гроссмейстера в турнире.
- 1992, впервые микрокомпьютер, Chessmachine Gideon 3.1, разработанный Эдом Шредером (Ed Schröder) выигрывает VII Всемирный Компьютерный Шахматный Чемпионат впереди суперкомпьютеров.
- 1997, Deep Blue выиграл матч против Гарри Каспарова 2-1 = 3.
- 2002, Владимир Крамник свел вничью матч против Deep Fritz.
- 2003, Каспаров сыграл вничью матч против Deep Junior.
- 2003, Каспаров сыграл вничью матч против X3D Fritz.
- 2005, Hydra выиграла матч с Майклом Адамсом со счетом 5,5-0,5.
- 2005, команда компьютеров (Hydra, Deep Junior и Fritz), обыграла 8.5-3.5 команду из шахматистов (Веселин Топалов, Руслан Пономарев и Сергей Карякин), которые имели средний ЭЛО рейтинг 2681.
- 2006, чемпион мира, Владимир Крамник, побежден 4-2 Deep Fritz.

## 21.10. Теоретики компьютерных шахмат

- Дэвид Леви
- Роберт Хайатт (автор шахматной программы Crafty)
- Ганс Берлинер
- Клод Шеннон
- Давид Бронштейн

## 21.11. См. также

- Шахматная программа

## 21.12. Примечания

[1] Alan Turing vs Alick Glennie // «Turing Test», 1952

[2] Early Computer Chess Programs // Bill Wall's Wonderful World of Chess

[3] Computer Chess History by Bill Wall

## 21.13. Ссылки

- История компьютерных шахмат (недоступная ссылка с 13-05-2013 (522 дня) — *история*)
- Защита Чести Человечества — статья Тима Краббе об «антикомпьютерном» стиле шахмат
- Computer-Chess Club — место, где профессиональные авторы обсуждают свои программы
- Компьютерная шахматная теория Колина Фрауна

# Глава 22

# Chess Engines Grand Tournament

**Chess Engines Grand Tournament**, также известный как **CEGT**, является одной из самых известных организаций, которая проверяет компьютерное шахматное программное обеспечение, запуская шахматные программы против друг друга и производя рейтинговую таблицу оценок.

CEGT обычно проверяет любительские и профессиональные шахматные программы в различных форматах времени, таких как 40/4 (40 ходов за 4 минуты, с повторениями), 40/40 (40 ходов за 40 минут, с повторениями) и 40/120 (40 ходов за 120 минут, с повторениями). Контроль времени 40/120 считают наиболее объективным форматом игры в компьютерные шахматы, свободно доступной он-лайн.

На сентябрь 2009 года тестирующая группа состояла из 8 участников, координированных Хайнцем ван Кемпеном, использовав в общей сложности 20 персональных компьютеров. [1] Группа выполнила больше чем 120,000 игр. Игры включают SMP тестирование.

На 23 марта 2010 года возглавляет GECT 40/120 — Quad рейтинг-лист шахматная программа Rybka 3 x64 4CPU с 3149 пунктами. Второе место занимает Naum 4 x64 4CPU с 3088 пунктами. Третье место занимает Deep Fritz 12 4CPU с 3071 пунктами [2].

## 22.1. Примечания

[1] CEGT testers. Архивировано из первоисточника 6 апреля 2012.

[2] GECT 40/120 - Quad Tournament Time Control. Архивировано из первоисточника 6 апреля 2012.

## 22.2. См. также

- Компьютерные шахматы
- Computer Chess Rating Lists
- Swedish Chess Computer Association

## 22.3. Ссылки

- Official site (англ.) (Проверено 21 августа 2009)
- the CEGT rating list (англ.) (Проверено 21 августа 2009)

# Глава 23

# ChessVegas

**ChessVegas** (по-русски произносится как чессвегас) — свободный шахматный сайт, предназначенный для онлайн игры в шахматы. Основан осенью 2007 года.

Не требуется скачивать и устанавливать какую-либо программу-клиент. Для игры используется любой браузер, поддерживающий JavaScript и Flash.

## 23.1. Основные возможности

- Проведение шахматных турниров в режиме реального времени. На данный момент реализованы такие схемы турниров, как командные, круговые, групповые и нокаут-турниры. Кроме того, реализована швейцарская система
- Игра не только в блиц, но и в экстрим-блиц с контролем времени от 10 до 45 секунд на партию.
- Трансляция партий в режиме реального времени.

## 23.2. Технологии

Серверная часть разработана на применении ряда перспективных технологий. В основе лежит использование открытого протокола AMQP и механизма OLTP в СУБД PostgreSQL. В качестве AMQP-брокера выбран сервер RabbitMQ. Для увеличения производительности активно используются такие решения, как mod_perl, memcached и nginx.

## 23.3. Интерфейс

Основа интерфейса - JavaScript-библиотека Dojo, встроенный AJAX-модуль которой позволяет минимизировать трафик.

## 23.4. Системные требования

- В браузере должен быть установлен Flash-плагин.
- Должен быть открыт исходящий трафик на порты 843 и 5672.

## 23.5. Ссылки

- Официальный сайт
- Английская версия
- Интервью с Риком Смитом о будущем стандарта HTML5, где упоминается ChessVegas, как пример реализации проекта на технологии WebSockets

# Глава 24

# Commodore Chessmate

**Commodore Chessmate** — один из первых шахматных компьютеров, выпущенный компанией Commodore в июне 1978 года в США, Англии и Германии[1]. Впервые был представлен на CES 1978 года в Чикаго[1].[2]

Commodore Chessmate является одним из первых шахматных компьютеров, поступивших в массовую продажу. Самая первая подобная машина — Fidelity Chess Challenger 10, выпущенная компанией Fidelity Electronics, — появилась на прилавках всего годом ранее — весной 1977 года[3].

Chessmate базируется на одноплатном компьютере KIM-1, произведённом также компанией Commodore. В нём используется программа Microchess 1.5, написанная 18 декабря 1976 года программистом Питером Дженнигсом (англ. *Peter Jennings*) для компьютеров KIM-1[4][5]. Позже на основе этой программы были выпущены другие шахматные компьютеры — Novag Chess Champion MK II в 1979 году и TEC Schachcomputer в 1981 году. Особым успехом компьютер не пользовался, поэтому уже в 1980 году Commodore остановила выпуск Chessmate.[2]

Commodore Chessmate имеет корпус из кремово-белого или голубого (намного реже) пластика размером 22x16x5 сантиметров[1]. Он использует 8-битный процессор MOS Technology 6504 на тактовой частоте 1 МГц, 5 Кб ROM и 320 байт RAM, питание от сети. Управление осуществляется с помощью 19-клавишной мембранной клавиатуры наподобие ZX80. Сборка компьютеров производилась в Гонконге.[1][6][2]

Наиболее популярен компьютер был в Германии[6], где он появился летом 1979 года по цене 199 немецких марок[7] (по другой версии — 395[2]). Немецкая газета ELO (de), посвящённая электротехнике, в майском номере 1979 года написала по этому поводу[2]:

> Компания Commodore собирается выпустить на рынок шахматный компьютер Chess-Mate. На компьютере есть возможность выбора из восьми уровней сложности, а его заявленная цена составляет 395 немецких марок. Управление производится с помощью клавиатуры, ходы отображаются на четырёх 7-сегментных светодиодных дисплеях. Игровое время также контролируется самой машиной.
>
> *Оригинальный текст* (нем.)
>
> *Einen Schachcomputer unter der Bezeichnung Chess-Mate wird die Firma Commodore auf den Markt bringen. Das Gerät soll 395 DM kosten und auf acht Spielstärken einstellbar sein. Die Züge werden mit Sensortasten eingegeben und von Siebensegmentziffern angezeigt. Auch die Spielzeit wird vom Chess-Mate kontrolliert.*

Chessmate имеет 8 уровней сложности[8] и библиотеку из 32 дебютов, каждый по 16 ходов, в том числе такие варианты начала партии как дебют слона, северный гамбит, итальянская партия, дебют Бёрда и другие[9]. Рейтинг Эло у Chessmate составляет 1050[10], что совсем немного для более-менее опытного игрока, но достаточно много для подобной машины[2]. В среднем на один ход компьютер затрачивает 3 минуты[2].

У пользователя существует возможность выбора игры как за белые, так и за чёрные фигуры. Также компьютер может играть и сам с собой. Кроме того, Chessmate способен проигрывать две небольшие мелодии с помощью пьезоэлектрического зуммера: одну в случае победы, другую — при поражении.[2]

## 24.1. Примечания


[1] Информация о компьютере с сайта ChessComputers.org (англ.)

[2] Commodore — Chessmate в коллекции сайта Schaakcomputers (нид.) (нем.)

[3] Информация о Fidelity Chess Challenger 10 на сайте Chess Computer UK (англ.)

[4] Microchess for the Kim-1 сайт Питера Дженнингса (англ.)

[5] Commodore info page (англ.)

[6] Commodore Chessmate на сайте Музея домашних компьютеров (нем.)

[7] Commodore Chessmate (1978) — Commodore … mehr als nur eine Leidenschaft (нем.)

[8] Уровни сложности игры в инструкции к Commodore Chessmate (нем.)

[9] Список доступных дебютов в инструкции к Commodore Chessmate (нем.)

[10] приблизительно соответствует пятому разряду игрока-человека

## 24.2. Ссылки

- Различная информация о программе Microchess на личном сайте Питера Дженнингса (англ.)
- Всё о Commodore Chessmate на сайте Schaakcomputers (нид.) (нем.)
- Rare Commodore Hardware (англ.)
- Все 8 страниц инструкции к Commodore Chessmate (нем.)

# Глава 25

# Computer Chess Rating Lists

**Computer Chess Rating Lists** (**CCRL**) — семейство рейтингов (рейтинг-листов) сильнейших компьютерных шахматных программ. В качестве методики применяется рейтинг Эло Bayesian [1], учитывающий цвет, процент набранных очков, включая ничьи, и различия в оценке шахматных программ при игре между собой.

На сегодняшний день является одним из самых авторитетных и известных методов оценки как коммерческих (проприетарных), так и бесплатных шахматных программ.

## 25.1. История создания и организация

Рейтинг-листы CCRL ведутся с 2006 года группой шахматных энтузиастов в следующем первоначальном составе: Грехам Бенкс (Graham Banks), Рей Бенкс (Ray Banks), Сара Бёрд (Sarah Bird), Кирилл Крюков (Kirill Kryukov) и Чарльс Смит (Charles Smith). Этих людей объединила вместе идея, что их хобби — тестирование шахматных программ будет более полезным и востребованным, если они объединят свои усилия и будут публиковать результаты на регулярной основе. В дальнейшем группа пополнялась за счёт добровольцев. На сегодняшний день, костяк группы состоит из двенадцати человек и объём выполняемой работы значительно перерос понятие хобби.

Предложить себя в качестве тестера может каждый, однако принимаются только люди хорошо известные в сообществе шахматных программ или имеют известных в этом кругу поручителей.

За время существования группа разработала целый свод формальных правил и допусков на основе которых сейчас и осуществляет тестирование. Такой подход более прозрачен и позволяет с большим доверием относиться к полученным результатам.

Авторы рейтинга щепетильно относятся к соблюдению прав разработчиков, например, они исключили из официальных листов достаточно сильную программу Strelka, посчитав её разновидностью программы Rybka.

## 25.2. Состав и методика

В настоящее время ведутся три основных независимых рейтинг-листа, различающихся по контролю времени и по некоторым другим параметрам:

- CCRL 40/40 — эквивалентно 40 ходов за 40 минут с повторением (CCRL 40/40)
- CCRL 40/4 — эквивалентно 40 ходов за 4 минуты с повторением (CCRL 40/4)
- CCRL 40/4FRC — эквивалентно 40 ходов за 4 минуты с повторением (CCRL 40/4FRC)

Во всех основных рейтинг-листах представлены все шахматные программы своими лучшими на момент оценки версиями. Оценка для основных рейтингов производится на архитектуре Athlon 64 X2 4600 + (2,4 ГГц).

При тестировании используются универсальные дебютные книги с глубиной оценки в 12 ходов вместо собственных прилагаемых к программам дебютных книг. При тестировании отключается возможность использования времени соперника на просчёт собственных ходов. Эти правила позволяют более объективно оценить качество работы собственно алгоритма программы при равных условиях.

Для повышения объективности проводится большое количество игр для каждой шахматной программы.

Кроме основных рейтингов ведётся множество других, например, для однопроцессорных систем, для 64-х и 32-битных систем, только для бесплатных программ и т. д. Таким образом не только специалисты, но и рядовой пользователь может произвести сравнение шахматных программ посмотрев результаты для своей конфигурации.

## 25.3. Лучшие шахматные программы

На 31 марта 2013 года в рейтинг-листе CCRL на первом месте находится шахматная программа Houdini 3 64-bit 4CPU, на втором Critter 1.6a 64-bit 4CPU, на третьем Stockfish 2.3.1 64-bit 4CPU [2].

## 25.4. См. также

- Компьютерные шахматы
- Chess Engines Grand Tournament
- Swedish Chess Computer Association

## 25.5. Примечания

[1] http://remi.coulom.free.fr/Bayesian-Elo/#elostat Рейтинг Эло Bayesian

[2] http://computerchess.org.uk/ccrl/4040/rating_list_pure.html Рейтинг компьютерных шахматных программ CCRL 40/40

## 25.6. Ссылки

- Домашняя страница CCRL
- Clash of the Computer Titans

# Глава 26

# Portable Game Notation

**Portable Game Notation (PGN)** — формат файла для сохранения шахматных партий. Он был разработан Стивеном Эдвардсом (англ. Steven J. Edwards) в 1994 году, чтобы облегчить обмен партиями (к примеру через интернет) между шахматными программами.

Формат PGN использует символы из ASCII-кодировки и состоит из двух частей: метаданные и нотация партии. В первой части стоит информация о партии: турнир, тур, имена игроков, результат и т. д. Вторая часть состоит из алгебраической нотации. Комментарии находятся между { } скобок.

Большинство шахматных программ поддерживает этот формат. Обработка файлов может совершаться и с помощью обычной программы для редактирования текста. В одном файле можно сохранять более, чем одну партию.

## 26.1. Seven Tag Roster

### 26.1.1. Event-Тег

[Event extquotedbl? extquotedbl] (Названия турнира, матча)

#### Примеры

- [Event “FIDE World Championship extquotedbl]
- [Event “Moscow City Championship extquotedbl]
- [Event “ACM North American Computer Championship extquotedbl]
- [Event “Casual Game extquotedbl]

### 26.1.2. Site-Тег

[Site extquotedbl? extquotedbl] (Место проведения)
Структура: «Город, Регион Страна»
Для обозначения страны используется список кодов МОК.

#### Примеры

- [Site “New York City, NY USA extquotedbl]
- [Site “St. Petersburg RUS extquotedbl]
- [Site “Riga LAT extquotedbl]

### 26.1.3. Date-Тег

[Date extquotedbl????.??.?? extquotedbl] (Дата)

#### Примеры

- [Date “1992.08.31 extquotedbl]
- [Date “1993.??.?? extquotedbl]
- [Date “2001.01.01 extquotedbl]

### 26.1.4. Round-Тег

[Round extquotedbl? extquotedbl] (Тур)

#### Примеры

- [Round “1 extquotedbl]
- [Round “3.1 extquotedbl]
- [Round “4.1.2 extquotedbl]

### 26.1.5. White-Тег

[White extquotedbl? extquotedbl] (Инициалы Игрока «Белые»)

#### Примеры

- [White “Tal, Mikhail N. extquotedbl]
- [White “van der Wiel, Johan extquotedbl]
- [White “Acme Pawngrabber v.3.2 extquotedbl]
- [White “Fine, R. extquotedbl]

### 26.1.6. Black-Тег

[Black extquotedbl? extquotedbl] (Инициалы Игрока «Чёрные»)

#### Примеры

- [Black “Lasker, Emmanuel extquotedbl]
- [Black “Smyslov, Vasily V. extquotedbl]
- [Black “Smith, John Q.:Woodpusher 2000 extquotedbl]
- [Black “Morphy extquotedbl]

### 26.1.7. Result-Тег

[Result extquotedbl* extquotedbl] (Результат)

#### Примеры

Все возможные результаты

- [Result “0-1 extquotedbl]
- [Result “1-0 extquotedbl]
- [Result “1/2-1/2 extquotedbl]
- [Result extquotedbl* extquotedbl]

## 26.2. Пример

[Event “IBM Kasparov vs. Deep Blue Rematch extquotedbl] [Site “New York, NY USA extquotedbl] [Date “1997.05.11 extquotedbl] [Round “6 extquotedbl] [White “Deep Blue extquotedbl] [Black “Kasparov, Garry extquotedbl] [Opening “Caro-Kann: 4...Nd7 extquotedbl] [ECO “B17 extquotedbl] [Result “1-0 extquotedbl] 1.e4 c6 2.d4 d5 3.Nc3 dxe4 4.Nxe4 Nd7 5.Ng5 Ngf6 6.Bd3 e6 7.N1f3 h6 8.Nxe6 Qe7 9.O-O fxe6 10.Bg6+ Kd8 {Каспаров встряхнул головой} 11.Bf4 b5 12.a4 Bb7 13.Re1 Nd5 14.Bg3 Kc8 15.axb5 cxb5 16.Qd3 Bc6 17.Bf5 exf5 18.Rxe7 Bxe7 19.c4 1-0

## 26.3. Ссылки

- Portable Game Notation Specification and Implementation Guide (англ.)

# Глава 27

# Swedish Chess Computer Association

**Swedish Chess Computer Association** (**SSDF**) — *Шведская шахматная компьютерная ассоциация* (швед. *Svenska schackdatorföreningen*) является организацией, тестирующей компьютерное программное обеспечение, играя шахматными программами против друг друга. Ведёт свой рейтинг-лист.

SSDF-лист является одной из немногих статистически значимых мер шахматной силы программ, особенно в сравнении с турнирами, поскольку включает в себя результаты тысяч игр, играемых на стандартных аппаратных средствах при обычном контроле времени. Список сообщает не только об абсолютной оценке, но также о процентах выигрышей и количестве учтенных игр. Рейтинг SSDF является независимым от всех национальных или международных рейтинговых систем, таких как ФИДЕ, поэтому компьютерные оценки могут только использоваться для относительного сравнения между собой.

Текущая (на март 2010 г.) платформа SSDF основана на Intel Core 2 Quad 6600 + 2.4 GHz и 64-битной операционной системой.

В июне 2006 года в листе появилась шахматная программа Rybka, впервые превысившая отметку в 2900 пунктов.

4 августа 2009 года в рейтинг-листе SSDF на 1 месте находилась программа Deep Rybka 3 x64 2GB Q6600 2,4 GHz с оценкой в 3224 пункта. На 2 месте находилась программа Naum 4 x64 2GB Q6600 2,4 GHz с оценкой 3134 пункта. На 3 месте находилась Zappa Mexico II x64 2GB Q6600 2,4 GHz с оценкой 3073 пункта.

## 27.1. Рейтинг-лист ежегодных победителей

- 1984: Novag Super Constellation 6502 4 MHz (1631)
- 1985: Mephisto Amsterdam 68000 12 MHz (1827)
- 1986: Mephisto Amsterdam 68000 12 MHz (1827)
- 1987: Mephisto Dallas 68020 14 MHz (1923)
- 1988: Mephisto MM 4 Turbo Kit 6502 16 MHz (1993)
- 1989: Mephisto Portorose 68020 12 MHz (2027)
- 1990: Mephisto Portorose 68030 36 MHz (2138)
- 1991: Mephisto Vancouver 68030 36 MHz (2127)
- 1992: Chess Machine 30 MHz Schröder 3.0 (2174)
- 1993: Mephisto Genius 2.0 486/50-66 MHz (2235)
- 1995: MChess Pro 5.0 Pentium 90 MHz (2306)
- 1996: REBEL 8.0 Pentium 90 MHz (2337)
- 1997: HIARCS 6.0 49MB P200 MMX (2418)
- 1998: Fritz 5.0 PB29% 67MB P200 MMX (2460)
- 1999: Chess Tiger 12.0 DOS 128MB K6-2 450 MHz (2594)
- 2000: Fritz 6.0 128MB K6-2 450 MHz (2607)
- 2001: Chess Tiger 14.0 CB 256MB Athlon 1200 (2709)
- 2002: Deep Fritz 7.0 256MB Athlon 1200 MHz (2759)
- 2003: Shredder 7.04 UCI 256MB Athlon 1200 MHz (2791)
- 2004: Shredder 8.0 CB 256MB Athlon 1200 MHz (2800)
- 2005: Shredder 9.0 UCI 256MB Athlon 1200 MHz (2808)
- 2006: Rybka 1.2 256MB Athlon 1200 MHz (2902)
- 2007: Rybka 2.3.1 Arena 256MB Athlon 1200 MHz (2935)
- 2008: Deep Rybka 3 2GB Q6600 2.4 GHz (3238)

## 27.2. См. также

- Компьютерные шахматы
- Chess Engines Grand Tournament
- Computer Chess Rating Lists

## 27.3. Ссылки

- Swedish Chess Computer Association
- PLY/SSDF – the story
- SSDF Computer Chess Rating Lists(недоступная ссылка — *история*, *копия*) (1996-2000)

# Глава 28

# UCI (протокол)

**UCI** (англ. *Universal Chess Interface*) — свободно распространяемый коммуникационный протокол, позволяющий движкам шахматных программ взаимодействовать с их графическим интерфейсом.

Был разработан и реализован Рудольфом Хубером (Rudolf Huber), автором программы SOS, и Стефаном Мейер-Каленом (Stefan Meyer-Kahlen), автором шахматной программы Shredder в ноябре 2000 года, и его можно рассматривать как серьёзного конкурента более старому и устоявшемуся коммуникационному протоколу Xboard/Winboard.

## 28.1. Ссылки

- Описание формата UCI

# Глава 29

# Международная ассоциация компьютерных игр

**Международная ассоциация компьютерных игр** (англ. *International Computer Games Association, ICGA*) была создана в 1977 году как **Международная ассоциация компьютерных шахмат** (англ. *ICCA*). Основана разработчиками в области компьютерных шахмат с целью организации будущих чемпионатов для компьютерных программ и облегчения обмена техническими знаниями через журнал ICCA.

Переименованная в 2002 году в ICGA, ассоциация стала широко способствовать развитию компьютерных игр и игр с искусственным интеллектом, проводя Компьютерные олимпиады, Чемпионаты мира по шахматам среди компьютерных программ и международные конференции по компьютерам и играм. ICGA ежеквартально публикует свой журнал, поддерживая отношения с информатикой, коммерцией и организацией игр во всем мире. Возглавляет ICGA Дэвид Леви.

## 29.1. Ссылки

- Официальный сайт ICGA

# Глава 30

# Продвинутые шахматы

**Адванс** (от англ. «advance»/«advanced chess» — «передовые»/«продвинутые шахматы») — разновидность шахмат, подразумевающая возможность использования во время игры какой-либо сторонней помощи (дебютные книги, шахматные программы, компьютерные движки, базы данных, эндшпильные таблицы и т.п.) и «легальность» её использования. Такие разновидности шахмат, как фристайл, заочные шахматы (если правилами предусмотрено использование помощи), а также игра шахматными движками — в той или иной степени относятся к адвансу.

## 30.1. Введение

В общем случае, кратко, адванс — это игра с расширенными возможностями.

## 30.2. История формирования адванса в шахматах, как отдельной дисциплины

Использование сторонних возможностей (подсказок) во время партии традиционно неприемлемо. Возможность использования подсказок в классические шахматы, если и допускается, то в основном изредка и при тренировочном или неофициальном характере игры. В связи с появлением компьютеров и появлением шахматных компьютерных программ, возникла возможность оценки их силы игры при игре против человека и при игре программ между собой. Шахматные программы (шахматные движки), приобретают самостоятельное обособленное значение, как отдельные шахматные игроки. Одновременно с этим появилась возможность игры людей-шахматистов между собой, при использовании компьютерной помощи. Такие новые виды и способы игры в шахматы начали получать распространение, получив название «advanced chess» (реже «cyborg chess» или «centaur chess»). (*прим.* Считается, что эти определения были введены в широкий обиход гроссмейстером Г.Каспаровым для обозначения игры людей между собой, при использовании компьютеров). Слово «адванс» в шахматах появилось для обозначения игры, при использовании компьютерных программ. Однако, если рассматривать современное понимание адванса, то шахматный адванс не подразумевает *обязательность* использования компьютерных программ в процессе игры, за исключением такой его разновидности, как «игра шахматными движками». Т.е. в шахматах адванс, как явление, фактически существовал, параллельно с существованием шахмат, но как самостоятельное явление традиционно не поощрялся, не воспринимался или не обозначался. А появление адванса, как самостоятельной, отдельно-обозначаемой дисциплины, начинается не с появлением адванса как такового, а, на самом деле, с появлением первого *компьютерного* шахматного адванса и является относительно новой шахматной разновидностью, а об адвансе, объективно существовавшем до появления первых компьютеров и шахматных программ, можно говорить, как о *докомпьютерном* шахматном адвансе.

## 30.3. Адванс и классические шахматы

Любое использование сторонней помощи по правилам FIDE не допустимо, что говорит об адвансе, как об особой разновидности шахматных правил, где использование сторонней помощи не ограничено.

## 30.4. Адванс и заочные шахматы

Если правилами заочного соревнования предусмотрено использование какой-либо сторонней помощи, такое соревнование является адвансом. Если условиями заочного соревнования использование какой-либо сторонней помощи не предусмотрено или запрещено, то такое соревнование адвансом не является.

## 30.5. Адванс (классический адванс)

В большинстве случаев соревнования по адвансу проводятся при больших временных контролях и методами заочных шахмат (чаще всего игрой на сервере через интернет). При этом, в подавляющем большинстве случаев, соревнования по заочным шахматам предусматривают использование какой-либо сторонней помощи, поэтому могут быть обозначены или обозначаются словом «адванс». В совокупности тождественность и сочетание этих особенностей и есть адванс в классическом его понимании. Классический адванс — это адванс, проводимый при больших временных контролях методами заочных шахмат. Адванс (классический адванс) — наиболее распространённый и популярный тип адванса.

## 30.6. Адванс и компьютерные шахматы

Использование сильнейших компьютерных программ на адванс-серверах подразумевается и фактически является неотъемлемым элементом при игре в современный адванс, однако обязательным элементом, например, классического адванса использование компьютерных программ не является.

## 30.7. Фристайл

Фристайл — тип адванса. С этой точки зрения «продвинутые шахматы» — это своеобразные симбиотические шахматы, объединяющие человеческую креативность и стратегическое мышление с машинным перебором вариантов и глубиной расчета. Игроки во время игры могут пользоваться компьютером и рассматривать варианты, предлагаемые шахматными программами. Как правило, данная версия шахмат используется для показательных выступлений сильнейших шахматистов мира (например, показательный матч из двух партий между В. Анандом и В. Крамником в ГУМе (Москва, ноябрь 2007 года) — зрители в итоге увидели две маловыразительные ничьи). Прелесть зрелища составляет демонстрация (на большом экране) тех вариантов, которые рассматривают в данный момент игроки. Вместе с тем «продвинутые шахматы» позволяют достичь значительно большей глубины игры, избежать просмотров и т.п.. Фристайлом также часто называется адванс при «стандартных» по меркам классических шахмат контролях, но малых по меркам классического адванса. Особенностью фристайла (в т.ч. при показательных выступлениях) является возможность введения ограниченного кол-ва возможных обращений к помощи компьютера во время партии, что обычно оговаривается в правилах заранее.

## 30.8. Игра шахматными движками

Является особым типом адванса, где участие человека в процессе игры не предусмотрено вовсе.

## 30.9. Расширенные шахматные турниры

В последние годы были проведены в глобальном масштабе турниры по адвансу для «кентавров» (где кентавр — это человек + компьютер) в роли кентавра обычно выступали разработчики компьютерных движков и их программа. Одним из основных таких турниров, можно отметить PAL / CSS Freestyle турнир, спонсором которого являлась группа PAL (Абу-Даби), на котором был очень высокий уровень игры, а победителями, в хронологическом порядке, были Закс (Стивен Cramton и Стивен Zackery, США), Zorchamp (Zorchamp, Объединенные Арабские Эмираты), Rajlich (Васик Райлих, Венгрия), Xakru (Иржи Dufek, Чехия), Flying Saucers (Даг Нильсен, Дания), Rajlich (Васик Райлих, Венгрия) Ibermax (Энсон Уильямс, Англия) и Ultima (Эрос Риччио, Италия).

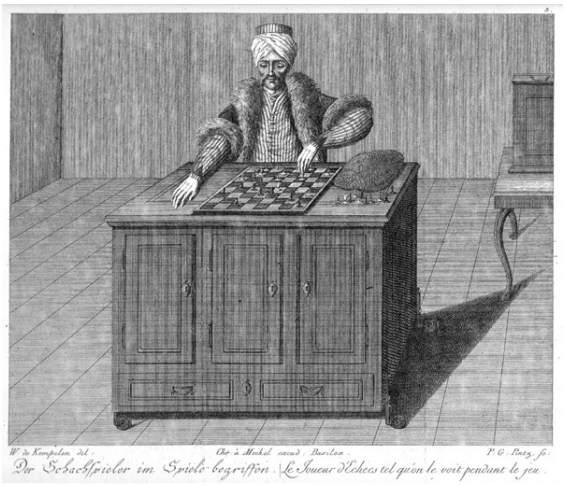
Похожие турниры были организованы FICGS (Chess Freestyle Cup), ChessBase (Computer Bild Spiele Scach Turnier) и Infinity Chess.

## 30.10. Ссылки

Адванс для начинающих. Краткое руководство

# Глава 31

# Шахматный автомат




*Шахматная машина «Турок» на рисунке Карла Готлиба фон Виндиш из книги «Briefe über den Schachspieler des Hrn. von Kempelen», 1784*

**Шахматный автомат** — общее название «иллюзионных устройств», в которых партию ведёт невидимый публике шахматист.

*Подписанный углём автопортрет В. Кемпелена, который сконструировал первый шахматный автомат*

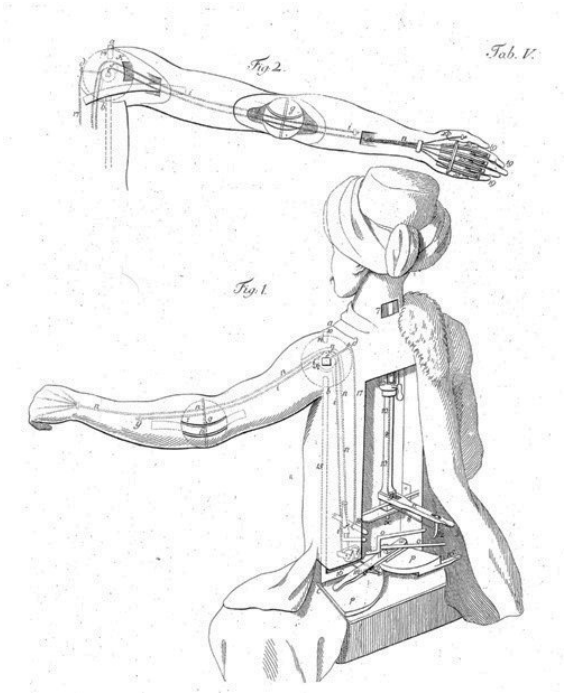
## 31.1. Автомат Кемпелена

Впервые сконструирован Вольфгангом фон Кемпеленом и продемонстрирован в Вене в 1769 году. Представлял собой выполненную в натуральную величину восковую фигуру, одетую в экзотический турецкий наряд, «турка», сидящего за деревянным ящиком (размером 1,2×0,6×0,9 м.) с шахматной доской на верхней крышке. Перед началом демонстрации дверцы ящика раскрывались и при помощи свечи публике показывался сложный механизм с различными узлами и деталями. Затем дверцы закрывались, механизм заводился ключом и начиналась игра, которую за автомат вёл спрятанный в ящике сильный шахматист. Система зеркал, расставленных под соответствующими углами, и маскирующими перегородками прекрасно скрывали сидящего внутри игрока. После каждых двадцати ходов конструктор лично заводил машину, давая, таким образом, спрятанному шахматисту некоторый резерв времени для анализа создавшейся на шахматной доске ситуации.

Автомат был способен решить задачу о ходе коня, демонстрируя публике обход конём всей шахматной доски так, что на каждую клетку приходился один ход.

### 31.1.1. Принцип работы

Плотно втиснутый в ящик автомата шахматист не мог непосредственно наблюдать за ходом игры. Эту проблему Кемпелен решил с помощью сигнализационной системы. В основание тяжёлых фигур, установленных на шахматной доске, были вмонтированы сильные магниты. Под доской, внутри ящика, под





*Объявление Мельцеля о появлении шахматного автомата в Лондоне*

каждым полем находился металлический шарик, надетый на вертикальную нитку. Когда фигуру поднимали, шарик опускался, сигнализируя о её перемещении. Как только фигура оказывалась на новом поле, магнит притягивал соответствующий шарик.

При своём ходе невидимый шахматист передвигал рычаг, заставляя руку «турка» опускаться над заданной точкой доски. В руке манекена находились гибкие тросики, которые управлялись движением пальцев шахматиста. Вращая втулку на конце рычага, шахматист брал фигуру, переносил её на нужное поле, разжимал пальцы и возвращал руку в исходное положение.

Шах королю «турок» объявлял троекратным кивком головы. Если противник совершал некорректный ход, к примеру, перемещал ферзя ходом коня, «турок» немедленно прекращал игру и оставался без движения.

## 31.2. Путешествие по Европе

В 1770—1773 годах Кемпелен демонстрировал автомат в Вене и Пожоне, в 1783—1784 — в Париже, Лондоне, Берлине, Лейпциге, Дрездене и других городах. Автомат играл успешно, выигрывая большинство партий. В Париже он проиграл несколько партий сильным шахматистам, в том числе Ф. Филидору.

## 31.3. Автомат у Мельцеля

После смерти Кемпелена (1804) автомат стал достоянием австрийского механика И. Мельцеля (1772—1838). В 1809 году в Шёрбрунне автомат имел честь играть против Наполеона. Мельцель демонстрировал «турка» во многих европейских странах, а 1826—1838 годах — в Америке. При Мельцеле за автомат играли сильные шахматисты того времени — И. Альгайер (именно он играл за автомат против Наполеона), У. Льюис, А. Александр и другие.

В 1850 году в Лондоне вышел сборник из 50 партий сыгранных автоматом, которые приведены французским шахматистом Ж. Муре, игравшим за автомат. Хотя Муре был доверенным лицом Мельцеля, он раскрыл секрет автомата, опубликовав о нём статью в парижском журнале «Магазин питтореск» (1834), а два года спустя автомат был полностью разоблачён в Америке писателем Эдгаром По, который опубликовал большую статью под заглавием «Игрок в шахматы Мельзеля». Интерес к автомату снизился, хотя Мельцель продолжал гастроли. После смерти Мельцеля автомат был продан с аукциона и попал в Китайский музей Филадельфии, где сгорел во время пожара (1854).

В течение почти 70 лет публичных выступлений «мозг» автомата заменяли поочерёдно несколько знаменитых шахматистов. О больших способностях и мастерстве этих игроков свидетельствует хотя бы то, что из трёхсот сыгранных ими партий они проиграли только шесть.

## 31.4. Другие автоматы

У изобретения Кемпелена появилось немало подражателей; при этом принцип «шахматиста-невидимки» оставался неизменным, менялся лишь внешний вид автомата и некоторые технические детали. Итальянец Мороси демонстрировал свой автомат в Париже (1798), однако успеха не имел: его автомат играл слабо и медленно. «Баварский мальчик» нюрнбергского часовщика А. Байера играл как в шахматы, так и в шашки; в 1820 году он выставлялся в Мюнхене. Американский «шахматный игрок», сконструированный музыкальным издателем У. Уокером (1827), был копией кемпеленовского «турка»; Мельцель, находившийся тогда в США, купил его, чтобы избежать конкуренции.

Автомат extquotedblАджиб extquotedbl был изготовлен бристольским[1] краснодеревщиком Ч. Хупером в виде восточной фигуры в экзотичном наряде (индуса в тюрбане), сидящей на 6-гранном ящике перед шахматной доской. Первоначальной игрой «Аджиба» руководил сам Хупер, затем его сын. В 1888 году за автомат играли американские шахматисты Ч. Мол и А. Ходжес. Наивысших успехов «Аджиб» достиг в 1891—1900 годах, когда его оператором был Г. Пильсбери, который не знал поражений. extquotedblАджиб extquotedbl сгорел в 1926 году.

В 1870 годах в Великобритании появился автомат под названием «Мефисто», сконструированный лондонским гомеопатом Ч. Гюмпелем. В 1878 году «Мефисто» выиграл турнир-гандикап, успешно провёл сеанс одновременной игры на 20 досках, в 1883 году выиграл партию у М. Чигорина, просмотревшего потерю качества. Хорошие результаты «Мефисто» не удивляют: его игрой руководил И. Гунсберг. Гюмпель не скрывал, что игра велась «шахматистом-невидимкой», который, по утверждению Гюмпеля, находился не внутри автомата, а в соседней комнате. Утверждение Гюмпеля проверить не удалось: после выступления в Париже (1889) «Мефисто» бесследно исчез.

Создание автоматов преследовало коммерческую цель, но они сыграли важную роль в популяризации шахмат в мире, явились прообразом современных шахматных компьютеров.

## 31.5. Современность

Появление современных ЭВМ, наконец, позволило воплотить идею шахматного автомата в реальность. Сегодняшний шахматный автомат представляет из себя робота, состоящего из шахматного компьютера и управляемой им руки-манипулятора, служащей для захвата и перемещения шахматных фигур. Как и у их предшественников, уровень игры автоматов очень высок, доказательством чего является недавний матч за звание чемпиона мира между немецким роботом Kuka Monstr и российским автоматом Chesska. Перед тем, как начать матч за звание чемпиона мира среди шахматных автоматов (проигранный Kuka со счетом 0,5 на 3,5), немецкий робот одержал убедительную победу в блице над международным гроссмейстером Александром Грищуком со счётом 4,5 на 1,5.[2]

## 31.6. См. также

- Компьютерные шахматы
- Шахматный компьютер
- Задача о ходе коня

## 31.7. Примечания

[1] *Hooper D., Whyld K.* Ajeeb // The Oxford companion to chess. — 2nd. ed.. — Oxford University Press, 1996. — P. 5. — 483 p. — (Oxford Companions Series). — ISBN 9780192800497.

[2] «Кука» обыграл Грищука, но уступил «Ческе» звание чемпиона мира среди роботов. // Chessdom (20 мая 2012). Проверено 7 июня 2012. Архивировано из первоисточника 27 июня 2012.

## 31.8. Литература


- Шахматный словарь / гл. ред. Л. Я. Абрамов; сост. Г. М. Гейлер. — М.: Физкультура и спорт, 1964. — С. 7. — 120 000 экз.
- Шахматы : энциклопедический словарь / гл. ред. А. Е. Карпов. — М.: Советская энциклопедия, 1990. — С. 8—9. — 100 000 экз. — ISBN 5-85270-005-3.
- *Гижицкий Е.* С шахматами через века и страны, 1970.
- Kempelen sakkautomataja, в книге: Magyar sakktörtenet, köt. 1, Bdpst, 1975.
- *Carroll C. M.* The great chess automation, N. Y., [1975].
- Der Schachautomat des Baron von Kempelen, mit einem Nachwort von Marion Faber, Dortmund, 1983.
- Эдгар Аллан По «Фон Кемпелен и его открытие» Von Kempelen and His Discovery (1849)

## 31.9. Ссылки

- Партии Шахматного автомата в базе *Chessgames*
- *Хенкин В.* «Одиссея шахматного автомата» (часть 1)
- *Хенкин В.* «Одиссея шахматного автомата» (часть 2)

# Глава 32

# Шахматный компьютер

**Шахматный компьютер** — специализированное устройство для игры в шахматы. Обычно используется для развлечения и практики игроков в шахматы при отсутствии партнёра-человека, в качестве средства для анализа различных игровых ситуаций, для соревнования компьютеров между собой. Представляют собой развитие идеи «шахматного автомата», возникшей в XVIII веке.

Потребительские шахматные компьютеры обычно выполняются в виде шахматной доски с дисплеем и набором клавиш для выполнения необходимых игровых действий. В зависимости от конструкции, доска может быть никак не связана с компьютерной частью и не иметь электроники, или наоборот, представлять собой дисплей, отображающий графическое представление игровой ситуации.

В СССР с середины 1980-х годов выпускались потребительские шахматные компьютеры Электроника ИМ−01, Электроника ИМ−05, Электроника ИМ−29 («Шахматный партнёр»), Интеллект−01 и Интеллект−02, Дебют, Феникс и другие.

Большинство программ основано на методе перебора ходов, существуют программы, основанные на эвристических методах. Попытка создать настоящую шахматную программу, на основе алгоритма шахматного мастера, предпринималась экс-чемпионом мира М. Ботвинником и его ассистентами-программистами — Б. Штильманом и А. Резницким.

## 32.1. См. также

- Компьютерные шахматы

## 32.2. Ссылки

Фотографии отечественных шахматных компьютеров

# Глава 33

# Эвристика нулевого хода

В компьютерных шахматах, **эвристика нулевого хода** — метод увеличения скорости алгоритма альфа-бета отсечения.

Альфа-бета отсечение ускоряет выполнение минимаксного алгоритма, распознавая точки отсечения вариантов, представляющихся бесперспективными. Это такая точка в игровом дереве, где текущая позиция настолько выгодна для стороны, которая сейчас ходит, что противоположная сторона будет избегать такую позицию. Поскольку такие позиции не могут быть результатом наилучшей игры, их и все ветви игрового дерева, которые идут от них, можно исключить из расчёта («отсечь»). Чем скорее программа делает отсечку, тем быстрее работает система поиска наилучшего хода.

Эвристика нулевого хода направлена на ускорение нахождения предполагаемых точек отсечения при сохранении разумного уровня аккуратности. Идея этой эвристики базируется на том предположении, что наиболее приемлемые ходы в шахматах улучшают позицию того, кто их сделал. Так, если игрок в данной точке может передать очередь хода противнику (сделать *нулевой ход*, что недопустимо в шахматах) и всё ещё имеет позицию, достаточно сильную для создания отсечения, тогда в данной точке почти наверняка возможно отсечение, поскольку данный игрок в действительности будет делать ход, и его позиция ещё более усилится.

Эвристика нулевого хода приводит к неверному результату в ситуациях цугцванга, когда игрок вынужден делать явно невыгодный ход при отсутствии вариантов улучшения своей позиции, поэтому игровые компьютерные программы вынуждены распознавать подобные ситуации и находить способы компенсации такого рода ошибок.

В частности extquotedblверифицированной эвристикой нулевого хода extquotedbl называется компьютерная стратегия не полного отсечения таких вариантов, а продолжение поиска, однако с сокращённой глубиной [1] .

## 33.1. Ссылки


- *Frayn C.* Null Move Heuristic // Computer Chess Programming Theory
- *David-Habibi O., Netanyahu N. S.* Verified null-move pruning (недоступная ссылка с 13-05-2013 (522 дня) — *история*) // ICGA Journal, September 2002


## 33.2. Примечания


[1] Omid David Tabibi and Nathan S. Netanyahu (2002), Verified Null-Move Pruning

# Глава 34

# Электроника ИМ−05

**«Электроника ИМ−05» —** советский шахматный компьютер. Создан на базе процессора КМ1801ВМ2. Является продолжением линейки шахматных компьютеров: «Электроника ИМ−01» и «Электроника ИМ−01Т». Выпускался объединением электронного приборостроения «Светлана», г. Ленинград.

Имел несколько уровней игры. Режим расстановки позиции и анализ позиции. Для отображения ходов имел люминесцентный индикатор зеленого цвета.

Доска была обычной, с магнитными фигурами. Никакой связи с электроникой доска не имела. Сила игры этого компьютера находилась примерно на уровне второго человеческого разряда.

## 34.1. Технические характеристики

- Процессор: КМ1801ВМ2
- Память: 4 КБ ОЗУ, 24 КБ ПЗУ

## 34.2. См. также

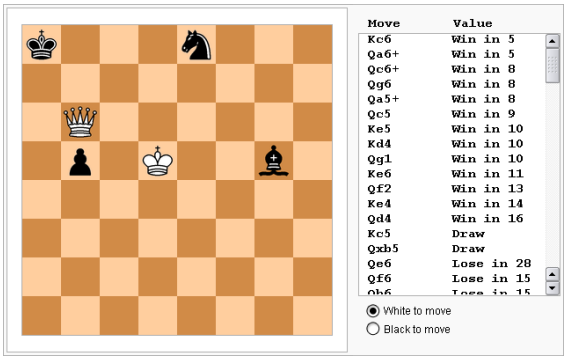
- Компьютерные шахматы

## 34.3. Ссылки

- Электроника ИМ−05 — фотографии
- Сергей Фролов. Шахматные компьютеры (рус.)
- Электроника ИМ−05 — фотографии

# Глава 35

# Эндшпильные таблицы Налимова




*Типичный интерфейс для использования баз данных эндшпиля. Для каждого хода белых таблицы показывают число ходов к выигрышу. В результате ходов Крсб или Фаб + белые выигрывают в 5 ходов, следовательно, это оптимальные ходы.*

**Эндшпильные таблицы Налимова** — базы данных шахматных окончаний. Эндшпильные таблицы Налимова названы именем новосибирского программиста Евгения Налимова, который предложил эффективный алгоритм для абсолютно точного расчёта шахматных окончаний. Созданные Налимовым удачные алгоритмы используются для генерации эндшпильных баз данных.

В настоящее время все ведущие компьютерные программы для игры в шахматы имеют опцию для подключения таблиц Налимова.

В таблицах Налимова имеются абсолютно точные варианты развития шахматной партии в эндшпиле. С помощью таблиц Налимова определяются все возможные варианты продолжения игры, все возможные результаты и число ходов, через которое при идеальной игре партия придёт к тому или иному результату.

## 35.1. Некоторые интересные позиции

### 35.1.1. Длиннейшие выигрыши

3-фигурные окончания. Мат в 28 ходов, FEN: 8/8/8/k7/8/K7/6P1/8 b.

4-фигурные окончания. Мат в 43 хода, FEN: 8/5k2/2PK4/5r2/8/8/8/8 w.

5-фигурные окончания. Мат в 127 ходов, FEN: 8/8/8/8/1p2P3/4P3/1k6/3K4 w.

6-фигурные окончания. Мат в 262 хода, FEN: 6k1/5n2/8/8/8/5n2/1RK5/1N6 w.

7-фигурные окончания. Мат в 549 ходов, FEN: 1n1k4/6Q1/5KP1/8/7b/1r6/8/8 w.

Мат в 549 ходов. Позиция и решение.

## 35.2. Расчёт

Время расчёта и объём таблиц Налимова экспоненциально возрастает с количеством участвующих фигур.

Для расчёта всех пятифигурных таблиц на компьютере с процессором «Атлон» 1,2 ГГц требуется 5 суток, для расчета шестифигурных таблиц на нём же потребовалось бы уже 860 дней, а всех семифигурных — около семи столетий. Таким образом, время и производительность компьютеров являются преградой для расчёта «эндшпильных» баз всех 32-х фигур. Надеждой программистов остаётся закон Мура и продолжение его долголетия.

К настоящему времени имеются базы данных, рассчитанные по таблицам Налимова, для всех трёх-, четырёх-, пяти-, шести- и семифигурных окончаний (включая двух королей). Расчет восьмифигурных окончаний пока ещё не планируется;

*7-фигурные таблицы для ситуаций распределения фигур четыре против трех и пять против двух уже рассчитаны. Таблицы названы таблицами Ломоносова и рассчитывались на суперкомпьютерах «Ломоносов» и IBM Blue Gene/P, установленных в Московском государственном университете имени М. В. Ломоносова. Расчеты проводились весной-летом 2012 года. Авторы таблиц — Владимир Махнычев и Виктор Захаров, сотрудники ВМК МГУ. Размер всех 7-фигурных таблиц — 140 ТБ. Публичный доступ к таблицам по состоянию на май 2013 года отсутствует. (Пока доступ есть только для пользователей Aquarium)* При этом таблицы Ломоносова уже ак-

тивно использовались при анализе партий Чемпионата Мира по шахматам[1][2].

## 35.3. Размер

- Все 3-фигурные окончания занимают 62,4 КБ.
- Все 4-фигурные окончания занимают 29,5 МБ.
- Все 5-фигурные окончания занимают 7,03 ГБ.
- Все 6-фигурные окончания занимают 1,205 ТБ.
- Все 7-фигурные окончания занимают 140 ТБ.
- Все 8-фигурные окончания будут занимать приблизительно 10 ПБ.

## 35.4. Исторические предшественники

Налимов был далеко не первым, кто высказал и реализовал идею игры компьютера в малофигурном окончании путём использования предварительно рассчитанной исчерпывающей таблицы возможных ходов. Ещё в 1977 году Кен Томпсон представил на конференции Международной федерации по обработке информации (en:International Federation for Information Processing) в Торонто похожую систему: путём перебора с возвратом была построена таблица всех возможных положений в эндшпиле «ладья и король против ферзя и короля». Общее число позиций для него составляет около 4 миллионов. Компьютер играл за игрока, владеющего ладьёй. Этот эндшпиль теоретически проигрышный, шахматист уровня мастера, владея ферзём, обычно легко выигрывает его у любого противника. Поэтому компьютеру была поставлена задача максимально оттянуть свой теоретически неизбежный проигрыш.

Результаты экспериментов, в которых компьютер играл с шахматистами, были довольно интересными. Против программы пытались играть Ханс Берлинер, экс-чемпион мира по переписке, и Лоренс Дей, чемпион Канады. Ни тот, ни другой не смогли выиграть у программы, хотя любая позиция была для них выигрышной. Дело в том, что теоретически безупречная игра компьютера часто выглядела нелогично, противоречила принципам, предписываемым шахматной теорией (например, обычно рекомендуется не уводить ладью далеко от короля, но программа нередко делала это), необычные ходы компьютера сбивали шахматиста с толку, и он упускал выигрыш.

В 1970-е годы идея предварительно рассчитанных эндшпилей не получила дальнейшего развития, так как быстродействие и объём памяти тогдашних компьютеров не позволяли получить подробные таблицы, ставшие доступными в настоящее время.

## 35.5. См. также

- Компьютерные шахматы

## 35.6. Ссылки

- Endgame Database shredderchess.com
- On-line анализ вплоть до 6-и фигурного эндшпиля.
- Lomonosov Endgame Tablebases (про 7-фигурные).
- A guide to Endgames Tablebase
- Эндшпильные таблицы Налимова.
- *Мичи Д., Джонстон Р.* Компьютер-творец / Пер. с англ.— М. Мир, 1987.— стр.68, «Странный случай с таблицей Томпсона».
- Lomonosov Endgame Tablebases (англ.) // ChessOK.com

## 35.7. Примечания

[1] Примеры анализа сложных ситуаций в семифигурных таблицах в блоге разработчиков 7ТВ, (ВМК МГУ)

[2] Официальная публикация решающей партии матча на звание Чемпиона Мира по шахматам 2012 года Ананд — Гельфанд с комментариями ключевых моментов партии на основе таблиц Ломоносова

# 35.8. Источники текстов и изображения, авторы и лицензии

## 35.8.1. Текст

- **Алгоритм Apriori** *Источник:* http://ru.wikipedia.org/wiki/Алгоритм_Apriori?oldid=59658388 *Авторы:* Neon, Valodzka, Poa, Toyota prius 2, Typhoonbreath, Thijs!bot, Ingwar JR, VolkovBot, Aleksandrit, Fleur de lis, Голем, РобоСтася, MystBot, LucienBOT, EmausBot, ZéroBot, W2, Dexbot, Robiteria, Addbot и Аноним: 7
- **Алгоритм Бога** *Источник:* http://ru.wikipedia.org/wiki/Алгоритм_Бога?oldid=61864759 *Авторы:* Nashev, WebCite Archiver, Д.Ильин, Bloodyritual, Stannic и Аноним: 3
- **Алгоритм выбора** *Источник:* http://ru.wikipedia.org/wiki/Алгоритм_выбора?oldid=64128461 *Авторы:* A5b, VolkovBot, FearChild, Peni, РобоСтася, Mihaild, LightOfDark, Roman Munich, Xqbot, Syarik, Addbot и Аноним: 3
- **Альфа-бета отсечение** *Источник:* http://ru.wikipedia.org/wiki/Альфа-бета_отсечение?oldid=62176485 *Авторы:* Maxal, JAnDbot, TXiKiBoT, Loveless, Q Valda, SergeyJ, MelancholieBot, Rubinbot, Xqbot, Foux, Alpunin, SEA99, Фидель22, WebCite Archiver, MerlIwBot, MBHbot, Висарик, Addbot, Stannic и Аноним: 2
- **Двоичный поиск** *Источник:* http://ru.wikipedia.org/wiki/Двоичный_поиск?oldid=65211296 *Авторы:* Maximaximax, Maxal, A.I., A5b, Albedo, Mercury, Valodzka, AndyVolykhov, Edwardspec TalkBot, AntonR, ZsergheiBot, JAnDbot, Soulbot, Dysangelium, CommonsDelinker, Leper Messiah, Freeaccount, VolkovBot, A.Savin, Newt, SieBot, YonaBot, Urutseg, Tim2, Sergey539, SolitaryDreamer, Gökhan, SilvonenBot, VlsergeyBot, Alkven, MDA, AVB, Luckas-bot, Abeshenkov, Gvsmirnov, Wizardist, Hairovich, Ghuron, X7q, WindBot, Андрей Куликов, Synopsis2009, Deadly pro, EmausBot, AVANG, OneLittleMouse, H2Bot, Abc41, MerlIwBot, MBHbot, KPu3uC B Poccuu, Кантакузин, Liveflowcharts, 762bot, Ekruten, Hint000, Cardinalofworlds, Bulatov и Аноним: 47
- **Двунаправленный поиск** *Источник:* http://ru.wikipedia.org/wiki/Двунаправленный_поиск?oldid=60445748 *Авторы:* ButkoBot, Melancholic, Четыре тильды, Alexbot, Musicien, РобоСтася, DaKick, Rubinbot, JackieBot, ESSch, Робот Филипп Вечеровский, Dinamik-bot, EmausBot, H2Bot, WikitanvirBot, Stannic и Аноним: 9
- **Дихотомия** *Источник:* http://ru.wikipedia.org/wiki/Дихотомия?oldid=64744584 *Авторы:* Andre Engels, Maxal, Maqs, Obersachse, Al Silonov, A5b, Maksim-bot, Infovarius, Altes, Ivan Vlasov, Пиотровский Юрий, AntonR, Thijs!bot, JAnDbot, Blacklake, Cypok, Oppo-rancisis, Shureg, VolkovBot, TXiKiBoT, Chath, Urutseg, Andromedus, Cobrafist, Chan, EvgenyGenkinBot, SolitaryDreamer, MBP, VlsergeyBot, РобоСтася, Abiyoyo, Luckas-bot, Юлия 70, Katets, Xqbot, Шуфель, LucienBOT, Ghuron, X7q, EmausBot, ZéroBot, Шадрин Денис, LarBot, WikitanvirBot, MerlIwBot, Humanitarian&, 762bot, Connexx, Bulatov и Аноним: 20
- **Задача поиска ближайшего соседа** *Источник:* http://ru.wikipedia.org/wiki/Задача_поиска_ближайшего_соседа?oldid=65763891 *Авторы:* Александр Сигачёв, Grey horse, VolkovBot, TXiKiBoT, Overrider, Lvova, Loveless, Голем, LaaknorBot, Krassotkin, Dinamik-bot, Ripchip Bot, Flammable, Movses-bot, Addbot и Аноним: 4
- **Интерполирующий поиск** *Источник:* http://ru.wikipedia.org/wiki/Интерполирующий_поиск?oldid=64956079 *Авторы:* AntonR, Ors archangel, Голем, Четыре тильды, EvgenyGenkinBot, BOTarate, Luckas-bot, Dinamik-bot, Addbot и Аноним: 13
- **Ключ для определения** *Источник:* http://ru.wikipedia.org/wiki/Ключ_для_определения?oldid=54634881 *Авторы:* Mercury, Cantor, Lige, РобоСтася, Glagolev, Bopsulai, ArtTrapeza, KrBot, WebCite Archiver, KLBot2, Sealle, Demidenko и Vcohen
- **Линейный поиск** *Источник:* http://ru.wikipedia.org/wiki/Линейный_поиск?oldid=53383048 *Авторы:* Nzeemin, Escarbot, AntonR, ZsergheiBot, Thijs!bot, VolkovBot, Aibot, Ors archangel, SieBot, Голем, EvgenyGenkinBot, VlsergeyBot, Kaenaru hito, Maaaks, Luckas-bot, Хитрый гнУс, Dinamik-bot, EmausBot, ChuispastonBot, Ekruten, Addbot и Аноним: 5
- **Математика кубика Рубика** *Источник:* http://ru.wikipedia.org/wiki/Математика_кубика_Рубика?oldid=64912477 *Авторы:* Jackie, Colt browning, Кубаноид, KrBot, Д.Ильин, Stannic и Аноним: 2
- **Метод золотого сечения** *Источник:* http://ru.wikipedia.org/wiki/Метод_золотого_сечения?oldid=62923716 *Авторы:* Torin, Obersachse, Roxis, Alex Smotrov, TXiKiBoT, Балахонов, SolitaryDreamer, Сусанин, VlsergeyBot, РобоСтася, Pessimist2006, Dimka chydo, X7q, Андрей Куликов, Khas, Kirya-90, EmausBot, ZéroBot, Tehnick, Nshulipa, МетаСкептик12 и Аноним: 37
- **Метод перебора** *Источник:* http://ru.wikipedia.org/wiki/Метод_перебора?oldid=56113781 *Авторы:* Maximaximax, ManN, Rasim, ZsergheiBot, Lawgiver, Vlad2000PlusBot, Newt, Голем, SolitaryDreamer, VlsergeyBot, РобоСтася, Rubinbot, Mikhael 666 mikhailovich, X7q, Tretyak и Аноним: 3
- **Поиск в глубину** *Источник:* http://ru.wikipedia.org/wiki/Поиск_в_глубину?oldid=65970561 *Авторы:* Piv, Chobot, Cheops, A5b, Mercury, Oyster, Александр Крайнов, AntonR, JAnDbot, Abatishchev, ButkoBot, РоманСузи, Порфирий, Дмитрий Джус, SieBot, Gribozavr, EvgenyGenkinBot, Centurn l, VanBot, SilvonenBot, Fractaler, Luckas-bot, Voitalex, Roman.Al.Petrov, Xqbot, DumZiBoT, X7q, ESSch, Робот Филипп Вечеровский, RedBot, BAZIL2, Green Giant, EmausBot, ZéroBot, ChuispastonBot, KrBot, HiW-Bot, LioN vlad, BeLove, Addbot, Stannic, Schulllz, Igor Grushevskiy и Аноним: 40
- **Поиск в пространстве состояний** *Источник:* http://ru.wikipedia.org/wiki/Поиск_в_пространстве_состояний?oldid=57344408 *Авторы:* Stannic
- **Поиск в ширину** *Источник:* http://ru.wikipedia.org/wiki/Поиск_в_ширину?oldid=66235534 *Авторы:* Rokur, Chobot, Cheops, A5b, Oyster, Александр Крайнов, MeaWr77, AntonR, JAnDbot, Candid, Vvvolf, -cid, Toto, VolkovBot, РоманСузи, Дмитрий Джус, Loveless, Knkd, Naagi, Sergey539, EvgenyGenkinBot, VanBot, VlsergeyBot, Qkowlew, Zorrobot, DaKick, Luckas-bot, Rubinbot, Rocker mipt, Xqbot, DumZiBoT, X7q, ESSch, MondalorBot, Робот Филипп Вечеровский, Dinamik-bot, BAZIL2, EmausBot, X-3me, Serbeh, LarBot, HiW-Bot, YFdyh-bot, Aced, Addbot, Stannic, АнШам и Аноним: 28
- **Пчелиный алгоритм** *Источник:* http://ru.wikipedia.org/wiki/Пчелиный_алгоритм?oldid=63182480 *Авторы:* Daemon2010, Anaxibia, KrBot и Tryerty
- **Радужная таблица** *Источник:* http://ru.wikipedia.org/wiki/Радужная_таблица?oldid=62922464 *Авторы:* Dodonov, Rafailka, A5b, Javalenok, Redline, Thijs!bot, Sseeaann, AVRS, VolkovBot, Toyota Dream House, Aydarov, Серж Тихомиров, Amirobot, Luckas-bot, Gromolyak, Polymorphm, Xqbot, SassoBot, MastiBot, Courbd, KamikazeBot, EmausBot, Damira Reicherd, MerlIwBot, Ghainmem, Sealle, Well-Informed Optimist, YFdyh-bot, Addbot, EvRubot и Аноним: 44

- **Троичный поиск** *Источник:* http://ru.wikipedia.org/wiki/Троичный_поиск?oldid=59069188 *Авторы:* ButkoBot, Yury Chekhovich, Голем, WikiStrider, SolitaryDreamer, Luckas-bot, Rubinbot, Mikhael 666 mikhailovich, X7q, Dalka, WikitanvirBot, Addbot и Аноним: 8
- **Компьютерные шахматы** *Источник:* http://ru.wikipedia.org/wiki/Компьютерные_шахматы?oldid=65095361 *Авторы:* Dm, Vald, Shamin Roman, Александр Цамутали, Q Valda, Uragan. TT, Veinarde, Berillium, DenisKrivosheev, The-city-not-present, Evatutin, EmausBot, Alpunin, Станислав Митичкин, Oleksiy.golubov, Darvesk, Flaflir, Kavalira, KrBot, Alexkachanov, MerlIwBot, W2Bot, MBHbot, Fifis, Rubilnik, Tatiana Matlina, Mustafaalmas, Здравствуйте!, SvetoslavHorobre, RotlinkBot, NLab, Addbot и Аноним: 24
- **Chess Engines Grand Tournament** *Источник:* http://ru.wikipedia.org/wiki/Chess_Engines_Grand_Tournament?oldid=58072519 *Авторы:* Softy, VolkovBot, Wevict, NBS, LaaknorBot, Qweedsa, Alpunin, WebCite Archiver, Mental Irbis, Addbot и Аноним: 2
- **ChessVegas** *Источник:* http://ru.wikipedia.org/wiki/ChessVegas?oldid=55596049 *Авторы:* INS Pirat, РобоСтася, U-bot, Helipilot, Shogiru и Аноним: 4
- **Commodore Chessmate** *Источник:* http://ru.wikipedia.org/wiki/Commodore_Chessmate?oldid=55424175 *Авторы:* Акутагава, Mozenrath, Letzte*Spieler, Felitsata, EmausBot и W2Bot
- **Computer Chess Rating Lists** *Источник:* http://ru.wikipedia.org/wiki/Computer_Chess_Rating_Lists?oldid=56127100 *Авторы:* Лика, VolkovBot, Wevict, NBS, Rubinbot, Ботильда, Krassotkin, Alpunin и Аноним: 7
- **Portable Game Notation** *Источник:* http://ru.wikipedia.org/wiki/Portable_Game_Notation?oldid=53431141 *Авторы:* Nerevar, Cheops, Thijs!bot, VolkovBot, TXiKiBoT, Volcanus, SieBot, NBS, PixelBot, Alexbot, Luckas-bot, EmausBot, Vlad Akila, ZéroBot, Addbot и Аноним: 3
- **Swedish Chess Computer Association** *Источник:* http://ru.wikipedia.org/wiki/Swedish_Chess_Computer_Association?oldid=64079563 *Авторы:* Wevict, Q Valda, AVB, D'ohBot, Alpunin, Kavalira, RotlinkBot, Addbot и Аноним: 1
- **UCI (протокол)** *Источник:* http://ru.wikipedia.org/wiki/UCI_(протокол)?oldid=64264974 *Авторы:* OckhamTheFox, Yaleks, VolkovBot, Vlad2000PlusBot, Wevict, VVVBot-temp, Peni, Голем, NBS, Cave, Ufim, LatitudeBot, Victor-435, РобоСтася, Luckas-bot, Roman Lagunov, AbiyoyoBot, Movses-bot, Addbot и Аноним: 1
- **Международная ассоциация компьютерных игр** *Источник:* http://ru.wikipedia.org/wiki/Международная_ассоциация_компьютерных_игр?oldid=53643794 *Авторы:* Wevict, Q Valda, MystBot, Rubinbot, Ботильда, EmausBot, ZéroBot и Addbot
- **Продвинутые шахматы** *Источник:* http://ru.wikipedia.org/wiki/Продвинутые_шахматы?oldid=63862661 *Авторы:* A5b, Nickispeaki, Volcanus, Artemka373, U-bot, Юрій25031994, KrBot, MBHbot, Shogiru, Addbot, Cinmad, Аширu, Старший Мастер и Аноним: 11
- **Шахматный автомат** *Источник:* http://ru.wikipedia.org/wiki/Шахматный_автомат?oldid=66131993 *Авторы:* Komap, Softy, Nzeemin, Mashiah Davidson, Переход Артур, Volcanus, Vlsergey, Q Valda, Stypex, VlsergeyBot, Fractaler, LaaknorBot, Amirobot, AVB, Luckas-bot, Rubinbot, Yonidebot, Xqbot, Markovka, TobeBot, EmausBot, Alpunin, ZéroBot, WebCite Archiver, Chessrobot, Kwame, Makecat-bot, Shogiru, Addbot, Citing Bot и Аноним: 17
- **Шахматный компьютер** *Источник:* http://ru.wikipedia.org/wiki/Шахматный_компьютер?oldid=65921343 *Авторы:* Komap, Sergei Frolov, APTEM, Volcanus, Chath, Анатолич1, Wevict, Analyzer (KODEP), Zaqq, U-bot, Structor, AlexVinS, Alpunin, LarBot, KrBot, Gipoza и Аноним: 3
- **Эвристика нулевого хода** *Источник:* http://ru.wikipedia.org/wiki/Эвристика_нулевого_хода?oldid=55424192 *Авторы:* Temaotheos, Neon, INS Pirat, Q Valda, Luckas-bot, Lazyhawk, EmausBot, Alpunin, W2Bot и Аноним: 1
- **Электроника ИМ−05** *Источник:* http://ru.wikipedia.org/wiki/Электроника_ИМ$-$05?oldid=48604291 *Авторы:* Sergei Frolov, Nzeemin, Grey Sever, Lazyhawk, U-bot, WindBot, Letzte*Spieler, Alpunin и Аноним: 4
- **Эндшпильные таблицы Налимова** *Источник:* http://ru.wikipedia.org/wiki/Эндшпильные_таблицы_Налимова?oldid=63104178 *Авторы:* Ygrek, Dm, Halyavin, Softy, Neon, Mercury, Pauk, Frei, Dmitry Fomin, Escarbot, TXiKiBoT, Volcanus, Wevict, Голем, NBS, Q Valda, PipepBot, DarkCherry, Rubinbot, Obersachsebot, Clockwork, Xqbot, Van Der Lokken, Evatutin, WindBot, ALEF7, Alpunin, Zamzavkaftyaf, ZéroBot, SEA99, Vladmax, I-go-spear, Fifis, Bloodyritual, Addbot и Аноним: 26

## 35.8.2. Изображения

- **Файл:AB_pruning.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/9/91/AB_pruning.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Transferred from en.wikipedia *Художник:* Original uploader was Jez9999 at en.wikipedia
- **Файл:Animated_BFS.gif** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/46/Animated_BFS.gif *Лицензия:* CC-BY-SA-3.0 *Авторы:* Originally from en.wikipedia; description page is/was here. *Художник:* Blake Matheny. Original uploader was Bmatheny at en.wikipedia
- **Файл:Breadth-First-Search-Algorithm.gif** *Источник:* http://upload.wikimedia.org/wikipedia/commons/5/5d/Breadth-First-Search-Algorithm.gif *Лицензия:* CC-BY-SA-3.0-2.5-2.0-1.0 *Авторы:* собственная работа *Художник:* Mre
- **Файл:Broom_icon.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/2/2c/Broom_icon.svg *Лицензия:* GPL *Авторы:* http://www.kde-look.org/content/show.php?content=29699 *Художник:* gg3po (Tony Tony), SVG version by User:Booyabazooka
- **Файл:C64c_system.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/8/84/C64c_system.jpg *Лицензия:* CC-BY-SA-2.5 *Авторы:* собственная работа *Художник:* Bill Bertram
- **Файл:Chess.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/0/05/Chess.svg *Лицензия:* LGPL *Авторы:* http://ftp.gnome.org/pub/GNOME/sources/gnome-themes-extras/0.9/gnome-themes-extras-0.9.0.tar.gz *Художник:* David Vignoni
- **Файл:Chess_tablebase_query.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/a/af/Chess_tablebase_query.png *Лицензия:* CC-BY-SA-3.0 *Авторы:* собственная работа *Художник:* User:ZeroOne
- **Файл:Chess_zver_26.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/46/Chess_zver_26.svg *Лицензия:* Public domain *Авторы:* собственная работа *Художник:* Ysangkok

- **Файл:Commons-logo.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/4a/Commons-logo.svg *Лицензия:* Public domain *Авторы:* This version created by Pumbaa, using a proper partial circle and SVG geometry features. (Former versions used to be slightly warped.) *Художник:* SVG version was created by User:Grunt and cleaned up by 3247, based on the earlier PNG version, created by Reidab.
- **Файл:Computer.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/d/d7/Computer.svg *Лицензия:* Public domain *Авторы:* The Tango! Desktop Project *Художник:* The people from the Tango! project
- **Файл:Crystal_Clear_mimetype_binary.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/3/39/Crystal_Clear_mimetype_binary.png *Лицензия:* LGPL *Авторы:* All Crystal icons were posted by the author as LGPL on kde-look *Художник:* Everaldo Coelho and YellowIcon
- **Файл:Depth-first-tree.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/1/1f/Depth-first-tree.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* собственная работа *Художник:* Alexander Drichel
- **Файл:DepthFirstSearchTimestampsRu.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/5/5f/DepthFirstSearchTimestampsRu.svg *Лицензия:* CC0 *Авторы:* собственная работа *Художник:* Mercury13
- **Файл:E-to-the-i-pi.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/3/35/E-to-the-i-pi.svg *Лицензия:* CC-BY-2.5 *Авторы:* ? *Художник:* ?
- **Файл:Golden_ratio.PNG** *Источник:* http://upload.wikimedia.org/wikipedia/ru/2/2a/Golden_ratio.PNG *Лицензия:* Общественное достояние *Авторы:* собственная работа *Художник:* solitary dreamer
- **Файл:Image-silk.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/6/65/Image-silk.png *Лицензия:* CC-BY-2.5 *Авторы:* ? *Художник:* ?
- **Файл:Images.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/c/c0/Images.png *Лицензия:* CC-BY-2.5 *Авторы:* ? *Художник:* ?
- **Файл:LampFlowchart.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/9/91/LampFlowchart.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* vector version of Image:LampFlowchart.png *Художник:* svg by Booyabazooka
- **Файл:Malzels-exhibition-ad.JPG** *Источник:* http://upload.wikimedia.org/wikipedia/commons/1/18/Malzels-exhibition-ad.JPG *Лицензия:* Public domain *Авторы:* ? *Художник:* ?
- **Файл:Megaminx12.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/2/2b/Megaminx12.jpg *Лицензия:* Public domain *Авторы:* wikipedia english *Художник:* User Tetracube on en.wikipedia
- **Файл:Nuvola_apps_kcmprocessor.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/5/50/Nuvola_apps_kcmprocessor.png *Лицензия:* LGPL *Авторы:* http://icon-king.com *Художник:* David Vignoni / ICON KING
- **Файл:Poda_alfa-beta.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/6/64/Poda_alfa-beta.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* ? *Художник:* ?
- **Файл:RS_Chess_Computer.JPG** *Источник:* http://upload.wikimedia.org/wikipedia/commons/7/7b/RS_Chess_Computer.JPG *Лицензия:* ? *Авторы:* English Wikipedia *Художник:* Model Citizen
- **Файл:Rainbow_table1.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/7/78/Rainbow_table1.svg *Лицензия:* CC-BY-SA-2.5 *Авторы:* Dake *Художник:* Dake
- **Файл:Rubik's_cube_scrambled.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/a/ae/Rubik%27s_cube_scrambled.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* ? *Художник:* ?
- **Файл:Rubik-3-facelet-kociemba.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/0/04/Rubik-3-facelet-kociemba.png *Лицензия:* CC0 *Авторы:* собственная работа *Художник:* Stannic
- **Файл:Rubiks_cube_solved.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/6/61/Rubiks_cube_solved.jpg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Work by Mike Gonzalez (TheCoffee) *Художник:* Mike Gonzalez (TheCoffee)
- **Файл:Rubiks_revenge_scrambled.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/6/67/Rubiks_revenge_scrambled.jpg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Work by Mike Gonzalez (TheCoffee) *Художник:* Mike Gonzalez (TheCoffee)
- **Файл:Searchtool.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/6/61/Searchtool.svg *Лицензия:* LGPL *Авторы:* http://ftp.gnome.org/pub/GNOME/sources/gnome-themes-extras/0.9/gnome-themes-extras-0.9.0.tar.gz *Художник:* David Vignoni, Ysangkok
- **Файл:Silk-film.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/1/1a/Silk-film.png *Лицензия:* CC-BY-3.0 *Авторы:* http://www.famfamfam.com/lab/icons/silk/ *Художник:* Mark James
- **Файл:Super_Nintendo_Entertainment_System-USA.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/2/24/Super_Nintendo_Entertainment_System-USA.jpg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Transferred from ja.wikipedia *Художник:* Original uploader was Muband at ja.wikipedia Later version(s) were uploaded by Hr, PiaCarrot at ja.wikipedia.
- **Файл:Superflip.gif** *Источник:* http://upload.wikimedia.org/wikipedia/commons/d/d2/Superflip.gif *Лицензия:* CC0 *Авторы:* http://commons.wikimedia.org/wiki/File:Rubics_cube_PLL.gif *Художник:* User:Internoob
- **Файл:Text_document_with_red_question_mark.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *Лицензия:* Public domain *Авторы:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Художник:* Benjamin D. Esham (bdesham)
- **Файл:Turk-engraving-figure.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/f/fe/Turk-engraving-figure.jpg *Лицензия:* Public domain *Авторы:* ? *Художник:* ?
- **Файл:Turk-engraving5.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/2/25/Turk-engraving5.jpg *Лицензия:* Public domain *Авторы:* ? *Художник:* ?
- **Файл:Wiki_letter_w.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/6/6c/Wiki_letter_w.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* Jarkko Piiroinen

- **Файл:Wikitext-ru.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/47/Wikitext-ru.svg *Лицензия:* Public domain *Авторы:* made by Inkscape *Художник:* self
- **Файл:Wiktionary-logo-ru.png** *Источник:* http://upload.wikimedia.org/wikipedia/commons/b/bc/Wiktionary-logo-ru.png *Лицензия:* ? *Авторы:* Russian Wiktionary *Художник:* One half 3544, VPliousnine
- **Файл:ai_template.gif** *Источник:* http://upload.wikimedia.org/wikipedia/ru/0/0e/Ai_template.gif *Лицензия:* GFDL 1.2+ *Авторы:* На базе File:Human_brain_NIH.png *Художник:* Сергей Яковлев
- **Файл:chess_d45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/3/37/Chess_d45.svg *Лицензия:* GFDL *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_kdd45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/d/da/Chess_kdd45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_kld45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/1/1c/Chess_kld45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_l45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/c/cd/Chess_l45.svg *Лицензия:* GFDL *Авторы:* собственная работа *Художник:* en:User:Cburnett
- **Файл:chess_ndd45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/b/b4/Chess_ndd45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_nld45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/d/d4/Chess_nld45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_pdd45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/41/Chess_pdd45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_pdl45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/0/07/Chess_pdl45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_pld45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/9/98/Chess_pld45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_pll45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/8/86/Chess_pll45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_qdd45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/e/ed/Chess_qdd45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_qll45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/9/9a/Chess_qll45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_rdd45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/4e/Chess_rdd45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_rll45.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/44/Chess_rll45.svg *Лицензия:* CC-BY-SA-3.0 *Авторы:* Это векторное изображение было создано с помощью Inkscape. *Художник:* en:User:Cburnett
- **Файл:chess_zhor_26.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/b/bf/Chess_zhor_26.svg *Лицензия:* Public domain *Авторы:* ? *Художник:* ?
- **Файл:chess_zver_26.svg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/4/46/Chess_zver_26.svg *Лицензия:* Public domain *Авторы:* собственная работа *Художник:* Ysangkok
- **Файл:kempelen-charcoal.jpg** *Источник:* http://upload.wikimedia.org/wikipedia/commons/2/20/Kempelen-charcoal.jpg *Лицензия:* Public domain *Авторы:* ? *Художник:* ?

### 35.8.3. Лицензия