Матлогика

PDF создан с использованием инструмента с открытым исходным кодом mwlib. См. http://code.pediapress.com/ для получения более подробной информации.
PDF generated at: Sat, 08 Jan 2011 23:30:21 UTC

Содержание

Статьи

Формальный язык 1
Конечный автомат 2
Регулярное множество 5
Теорема Клини 6
Граф (математика) 7
Теория графов 12
Словарь терминов теории графов 15
Ориентированный граф 23
Полный граф 25
Задача о клике 26
Разрез графа 27
Шарнир (теория графов) 28
Дерево (теория графов) 29
Теорема Кэли о числе деревьев 32
Остовы графа 33
Алгоритм Шеннона — Фано 34
Алгоритм Флойда — Уоршелла 36
Паросочетание 38
Двудольный граф 39
Трансверсаль 40
Эйлеров цикл 42
Гамильтонов граф 44
Поиск с возвратом 46
Гиперкуб 47
Код Грея 49
Теория сложности вычислений 55
Класс сложности 59
Класс P 61
Класс NP 63
Равенство классов P и NP 65
NP-полная задача 68
Венгерский алгоритм 69
Сведение (теория сложности вычислений) 76
Задача выполнимости булевых формул 77

Задача коммивояжёра 79

Примечания

Источники и основные авторы 81
Источники, лицензии и редакторы изображений 83

Лицензии статей

Лицензия 85

Формальный язык

В математической логике и информатике **формальный язык** — это множество конечных слов (*син.* строк, цепочек) над конечным алфавитом. Понятие языка чаще всего используется в теории автоматов, теории вычислимости и теории алгоритмов. Научная теория, которая имеет дело с этими объектами, называется *теорией формальных языков*.

Например, если алфавит задан как {*a*, *b*}, а язык *L* включает в себя все слова над ним, то слово *ababba* принадлежит *L*.

Пустое слово (то есть строка нулевой длины) допускается и часто обозначается как *e*, ε или Λ.

Некоторые примеры формальных языков:

- множество всех слов над {*a*, *b*}
- множество $\{a^n\}$, где *n* — простое число, а a^n означает, что *a* повторяется *n* раз
- множество синтаксически корректных программ в данном языке программирования

Формальный язык может быть определен по-разному, например:

- Простым перечислением слов, входящих в данный язык. Этот способ, в основном, применим для определения конечных языков и языков простой структуры.
- Словами, порождёнными некоторой формальной грамматикой (см. иерархия Хомского)
- Словами, порождёнными регулярным выражением
- Словами, распознаваемыми некоторым конечным автоматом
- Словами, порождёнными БНФ-конструкцией

Некоторые операции могут быть использованы для того, чтобы порождать новые языки из данных. Предположим, что L_1 и L_2 являются языками, определёнными над некоторым общим алфавитом.

- *Конкатенация (сцепление)* L_1L_2 содержит все слова, удовлетворяющие форме vw, где v — это слово из L_1, а w — слово из L_2.
- *Пересечение* $L_1 \cap L_2$ содержит все слова, содержащиеся и в L_1, и в L_2.
- *Объединение* $L_1 \cup L_2$ содержит все слова, содержащиеся или в L_1, или в L_2.
- *Дополнение* языка L_1 содержит все слова алфавита, которые не содержатся в L_1.
- *Правое отношение* L_1/L_2 содержит все слова v, для которых существует слово w в L_2 такое, что vw находидось в L_1.
- *Замыкание Клини* L_1^* содержит все слова, которые могут быть записаны в форме $w_1w_2...w_n$, где w_i содержится в L_1 и $n \geq 0$. Следует помнить, что это включает и пустое слово ϵ, так как $n = 0$ допустимо по условию.
- *Обращение* L_1^R содержит обращенные слова из L_1.
- *Смешение* L_1 и L_2 содержит все слова, которые могут быть записаны в форме $v_1w_1v_2w_2...v_nw_n$, где $n \geq 1$ и $v_1, ..., v_n$ являются такими словами, что связь $v_1...v_n$ находится в L_1, а $w_1, ..., w_n$ являются такими словами, что $w_1...w_n$ находятся в L_2.

Другие значения

Следует помнить, что мы можем говорить о *формальном языке* во многих контекстах (научном, юридическом, лингвистическом и других), подразумевая способ выражения более осторожный и аккуратный или же более манерный, чем повседневная речь. Смысл формального языка, подразумеваемый здесь, соответствует его определению в теории формальных языков.

В терминологии теории моделей, **язык** соответствует не языку в информатике, а скорее алфавиту. Язык состоит из множеств символов, функций и отношений вместе с их арностью, а также множество переменных. Каждое из этих множеств может быть бесконечным. Из языка вместе с универсальными логическими символами составляются логические высказывания.

Список литературы

- Хопкрофт, Дж., Мотвани, Р., Дж. Ульман (2002). *Введение в теорию автоматов, языков и вычислений.* Addison Wesley. ISBN 5-8459-0261-4
- *Серебряков В.А., Галочкин М.П., Гончар Д.Р., Фуругян М.Г.* Теория и реализация языков программирования [1] //М.: МЗ-Пресс, 2006 г., 2-е изд. ISBN 94073-094-9.
- Подборка ссылок на литературу по направлению. [2]

Примечания

[1] http://trpl.narod.ru/TRYAP_BOOK_Details.htm

[2] http://trpl.narod.ru/printed_guides.htm

Конечный автомат

Конечный автомат — абстрактный автомат без выходного потока, число возможных состояний которого конечно. Результат работы автомата определяется по его конечному состоянию.

Существуют различные варианты задания конечного автомата. Например, конечный автомат может быть задан с помощью пяти параметров: $M = (Q, \Sigma, \delta, q_0, F)$, где:

- Q — конечное *множество состояний* автомата;
- q_0 — *начальное состояние* автомата ($q_0 \in Q$);
- F — множество *заключительных* (или *допускающих*) состояний, таких что $F \subseteq Q$;
- Σ — допустимый *входной алфавит* (конечное множество допустимых входных символов), из которого формируются строки, считываемые автоматом;
- δ — заданное отображение множества $Q \times \Sigma$ во множество $\mathcal{P}(Q)$ подмножеств Q:

$$\delta : Q \times \Sigma \to \mathcal{P}(Q)$$

(иногда δ называют *функцией переходов автомата*).

Автомат начинает работу в состоянии q_0, считывая по одному символу входной строки. Считанный символ переводит автомат в новое состояние из Q в соответствии с функцией переходов. Если по завершении считывания входного слова (цепочки символов) автомат оказывается в одном из допускающих состояний, то слово «принимается» автоматом. В этом случае говорят, что оно принадлежит языку данного автомата. В противном случае слово «отвергается».

Конечные автоматы широко используются на практике, например в синтаксических, лексических анализаторах, и тестировании программного обеспечения на основе моделей.

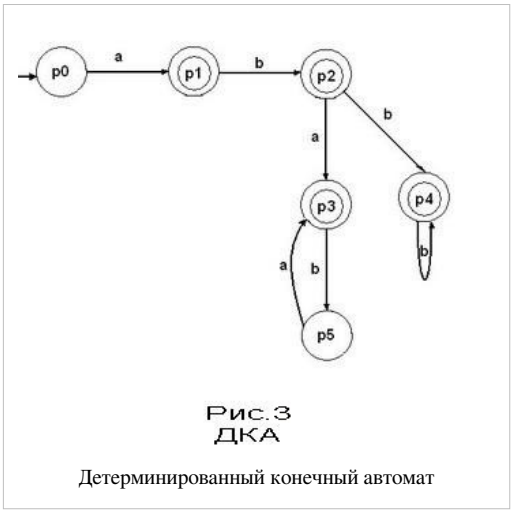
Другие способы описания

1. **Диаграмма состояний** (или иногда **граф переходов**) — графическое представление множества состояний и функции переходов. Представляет собой нагруженный однонаправленный граф, вершины которого — состояния КА, ребра — переходы из одного состояния в другое, а *нагрузка* — символы, при которых осуществляется данный переход. Если переход из состояния q1 в q2 может быть осуществлен при появлении одного из *нескольких* символов, то над дугой диаграммы (ветвью графа) должны быть надписаны все они.
2. **Таблица переходов** — табличное представление функции δ. Обычно в такой таблице каждой строке соответствует одно состояние, а столбцу — один допустимый входной символ. В ячейке на пересечении строки и столбца записывается действие, которое должен выполнить автомат, если в ситуации, когда он находился в данном состоянии на входе он получил данный символ.

Детерминированность

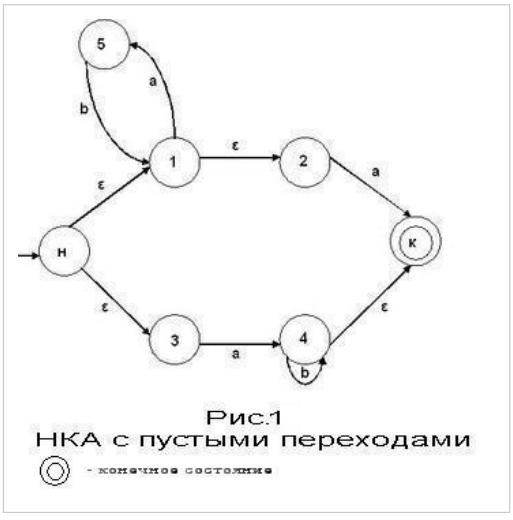
Конечные автоматы подразделяются на детерминированные и недетерминированные.

- Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором для каждой последовательности входных символов существует лишь одно состояние, в которое автомат может перейти из текущего.
- Недетерминированный конечный автомат (НКА) является обобщением детерминированного. Недетерминированность автоматов достигается двумя способами:

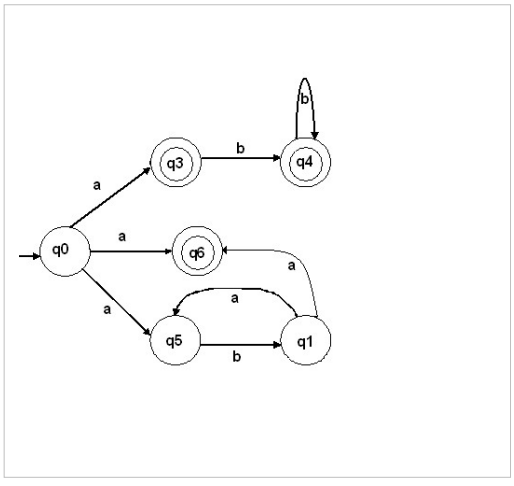


Детерминированный конечный автомат

Существуют переходы, помеченные пустой цепочкой ε



Из одного состояния выходит несколько переходов, помеченных одним и тем же символом



I+

Существует теорема, гласящая, что «Любой недетерминированный конечный автомат может быть преобразован в детерминированный так, чтобы их языки совпадали» (такие автоматы называются эквивалентными). Однако, поскольку количество состояний в эквивалентном ДКА в худшем случае растёт экспоненциально с ростом количества состояний исходного НКА, на практике подобная детерминизация не всегда возможна. Кроме того, конечные автоматы с выходом в общем случае не поддаются детерминизации.

В силу последних двух замечаний, несмотря на бóльшую сложность недетерминированных конечных автоматов, для задач, связанных с обработкой текста, преимущественно применяются именно НКА.

Автоматы и регулярные языки

Для автомата можно определить язык (множество слов) в алфавите Σ, который он **представляет** — так называется множество слов, при вводе которых автомат переходит из начального состояния в одно из состояний множества F.

Теорема Клини гласит, что класс языков, представимых конечными автоматами, совпадает с классом регулярных языков. Кроме того, этот класс совпадает с классом языков, задаваемых регулярными грамматиками.

Специализированные языки программирования

- Язык последовательных функциональных схем SFC (Sequential Function Chart) — графический язык программирования широко используется для программирования промышленных логических контроллеров (ПЛК).

В SFC программа описывается в виде схематической последовательности шагов, объединенных переходами.

Примечания

См. также

- Автоматное программирование
- SFC
- Компилятор
- Транслятор
- Сеть Петри
- Секвенциальная логика (Последовательностная логика)
- Таблица принятия решений

Ссылки

- М. И. Дехтярь Введение в схемы, автоматы и алгоритмы (http://www.intuit.ru/department/ds/introsaa/)
- Open source генератор конечных автоматов на языках C++ и Java по XML файлам описания (http://sourceforge.net/projects/genfsm/)
- http://www.rsdn.ru/article/alg/nka.xml Недетерминированные конечные автоматы]
- С. Ю. Подзоров Курс лекции по теории алгоритмов (http://www.nsu.ru/education/podzorov/Alg/Course.pdf)
- Теория автоматов / Э. А. Якубайтис, В. О. Васюкевич, А. Ю. Гобземис, Н. Е. Зазнова, А. А. Курмит, А. А. Лоренц, А. Ф. Петренко, В. П. Чапенко // Теория вероятностей. Математическая статистика. Теоретическая кибернетика. — М.: ВИНИТИ, 1976. — Т. 13. — С. 109–188. — URL http://www.mathnet.ru/php/getFT.phtml?jrnid=intv&paperid=28&what=fullt&option_lang=rus

Регулярное множество

В теории языков **регуля́рным мно́жеством** (или, **регуля́рным языком**) называется формальный язык, который удовлетворяет приведённым ниже свойствам. Эти простые свойства таковы, что класс регулярных множеств удобно изучать в целом и полученные результаты оказываются применимы во многих важных случаях формальных языков. То есть, понятие регулярного множества является примером математической структуры.

Определение регулярного множества

Пусть Σ — конечный алфавит. *Регулярное множество* R(Σ) в алфавите Σ определяется следующими рекурсивными свойствами:

№.	Свойство	Описание
1	$\emptyset \in R(\Sigma)$	Пустое множество является регулярным множеством в алфавите Σ
2	$\{\epsilon\} \in R(\Sigma)$	Множество, состоящее из одной лишь пустой строки является регулярным множеством в алфавите Σ
3	$\forall a \in \Sigma : \{a\} \in R(\Sigma)$	Множество, состоящее из одного любого символа алфавита Σ является регулярным множеством в алфавите Σ
4	$P \in R(\Sigma) \wedge Q \in R(\Sigma) \Rightarrow (P \cup Q) \in R(\Sigma)$	Если два какие-либо множества являются регулярными в алфавите Σ, то и их объединение тоже является регулярным множеством в алфавите Σ
5	$P \in R(\Sigma) \wedge Q \in R(\Sigma) \Rightarrow PQ \in R(\Sigma)$	Если два какие-либо множества являются регулярными в алфавите Σ, то и множество, составленное из всевозможных сцеплений пар их элементов тоже является регулярным множеством в алфавите Σ
6	$P \in R(\Sigma) \Rightarrow P^* \in R(\Sigma)$	Если какое-либо множество является регулярным в алфавите Σ, то множество всевозможных сцеплений его элементов тоже является регулярным множеством в алфавите Σ

Ничто другое, кроме следующего из перечисленного, не является регулярным множеством в алфавите Σ

См. также

- Построение синтаксического анализатора на основе автоматного подхода

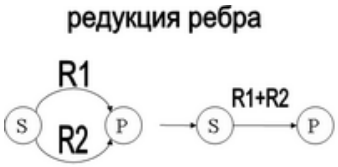
Теорема Клини

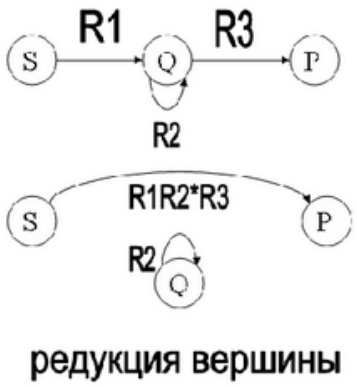
Главный тезис **Теоремы Клини**: «Классы регулярных множеств и автоматных языков совпадают».

Доказательство теоремы Клини

Любой граф переходов конечного автомата всегда можно представить в нормализованной форме, в которой только одна начальная вершина только с исходящими ребрами и только одна заключительная вершина только с входящими ребрами.

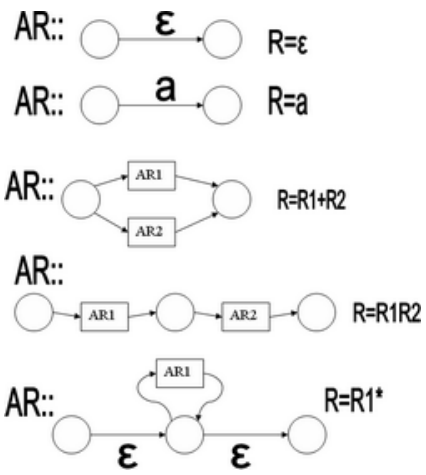
Над графом переходов, представленным в нормализованной форме, могут быть выполнены две операции редукции — редукция ребра и редукция вершины — с сохранением допускаемого этим графом переходов языка. Каждая операция редукции не меняет языка, распознаваемого графом переходов, но уменьшает либо число ребер, либо число вершин.





Следовательно, **каждый автоматный язык является регулярным множеством.**

Для каждого регулярного выражения R может быть построен конечный автомат (возможно недетерминированный), распознающий язык, задаваемый R. Определение таких автоматов дадим рекурсивно.



Следовательно, **каждое регулярное множество является автоматным языком.** Теорема доказана.

Ссылки

- Карпов Ю. Г. Теория автоматов. — СПб.: Питер, 2002. С. 224. ISBN 5-318-00537-3

См. также

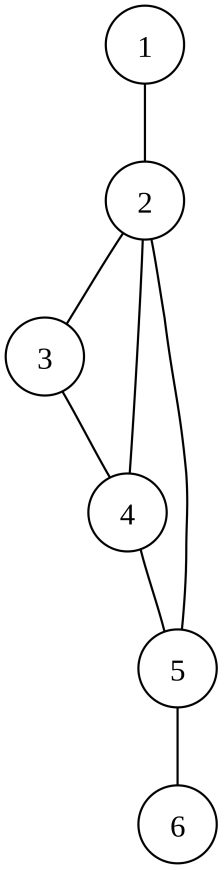
- Конечные автоматы
- Эквивалентность детерминированных и недетерминированных конечных автоматов
- Стивен Клини

Граф (математика)

В математической теории графов и информатике **граф** — это совокупность непустого множества вершин и множества пар вершин.

Объекты представляются как **вершины**, или **узлы** графа, а связи — как **дуги**, или **рёбра**. Для разных областей применения виды графов могут различаться направленностью, ограничениями на количество связей и дополнительными данными о вершинах или рёбрах.

Многие структуры, представляющие практический интерес в математике и информатике, могут быть представлены графами. Например, строение Википедии можно смоделировать при помощи ориентированного графа (орграф), в котором вершины — это статьи, а дуги (ориентированные рёбра) — гиперссылки (см. Тематическая карта).

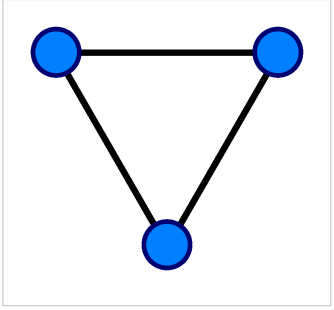


Неориентированный граф с шестью вершинами и семью рёбрами

Определения

Теория графов не обладает устоявшейся терминологией. В различных статьях под одними и теми же терминами понимаются разные вещи. Приводимые ниже определения — наиболее часто встречаемые.

Граф



Граф или **неориентированный граф** G — это упорядоченная пара $G := (V, E)$, для которой выполнены следующие условия:

- V это непустое множество **вершин** или **узлов**,
- E это множество пар (в случае неориентированного графа — неупорядоченных) вершин, называемых **рёбрами**.

V (а значит и E, иначе оно было бы мультимножеством) обычно считаются конечными множествами. Многие хорошие результаты, полученные для конечных графов, неверны (или каким-либо образом отличаются) для *бесконечных графов*. Это происходит потому, что ряд соображений становится ложным в случае бесконечных множеств.

Вершины и рёбра графа называются также **элементами** графа, число вершин в графе $|V|$ — **порядком**, число рёбер $|E|$ — **размером** графа.

Вершины u и v называются **концевыми** вершинами (или просто **концами**) ребра $e = \{u, v\}$. Ребро, в свою очередь, **соединяет** эти вершины. Две концевые вершины одного и того же ребра называются **соседними**. Два ребра называются **смежными**, если они имеют общую концевую вершину.

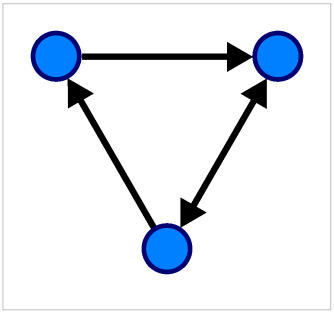
Два ребра называются **кратными**, если множества их концевых вершин совпадают.

Ребро называется **петлёй**, если его концы совпадают, то есть $e = \{v, v\}$.

Степенью $\deg V$ вершины V называют количество рёбер, для которых она является концевой (при этом петли считают дважды).

Вершина называется **изолированной**, если она не является концом ни для одного ребра; **висячей** (или **листом**), если она является концом ровно одного ребра.

Ориентированный граф



Ориентированный граф (сокращённо **орграф**) G — это упорядоченная пара $G := (V, A)$, для которой выполнены следующие условия:

- V это непустое множество **вершин** или **узлов**,
- A это множество (упорядоченных) пар различных вершин, называемых **дугами** или **ориентированными рёбрами**.

Дуга — это упорядоченная пара вершин *(v, w)*, где вершину *v* называют началом, а *w* — концом дуги. Можно сказать, что дуга $v \to w$ ведёт от вершины *v* к вершине *w*.

Смешанный граф

Смешанный граф G — это граф, в котором некоторые рёбра могут быть ориентированными, а некоторые — неориентированными. Записывается упорядоченной тройкой $G := (V, E, A)$, где *V*, *E* и *A* определены так же, как выше.

Ориентированный и неориентированный графы являются частными случаями смешанного.

Изоморфные графы

Граф G называется изоморфным графу H, если существует биекция f из множества вершин графа G в множество вершин графа H, обладающая следующим свойством: если в графе G есть ребро из вершины A в вершину B, то в графе H должно быть ребро из вершины $f(A)$ в вершину $f(B)$ и наоборот — если в графе H есть ребро из вершины A в вершину B, то в графе G должно быть ребро из вершины $f^{-1}(A)$ в вершину $f^{-1}(B)$. В случае ориентированного графа эта биекция также должна сохранять ориентацию ребра.

В случае взвешенного графа биекция также должна сохранять вес ребра.

Прочие связанные определения

Путём (или **цепью**) в графе называют конечную последовательность вершин, в которой каждая вершина (кроме последней) соединена со следующей в последовательности вершин ребром.

Ориентированным путём в орграфе называют конечную последовательность вершин $v_i \ (i = 1, \ldots, k)$, для которой все пары $(v_i, v_{i+1}) \ (i = 1, \ldots k-1)$ являются (ориентированными) рёбрами.

Циклом называют путь, в котором первая и последняя вершины совпадают. При этом **длиной** пути (или цикла) называют число составляющих его *рёбер*. Заметим, что если вершины u и v являются концами некоторого ребра, то согласно данному определению, последовательность (u, v, u) является циклом. Чтобы избежать таких «вырожденных» случаев, вводят следующие понятия.

Путь (или цикл) называют **простым**, если ребра в нём не повторяются; **элементарным**, если он простой и вершины в нём не повторяются. Несложно видеть, что:

- Всякий путь, соединяющий две вершины, содержит элементарный путь, соединяющий те же две вершины.
- Всякий простой *неэлементарный* путь содержит элементарный *цикл*.
- Всякий *простой* цикл, проходящий через некоторую вершину (или ребро), содержит *элементарный* (под-)цикл, проходящий через ту же вершину (или ребро).
- Петля — элементарный цикл.

Бинарное отношение на множестве вершин графа, заданное как «существует путь из u в v», является отношением эквивалентности, и, следовательно, разбивает это множество на классы эквивалентности, называемые **компонентами связности** графа. Если у графа ровно одна компонента связности, то граф связный. На компоненте связности можно ввести понятие **расстояния** между вершинами как минимальную длину пути, соединяющего эти вершины.

Всякий максимальный связный подграф графа G называется **связной компонентой** (или просто компонентой) графа G. Слово «максимальный» означает максимальный относительно включения, то есть не содержащийся в связном подграфе с большим числом элементов

Ребро графа называется **мостом**, если его удаление увеличивает число компонент.

Дополнительные характеристики графов

Граф называется:

- **связным**, если для любых вершин u , v есть путь из u в v .
- **сильно связным** или **ориентированно связным**, если он ориентированный, и из любой вершины в любую другую имеется ориентированный путь.
- **деревом**, если он связный и не содержит *простых* циклов.
- **полным**, если любые его две (различные, если не допускаются петли) вершины соединены ребром.
- **двудольным**, если его вершины можно разбить на два непересекающихся подмножества V_1и V_2так, что всякое ребро соединяет вершину из V_1с вершиной из V_2.
- **k-дольным**, если его вершины можно разбить на k непересекающихся подмножества V_1, V_2, …, V_k так, что не будет рёбер, соединяющих вершины одного и того же подмножества.
- **полным двудольным**, если каждая вершина одного подмножества соединена ребром с каждой вершиной другого подмножества.
- **планарным**, если граф можно изобразить диаграммой на плоскости без пересечений рёбер.
- **взвешенным**, если каждому ребру графа поставлено в соответствие некоторое число, называемое весом ребра.

Также бывает:

- **k-раскрашиваемым**
- **k-хроматическим**

Способы представления графа в информатике

Матрица смежности

Таблица, где как столбцы, так и строки соответствуют вершинам графа. В каждой ячейке этой матрицы записывается число, определяющее наличие связи от вершины-строки к вершине-столбцу (либо наоборот).

Недостатком являются требования к памяти, прямо пропорциональные квадрату количества вершин.

- Двумерный массив;
- Матрица с пропусками;
- Неявное задание (при помощи функции).

Матрица инцидентности

Каждая строка соответствует определённой вершине графа, а столбцы соответствуют связям графа. В ячейку на пересечении i -ой строки с j -м столбцом матрицы записывается:

1

: в случае, если связь j «выходит» из вершины i ,

−1,

: если связь «входит» в вершину,

0

: во всех остальных случаях (то есть если связь является петлёй или связь не инцидентна вершине)

Данный способ является самым ёмким (размер пропорционален $|E||V|$) для хранения, но облегчает нахождение циклов в графе.

Список рёбер

Список рёбер — это тип представления графа в памяти, подразумевающий, что каждое ребро представляется двумя числами — номерами вершин этого ребра. Список рёбер более удобен для реализации различных алгоритмов на графах по сравнению с матрицей смежности.

Обобщение понятия графа

Простой граф является одномерным симплициальным комплексом.

Более абстрактно, граф можно задать как тройку (V, E, φ), где V и E — некоторые множества (*вершин* и *рёбер*, соотв.), а φ — **функция инцидентности** (или **инцидентор**), сопоставляющая каждому ребру $e \in E$ (упорядоченную или неупорядоченную) пару вершин u и v из V (его *концов*). Частными случаями этого понятия являются:

- ориентированные графы (орграфы) — когда $\varphi(e)$ всегда является упорядоченной парой вершин;
- неориентированные графы — когда $\varphi(e)$ всегда является неупорядоченной парой вершин;
- смешанные графы — в котором встречаются как ориентированные, так и неориентированные рёбра и петли;
- Эйлеровы графы — граф в котором существует циклический эйлеров путь (Эйлеров цикл).
- мультиграфы — графы с *кратными* рёбрами, имеющими своими концами одну и ту же пару вершин;
- псевдографы — это мультиграфы, допускающие наличие петель;
- простые графы — не имеющие петель и кратных рёбер.

Под данное выше определение не подходят некоторые другие обобщения:

- гиперграф — если ребро может соединять более двух вершин.
- ультраграф — если между элементами xi и uj существуют бинарные отношения инцидентности.

Литература

- *Оре О.* Теория графов. М.: Наука, 1968. 336с. http://eqworld.ipmnet.ru/ru/library/books/Ore1965ru.djvu
- *Уилсон Р.* Введение в теорию графов. Пер с англ. М.: Мир, 1977. 208с. http://eqworld.ipmnet.ru/ru/library/books/Uilson1977ru.djvu
- *Харари Ф.* Теория графов. М.: Мир, 1973. http://eqworld.ipmnet.ru/ru/library/books/Harari1973ru.djvu
- *Кормен Т. М.и др.* **Часть VI. Алгоритмы для работы с графами** // Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 0-07-013151-1
- *Салий В. Н. Богомолов А. М.* Алгебраические основы теории дискретных систем. — М.: Физико-математическая литература, 1997. — ISBN 5-02-015033-9
- *Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И.* Лекции по теории графов. М.: Наука, 1990. 384с. (Изд.2, испр. М.: УРСС, 2009. 392 с.)
- *Кирсанов М. Н.* Графы в Maple. М.: Физматлит, 2007. — 168 с. http://vuz.exponenta.ru/PDF/book/GrMaple.pdf http://eqworld.ipmnet.ru/ru/library/books/Kirsanov2007ru.pdf

См. также

- Словарь терминов теории графов
- Прямое произведение графов
- Теоремы теории графов
- Boost Graph — Библиотека для работы с графами на языке C++

Ссылки

Популярные программы для визуализации графов

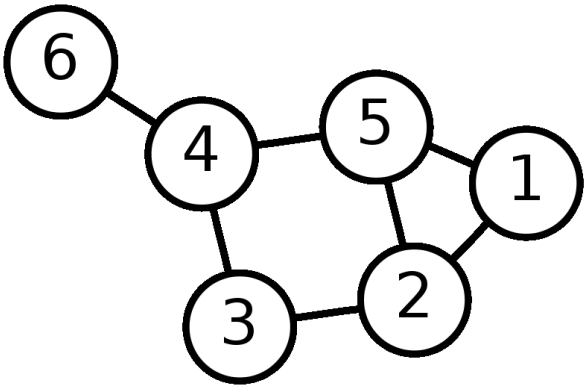
- Graphviz
- aiSee (на русском) [1]
- Intelligent Graph Visualizer (автоматическое размещение на плоскости, поиск кратчайшего пути, поиск центра, раскраска и др.) [2]
- Список Давида Эпстейна (на английском) [3]
- Список Роберто Тамассиа (на английском) [4]
- Список Георга Зандера (на английском) [5]

Примечания

[1] http://www.absint.com/aisee/index_ru.htm
[2] https://sourceforge.net/projects/igv-intelligent/
[3] http://www.ics.uci.edu/~eppstein/gina/gdraw.html
[4] http://cs.brown.edu/people/rt/gd.html
[5] http://rw4.cs.uni-sb.de/~sander/html/gstools.html

Теория графов

Тео́рия гра́фов — раздел дискретной математики, изучающий свойства графов. В общем смысле граф представляется как множество *вершин* (узлов), соединённых *рёбрами*. В строгом определении графом называется такая пара множеств G=(V,E), где V есть подмножество любого счётного множества, а E — подмножество V×V.



Граф с шестью вершинами и семью рёбрами

Теория графов находит применение, например, в геоинформационных системах (ГИС). Существующие или вновь проектируемые дома, сооружения, кварталы и т. п. рассматриваются как вершины, а соединяющие их дороги, инженерные сети, линии электропередачи и т. п. — как рёбра. Применение различных вычислений, производимых на таком графе, позволяет, например, найти кратчайший объездной путь или ближайший продуктовый магазин, спланировать оптимальный маршрут.

Теория графов содержит большое количество нерешённых проблем и пока не доказанных гипотез.

История возникновения теории графов

Родоначальником теории графов считается Леонард Эйлер. В 1736 году в одном из своих писем он формулирует и предлагает решение задачи о семи кёнигсбергских мостах, ставшей впоследствии одной из классических задач теории графов.

Терминология теории графов

Терминология теории графов поныне не определена строго. В частности в монографии *Гудман, Хидетниеми, 1981* сказано: «В программистском мире нет единого мнения о том, какой из двух терминов „граф“ или „сеть“. Мы выбрали термин „сеть“, так как он, по-видимому, чаще встречается в прикладных областях». Аналогичная ситуация для «вершина/точка»

Изображение графов на плоскости

При изображении графов чаще всего используется следующая система обозначений: каждой вершине сопоставляется точка на плоскости, и если между вершинами существует ребро, то соответствующие точки соединяются отрезком. В случае ориентированного графа отрезки заменяют стрелками.

Не следует путать изображение графа с собственно графом (абстрактной структурой), поскольку одному графу можно сопоставить не одно графическое представление. Изображение призвано лишь показать, какие пары вершин соединены рёбрами, а какие — нет. Часто на практике бывает трудно ответить на вопрос, являются ли два изображения моделями одного и того же графа или нет. В зависимости от задачи, одни изображения могут давать более наглядную картину, чем другие.

Некоторые задачи теории графов

- Семь мостов Кёнигсберга — один из первых результатов в теории графов, опубликован Эйлером в 1736.
- Проблема четырёх красок — была сформулирована в 1852 году, но неклассическое доказательство получено лишь в 1976 году (достаточно 4-х красок для карты на сфере (плоскости).
- Задача коммивояжёра — одна из наиболее известных NP-полных задач.
- Задача о клике — ещё одна NP-полная задача.
- Нахождение минимального стягивающего (остовного) дерева.
- Изоморфизм графов — можно ли путем перенумерации вершин одного графа получить другой или его часть.
- Планарность графа — можно ли изобразить граф на плоскости без пересечений ребер (или с минимальным числом слоев, что находит применение при трассировке межсоединений элементов печатных плат или микросхем).

К теории графов также относится целый ряд математических проблем, не решенных на сегодняшний день.

Применение теории графов

- В химии (для описания структур, путей сложных реакций[1] , правило фаз также может быть интерпретировано как задача теории графов); компьютерная химия — сравнительно молодая область химии, основанная на применении теории графов. Теория графов представляет собой математическую основу хемоинформатики. Теория графов позволяет точно определить число теоретически возможных изомеров у углеводородов и других органических соединений.
- В информатике и программировании (граф-схема алгоритма)
- В коммуникационных и транспортных системах. В частности, для маршрутизации данных в Интернете.
- В экономике

- В логистике
- В схемотехнике (топология межсоединений элементов на печатной плате или микросхеме представляет собой граф или гиперграф) [2] .

Литература

- *Басакер Р., Саати Т.* Конечные графы и сети. М.: Наука, 1974. 368с. [3]
- *Белов В. В., Воробьев Е. М., Шаталов В. Е.* Теория графов. — М.: Высш. школа, 1976. — С. 392.
- *Берж К.* Теория графов и ее приложения. М.: ИЛ, 1962. 320с. [4]
- *Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И.* Лекции по теории графов. М.: Наука, 1990. 384с. (Изд.2, испр. М.: УРСС, 2009. 392 с.)
- *Зыков А. А.* Основы теории графов [5]. — М.: «Вузовская книга», 2004. — С. 664. — ISBN 5-9502-0057-8(М.: Наука, 1987. 383с.)
- Химические приложения топологии и теории графов. Под ред. Р. Кинга. Пер. с англ. М.: Мир, 1987.
- *Кирсанов М. Н.* Графы в Maple. М.: Физматлит, 2007. 168 с. http://vuz.exponenta.ru/PDF/book/GrMaple.pdf http://eqworld.ipmnet.ru/ru/library/books/Kirsanov2007ru.pdf
- *Кристофидес Н.*Теория графов. Алгоритмический подход. М.: Мир, 1978. 429с. [6]
- *Кормен Т. Х. и др.* Часть VI. Алгоритмы для работы с графами // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд.. — М.: Вильямс, 2006. — С. 1296. — ISBN 0-07-013151-1
- *Оре О.* Теория графов [7]. — 2-е изд.. — М.: Наука, 1980. — С. 336.
- *Салий В. Н. Богомолов А. М.* Алгебраические основы теории дискретных систем [8]. — М.: Физико-математическая литература, 1997. — ISBN 5-02-015033-9
- Свами М., Тхулалираман К. Графы, сети и алгоритмы. М: Мир, 1984. 455с. [9]
- Татт У. Теория графов. Пер. с англ. М.: Мир, 1988. 424 с. [10]
- *Уилсон Р.* Введение в теорию графов. Пер с англ. М.: Мир, 1977. 208с. [11]
- *Харари Ф.* Теория графов [12]. — М.: Мир, 1973. (Изд. 3, М.: КомКнига, 2006. — 296 с.)
- *Харари Ф., Палмер Э.* Перечисление графов [13]. — Мир, 1977.
- *Diestel R.* Graph Theory, Electronic Edition [14]. — NY: Springer-Verlag, 2005. — С. 422.

Примечания

[1] Г. С. Яблонский, В. И. Быков, А. Н. Горбань, Кинетические модели каталитических реакций (http://thermotree.narod.ru/contybg1983.htm), Новосибирск: Наука (Сиб. отделение), 1983.- 255 с.

[2] Курейчик В.М., Глушань В.М., Щербаков Л.И. Комбинаторные аппаратные модели и алгоритмы в САПР. М.: Радио и связь, 1990. 216 с.

[3] http://eqworld.ipmnet.ru/ru/library/books/BasakerSaati1974ru.djvu

[4] http://eqworld.ipmnet.ru/ru/library/books/Berzh1962ru.djvu

[5] http://www.dleex.com/read/?5561

[6] http://eqworld.ipmnet.ru/ru/library/books/Kristofides1978ru.djvu

[7] http://eqworld.ipmnet.ru/ru/library/books/Ore1965ru.djvu

[8] http://www.krelib.com/?cat=106&file=3816

[9] http://dlc.crimea.edu/catalogue/informatika/ocr-030801052953.djvu

[10] http://ru.dleex.com/read/5623

[11] http://eqworld.ipmnet.ru/ru/library/books/Uilson1977ru.djvu

[12] http://eqworld.ipmnet.ru/ru/library/books/Harari1973ru.djvu

[13] http://eqworld.ipmnet.ru/ru/library/books/HarariPalmer1977ru.djvu

[14] http://diestel-graph-theory.com/GrTh.html

См. также

- Словарь терминов теории графов
- Теоремы теории графов
- Дискретная математика
- Связность графов

Ссылки

- Толковый словарь по теории графов (http://pco.iis.nsk.su/grapp/)
- Лекции 5 и 6 (http://www.csin.ru/courses/informatika-versiya-2002-2003) «Алгоритмы на графах» вводного курса «Информатика»
- Алгоритмы и краткие описания и программ на C++ (http://e-maxx.ru/algo/)
- Дискретная математика, алгоритмы, апплеты, визулизация графов (http://rain.ifmo.ru/cat/view.php/theory/list)
- Графы в химии (http://www.xumuk.ru/encyklopedia/1148.html)
- Intelligent Graph Visualizer (автоматическое размещение на плоскости, поиск кратчайшего пути, поиск центра и др.) (https://sourceforge.net/projects/igv-intelligent/)

Словарь терминов теории графов

Здесь собраны определения терминов из теории графов. Курсивом выделены *ссылки* на термины в этом словаре (на этой странице).

А Б В Г Д Е Ё Ж З И К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Э Ю Я

А

- **Автоморфизм —** Изоморфизм графа с самим собой.

Б

- **Биграф —** см. *двудольный граф*.
- **Блок-дизайн** с параметрами (v, k, λ) **—** покрытие с кратностью λ полного графа на v вершинах полными графами на k вершинах.

В

- **Валентность вершины —** см. *Степень вершины*.
- **Вершина**, **Узел —** базовое понятие: точка, где могут сходиться/выходить *рёбра* и/или *дуги*. Множество вершин графа G обозначается V(G).
- **Взвешенный граф —** граф, каждому *ребру* которого поставлено в соответствие некое значение (*вес ребра*).
- **Висячая вершина —** вершина, *степень* которой равна 1 (то есть $d(v) = 1$).
- **Вполне несвязный граф** (**пустой граф**, **нуль-граф**) **—** регулярный граф степени 0, то есть граф без рёбер.
- **Вес ребра —** значение, поставленное в соответствие данному *ребру взвешенного графа*. Обычно, вес **—** вещественное число, в таком случае его можно интерпретировать как «длину» ребра.

Г

- **Гамильтонов граф —** граф, в котором есть гамильтонов цикл.
- **Гамильтонов путь —** *простой путь* в графе, содержащий все *вершины* графа ровно по одному разу.
- **Гамильтонов цикл —** *простой цикл* в графе, содержащий все вершины графа ровно по одному разу.
- **Геометрическая реализация —** фигура, вершинам которой соответствуют вершины графа, рёбрам — рёбра графа и рёбра в фигуре соединяют вершины, соответствующие вершинам в графе.
- **Геометрический граф —** плоская фигура из вершин — точек плоскости и рёбер — линий, соединяющих некоторые пары вершин. Может представлять многими способами всякий граф.
- **Гиперграф —** совокупность из множества вершин и множества гиперрёбер (подмножество n-й евклидовой степени множества вершин, то есть гиперрёбра соединяют произвольное количество вершин).
- **Гомеоморфные графы —** графы, получаемые из одного графа с помощью последовательности подразбиений рёбер.
- **Грань —** область, ограниченная рёбрами в *плоском графе*, и не содержащая внутри себя вершин и рёбер графа. Внешняя часть плоскости тоже образует грань.
- **Граф —** базовое понятие. Включает множество *вершин* и множество *рёбер*, являющееся подмножеством декартова квадрата множества вершин (то есть каждое ребро соединяет ровно две вершины).
- **Граф рода** g **—** граф, который можно изобразить без пересечений на поверхности рода g и нельзя изобразить без пересечений ни на одной поверхности рода g-1.

Д

- **Двойственный граф**. Граф *A* называется двойственным к планарному графу *B*, если вершины графа *A* соответствуют граням графа *B*, и две вершины графа *A* соединены ребром тогда и только тогда, когда соответствующие грани графа *B* имеют хотя бы одно общее ребро.
- **Двудольный граф** (или **биграф**, или **чётный граф**) — это граф $G(V, E)$, такой что множество вершин *V* разбито на два непересекающихся подмножества V_1 и V_2, причём всякое ребро *E* инцидентно вершине из V_1 и вершине из V_2 (то есть соединяет вершину из V_1 с вершиной из V_2). То есть, *правильная раскраска графа* двумя цветами. Множества V_1 и V_2 называются «долями» двудольного графа. Двудольный граф называется «полным», если любые две вершины из V_1 и V_2 являются смежными. Если $|V_1| = a$, $|V_2| = b$, то полный двудольный граф обозначается $K_{a,b}$.
- **Двусвязный граф —** *связный граф*, в котором нет *шарниров*.
- **Дерево —** *связный граф*, не содержащий циклов.
- **Диаметр графа —** это максимум расстояния между вершинами для всех пар вершин. Расстояние между вершинами — наименьшее число рёбер *пути*, соединяющего две вершины.
- **Длина** *маршрута* **—** количество рёбер в маршруте (с повторениями). Если маршрут $M = v_0, e_1, v_1, e_2, v_2, ..., e_k, v_k$, то длина *M* равна *k* (обозначается $|M| = k$).
- **Длина** *пути* **—** число дуг пути (или сумма длин его дуг, если последние заданы). Так для пути v_1, v_2, …, v_n длина равна n-1.
- **Дуга —** это ориентированное *ребро*.
- **Дополнение графа —** граф над тем же множеством вершин, что и исходный, но вершины соединены ребром тогда и только тогда, когда в исходном графе ребра нет.

З

- **Граф-звезда —** полный двудольный граф $K_{1,b}$.

И

- **Изолированная вершина —** вершина, *степень* которой равна 0 (то есть нет ребер *инцидентных* ей).
- **Изоморфизм**. Два графа называются **изоморфными**, если существует перестановка вершин, при которой они совпадают. Иначе говоря, два графа называются **изоморфными**, если существует взаимно-однозначное соответствие между их вершинами и рёбрами, которое сохраняет *смежность* и *инцидентность* (графы отличаются только названиями своих вершин).
- **Интервальный граф —** граф, вершины которого могут быть взаимно однозначно поставлены в соответствие отрезкам на действительной оси таким образом, что две вершины инцидентны одному ребру тогда и только тогда, когда отрезки, соответствующие этим вершинам, пересекаются.
- **Инцидентность —** понятие, используемое только в отношении ребра и вершины: если v_1, v_2 — вершины, а $e = (v_1, v_2)$ — соединяющее их ребро, тогда вершина v_1 и ребро *e* инцидентны, вершина v_2 и ребро *e* тоже инцидентны. Две вершины (или два ребра) инцидентными быть не могут. Для обозначения ближайших вершин (рёбер) используется понятие *смежности*.

К

- **Клетка —** *регулярный граф* наименьшего *обхвата* для заданной степени вершин.
- **Клика —** подмножество вершин графа, полностью соединённых друг с другом, то есть подграф, являющийся *полным графом*.
  - **Кликовое число**(*Clique number*) **—** число (G) вершин в наибольшей клике. *Другие названия — густота, плотность.*
  - **Максимальная клика —** клика с максимально возможным числом вершин среди клик графа.
- **Конструктивное перечисление графов —** получение полного списка графов в заданном классе.
- **Компонента связности графа —** некоторое подмножество *вершин графа* такое, что для любых двух вершин из этого множества существует *путь* из одной в другую, и не существует пути из вершины этого множества в вершину не из этого множества.
- **Компонента сильной связности графа**, **слой —** максимальное множество вершин *ориентированного графа* такое, что для любых двух вершин из этого множества существует *путь* как из первой во вторую, так и из второй в первую.
- **Контур —** *замкнутый путь в орграфе*.
- **Коцикл —** минимальный *разрез*, минимальное множество ребер, удаление которого делает граф несвязным.
- **Кратные рёбра —** несколько рёбер, *инцидентных* одной и той же паре вершин. Встречаются в *мультиграфах*.
- **Кубический граф —** регулярный граф степени 3, то есть граф в котором каждой вершине инцидентно ровно три ребра.
- **k-дольный граф —** граф G, у которого хроматическое число c(G)=k
- **k-связный граф —** *связный граф*, в котором не существует набора V' из $k-1$ или менее *вершин*, такого, что удаление всех вершин V' и инцидентных им *рёбер* нарушает связность графа. В частности, связный граф является 1-связным, а двусвязный — 2-связным.

Л

- **Лама́нов граф** с *n* вершинами — такой граф *G*, что, во-первых, для каждого *k* любой подграф графа *G*, содержащий *k* вершин, имеет не более, чем 2*k* −3 ребра и, во-вторых, граф *G* имеет ровно 2*n* −3 ребра.
- **Лес** — неориентированный граф без циклов. *Компонентами связности* леса являются *деревья*.
- **Локальная степень вершины** — число рёбер ей инцидентных. Петля дает вклад, равный "2" в степень вершины.

М

- **Маршрут** в графе — это чередующаяся последовательность вершин и рёбер $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, в которой любые два соседних элемента *инцидентны*. Если $v_0 = v_k$, то маршрут **замкнут**, иначе **открыт**.
- **Матрица достижимости** орграфа — это матрица, содержащая информацию о существовании путей между вершинами в орграфе.
- **Матрица инцидентности** графа — это матрица, значения элементов которой характеризуется инцидентностью соответствующих вершин графа (по вертикали) и его рёбер (по горизонтали). Для неориентированного графа элемент принимает значение **1**, если соответствующие ему вершина и ребро инцидентны. Для *ориентированного* графа элемент принимает значение **1**, если инцидентная вершина является началом ребра, значение **-1**, если инцидентная вершина является концом ребра; в остальных случаях (в том числе и для *петель*) значению элемента присваивается **0**.
- **Матрица смежности** графа — это матрица, значения элементов которой характеризуются смежностью вершин графа. При этом значению элемента матрицы присваивается количество рёбер, которые соединяют соответствующие вершины (то есть которые *инцидентны* обоим вершинам). *Петля* считается сразу двумя соединениями для вершины, то есть к значению элемента матрицы в таком случае следует прибавлять **2**.
- **Минимальный каркас** (или **Каркас минимального веса**, **Минимальное остовное дерево**) графа — ациклическое множество рёбер в связном, взвешенном и неориентированном графе, соединяющих между собой все вершины данного графа, при этом сумма весов всех рёбер в нем минимальна.
- **Множество смежности** вершины *v* — множество вершин, смежных с вершиной *v*. Обозначается $\Gamma^+(v)$
- **Мост** — *ребро*, удаление которого увеличивает количество *компонент связности* в графе.
- **Мультиграф** — граф, в котором существует пара вершин, которая соединена более чем одним ребром (ненаправленным), либо более чем двумя дугами противоположных направлений.

Н

- **Направленный граф** — *ориентированный граф* в котором две вершины соединяются не более чем одной дугой.
  - **Направленный ациклический граф** — ориентированный граф без *контуров*.
- **Независимое множество вершин** (известное также как **внутренне устойчивое множество**) — есть множество вершин графа G, такое, что любые две вершины в нем не смежны (никакая пара вершин не соединена ребром).
  - Независимое множество называется **максимальным**, когда нет другого независимого множества, в которое оно бы входило.
  - Если Q является семейством всех независимых множеств графа G, то число a(G) = max |S| (где S принадлежит Q) называется **числом независимости графа** G, а множество S*, на котором этот максимум достигается, называется **наибольшим независимым множеством**.
- **Нормированный граф** — *ориентированный граф* без *циклов.*
- **Нуль-граф** (**вполне несвязный граф**, **пустой граф**) — регулярный граф степени 0, то есть граф, не содержащий рёбер.

О

- **Обхват —** длина наименьшего *цикла* в графе.
- **Окружение —** длина наибольшего *простого цикла в графе.*
- **Орграф**, **ориентированный граф** G = (V,E) есть пара множеств, где V — множество вершин (узлов), E — множество дуг (ориентированных рёбер). Дуга — это упорядоченная пара вершин (v, w), где вершину v называют началом, а w — концом дуги. Можно сказать, что дуга v → w ведет от вершины v к вершине w, при этом вершина w смежная с вершиной v.
- **Остовом** (неориентированного) связного графа G=(V,E) называется его *частичный граф* S=(V,T), являющийся *деревом*.
- **Остовный подграф —** подграф, содержащий все вершины.

П

- **Петля —** ребро, начало и конец которого находятся в одной и той же вершине.
- **Пересечение графов** (помеченных графов) — G1=(X1, U1) и G2=(X2, U2) — граф G1∩G2, множеством вершин которого является X=X1∩X2, а множеством ребер U=U1∩U2, где U1 и U2 — множество неупорядоченных пар вершин.
- **Перечисление графов —** подсчет числа неизоморфных графов в заданном классе (с заданными характеристиками).
- **Периферийная вершина** - вершина, эксцентриситет которой равен диаметру графа.
- **Планарный граф —** граф, который может быть изображён (*уложен*) на плоскости без пересечения рёбер. *Изоморфен* плоскому графу, то есть, является графом с пересечениями, но допускающий его плоскую укладку, поэтому может отличаться от *плоского графа* изображением на плоскости . Таким образом, может быть разница между плоским графом и планарным графом при изображении на плоскости.
- **Плоский граф —** *геометрический граф*, в котором никакие два ребра не имеют общих точек, кроме инцидентной им обоим вершины (не пересекаются). Является *уложенным* графом на плоскости.
- **Подграф** исходного графа — граф, содержащий некое подмножество вершин данного графа и некое подмножество *инцидентных* им рёбер. (ср. *Суграф*.)
- **Полным графом** называется граф, в котором для каждой пары вершин v_1, v_2, существует ребро, инцидентное v_1 и инцидентное v_2 (каждая вершина соединена ребром с любой другой вершиной)
- **Полным двудольным** называется двудольный граф, в котором каждая вершина одного подмножества соединена ребром с каждой вершиной другого подмножества
- **Полустепень захода** в *орграфе* для вершины *v* — число дуг, входящих в вершину. Обозначается $d^+(v)$.
- **Полустепень исхода** в *орграфе* для вершины *v* — число дуг, исходящих из вершины. Обозначается $d^-(v)$.
- **Помеченный граф —** граф, вершинам которого присвоены какие-либо метки, например, натуральные числа или символы какого-нибудь алфавита.
- **Порождённый подграф —** подграф, порождённый множеством вершин исходного графа.
- **Правильная раскраска графа —** *раскраска*, при которой каждый цветной класс является независимым множеством. Иначе говоря, в правильной раскраске любые две смежные вершины должны иметь разные цвета.
- **Простая цепь —** *маршрут*, в котором все вершины различны.
- **Простой граф —** *граф*, в котором нет *кратных рёбер* и *петель*.
- **Простой путь —** *путь*, все рёбра которого попарно различны.[*источник не указан 600*] Другими словами, простой путь не проходит дважды через одно ребро.
  - **Простой цикл —** *цикл*, не проходящий дважды через одну вершину.
- **Псевдограф —** граф, содержащий *петли* и/или кратные рёбра.

- **Пустой граф** (**вполне несвязный граф**, **нуль-граф**) **—** регулярный граф степени 0, то есть граф, не содержащий *рёбер.*
- **Путь —** последовательность *рёбер* (в неориентированном графе) и/или дуг (в ориентированном графе), такая, что конец одной дуги (ребра) является началом другой дуги (ребра). Или последовательность вершин и дуг (рёбер) в которой каждый элемент инцидентен предыдущему и последующему.[*источник не указан 600*] Может рассматриваться как частный случай *маршрута*.
- **Путь в орграфе —** это последовательность вершин v_1, v_2, …, v_n, для которой существуют дуги $v_1 \to v_2$, $v_2 \to v_3$, …, $v_{n-1} \to v_n$. Говорят, что этот путь начинается в вершине v_1, проходит через вершины v_2, v_3, …, v_{n-1}, и заканчивается в вершине v_n.

P

- **Радиус графа —** минимальный из эксцентриситетов вершин связного графа; вершина, на которой достигается этот минимум называется центральной вершиной.
- **Разбиение графа —** представление исходного графа в виде множества подмножеств вершин по определенным правилам.
- **Разделяющая вершина —** то же, что и *шарнир*.
- **Развёртка графа —** функция, заданная на вершинах ориентированного графа.
- **Размеченный граф —** граф, для которого задано множество меток S, функция разметки вершин $f : A \to S$ и функция разметки дуг $g : R \to S$. Графически эти функции представляются надписыванием меток на вершинах и дугах. Множество меток может разделяться на два непересекающихся подмножества меток вершин и меток дуг.
- **Разрез —** множество *ребер*, удаление которого делает граф несвязным.
- **Раскраска графа —** разбиение вершин на множества (называемые цветами). Если при этом нет двух смежных вершин принадлежащих одному и тому же множеству (то есть две смежные вершины всегда разного цвета), то такая раскраска называется правильной.
- **Ребро графа —** базовое понятие. Ребро соединяет две *вершины* графа.
- **Регулярный граф —** граф, *степени* всех вершин которого равны. Степень регулярности является инвариантом графа и обозначается $r(G)$. Для нерегулярных графов $r(G)$не определено. Регулярные графы представляют особую сложность для многих алгоритмов.
  - **Регулярный граф степени 0** (**вполне несвязный граф**, **пустой граф**, **нуль-граф**) **—** граф без рёбер.

C

- **Самодвойственный граф —** граф, *изоморфный* своему *двойственному графу*.
- **Связность**. Две вершины в графе **связаны**, если существует соединяющая их (простая) *цепь*.
- **Связный граф —** граф, в котором все вершины связаны.
- **Сечение графа —** множество рёбер, удаление которых делит граф на два изолированных подграфа, один из которых, в частности, может быть тривиальным графом.
- **Сеть —** в принципе, то же, что и *граф*, хотя сетями обычно называют графы, вершины которых определённым образом помечены.
- **Сильная связность**. Две вершины в *ориентированном графе* **сильно связаны**, если существует путь из первой во вторую и из второй в первую.
  - **Сильно связный орграф —** *орграф*, в котором все вершины сильно связаны.
- **Смежность —** понятие, используемое в отношении только двух *рёбер* либо только двух *вершин*: Два ребра, *инцидентные* одной вершине, называются **смежными**; две вершины, *инцидентные* одному ребру, также называются **смежными**. (ср. *Инцидентность*.)
- **Смешанный граф —** граф, содержащий как ориентированные, так и неориентированные *рёбра*.

- **Спектр графа —** множество всех собственных значений матрицы смежности с учетом кратных рёбер.
- **Степень вершины —** количество рёбер графа *G*, *инцидентных* вершине *x*. Обозначается $d(x)$.
  Минимальная степень вершины графа *G* обозначается $\delta(G)$. а *максимальная* — $\Delta(G)$.
- **Стягивание** ребра графа — замена концов ребра одной вершиной, соседями новой вершины становятся соседи этих концов. Граф G_1 **стягиваем** к G_2, если второй можно получить из первого последовательностью стягиваний рёбер.
- **Суграф** (**частичный граф**) исходного графа — граф, содержащий все вершины исходного графа и подмножество его *рёбер*. (ср. *Подграф*.)

Т

- **Тождественный граф —** граф, у которого возможен один единственный *автоморфизм* — тождественный. Образно говоря, тождественный граф — это «абсолютно несимметричный» граф.
- **Точка сочленения —** то же, что и *шарнир*.
- **Триангуляция поверхности —** *укладка* графа на поверхность, разбивающая её на треугольные области; частный случай топологической триангуляции.
- **Тривиальный граф —** граф, состоящий из одной *вершины*.
- **Турнир —** *ориентированный граф*, в котором каждая пара вершин соединена одним ребром.

У

- **Узел —** то же, что и *Вершина*.
- **Укладка**: граф укладывается на некоторой поверхности, если его можно нарисовать на этой поверхности так, чтобы *рёбра* графа при этом не пересекались. (См. *Планарный граф*, *Плоский граф*.)
- **Упорядоченный граф —** граф, в котором рёбра, выходящие из каждой вершины, однозначно пронумерованы, начиная с 1. Рёбра считаются упорядоченными в порядке возрастания номеров. При графическом представлении часто рёбра считаются упорядоченными в порядке некоторого стандартного обхода (например, против часовой стрелки).

Ф

- **n-Фактор** графа — регулярный остовный подграф степени n.
- **n-Факторизация** графа — разбиение графа на независимые *n-факторы*.

Х

- **Хроматическое число** графа — минимальное количество цветов, требуемое для *раскраски* вершин графа, при которой любые вершины, соединенные ребром, раскрашены в разные цвета.

Ц

- **Центр графа** - вершина, расстояние от которой до самой дальней вершины минимально.
- **Цепь** в графе — *маршрут*, все рёбра которого различны. Если все *вершины* (а тем самым и рёбра) различны, то такая цепь называется **простой** (**элементарной**). В цепи $v_0, e_1, \ldots, e_k, v_k$ вершины v_0 и v_k называются **концами** цепи. Цепь с концами *u* и *v* **соединяет** вершины *u* и *v*. Цепь, соединяющая вершины *u* и *v* обозначается $\langle u, v \rangle$. Для *орграфов* цепь называется орцепью.

  В некоторых источниках **простая цепь** — цепь, *рёбра* которой различны, что является более слабым условием.
- **Цикл —** замкнутая *цепь*. Для *орграфов* цикл называется *контуром*.

- **Цикл** (**простой цикл**) в *орграфе* — это *простой путь длины* не менее 1, который начинается и заканчивается в одной и той же вершине.
- **Цикл Гамильтона** — то же, что и *Гамильтонов цикл*.

- **Цикломатическое число графа** — минимальное число ребер, которые надо удалить, чтобы граф стал ациклическим. Существует соотношение: $p_1(G) = p_0(G) + |E(G)| - |V(G)|$, где $p_1(G)$ — цикломатическое число, p_0 — число *компонент связности* графа, $|E(G)|$ — число *рёбер*, а $|V(G)|$ — число *вершин*.

Ч

- **Частичный граф** — то же, что и *суграф*.
- **Чётный граф** — то же, что и *двудольный граф*.

Ш

- **Шарнир** — *вершина графа*, в результате удаления которой вместе со всеми *инцидентными* ей *рёбрами* количество *компонент связности* в графе возрастает.

Э

- **Эйлеров граф** — это граф, в котором существует *цикл*, содержащий все рёбра графа по одному разу (вершины могут повторяться).
- **Эйлерова цепь** (или **Эйлеров цикл**) — это *цепь* (*цикл*), которая содержит все рёбра графа (вершины могут повторяться).
- **Эксцентриситет** вершины — максимальное из расстояний от данной вершины до других вершин.
- **Элементарный путь** — *путь*, вершины которого, за исключением быть может, первой и последней, различны. Другими словами, простой путь не проходит дважды через одну вершину, но может начаться и закончиться в одной и той же вершине, в таком случае он называется *циклом* (элементарным циклом).
- **Элементарным стягиванием** называется такая процедура: берем *ребро* (вместе с инцидентными ему *вершинами*, например, u и v) и «стягиваем» его, то есть удаляем ребро и отождествляем вершины u и v. Полученная при этом вершина инцидентна тем ребрам (отличным от), которым первоначально были инцидентны u или v.

Ссылки

- Толковый словарь по теории графов [1]
- Теория графов [2]

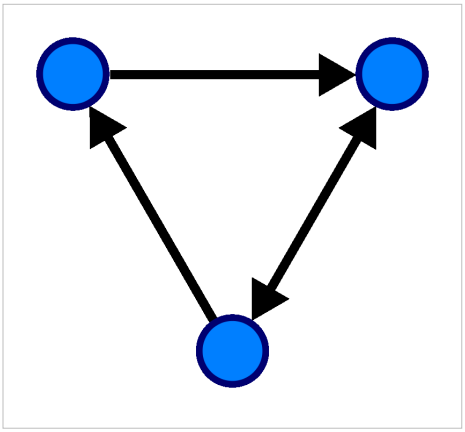
[1] http://pco.iis.nsk.su/grapp/

[2] http://vuz.exponenta.ru/PDF/teogr.html

Литература

- *Харари Ф.* Теория графов. — М.: УРСС, 2003. — 300 с. — ISBN 5-354-00301-6

Ориентированный граф



Ориентированный граф (кратко **орграф**) — (мульти) граф, рёбрам которого присвоено направление. Направленные рёбра именуются также *дугами*, а в некоторых источниках (Оре) и просто рёбрами.

Основные понятия

Формально, орграф D=(V, E) есть множество E упорядоченных пар вершин $v \in V$.

Дуга {u, v} **инцидентна** вершинам u и v. При этом говорят, что u — **начальная вершина** дуги, а v — **конечная вершина**.

Орграф, полученный из простого графа ориентацией ребер называется **направленным**. В отличие от последнего, в произвольном **простом орграфе** две вершины могут соединяться двумя разнонаправленными дугами.

Направленный полный граф называется **турниром**.

Связность

Маршрутом в орграфе называют чередующуюся последовательность вершин и *дуг*, вида $v_0\{v_0, v_1\}v_1\{v_1, v_2\}v_2 \ldots v_n$ (вершины могут повторяться). **Длина маршрута** — количество дуг в нем.

Путь есть *маршрут* в орграфе без повторяющихся дуг, **простой путь** — без повторяющихся вершин. Если существует путь из одной вершины в другую, то вторая вершина **достижима** из первой.

Контур есть замкнутый *путь*.

Для **полумаршрута** снимается ограничение на направление дуг, аналогично определяются **полупуть** и **полуконтур**.

Орграф **сильно связный**, или просто **сильный** если все его вершины взаимно *достижимы*; **односторонне связный**, или просто **односторонний** если для любых двух вершин, по крайней мере одна достижима из другой; **слабо связный**, или просто **слабый**, если при игнорировании направления дуг получается связный (мульти)граф;

Максимальный *сильный* подграф называется **сильной компонентой**; **односторонняя компонента** и **слабая компонента** определяются аналогично.

Конденсацией орграфа D называют орграф D*, вершинами которого служат сильные компоненты D, а дуга в D* показывает наличие хотя бы одной дуги между вершинами, входящими в соответствующие компоненты.

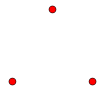
Дополнительные определения

Направленный ациклический граф или **гамак** есть бесконтурный орграф.

Ориентированный граф, полученный из заданного сменой направления ребер на противоположное, называется **обратным**.

Изображение и свойства всех орграфов с тремя узлами

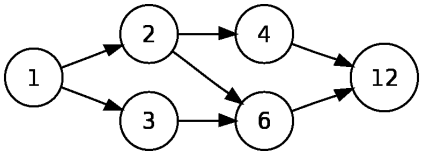
Легенда: **С** — слабый, **ОС** — односторонний, **СС** — сильный, **Н** — является направленным графом, **Г** — является гамаком, **Т** — является турниром

0 дуг	1 дуга	2 дуги	3 дуги	4 дуги	5 дуг	6 дуг
 **пустой, Н, Г**	 **Н, Г**		 **ОС**	 **СС**	 **СС**	 **полный, СС**
		 **ОС, Н, Г**	 **СС, Н, Т**	 **СС**		
		 **С, Н, Г**	 **ОС, Н, Г, Т**	 **ОС**		
		 **С, Н, Г**	 **ОС**	 **ОС**		

Применение орграфов

Орграфы широко применяются в программировании как способ описания систем со сложными связями. К примеру, одна из основных структур, используемых при разработке компиляторов и вообще для представления компьютерных программ — граф потоков данных.

Бинарные отношения



орграф отношения делимости

Бинарное отношение над конечным носителем может быть представлено в виде *орграфа*. *Простым орграфом* представимы антирефлексивные отношения, в общем случае требуется орграф с петлями. Если отношение симметрично, то его можно представить неориентированным графом, а если антисимметрично, то *направленным графом*.

Примечания

Литература

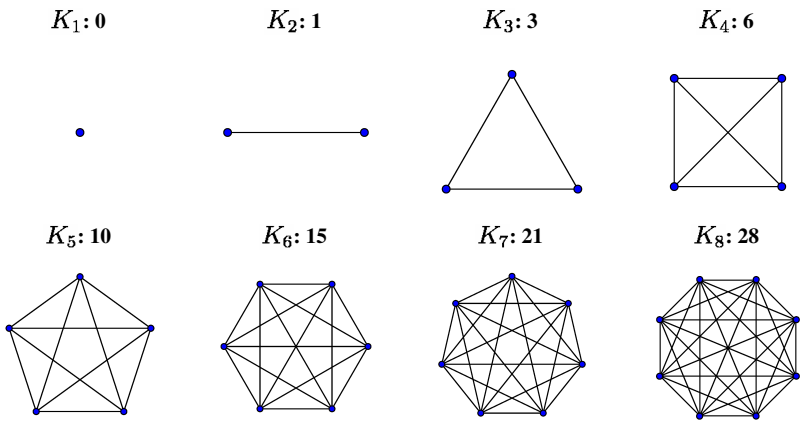
- *Харари Ф.* Теория графов. — М.: УРСС, 2003. — 300 с. — ISBN 5-354-00301-6
- *Оре, Ойстин* Теория графов. — М.: УРСС, 2008. — 352 с. — ISBN 978-5-397-00044-4
- *Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман* Компиляторы: принципы, технологии и инструменты, 2 издание = Compilers: Principles, Techniques, and Tools. — 2 изд. — М.: «Вильямс», 2008. — ISBN 978-5-8459-1349-4

Полный граф

Полный граф — простой граф, в котором каждая пара различных вершин смежна. Полный граф с n вершинами имеет $n(n-1)/2$ рёбер и обозначается K_n. Является регулярным графом степени $n-1$.

Графы с K_1 по K_4 являются планарными. Полные графы с большим количеством вершин не являются планарными, так как содержат подграф K_5 и, следовательно, не удовлетворяют критерию Понтрягина-Куратовского.

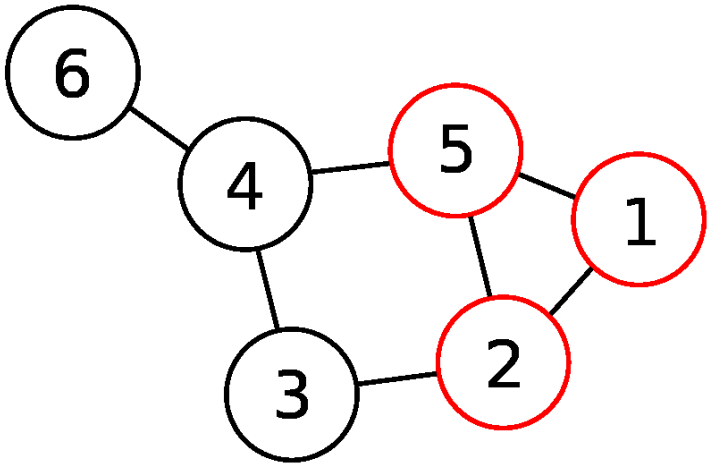
Ниже приведены полные графы с числом вершин от 1 до 8 и количества их рёбер.



Задача о клике

Задача о клике относится к классу NP-полных задач в области теории графов. Впервые она была сформулирована в 1972 году Ричардом Карпом.[1]

Кликой в неориентированном графе называется подмножество вершин, каждые две из которых соединены ребром графа. Иными словами, это *полный подграф* первоначального графа. Размер клики определяется как число вершин в ней. Задача о клике существует в двух вариантах: в задаче распознавания (англ.) требуется определить, существует ли в заданном графе G клика размера k, в то время как в вычислительном варианте требуется найти в заданном графе G клику максимального размера.



Граф с кликой размера 3.

NP-полнота

NP-полнота задачи о клике следует из NP-полноты задачи о независимом множестве (вершин). Легко показать, что необходимым и достаточным условием для существования клики размера k является наличие независимого множества размера не менее k в дополнении графа. Это очевидно, поскольку полнота подграфа означает, что его дополнение не содержит ни одного ребра.

Другое доказательство NP-полноты можно найти в книге «Алгоритмы: построение и анализ».[2]

Алгоритмы

Как и для других NP-полных задач, эффективного алгоритма для поиска клики заданного размера на данный момент не найдено. Полный перебор всех возможных подграфов размера k с проверкой того, является ли хотя бы один из них полным, — неэффективен, поскольку полное число таких подграфов в графе с v вершинами равно биномиальному коэффициенту $\binom{v}{k} = \frac{v!}{k!(v-k)!}$.

Другой алгоритм работает так: две клики размера n и m «склеиваются» в большую клику размера $n+m$, причём кликой размера 1 полагается отдельная вершина графа. Алгоритм завершается, как только ни одного слияния больше произвести нельзя. Время работы данного алгоритма линейно, однако он является эвристическим, поскольку не всегда приводит к нахождению клики максимального размера. В качестве примера неудачного завершения можно привести случай, когда вершины, принадлежащие максимальной клике, оказываются разделены и находятся в кликах меньшего размера, причём последние уже не могут быть «склеены» между собой.

См. также

- Алгоритм Брона — Кербоша — быстрое нахождение клик

Примечания

[1] Karp, Richard (1972). " Reducibility Among Combinatorial Problems (http://www.cs.berkeley.edu/~luca/cs172/karp.pdf)". *Proceedings of a Symposium on the Complexity of Computer Computations*, Plenum Press.

[2] *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

Литература

- Cook, Stephen A. (1971). " The Complexity of Theorem-Proving Procedures (http://www.cs.toronto.edu/~sacook/homepage/1971.pdf.gz)". *Proceedings of the Third Annual ACM Symposium on Theory of Computing*: 151-158. Проверено 2007-06-11.
- Garey, Michael R. & Johnson, David S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, ISBN 0-7167-1045-5 A1.2: GT19, pg.194.

Ссылки

- Challenging Benchmarks for Maximum Clique, Maximum Independent Set, Minimum Vertex Cover and Vertex Coloring (http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm)

Разрез графа

Разре́з гра́фа в задачах о потоке — такая пара множеств вершин (S,T), что

1. $S \cup T = V$, где V — множество вершин графа
2. $S \cap T = \emptyset$
3. $s \in S, t \in T$, где s — исток, t — сток.

Величиной разреза называется сумма пропускных способностей таких рёбер (i, j), что $i \in S, j \in T$.

Другие определения разреза (сечения) графа

- Разрез графа — множество рёбер, удаление которых делит граф на два изолированных подграфа, один из которых, в частности, может быть отдельным узлом.
- Разрез графа — линия, делящая граф на две несвязанные части.

Характеристики

- Линии сечения могут пересекать произвольное число ребер и хорд.
- Для получения главного сечения графа нужно линию сечения графа провести таким образом, чтобы она пересекала только одну ветвь графа при произвольном пересечении хорд.

См. также

- Граф (математика)
- Поток в графе
- Теорема Форда — Фалкерсона

Шарнир (теория графов)

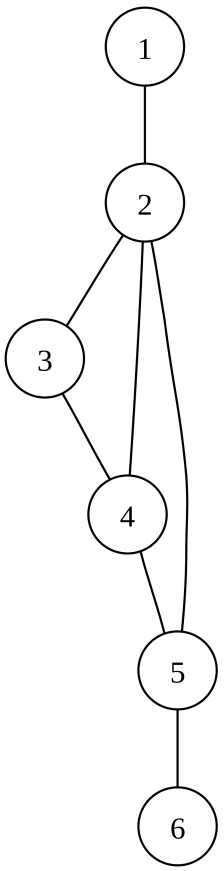
Шарниром в теории графов называется вершина графа, при удалении которой количество компонент связности возрастает. Для обозначения этого понятия также используются термины «разделяющая вершина» и «точка сочленения».

Определения

Вершина v графа G называется **шарниром**, если подграф G_1, полученный из графа G удалением вершины v и всех инцидентных ей рёбер, состоит из большего количества компонент связности, чем исходный граф G.

С понятием шарнира также связано понятие двусвязности. Связный граф, не содержащий шарниров, называется **двусвязным**. Максимальный двусвязный подграф графа называется **компонентой двусвязности**. Компоненты двусвязности иногда называют блоками.

Рёберным аналогом шарнира является **мост**. Мостом называется такое ребро графа, в результате удаления которого количество компонент связности в графе возрастает.



Граф, содержащий два шарнира (вершины 2 и 5) и три блока (12, 2345, 56).

Поиск шарниров

Эффективное решение задачи поиска всех шарниров графа основано на алгоритме поиска в глубину.

Пусть дан граф G. Через $Adj(v)$ обозначим множество всех вершин графа, смежных с v. Предположим, что мы просмотрели граф в глубину, начав с некоторой произвольной вершины. Занумеруем все вершины графа в том порядке, в котором мы их обошли, и каждой вершине v припишем соответствующий номер $n(v)$. Если в вершину u мы впервые попали из вершины v, то вершину u будем называть потомком v, а v — предком u. Множество всех потомков вершины v обозначим через $Ch(v)$. Через $Low(v)$ обозначим минимальный номер среди всех вершин, смежных с v и с теми вершинами, в которые мы пришли по пути, проходящем через v.

Ясно, что величину $Low(v)$ можно вычислить рекурсивно, непосредственно в процессе обхода в глубину: если в настоящий момент рассматривается вершина v, и из неё нельзя перейти в ещё не посещённую вершину (т.е. нужно вернуться к предку v, или прекратить обход, если v — стартовая вершина), то для всех её потомков Low уже посчитано, а значит, и для неё можно можно провести соответствующие вычисления по формуле

$$Low(v) = \min \left\{ \min_{x \in Ch(v)} Low(x), \min_{y \in Adj(v) / Ch(v)} n(y) \right\}.$$

Зная величину $Low(v)$ для всех вершин графа, можно однозначным образом определить все его шарниры согласно следующим двум правилам:

1. Стартовая вершина (т.е. та, с которой мы начали обход) является шарниром тогда и только тогда, когда у неё больше одного потомка.
2. Вершина v, отличная от стартовой, является шарниром тогда и только тогда, когда у неё есть потомок u такой, что $Low(u) = n(v)$.

В качестве примера рассмотрим применение описанного алгоритма к графу, изображённому на рисунке справа. Числа, которыми помечены вершины, соответствуют одному из возможных вариантов обхода в глубину. При таком порядке у каждой из вершин ровно один потомок, за исключением вершины 6, у которой потомков нет. Стартовая вершина 1 имеет с единственного потомка, следовательно, шарниром она не является. Для остальных вершин вычислим значения интересующей нас функции:

$Low(6) = 5, Low(5) = 2, Low(4) = 2, Low(3) = 2, Low(2) = 1.$

У вершины 2 есть потомок 3, а у 5 потомок 6 — в обоих случаях выполнено искомое соотношение $Low(u) = n(v)$. Следовательно, 2 и 5 являются шарнирами. Других шарниров в этом графе нет.

См. также

- Компонента связности

Литература

- *В. Е. Алексеев, В. А. Таланов.* Графы и алгоритмы. Структуры данных. Модели вычислений.. — М.: БИНОМ. Лаборатория знаний, 2006. — ISBN 5-94774-543-7

Дерево (теория графов)

Дерево — это связный граф (то есть такой граф, между любой парой вершин которого существует по крайней мере один путь), не содержащий циклов (то есть ациклический граф).[1] Ацикличность означает, что между любой парой вершин в дереве существует только один путь.

Ориентированное (направленное) дерево — ацикличный орграф (ориентированный граф, не содержащий циклов), в котором только одна вершина имеет нулевую степень захода (в неё не ведут дуги), а все остальные вершины имеют степень захода 1 (в них ведёт ровно по одной дуге). Вершина с нулевой степенью захода называется *корнем* дерева, вершины с нулевой степенью исхода (из которых не исходит ни одна дуга) называются *концевыми вершинами* или *листьями*.[2]

Формально дерево определяется как конечное множество T одного или более узлов со следующими свойствами:

1. существует один корень дерева T
2. остальные узлы (за исключением корня) распределены среди $m \geq 0$ непересекающихся множеств $T_1, ..., T_m$, и каждое из множеств является деревом; деревья $T_1, ..., T_m$ называются поддеревьями данного корня T

Связанные определения

- *Степень узла* — количество исходящих дуг (или, иначе, количество поддеревьев узла).
- *Концевой узел* (*лист*) — узел со степенью 1 (то есть узел, в который ведёт только одно ребро; в случае ориентированного дерева — узел, в который ведёт только одна дуга и не исходит ни одной дуги).
- *Узел ветвления* — неконцевой узел.
- *Уровень узла* — длина пути от корня до узла. Можно определить рекурсивно:

1. уровень корня дерева T равен 0;
2. уровень любого другого узла на единицу больше, чем уровень корня ближайшего поддерева дерева T, содержащего данный узел.

- Дерево с отмеченной вершиной называется *корневым деревом*.
  - m -й *ярус* дерева T — множество узлов дерева, на уровне m от корня дерева.
  - частичный порядок на вершинах: $u \prec v$, если вершины u и v различны и вершина u лежит на (единственной!) элементарной цепи, соединяющей корень с вершиной v.
  - *корневое поддерево* с корнем v — подграф $\{v\} \cup \{w \mid v < w\}$.
- *Остовное дерево* (*остов*) — это подграф данного графа, содержащий все его вершины и являющийся деревом. Рёбра графа, не входящие в остов, называются **хордами** графа относительно остова.
- *Лес* — множество (обычно упорядоченное), не содержащее ни одного непересекающегося дерева или содержащее несколько непересекающихся деревьев.

Двоичное дерево

Термин *двоичное дерево* (оно же *бинарное дерево*) имеет несколько значений:

- Неориентированное дерево, в котором степени вершин не превосходят 3.
- Ориентированное дерево, в котором исходящие степени вершин (число исходящих рёбер) не превосходят 2.
- Абстрактная структура данных, используемая в программировании. На двоичном дереве основаны такие структуры данных, как двоичное дерево поиска, двоичная куча, красно-чёрное дерево, АВЛ-дерево, фибоначчиева куча и др.

N-арные деревья

N-арные деревья определяются по аналогии с двоичным деревом. Для них также есть ориентированные и неориентированные случаи, а также соответствующие абстрактные структуры данных.

- **N-арное дерево** (неориентированное) — это дерево (обычное, неориентированное), в котором степени вершин не превосходят N+1.
- **N-арное дерево** (ориентированное) — это ориентированное дерево, в котором исходящие степени вершин (число исходящих рёбер) не превосходят N.

Свойства

- Дерево не имеет кратных рёбер и петель.
- Любое дерево с n вершинами содержит $n - 1$ ребро. Более того, конечный связный граф является деревом тогда и только тогда, когда $B - P = 1$, где B — число вершин, P — число рёбер графа.
- Граф является деревом тогда и только тогда, когда любые две различные его вершины можно соединить единственным *элементарным* путём.
- Любое дерево однозначно определяется расстояниями (длиной наименьшей цепи) между его концевыми (степени 1) вершинами.

- Любое дерево является двудольным графом. Любое дерево, содержащее счётное количество вершин, является планарным графом.
- Для любых трех вершин дерева, пути между парами этих вершин имеют ровно одну общую вершину.

Подсчёт деревьев

- Число различных деревьев, которые можно построить на n нумерованных вершинах, равно n^{n-2} (**Теорема Кэли**[3]).
- Производящая функция

$$T(z) = \sum_{n=1}^{\infty} T_n z^n$$

для числа T_n неизоморфных корневых деревьев с n вершинами удовлетворяет функциональному уравнению

$$T(z) = x \exp \sum_{r=1}^{\infty} \frac{1}{r} T(x^r).$$

- Производящая функция

$$t(z) = \sum_{n=1}^{\infty} t_n z^n$$

для числа t_n неизоморфных деревьев с n вершинами можно представить с помощью перечисляющего ряда для корневых деревьев:

$$t(z) = T(z) - \frac{1}{2}\left(T^2(z) - T(z^2)\right).$$

- При $n \to \infty$ верна следующая ассимптотика

$$t_n \sim C\alpha^n / n^{5/2}$$

где C и α определённые константы, $C = 0,534948...$, $\alpha = 2,95576...$.

Кодирование деревьев

Дерево можно кодировать наборами из нулей и единиц. Рассмотрим, например, укладку дерева на плоскости. Начиная с какой либо вершины, будем двигаться по ребрам дерева, сворачивая в каждой вершине на ближайшее справа ребро и поворачивая назад в концевых вершинах дерева. Проходя по некоторому ребру, записываем 0 при движении по ребру в первый раз и 1 при движении по ребру второй раз (в обратном направлении). Если m — число рёбер дерева, то через $2m$ шагов мы вернемся в исходную вершину, пройдя по каждому ребру дважды. Полученная при этом последовательность из 0 и 1 (код дерева) длины $2m$ позволяет однозначно восстанавливать не только само дерево D, но и его укладку на плоскости. Произвольному дереву соответствуют несколько таких кодов. В частности, из этого способа кодирования вытекает следующая грубая оценка на число деревьев с n вершинами:

$$t_n \leq T_n < 2^{2n}$$

См. также

- Словарь терминов теории графов
- Лес непересекающихся множеств

Примечания

[1] § 13. Определение дерева // Лекции по теории графов / Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И.. — М.: Наука, Физматлит, 1990. — С. 53. — 384 с. — 22 000 экз. — ISBN 5-02-013992-0

[2] *Альфс Берзтисс.* Глава 3. Теория графов. 3.6. Деревья // Структуры данных = A. T. Berztiss. Data structures. Theory and practice. — М.: Статистика, 1974. — С. 131. — 10 500 экз.

[3] Дискретная математика: алгоритмы. Формула Кэли (http://rain.ifmo.ru/cat/view.php/theory/graph-general/cayley-2008/)

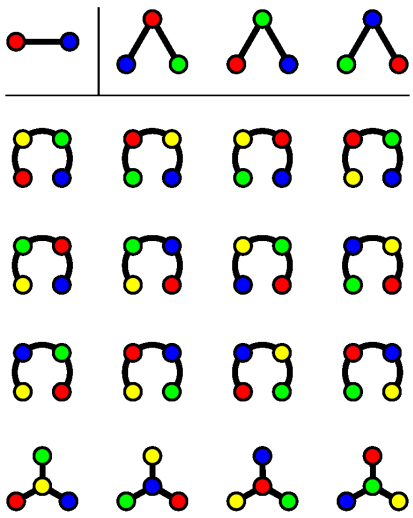
Литература

- *Дональд Кнут.* Искусство программирования, том = The Art of Computer Programming, vol. 1. Fundamental Algorithms. — 3-е изд. — М.: Вильямс, 2006. — Т. 1. Основные алгоритмы. — 720 с. — ISBN 0-201-89683-4
- *Оре О.* Теория графов. — 2-е изд. — М.: Наука, 1980. — 336 с.

Теорема Кэли о числе деревьев

Теорема Кэли о числе деревьев — названное в честь А. Кэли явное выражение в теории графов для числа деревьев с данным числом пронумерованных вершин. А именно, оказывается, что на n вершинах, пронумерованных числами от 1 до n, существует ровно n^{n-2}различных деревьев.

Количество деревьев на n пронумерованных вершинах оказывается также равным числу разложений n-цикла (12…n) в произведение (n-1) транспозиции, а также числу (соответствующим образом нормированных) многочленов степени n с заданными (n-1) критическими значениями общего положения. Наконец, это последнее является частным случаем топологической классификации разветвлённых накрытий сферы Римана — тем самым, подсчёт числа деревьев оказывается частным случаем вычисления чисел Гурвица, соответствующим случаю накрывающей поверхности рода 0.



Полный список деревьев на 2, 3 и 4 пронумерованных вершинах: $2^{2-2} = 1$ дерево с 2 вершинами, $3^{3-2} = 3$ дерева с 3 вершинами и $4^{4-2} = 16$ деревьев с 4 вершинами.

Литература

- Ю. М. Бурман, записки курса «Критические значения многочленов [1]»: [2], [3], [4], [5].
- М. Э. Казарян, записки [6] курса «Геометрия, топология и комбинаторика разветвленных накрытий сферы [7]».
- A. Cayley (1889). «A theorem on trees [8]». *Quart. J. Math* **23**: 376–378.
- T. Ekedahl, S. Lando, M. Shapiro, A. Vainshtein. *Hurwitz numbers and Hodge integrals* [9].

Примечания

[1] http://www.mccme.ru/dubna/2007/courses/burman.htm
[2] http://www.mccme.ru/dubna/2007/notes/burman/Lect1.pdf
[3] http://www.mccme.ru/dubna/2007/notes/burman/Lect2.pdf
[4] http://www.mccme.ru/dubna/2007/notes/burman/Lect3.pdf
[5] http://www.mccme.ru/dubna/2007/notes/burman/Lect4.pdf
[6] http://www.mccme.ru/dubna/2005/notes/kazarian.zip
[7] http://www.mccme.ru/dubna/2005/courses/kazarian.html
[8] http://books.google.com/books?id=M7c4AAAAIAAJ&pg=PA26
[9] http://arxiv.org/abs/math/9902104

Остовы графа

Остов графа

Пусть G — произвольный (n, m)- граф с k компонентами связности. Если G — не лес, то в нем (его компонентах связности) существуют циклы. Рассмотрим какой-либо цикл и удалим из него некоторое ребро. При этом количество компонент связности не увеличится. Если после этого еще останутся циклы, то рассмотрим следующий из них и снова удалим какое-либо его ребро. Продолжим этот процесс до тех пор, пока не исчезнут все циклы. Полученный в результате подграф, который, очевидно, является лесом и имеет столько же компонент связности, как и исходный граф G, называется **остовом графа G**.

Остовым (покрывающим) деревом графа G называется любое дерево T, являющееся суграфом графа G. Покрывающее дерево T в связном графе определяется неоднозначно.

Связный граф имеет единственное покрывающее дерево тогда и только тогда, когда он сам является деревом.

Остов графа G является минимальным связным суграфом графа G, то есть содержит минимальное количество ребер, связывающих вершины графа.

Суграф T графа G называется **каркасом** этого графа, если он является лесом, который на любом компоненте связности графа G образует дерево.

Теорема

Число ребер произвольного графа G с n вершинами и m ребрами, имеющего k компонент связности, который необходимо удалить для получения остова, не зависит от последовательности их удаления и равно m-n+k.

Доказательство теоремы

Пусть H_i, i=i, k-компоненты связности графа G, и пусть H_i-(n_i,m_i)-графы. После удаления ребер из циклов компоненты H_i она превратится в дерево, которое имеет n_i-1 ребер. Значит, из H_i необходимо удалить $m_i-(n_i-1)$ ребер. Суммируя по всем компонентам, находим, что для получения остова из графа G необходимо удалить $\sum_{i=1}^{k}(m_i-n_i+1)=\sum_{i=1}^{k}m_i-\sum_{i=1}^{k}n_i+\sum_{i=1}^{k}1$=m-n+k ребер, что и требовалось доказать.

Определения

Число $\nu(G)=m-n+k$ называется **цикломатическим числом (циклическим рангом)** графа G. Число $\nu^*(G)=n-k$, то есть число ребер, входящих в любой остов графа G называется его **коциклическим рангом**. $\nu(G)+\nu^*(G)=m$. **Корневым деревом** называется дерево $T=(V,E)$ с выделенной вершиной ν, принадлежащая V, при этом вершина ν называется корнем дерева. Для каждой вершины корневого дерева её уровень — это длина цепи от корня до этой вершины.

Следствия

Следствие 1 Граф G является лесом тогда и только тогда, когда $\nu(G)=0$.

Следствие 2 Граф G имеет единственный цикл тогда и только тогда, когда $\nu(G)=1$.

Следствие 3 Граф G, в котором число ребер не меньше, чем число вершин, имеет цикл.

Теорема(Кирхгоф)

Число остовов в связном графе G порядка $n\geq 2$ равно алгебраическому дополнению любого элемента матрицы Кирхгофа K(G) графа G.

Теорема. Орграф сильно связен, если в нем существует остовной циклический маршрут.

Алгоритм Шеннона — Фано

Алгоритм Шеннона — Фано — один из первых алгоритмов сжатия, который впервые сформулировали американские учёные Шеннон и Фано (англ. *Fano*). Данный метод сжатия имеет большое сходство с алгоритмом Хаффмана, который появился на несколько лет позже. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины. Коды Шеннона — Фано префиксные, то есть, никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов.

Основные сведения

Кодирование Шеннона — Фано (англ. *Shannon-Fano coding*) — алгоритм префиксного неоднородного кодирования. Относится к вероятностным методам сжатия (точнее, методам контекстного моделирования нулевого порядка). Подобно алгоритму Хаффмана, алгоритм Шеннона — Фано использует избыточность сообщения, заключённую в неоднородном распределении частот символов его (первичного) алфавита, то есть заменяет коды более частых символов короткими двоичными последовательностями, а коды более редких символов — более длинными двоичными последовательностями.

Алгоритм был независимо друг от друга разработан Шенноном (публикация «Математическая теория связи», 1948 год) и, позже, Фано (опубликовано как технический отчёт).

Основные этапы

1. Символы первичного алфавита m_1 выписывают в порядке убывания вероятностей.
2. Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
3. В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
4. Полученные части рекурсивно делятся и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Когда размер подалфавита становится равен нулю или единице, то дальнейшего удлинения префиксного кода для соответствующих ему символов первичного алфавита не происходит, таким образом, алгоритм присваивает различным символам префиксные коды разной длины. На шаге деления алфавита существует неоднозначность, так как разность суммарных вероятностей $p_0 - p_1$ может быть одинакова для двух вариантов разделения (учитывая, что все символы первичного алфавита имеют вероятность, большую нуля).

Алгоритм вычисления кодов Шеннона — Фано

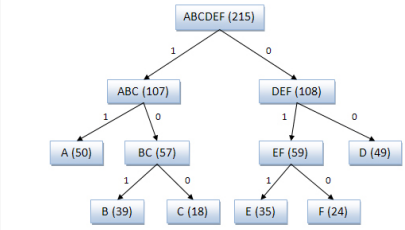
Код Шеннона — Фано строится с помощью дерева. Построение этого дерева начинается от корня. Всё множество кодируемых элементов соответствует корню дерева (вершине первого уровня). Оно разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое из этих подмножеств разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует концевая вершина кодового дерева; такое подмножество разбиению не подлежит. Подобным образом поступаем до тех пор, пока не получим все концевые вершины. Ветви кодового дерева размечаем символами 1 и 0, как в случае кода Хаффмана.

При построении кода Шеннона — Фано разбиение множества элементов может быть произведено, вообще говоря, несколькими способами. Выбор разбиения на уровне n может ухудшить варианты разбиения на следующем уровне (n + 1) и привести к неоптимальности кода в целом. Другими словами, оптимальное поведение на каждом шаге пути ещё не гарантирует оптимальности всей совокупности действий. Поэтому код Шеннона — Фано не является оптимальным в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона — Фано, и все они могут дать различные результаты. Если построить все возможные коды Шеннона — Фано для данного распределения вероятностей, то среди них будут находиться и все коды Хаффмана, то есть оптимальные коды.

Пример кодового дерева

Исходные символы:

- A (частота встречаемости 50)
- B (частота встречаемости 39)
- C (частота встречаемости 18)
- D (частота встречаемости 49)
- E (частота встречаемости 35)
- F (частота встречаемости 24)



Кодовое дерево

Полученный код: A — 11, B — 101, C — 100, D — 00, E — 011, F — 010.

Кодирование Шеннона — Фано является достаточно старым методом сжатия, и на сегодняшний день оно не представляет особого практического интереса. В большинстве случаев, длина последовательности, сжатой по данному методу, равна длине сжатой последовательности с использованием кодирования Хаффмана. Но на некоторых последовательностях могут сформироваться неоптимальные коды Шеннона — Фано, поэтому более эффективным считается сжатие методом Хаффмана.

Литература

- А. М. Яглом, И. М. Яглом. Вероятность и информация. — М.: «Наука», 1973.

Алгоритм Флойда — Уоршелла

Алгоритм Флойда — Уоршелла — динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа. Разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом.

Алгоритм

Пусть вершины графа $G = (V,\ E),\ |V| = n$ пронумерованы от 1 до n и введено обозначение d_{ij}^k для длины кратчайшего пути от i до j, который кроме самих вершин $i,\ j$ проходит только через вершины $1 \dots k$. Очевидно, что d_{ij}^0 — длина (вес) ребра $(i,\ j)$, если таковое существует (в противном случае его длина может быть обозначена как ∞)
Существует два варианта значения $d_{ij}^k,\ k \in (1,\ \dots,\ n)$:

1. Кратчайший путь между $i,\ j$ не проходит через вершину k, тогда $d_{ij}^k = d_{ij}^{k-1}$
2. Существует более короткий путь между $i,\ j$, проходящий через k, тогда он сначала идёт от i до k, а потом от k до j. В этом случае, очевидно, $d_{ij}^k = d_{ik}^{k-1} + d_{kj}^{k-1}$

Таким образом, для нахождения значения функции достаточно выбрать минимум из двух обозначенных значений.

Тогда рекуррентная формула для d_{ij}^k имеет вид:

d_{ij}^0 — длина ребра $(i,\ j)$

$d_{ij}^k = \min(d_{ij}^{k-1},\ d_{ik}^{k-1} + d_{kj}^{k-1})$

Алгоритм Флойда — Уоршелла последовательно вычисляет все значения $d_{ij}^k,\ \forall i,\ j$ для k от 1 до n. Полученные значения d_{ij}^n являются длинами кратчайших путей между вершинами $i,\ j$.

Псевдокод

На каждом шаге алгоритм генерирует двумерную матрицу W, $w_{ij} = d_{ij}^n$. Матрица W содержит длины кратчайших путей между всеми вершинами графа. Перед работой алгоритма матрица W заполняется длинами рёбер графа.

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      W[i][j] = min(W[i][j], W[i][k] + W[k][j])
```

Сложность алгоритма

Три вложенных цикла содержат операцию, исполняемую за константное время. $\sum_{n,\ n,\ n} O(1) = O(n^3),$ то есть алгоритм имеет кубическую сложность, при этом простым расширением можно получить также информацию о кратчайших путях — помимо расстояния между двумя узлами записывать в матрицу идентификатор первого узла в пути.

Вариации алгоритма

Построение матрицы достижимости

Алгоритм Флойда — Уоршелла может быть использован для нахождения замыкания отношения E по транзитивности. Для этого в качестве W[0] используется *бинарная матрица* смежности графа, $(w^0{}_{ij})_{n \times n} = 1 \Leftrightarrow (i,\ j) \in E$; оператор min заменяется дизъюнкцией, сложение заменяется конъюнкцией:

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      W[i][j] = W[i][j] or (W[i][k] and W[k][j])
```

После выполнения алгоритма матрица W является матрицей достижимости.

Использование битовых масок при реализации алгоритма позволяет существенно ускорить алгоритм. При этом сложность алгоритма снижается до $O(n^3/k)$, где k - длина битовой маски (в модели вычислений RAM). На практике, еще бóльшего ускорения можно достичь, используя такие специализированные наборы микропроцессорных команд, как SSE.

См. также

- Алгоритм Дейкстры
- Алгоритм Джонсона

Ссылки

- Реализация алгоритма Флойда на C++ [1]
- Реализация алгоритма Флойда на Delphi [2]
- Визуализатор [3]
- Подробное описание [4]

Литература

- *Ананий В. Левитин* **Глава 8. Динамическое программирование: Алгоритм Флойда поиска кратчайших путей между всеми парами вершин //** Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Aigorithms. — М.: «Вильямс», 2006. — С. 349 — 353. — ISBN 0-201-74395-7
- *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

Примечания

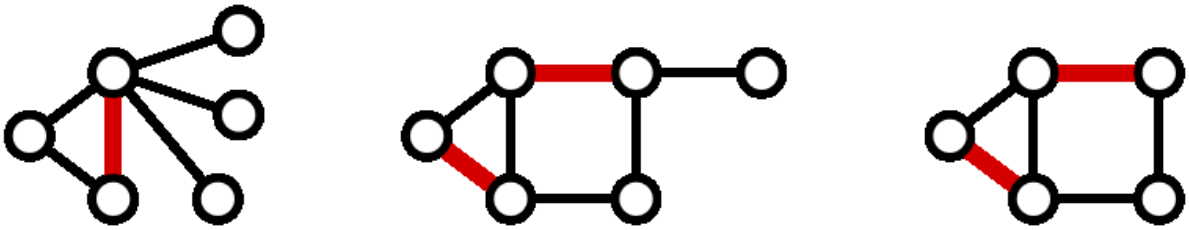
[1] http://e-maxx.ru/algo/floyd_warshall_algorithm
[2] http://plagiata.net.ru/?p=57
[3] http://rain.ifmo.ru/cat/data/vis/graph-paths/floyd-warshall-2004/code.jar
[4] http://habrahabr.ru/blogs/algorithm/105825/

Паросочетание

В теории графов, **паросочетание** или **независимое множество ребер** в графе - это набор попарно несмежных ребер.

Определение

Пусть дан граф $G = (V,E)$, **паросочетание** M в G это множество попарно несмежных ребер, то есть ребер не имеющих общих вершин.



Наибольшее паросочетание - это такое паросочетание, которое содержит максимальное количество ребер.

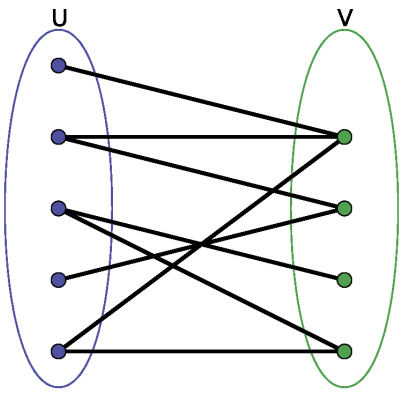
Алгоритмы

Ссылки

- Словарь терминов теории графов

Двудольный граф

Двудо́льный граф или **бигра́ф** — это математический термин теории графов, обозначающий граф, множество вершин которого можно разбить на две части таким образом, что каждое ребро графа соединяет какую-то вершину из одной части с какой-то вершиной другой части, то есть не существует ребра, соединяющего две вершины из одной и той же части.

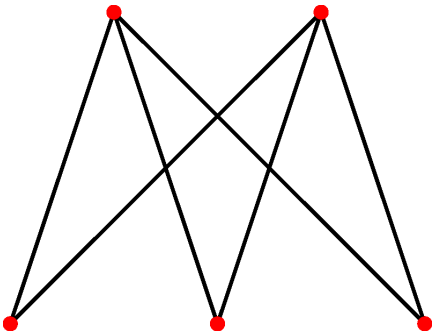


Биграф

Определение

Неориентированный граф $G = (W, E)$ называется двудольным, если множество его вершин можно разбить на две части $U \cup V = W$, $|U| > 0$, $|V| > 0$, так, что

- ни одна вершина в U не соединена с вершинами в U и
- ни одна вершина в V не соединена с вершинами в V


Полный двудольный граф $K_{3,2}$

Двудольный граф называется **полным**, если для каждой пары вершин $u \in U, v \in V$ существует ребро $(u, v) \in E$. Для

$$|U| = i, |V| = j$$

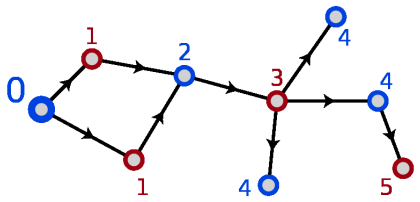
такой граф называется $K_{i,j}$

Свойства

- Граф является двудольным тогда и только тогда, когда он не содержит цикла нечётной длины. Поэтому двудольный граф не может содержать клику размером более 2.
- Граф является двудольным тогда и только тогда, когда он 2-раскрашиваем (то есть его хроматическое число равняется двум)
- Граф разбивается на пары разноцветных вершин тогда и только тогда, когда любые k элементов одной из долей связаны по крайней мере с k элементами другой (Теорема Холла).

Проверка двудольности

Для того, чтобы проверить граф на предмет двудольности, достаточно в каждой компоненте связности выбрать любую вершину и помечать оставшиеся вершины во время обхода графа (например, поиском в ширину или в глубину) поочерёдно как чётные и нечётные (см. иллюстрацию). Если при этом не возникнет конфликта, все чётные вершины образовывают множество U, а все нечётные — V.



Проверка двудольности с помощью чётности расстояний

Применения

- Сети Петри
- Граф Леви
- Теория кодирования
  - Factor graph
  - Tanner graph

См. также

- Словарь терминов теории графов
- Биекция

Трансверсаль

Не путать с трансверсалью треугольника.

Трансверса́ль (**система различных представителей**) — понятие из теории множеств, которое является достаточно важным для всей дискретной математики. Оно также существует в логике и линейной алгебре.

Определение

Обозначим как E конечное непустое множество, а как $S = (S_1, S_2, \ldots, S_m)$ — семейство его подмножеств, то есть последовательность непустых подмножеств E длины m.

Трансверсалью семейства S является такое подмножество $T \subseteq E$, для которого существует биекция $\phi : T \to 1, 2, \ldots, m$, причём для $\forall t \in T$ выполняется условие $t \in S_{\phi(t)}$.

Другими словами, для элементов трансверсали существует такая нумерация, при которой $t_i \in S_i, i = 1, 2, \ldots, m$.

Поскольку T — множество, то все его элементы различны, отсюда и название «система *различных* представителей».

Пример

Пусть заданы множество $E = \{1, 2, 3, 4, 5\}$ и семейство подмножеств $S = \{S_1 = \{1, 4, 5\}, S_2 = \{1\}, S_3 = \{2, 3, 4\}, S_4 = \{2, 4\}\}$. Примером трансверсали для такого семейства может служить $T = \{1, 2, 3, 4\}$, в котором $1 \in S_2, 2 \in S_4, 3 \in S_3, 4 \in S_1$.

Частичная трансверсаль

В случае, если ϕ — инъекция, то трансверсаль называется *частичной*. Частичные трансверсали семейства S являются трансверсалями его подсемейств. Любое подмножество трансверсали является частичной трансверсалью.

Существование

На практике чаще важен ответ на вопрос, существует ли трансверсаль для конкретного семейства. Некоторой шуточной формулировкой этого вопроса является задача о свадьбах:

> Пусть имеется некоторое множество юношей и некоторое множество девушек. При этом каждый юноша из первого множества знаком с несколькими девушками из второго. Требуется женить всех юношей так, чтобы каждый сочетался моногамным браком со знакомой ему девушкой.

Эта задача имеет решение, если для семейства множеств, образованных из девушек, знакомых с конкретным юношей, существует трансверсаль.

Строгим решением задачи о существовании трансверсали является теорема Холла, а также её обобщение — теорема Радо.

Обобщение

Понятие трансверсали можно обобщить.

Пусть имеется последовательность целых положительных чисел $(k_1, k_2, \ldots, k_m)$. Тогда $(k_1, k_2, \ldots, k_m)$-трансверсалью семейства S будет семейство $P = (P_1, P_2, \ldots, P_m)$ подмножеств множества E, для которого выполняются условия:

1. $P_i \subseteq S_i, i = 1, 2, \ldots, m$;
2. $|P_i| = k_i$;
3. $P_i \bigcap P_j = \varnothing, i \neq j$.

Литература

- *М. Холл* Глава №5 «Системы различных представителей» // Комбинаторика = Combinatorial Theory / пер. с англ. С. А. Широковой; под ред. А. О. Гельфонда и В. Е. Тараканова. — М.: Мир, 1970. — С. 65—78. — 424 с.
- *М. Свами, К. Тхуласираман* Глава №8.6 «Паросочетания в двудольных графах» // Графы, сети и алгоритмы = Graphs, Networks, and Algorithms / пер. с англ. М. В. Горбатовой, В. Л. Торхова, С. А. Фролова, В. Н. Четверикова; под ред. В. А. Горбатова. — М.: Мир, 1984. — С. 165—166. — 455 с. — 11 000 экз.
- *В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич* Лекции по теории графов. — М.: Наука, 1990. — С. 87—92. — 384 с. — ISBN 5-02-013992-0
- *Н. И. Костюкова* Графы и их применение [1]. — Лекция №16: Теория трансверсалей. Проверено 29 x12 2009.
- *Alexander Schrijver* Chapter 22 «Transversals», chapter 23 «Common transversals» // Combinatorial optimization [2]. — Springer, 2003. — P. 378—409. — 1800 p. — ISBN 978-3540443896

Примечания

[1] http://www.intuit.ru/department/algorithms/graphsuse/16/
[2] http://books.google.ru/books?id=mqGeSQ6dJycC&pg=PA378&lpg=PA378&dq=transversal+hall&source=bl&ots=xOJTO1gqMc&sig=vXwkhhmQgYZLrlVuYHgAYS7aOw4&hl=ru&ei=gtdxSufjMaDMjAei56SbDA&sa=X&oi=book_result&ct=result

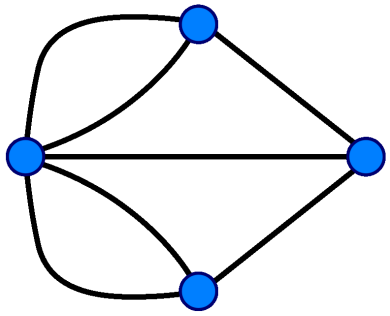
Эйлеров цикл

Эйлеров путь (эйлерова цепь) в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.

Эйлеров цикл — это эйлеров путь, являющийся циклом.

Эйлеров граф — граф, содержащий эйлеров цикл.

Полуэйлеров граф — граф, содержащий эйлеров путь (цепь).



Граф Кёнигсбергских мостов. Этот граф не является эйлеровым, поэтому решения не существует.

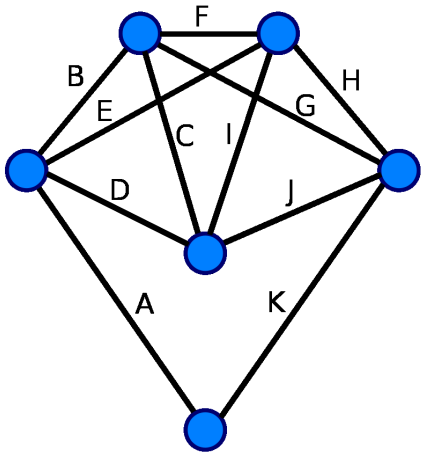
Существование эйлерова цикла и эйлерова пути

Разумеется, эйлеров цикл/путь существуют только в связных графах или в графах, которые после удаления всех одиночных вершин превратятся в связные.

В неориентированном графе

Кроме того, согласно теореме, доказанной Эйлером, эйлеров цикл существует тогда и только тогда, когда граф связный и в нём отсутствуют вершины нечётной степени.

Эйлеров путь в графе существует тогда и только тогда, когда граф связный и содержит не более чем две вершины нечётной степени.[1] [2] Ввиду леммы о рукопожатиях, число вершин с нечётной степенью должно быть четным. А значит Эйлеров путь существует только тогда, когда это число равно нулю или двум. Причём когда оно равно нулю, эйлеров путь вырождается в эйлеров цикл.



Каждая вершина этого графа имеет чётную степень, поэтому этот граф — эйлеров. Обход рёбер в алфавитном порядке даёт эйлеров цикл.

В ориентированном графе

Ориентированный граф содержит эйлеров цикл тогда и только тогда, когда он сильно-связан и для каждой вершины графа её полустепень захода равна её полустепени исхода, то есть в вершину входит столько же ребер, сколько из неё и выходит.

Поиск эйлерова пути в графе

Можно всегда свести задачу поиска эйлерова пути к задаче поиска эйлерова цикла. Действительно, предположим, что эйлерова цикла не существует, а эйлеров путь существует. Тогда в графе будет ровно 2 вершины нечётной степени. Соединим эти вершины ребром, и получим граф, в котором все вершины чётной степени, и эйлеров цикл в нём существует. Найдём в этом графе эйлеров цикл (алгоритмом, описанным ниже), а затем удалим из ответа несуществующее ребро.

Поиск эйлерова цикла в графе

Будем рассматривать самый общий случай — случай ориентированного мультиграфа, возможно, с петлями. Также мы предполагаем, что эйлеров цикл в графе существует (и состоит хотя бы из одной вершины). Для поиска эйлерова цикла воспользуемся тем, что эйлеров цикл — это объединение всех простых циклов графа. Следовательно, наша задача — эффективно найти все циклы и эффективно объединить их в один.

Реализовать это можно, например, так, рекурсивно:

```
procedure find_all_cycles (v)
var массив cycles
1. пока есть цикл, проходящий через v, находим его
    добавляем все вершины найденного цикла в массив cycles (сохраняя порядок обхода)
    удаляем цикл из графа
2. идем по элементам массива cycles
    каждый элемент cycles[i] добавляем к ответу
    из каждого элемента рекурсивно вызываем себя: find_all_cycles (cycles[i])
```

Достаточно вызвать эту процедуру из любой неодиночной вершины графа, и она найдёт все циклы в графе, удалит их из графа и объединит их в один эйлеров цикл.

Для поиска цикла на шаге 1 используем поиск в глубину.

Сложность полученного алгоритма — O(M), то есть линейная относительно количества рёбер M в данном графе.

Примечания

[1] Эйлеровы пути (http://www.msclub.ce.cctpu.edu.ru/bibl/ODM/Ll2_5.html)
[2] Маршруты и связность в орграфах (http://www.intuit.ru/department/algorithms/gaa/2/3.html)

См. также

- Гамильтонов цикл
- Граф (математика)
- Задача о ходе коня
- Дискретная математика
- Семь мостов Кёнигсберга
- Список объектов, названных в честь Леонарда Эйлера

Ссылки

- Реализация алгоритма поиска эйлерова цикла (краткие описания и программы на C++) (http://e-maxx.ru/algo/euler_path)
- Реализация алгоритма поиска эйлерова цикла на codenet.ru (http://sources.codenet.ru/download/936/graph3.html)
- Теория графов и комбинаторика (http://belsky.narod.ru/v2/download/mathemat/courses/tgik/)
- Эйлеровы и Гамильтоновы графы (http://www.refstar.ru/data/r/id.16731_1.html)
- Графы. Циклы и разрезы (ДИСКРЕТНАЯ МАТЕМАТИКА: АЛГОРИТМЫ, Визуализаторы) (http://rain.ifmo.ru/cat/view.php/vis/graph-circuits-cuts)
- Е. Гик. «Шахматы и математика» (http://golovolomka.hobby.ru/books/gik/index.shtml) Конь-хамелеон (http://golovolomka.hobby.ru/books/gik/02.shtml)

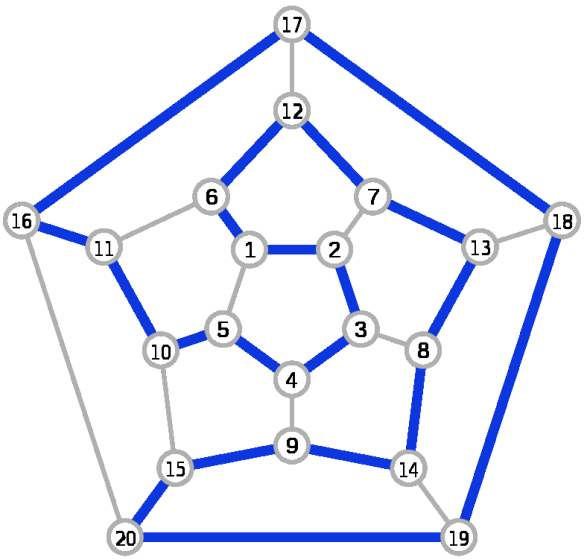
- *Weisstein, Eric W.* Eulerian Circuit (http://mathworld.wolfram.com/EulerianCircuit.html) (англ.) на сайте Wolfram MathWorld.

Гамильтонов граф

Гамильтонов граф — в теории графов это граф, содержащий *гамильтонову цепь* или *гамильтонов цикл*.

Гамильтонов путь (или **гамильтонова цепь**) — путь (цепь), содержащий каждую вершину графа ровно один раз. Гамильтонов путь, начальная и конечная вершины которого совпадают, называется **гамильтоновым циклом**. Гамильтонов цикл является простым остовным циклом (см. Словарь терминов теории графов).

Гамильтоновы путь, цикл и граф названы в честь ирландского математика У. Гамильтона, который впервые определил эти классы, исследовав задачу «кругосветного путешествия» по додекаэдру, узловые вершины которого символизировали крупнейшие города Земли, а рёбра — соединяющие их дороги.



Граф додекаэдра с выделенным циклом Гамильтона

Условия существования

Условие Дирака (англ.) (1952)

Пусть p — число вершин в данном графе; если степень каждой вершины не меньше, чем $\frac{p}{2}$, то граф называется графом Дирака. Граф Дирака — гамильтонов.

Условие Оре (1960)

p — количество вершин в данном графе. Если для любой пары несмежных вершин x, y выполнено неравенство $d(x) + d(y) \geqslant p$, то граф называется графом Оре (словами: степени любых двух несмежных вершин не меньше общего числа вершин в графе). Граф Оре — гамильтонов.

Теорема Бонди-Хватала

Теорема Бонди-Хватала (англ.) обобщающает утверждения Дирака и Оре. Для графа G с n вершинами замыкание определяется добавлением в G ребра (u,v) для каждой пары несмежных вершин u и v, сумма степеней которых не меньше n.

Граф является гамильтоновым тогда и только тогда, когда его замыкание — гамильтонов граф.

Условие Поша

Введем следующую функцию $f(x)$ целого неотрицательного аргумента x на графе $G = [A, B]$:

$$f(x) = |\{a \in A | d(a) \leq x\}| .$$

Написанное означает, что функция $f(x)$ в каждом целом неотрицательном x принимает значение, равное количеству вершин графа $G = [A, B]$, степень которых не превосходит x. Такую функцию $f(x)$ называют функцией Поша графа G.

См. также

- Эйлеров цикл
- Задача коммивояжёра
- Задача о ходе коня
- Семь мостов Кёнигсберга
- Китайская стена (головоломка)

Ссылки

- *Weisstein, Eric W.* Hamiltonian Circuit [1] (англ.) на сайте Wolfram MathWorld.
- Теория графов и комбинаторика [2]
- Эйлеровы и Гамильтоновы графы [3]

Примечания

[1] http://mathworld.wolfram.com/HamiltonianCircuit.html
[2] http://belsky.narod.ru/v2/download/mathemat/courses/tgik/
[3] http://www.refstar.ru/data/r/id.16731_1.html

Поиск с возвратом

Поиск с возвратом (англ. *Backtracking*) — общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве M. Как правило позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты …», «Сколько существует способов …», «Есть ли способ …», «Существует ли объект…» и т. п.

Термин *backtrack* был введен в 1950 году американским математиком Дерриком Генри Лемером.

Незначительные модификации метода поиска с возвратом, связанные с представлением данных или особенностями реализации, имеют и иные названия: метод ветвей и границ, поиск в глубину, метод проб и ошибок и т. д. Поиск с возвратом практически одновременно и независимо был изобретен многими исследователями еще до его формального описания.

Описание метода

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения.

Использование метода

Классическим примером использования алгоритма поиска с возвратом является задача о восьми ферзях. Её формулировка такова: «Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого». Сперва на доску ставят одного ферзя, а потом пытаются поставить каждого следующего ферзя так, чтобы его не били уже установленные ферзи. Если на очередном шаге такую установку сделать нельзя — возвращаются на шаг назад и пытаются поставить ранее установленного ферзя на другое место.

Помимо этого, метод поиска с возвратом позволяет решать множество других переборных задач. Например, с помощью него можно получить все подмножества, размещения, перестановки, сочетания данного множества M.

Недостатки

Метод поиска с возвратом является универсальным. Достаточно легко проектировать и программировать алгоритмы решения задач с использованием этого метода. Однако время нахождения решения может быть очень велико даже при небольших размерностях задачи (количестве исходных данных), причём настолько велико (может составлять годы или даже века), что о практическом применении не может быть и речи. Поэтому при проектировании таких алгоритмов, обязательно нужно теоретически оценивать время их работы на конкретных данных. Существуют также задачи выбора, для решения которых можно построить уникальные, «быстрые» алгоритмы, позволяющие быстро получить решение даже при больших размерностях задачи. Метод поиска с возвратом в таких задачах применять неэффективно.

См. также

- Рекурсия
- Метод ветвей и границ

Ссылки

- interactive animation of a backtracking algorithm [1]
- Перебор с возвратом [2]

Примечания

[1] http://www.hbmeyer.de/backtrack/backtren.htm
[2] http://www.mgopu.ru/PVU/2.1/Recurs/BacketTm/CnReturn/cn_retrn.htm

Гиперкуб

Гиперкуб — обобщение куба на случаи с числом измерений больше 3. Формально гиперкуб определяется как декартово произведение N равных отрезков.

Также можно сказать, что **N-куб —** это фигура, каждая вершина которой связана ребрами с N других вершин; N в свою очередь определяет размерность этой фигуры. Или же, N-мерный куб образуется N парами параллельных (N-1)-плоскостей, то есть имеет 2N граней, каждая из которых является (N-1)-кубом.

Свойства гиперкуба

Свойство	Значение
Длина ребра	a
Размерность	N
Объём	$V_N = a^N$
Площадь поверхности	$S_N = 2Na^{N-1}$

В информатике

Гиперкуб — сетевая топология, в которой узлы являются вершинами графа многомерного куба.

Различные гиперкубы

N-Куб	Изображение (двумерная проекция)	Название	Точек (*0*)	Отрезков (*1*)	Квадратов (*2*)	Кубов (*3*)	Тессерактов (*4*)	Пентерактов (*5*)	Хексерактов (*6*)	Хептерактов (*7*)	Октерактов (*8*)	Энтенерактов (*9*)	10-кубов (*10*)
0-куб		Точка	1										
1-куб		Отрезок	2	1									
2-куб		Квадрат	4	4	1								
3-куб		Куб	8	12	6	1							
4-куб		Тессеракт	16	32	24	8	1						
5-куб		Пентеракт	32	80	80	40	10	1					
6-куб		Хексеракт	64	192	240	160	60	12	1				
7-куб		Хептеракт	128	448	672	560	280	84	14	1			
8-куб		Октеракт	256	1024	1792	1792	1120	448	112	16	1		
9-куб		Энтенеракт	512	2304	4608	5376	4032	2016	672	144	18	1	
10-куб		10-куб	1024	5120	11520	15360	13440	8064	3360	960	180	20	1

Гиперкуб в художественной литературе

- Роберт Хайнлайн. Дом, который построил Тил.
- Роберт Шекли. Мисс Мышка и четвертое измерение.

См. также

- Куб 2: Гиперкуб
- Правильные N-мерные многогранники

Код Грея

2-битный код Грея
00 01 11 10
3-битный код Грея
000 001 011 010 110 111 101 100
4-битный код Грея
0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

Код Грея — система нумерования, в которой два соседних значения различаются только в одном разряде. Наиболее часто на практике применяется **рефлексный двоичный код Грея**, хотя в общем случае существует бесконечное множество кодов Грея для систем счисления с любым основанием. В большинстве случаев, под термином "код Грея" понимают именно рефлексивный бинарный код Грея.

Изначально предназначался для защиты от ложного срабатывания электромеханических переключателей. Сегодня коды Грея широко используются для упрощения выявления и исправления ошибок в системах связи, а также в формировании сигналов обратной связи в системах управления.

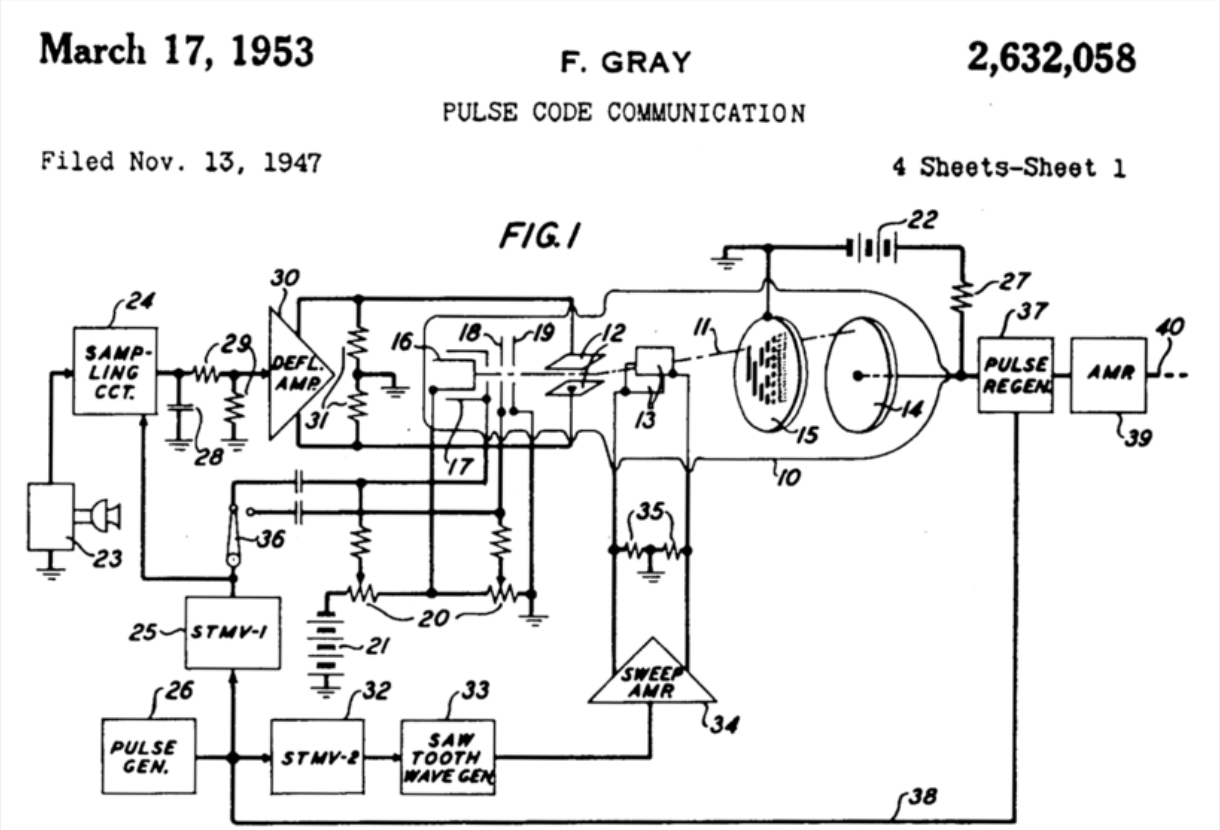
Название

Название *рефлексный (отражённый) двоичный код* происходит от факта, что вторая половина значений в коде Грея эквивалентна первой половине, только в обратном порядке, за исключением старшего бита, который просто инвертируется. Если же разделить каждую половину ещё раз пополам, свойство будет сохраняться для каждой из половин половины и т. д.

Код получил имя исследователя лабораторий Bell Labs Фрэнка Грея. Он использовал этот код в своей импульсной системе связи, для чего был написан патент за номером 2632058.

Применения

Использование кодов Грея основано прежде всего на том, что он минимизирует эффект ошибок при преобразовании аналоговых сигналов в цифровые (например, во многих видах датчиков).

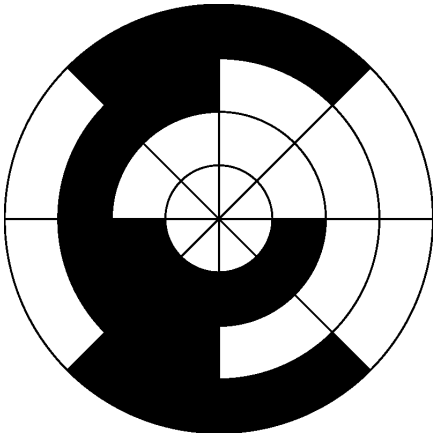


Фрагмент главной страницы патента Грея

Коды Грея часто используются в датчиках-энкодерах. Их использование удобно тем, что два соседних значения шкалы сигнала отличаются только в одном разряде. Также они используются для кодирования номера дорожек в жёстких дисках.

Код Грея можно использовать также и для решения задачи о Ханойских башнях .

Широко применяются коды Грея и в теории генетических алгоритмов [1] для кодирования генетических признаков, представленных целыми числами.



Круговой энкодер с трёхбитным кодом грея

Алгоритмы преобразования

Преобразование двоичного кода в код Грея

Коды Грея легко получаются из двоичных чисел путём побитовой операции «Исключающее ИЛИ» с тем же числом, сдвинутым вправо на один бит. Следовательно, i-й бит кода Грея G_i выражается через биты двоичного кода B_i следующим образом:

$$G_i = B_i \oplus B_{i+1},$$

где $\oplus$ – операция «исключающее ИЛИ»; биты нумеруются справа налево, начиная с младшего.

Ниже приведён алгоритм преобразования из двоичной системы счисления в код Грея, записанный на языке C:

```c
unsigned int grayencode(unsigned int g)
{
    return g ^ (g >> 1);
}
```

Тот же самый алгоритм, записанный на языке Паскаль:

```pascal
function BinToGray(b:integer):integer;
begin
  BinToGray:=b xor (b shr 1)
end;
```

Пример: преобразовать двоичное число 10110 в код Грея.

```
10110
01011
-----
11101
```

Преобразование кода Грея в двоичный код

Обратный алгоритм – преобразование кода Грея в двоичный код – можно выразить рекуррентной формулой

$$B_i = B_{i+1} \oplus G_i,$$

причём преобразование осуществляется побитно, начиная со старших разрядов, и значение B_{i+1}, используемое в формуле, вычисляется на предыдущем шаге алгоритма. Действительно, если подставить в эту формулу вышеприведённое выражение для *i*-го бита кода Грея, получим

$$B_i = B_{i+1} \oplus G_i = B_{i+1} \oplus (B_i \oplus B_{i+1}) = B_i \oplus (B_{i+1} \oplus B_{i+1}) = B_i \oplus 0 = B_i.$$

Однако приведённый алгоритм, связанный с манипуляцией отдельными битами, неудобен для программной реализации, поэтому на практике используют видоизменённый алгоритм:

$$B_k = \bigoplus_{i=k}^{N} G_i,$$

где *N* – число битов в коде Грея (для увеличения быстродействия алгоритма в качестве *N* можно взять номер старшего ненулевого бита кода Грея); знак $\bigoplus$ означает суммирование при помощи операции «исключающее ИЛИ», то есть

$$\bigoplus_{i=k}^{N} G_i = G_k \oplus G_{k+1} \oplus \ldots \oplus G_{N-1} \oplus G_N.$$

Действительно, подстатавив в формулу выражение для *i*-го бита кода Грея, получим

$$B_{i+1}) = (B_k \oplus B_{k+1}) \oplus (B_{k+1} \oplus B_{k+2}) \oplus \ldots \oplus (B_{N-1} \oplus B_N) \oplus (B_N \oplus B_{N+1}) = = B_k \oplus (B_{k+1} \oplus B_{k+1}) \oplus \ldots \oplus (B_N \oplus B_N) \oplus B_I$$

Здесь предполагается, что бит, выходящий за рамки разрядной сетки (B_{N+1}), равен нулю.

Ниже приведена функция на языке C, реализующаяя данный алгоритм. Она осуществляет последовательный сдвиг вправо и суммирование исходного двоичного числа, до тех пор, пока очередной сдвиг не обнулит слагаемое.

```c
unsigned int graydecode(unsigned int gray)
{
    unsigned int bin;
    for (bin = 0; gray; gray >>= 1) {
      bin ^= gray;
    }
    return bin;
}
```

Тот же самый алгоритм, записанный на языке Паскаль:

```pascal
function GrayToBin(b:integer):integer;
 var g:integer;
begin
  g:=0;
  while b>0 do begin
    g:=g xor b;
    b:=b shr 1;
  end;
  GrayToBin:=g;
end;
```

Пример: преобразовать код Грея 11101 в двоичный код.

```
11101
01110
00111
00011
00001
-----
10110
```

Быстрое преобразование 8/16/24/32-разрядного значения кода Грея в двоичный код на языке BlitzBasic:

```
Function GRAY_2_BIN%(X%)
Return X Xor ((X And $88888888) Shr 4) Xor ((X And $CCCCCCCC) Shr 2) Xor ((X And $EEEEEEEE) Shr 1)
End Function
```

Генерация кодов Грея

Ниже представлен один из алгоритмов создания последовательности кода Грея заданной глубины, записанный на языке Perl:

```
  my $depth = 16; # generate 16 Gray codes, 4 bits wide each
  my @gray_codes = ( '0', '1' );
  while(scalar(@gray_codes)<$depth)
      {
      my @forward_half=map{'0'.$_} @gray_codes;
      my @reverse_half=map{'1'.$_} reverse(@gray_codes);
      @gray_codes=(@forward_half,@reverse_half);
      }
```

Рекурсивная функция построение кода Грея на языке C:

```
//n -- требуемая длина кода,
//m -- указатель на массив, способный хранить
// все коды Грея, длиной до n
// (должен быть выделен до вызова функции)
//depth -- параметр рекурсии

int gray (int n, int* m, int depth)

{
      int i, t = (1 << (depth - 1));

      if (depth == 0)
            m[0] = 0;

      else {
        //массив хранит десятичные записи двоичных слов
            for (i = 0; i < t; i++)
                  m[t + i] = m[t - i - 1] + (1 << (depth - 1));
      }
```

```
      if (depth != n)
            gray(n, m, depth + 1);

      return 0;
}
```

Быстрое преобразование 8/16/24/32-разрядного бинарного кода в код Грея на языке BlitzBasic:

```
Function BIN_2_GRAY%(X%)
Return X Xor ((X And $EEEEEEEE) Shr 1)
End Function
```

См. также

- Код Хемминга
- Код Джонсона

Примечания

[1] BaseGroup.ru :: Генетические алгоритмы — математический аппарат (http://www.basegroup.ru/genetic/math_print.htm)

Библиография

- Black, Paul E. *Gray code*. 25 февраля 2004. NIST. (http://www.nist.gov/dads/HTML/graycode.html) (англ.).

Ссылки

- NIST Dictionary of Algorithms and Data Structures: Gray code (http://www.nist.gov/dads/HTML/graycode.html) (англ.)
- Коды Грея (http://alglib.sources.ru/articles/graycode.php)

Теория сложности вычислений

В информатике, **теория сложности вычислений** является разделом теории вычислений, изучающим стоимость работы, требуемой для решения вычислительной проблемы. Стоимость обычно измеряется абстрактными понятиями времени и пространства, называемыми вычислительными ресурсами. Время определяется количеством элементарных шагов, необходимых для решения проблемы, тогда как пространство определяется объёмом памяти или места на носителе данных. Таким образом, в этой области предпринимается попытка ответить на центральный вопрос разработки алгоритмов: «как изменится время исполнения и объём занятой памяти в зависимости от размера входа и выхода?». Здесь под размером входа понимается длина описания данных задачи в битах (например, в задаче коммивояжера длина входа пропорциональна количеству городов и дорог между ними), а под размером выхода — длина описания решения задачи (оптимального маршрута в задаче коммивояжера).

В частности, теория сложности вычислений определяет NP-полные задачи, которые недетерминированная машина Тьюринга может решить за полиномиальное время, тогда как для детерминированной машины Тьюринга полиномиальный алгоритм неизвестен. Обычно это сложные проблемы оптимизации, например, задача коммивояжера.

Временная и пространственная сложности

Теория сложности вычислений возникла из потребности сравнивать быстродействие алгоритмов, чётко описывать их поведение (время исполнения и объём необходимой памяти) в зависимости от размера входа и выхода.

Количество элементарных операций, затраченных алгоритмом для решения конкретного экземпляра задачи, зависит не только от размера входных данных, но и от самих данных. Например, количество операций алгоритма сортировки вставками значительно меньше в случае, если входные данные уже отсортированы. Чтобы избежать подобных трудностей, рассматривают понятие временной сложности алгоритма *в худшем случае*.

Временная сложность алгоритма (в худшем случае) — это функция размера входных и выходных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера. В задачах, где размер выхода не превосходит или пропорционален размеру входа, можно рассматривать временную сложность как функцию размера только входных данных.

Аналогично понятию *временной сложности в худшем случае* определяется понятие *временная сложность алгоритма в наилучшем случае*. Также рассматривают понятие *среднее время работы алгоритма*, то есть математическое ожидание времени работы алгоритма. Иногда говорят просто: «*Временная сложность алгоритма*» или «*Время работы алгоритма*», имея в виду временную сложность алгоритма в худшем, наилучшем или среднем случае (в зависимости от контекста).

По аналогии с временной сложностью, определяют *пространственную сложность алгоритма*, только здесь говорят не о количестве элементарных операций, а об объёме используемой памяти.

Асимптотическая сложность

Несмотря на то, что функция временной сложности алгоритма в некоторых случаях может быть определена точно, в большинстве случаев искать точное её значение бессмысленно. Дело в том, что во-первых, точное значение временной сложности зависит от определения *элементарных операций* (например, сложность можно измерять в количестве арифметических операций, битовых операций или операций на машине Тьюринга), а во-вторых, при увеличении размера входных данных вклад постоянных множителей и слагаемых низших порядков, фигурирующих в выражении для точного времени работы, становится крайне незначительным.

Рассмотрение входных данных большого размера и оценка порядка роста времени работы алгоритма приводят к понятию **асимптотической сложности** алгоритма. При этом алгоритм с меньшей асимптотической сложностью является более эффективным для всех входных данных, за исключением лишь, возможно, данных малого размера. Для записи асимптотической сложности алгоритмов используются асимптотические обозначения:

Обозначение	Интуитивное объяснение	Определение
$f(n) \in O(g(n))$	f ограничена сверху функцией g (с точностью до постоянного множителя) асимптотически	$\exists (C > 0), n_0 : \forall (n > n_0)\ \lvert f(n)\rvert \leq \lvert Cg(n)\rvert$ или $\exists (C > 0), n_0 : \forall (n > n_0)\ f(n) \leq Cg(n)$
$f(n) \in \Omega(g(n))$	f ограничена снизу функцией g (с точностью до постоянного множителя) асимптотически	$\exists (C > 0), n_0 : \forall (n > n_0)\ \lvert Cg(n)\rvert \leq \lvert f(n)\rvert$
$f(n) \in \Theta(g(n))$	f ограничена снизу и сверху функцией g асимптотически	$\exists (C, C' > 0), n_0 : \forall (n > n_0)\ \lvert Cg(n)\rvert < \lvert f(n)\rvert < \lvert C'g(n)\rvert$
$f(n) \in o(g(n))$	g доминирует над f асимптотически	$\forall (C > 0), \exists n_0 : \forall (n > n_0)\ \lvert f(n)\rvert < \lvert Cg(n)\rvert$
$f(n) \in \omega(g(n))$	f доминирует над g асимптотически	$\forall (C > 0), \exists n_0 : \forall (n > n_0)\ \lvert Cg(n)\rvert < \lvert f(n)\rvert$
$f(n) \sim g(n)$	f эквивалентна g асимптотически	$\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = 1$

Примеры

- **«пропылесосить ковер»** требует время, линейно зависящее от его площади ($\Theta(A)$), то есть на ковер, площадь которого больше в два раза, уйдет в два раза больше времени. Соответственно, при увеличении площади ковра в сто тысяч раз, объем работы увеличивается строго пропорционально в сто тысяч раз, и т. п.
- **«найти имя в телефонной книге»** требует всего лишь время, логарифмически зависящее от количества записей ($O(\log_2(n))$), так как открыв книгу примерно в середине, мы уменьшаем размер «оставшейся проблемы» вдвое (за счет сортировки имен по алфавиту). Таким образом, в книге, толщиной в 1000 страниц, любое имя находится не больше чем за $\log_2 1000 \approx 10$раз (открываний книги). При увеличении объема страниц до ста тысяч, проблема все еще решается за $\log_2 100000 \approx 17$заходов. (См. Двоичный поиск.)

Замечания

Поскольку $\log_a b = \frac{log_c b}{log_c a}$, в асимптотической оценке сложности часто пишут «логарифм» без упоминания основания — например, $O(n \log n)$.

Необходимо подчеркнуть, что степень роста наихудшего времени выполнения — не единственный или самый важный критерий оценки алгоритмов и программ. Приведем несколько соображений, позволяющих посмотреть на критерий времени выполнения с других точек зрения:

Если создаваемая программа будет использована только несколько раз, тогда стоимость написания и отладки программы будет доминировать в общей стоимости программы, то есть фактическое время выполнения не окажет существенного влияния на общую стоимость. В этом случае следует предпочесть алгоритм, наиболее простой для реализации.

Если программа будет работать только с «малыми» входными данными, то степень роста времени выполнения будет иметь меньшее значение, чем константа, присутствующая в формуле времени выполнения[1] . Вместе с тем и понятие «малости» входных данных зависит от точного времени выполнения конкурирующих

алгоритмов. Существуют алгоритмы, такие как алгоритм целочисленного умножения, асимптотически самые эффективные, но которые никогда не используют на практике даже для больших задач, так как их константы пропорциональности значительно превосходят подобные константы других, более простых и менее «эффективных» алгоритмов. Другой пример — фибоначчиевы кучи, несмотря на асимптотическую эффективность, с практической точки зрения программная сложность реализации и большие значения констант в формулах времени работы делают их менее привлекательными, чем обычные бинарные пирамиды[1].

> *Если решение некоторой задачи для n-вершинного графа при одном алгоритме занимает время (число шагов) порядка n^C, а при другом — порядка n+n!/C, где C — постоянное число, то согласно «полиномиальной идеологии» первый алгоритм практически эффективен, а второй — нет, хотя, например, при $C=10^{(10^{10})}$ дело обстоит как раз наоборот.*[2]
>
> А. А. Зыков

Эффективные, но сложные алгоритмы могут быть нежелательными, если готовые программы будут поддерживать лица, не участвующие в написании этих программ.

Известны примеры, когда эффективные алгоритмы требуют таких больших объемов машинной памяти (без возможности использования более медленных внешних средств хранения), что этот фактор сводит на нет преимущество «эффективности» алгоритма. Таким образом, часто важна не только «сложность по времени», но и «сложность по памяти» (пространственная сложность).

В численных алгоритмах точность и устойчивость алгоритмов не менее важны, чем их временная эффективность.

Классы сложности

Класс сложности — это множество задач распознавания, для решения которых существуют алгоритмы, схожие по вычислительной сложности. Два важных представителя:

Класс P

Класс P вмещает все те проблемы, решение которых считается «быстрым», то есть полиномиально зависящим от размера входа. Сюда относится сортировка, поиск во множестве, выяснение связности графов и многие другие.

Класс NP

Класс NP содержит задачи, которые недетерминированная машина Тьюринга в состоянии решить за полиномиальное количество времени. Следует заметить, что недерминированная машина Тьюринга является лишь абстрактной моделью, в то время как современные компьютеры соответствуют детерминированной машине Тьюринга с ограниченной памятью. Таким образом, класс NP включает в себя класс P, а также некоторые проблемы, для решения которых известны лишь алгоритмы, экспоненциально зависящие от размера входа (т.е. неэффективные для больших входов). В класс NP входят многие знаменитые проблемы, такие как задача коммивояжёра, задача выполнимости булевых формул, факторизация и др.

Проблема равенства классов P и NP

Вопрос о равенстве этих двух классов считается одной из самых сложных открытых проблем в области теоретической информатики. Математический институт Клэя включил эту проблему в список проблем тысячелетия, предложив награду размером в один миллион долларов США за её решение.

Знаменитые учёные

- Стивен Кук
- Ричард Карп
- Леонид Левин
- Александр Разборов
- Ади Шамир

Примечания

[1] *Кормен, Томас Х.; Лейзерсон, Чарльз И.; Ривест, Рональд Л.; Штайн, Клифорд* Алгоритмы: построение и анализ, 2-е издание = Introduction to Algorithms second edition. — М.: «Вильямс», 2005. — ISBN 5-8459-0857-4

[2] *А. А. Зыков* Основы теории графов. — 3-е изд. — М.: Вузовская книга, 2004. — С. 10. — 664 с. — ISBN 5-9502-0057-8

См. также

- Класс сложности
- Алгоритм
- Аппроксимация
- Сведение (теория сложности вычислений)
- «О» большое и «о» малое

Ссылки

- Курсы по теории вычислимости и сложности вычислений (http://www.csin.ru/courses#computability)
- Юрий Лифшиц *« Современные задачи теоретической информатики* (http://yury.name/modern.html)». Курс лекций по алгоритмам для NP-трудных задач.
- *А. А. Разборов* Theoretical Computer Science: взгляд математика (http://offline.computerra.ru/print/offline/2001/379/6782/) // *Компьютерра*. — 2001. — № 2. (альтернативная ссылка (http://www.mi.ras.ru/~razborov/computerra.ps))
- *А. А. Разборов* О сложности вычислений (http://www.mccme.ru/free-books/matpros/i4127141.pdf.zip) // *Математическое просвещение*. — МЦНМО, 1999. — № 3. — С. 127-141.

Классы сложности

P • NP • co-NP • NP-C • co-NP-C • UP • #P • #P-C • L • NL • NC • P-C • PSPACE • PSPACE-C • EXPTIME • NEXPTIME • EXPSPACE • 2-EXPTIME • PR • RE • Co-RE • RE-C • Co-RE-C • R • BQP • BPP • RP • ZPP • PP • PCP • IP • PH

Класс сложности

В теории алгоритмов **классами сложности** называются множества вычислительных задач, примерно одинаковых по сложности вычисления. Говоря более узко, классы сложности — это множества предикатов (функций, получающих на вход слово и возвращающих ответ 0 или 1), использующих для вычисления примерно одинаковые количества ресурсов.

Для каждого класса существует категория задач, которые являются «самыми сложными». Это означает, что любая задача из класса сводится к такой задаче, и притом сама задача лежит в классе. Такие задачи называют **полными** задачами для данного класса. Наиболее известными являются NP-полные задачи.

Полные задачи — хороший инструмент для доказательства равенства классов. Достаточно для одной такой задачи предоставить алгоритм, решающий её и принадлежащий более маленькому классу, и равенство будет доказано.

Определение

Каждый класс сложности (в узком смысле) определяется как множество предикатов, обладающих некоторыми свойствами. Типичное определение класса сложности выглядит так:

> Классом сложности *X* называется множество предикатов *P(x)*, вычислимых на машинах Тьюринга и использующих для вычисления *O(f(n))* ресурса, где *n* — длина слова *x*.

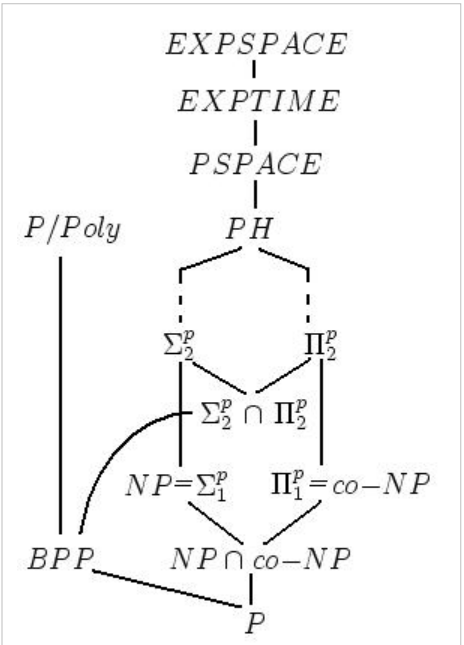
В качестве ресурсов обычно берутся время вычисления (количество рабочих тактов машины Тьюринга) или рабочая зона (количество использованных ячеек на ленте во время работы). Языки, распознаваемые предикатами из некоторого класса (то есть множества слов, на которых предикат возвращает 1), также называются принадлежащими тому же классу.

Кроме того, многие классы могут также быть описаны в терминах математической логики или теории игр.

Классы принято обозначать прописными буквами. Дополнение к классу *C* (то есть класс языков, дополнения которых принадлежат *C*) обозначается *co-C*.

Отношения между классами

Все классы сложности находятся в иерархическом отношении: одни включают в себя другие. Однако про большинство включений неизвестно, являются ли они строгими. Одна из наиболее известных открытых проблем в этой области — равенство классов P и NP. Если это предположение верно (в чём большинство учёных сомневается), то представленная справа иерархия классов сильно свернётся. На данный момент наиболее распространённой является гипотеза о невырожденности иерархии (то есть все классы различны). Кроме того, известно, что EXPSPACE не равен классу PSPACE.



Иерархия классов сложности

Иерархия по времени и иерархия по памяти

Рассмотрим функцию *f* и входную цепочку длиной *n*. Тогда класс DTIME*(f(n))* определяют как класс языков, принимаемых детерминированными машинами Тьюринга, заканчивающими свою работу за время, не превосходящее *f(n)*. Класс NTIME*(f(n))*, в свою очередь, определяют как класс языков, принимаемых недетерминированной машиной Тьюринга, заканчивающими свою работу за время, не превосходящее *f(n)*. Отметим, что ограничения на память при определении данных классов отсутствуют.

Аналогично иерархии по времени вводится иерархия по памяти. Класс DSPACE*(f(n))* обозначает класс языков, принимаемых детерминированными машинами Тьюринга, использующих не более *f(n)* ячеек памяти на рабочих лентах. Класс NSPACE*(f(n))* определяют как класс языков, принимаемых недетерминированными машинами Тьюринга, использующих не более *f(n)* ячеек памяти на рабочих лентах. Временные ограничения при определении данных классов отсутствуют.

Аналогично определяются и другие подобные рассмотренным выше классы. Приведем использующиеся сокращения:

- **D** - детерминированный (детерминистический)
- **N** - недетерминированный
- **R** - вероятностный с ограниченной односторонней ошибкой
- **B** - вероятностный с ограниченной двусторонней ошибкой
- **BQ** - квантовый с ограниченной двусторонней ошибкой

См. также

- AI-полная задача

Классы сложности

P • NP • co-NP • NP-C • co-NP-C • UP • #P • #P-C • L • NL • NC • P-C • PSPACE • PSPACE-C • EXPTIME • NEXPTIME • EXPSPACE • 2-EXPTIME • PR • RE • Co-RE • RE-C • Co-RE-C • R • BQP • BPP • RP • ZPP • PP • PCP • IP • PH

Литература

- *Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман* Введение в теорию автоматов, языков и вычислений = Introduction to Automata Theory, Languages, and Computation. — М.: «Вильямс», 2002. — С. 528. — ISBN 0-201-44124-1

Ссылки

- А.Китаев, А.Шень, М.Вялый. Классические и квантовые вычисления [1]

Примечания

[1] ftp://ftp.mccme.ru/users/vyalyi/qcomp/qbook.ps.gz

Класс P

В теории алгоритмов **классом P** (от англ. *polynomial*) называют множество задач, для которых существуют «быстрые» алгоритмы решения (время работы которых полиномиально зависит от размера входных данных). Класс P включён в более широкие классы сложности алгоритмов.

Определения

Формальное определение

Алгоритм отождествляется с детерминированной машиной Тьюринга, которая вычисляет ответ по данному на входную ленту слову из входного алфавита Σ. Временем работы алгоритма $T_M(x)$ при фиксированном входном слове *x* называется количество рабочих тактов машины Тьюринга от начала до остановки машины. Сложностью функции $f : \Sigma^* \to \Sigma^*$, вычисляемой некоторой машиной Тьюринга, называется функция $C : \mathbb{N} \to \mathbb{N}$, зависящая от длины входного слова и равная максимуму времени работы машины по всем входным словам фиксированной длины:

$$C_M(n) = \max_{x:|x|=n} T_M(x).$$

Если для функции *f* существует машина Тьюринга *M* такая, что $C_M(n) < n^c$ для некоторого числа *c* и достаточно больших *n*, то говорят, что она принадлежит классу P, или полиномиальна по времени.
Согласно тезису Чёрча — Тьюринга, любой мыслимый алгоритм можно реализовать на машине Тьюринга. Для любого языка программирования можно определить класс P подобным образом (заменив в определении машину Тьюринга на реализацию языка программирования). Если компилятор языка, на котором реализован алгоритм, замедляет исполнение алгоритма полиномиально (то есть время выполнения алгоритма на машине Тьюринга меньше некоторого многочлена от времени выполнения его на языке программирования), то определения классов P для этого языка и для машины Тьюринга совпадают. Код на ассемблере допускает преобразование в машину Тьюринга с небольшим полиномиальным замедлением, а поскольку все существующие языки допускают компиляцию в ассемблер (опять же, с полиномиальным замедлением), то определения класса P для машин Тьюринга и для любого существующего языка программирования совпадают.

Более узкое определение

Иногда под классом P имеют в виду более узкий класс функций, а именно класс предикатов (функций $f : \Sigma^* \to \{0, 1\}$). В таком случае **языком** *L*, который распознаёт данный предикат, называется множество слов, на которых предикат равен 1. Языками класса P называются языки, для которых существуют распознающие их предикаты класса P. Очевидно, что если языки L_1 и L_2 лежат в классе P, то и их объединение, пересечение и дополнения также лежат в классе P.

Включения класса P в другие классы

Класс P является одним из самых узких классов сложности. Алгоритмы, принадлежащие ему, принадлежат также классу NP, классу BPP (как допускающие полиномиальную реализацию с нулевой ошибкой), классу PSPACE (т.к. зона работы на машине Тьюринга всегда меньше времени), классу P/Poly (для доказательства этого факта используется понятие протокола работы машины, который переделывается в булеву схему полиномиального размера).

Уже более 30 лет остаётся нерешённой задача о равенстве классов P и NP. Если они равны, то любую задачу из класса NP можно будет решить быстро (за полиномиальное время). Однако научное сообщество склоняется в сторону отрицательного ответа на этот вопрос. Кроме того, не доказана и строгость включения в более

широкие классы, например, в PSPACE, но равенство P и PSPACE выглядит на данный момент очень сомнительно.

Примеры задач

Задачи, принадлежащие классу P

Примерами задач из класса P являются целочисленное сложение, умножение, деление, взятие остатка от деления, умножения матриц, выяснение связности графов и некоторые другие.

Задачи, принадлежность которых классу P неизвестна

Существует много задач, для которых не найдено полиномиального алгоритма, но не доказано, что его не существует. Соответственно, неизвестно, принадлежат ли такие задачи классу P. Вот некоторые из них:

1. Задача коммивояжёра (а также все остальные NP-полные задачи). Эта задача равносильна установлению равенства классов P и NP.
2. Разложение числа на простые множители.
3. Дискретное логарифмирование в конечном поле.
4. Задача о скрытой подгруппе с n образующими.
5. Дискретное логарифмирование в аддитивной группе точек на эллиптической кривой.

Практическое значение

Поскольку часто приходится вычислять значения функций на входных данных большого объёма, нахождение полиномиальных алгоритмов для вычисления функций является очень важной задачей. Считается, что вычислять функции, не лежащие в классе P, заметно сложнее, чем лежащие. Большинство алгоритмов, лежащих в классе P, имеют сложность, не превосходящую многочлен небольшой степени от размера входных данных. Например, стандартный алгоритм перемножения матриц требует n^3 умножений (хотя существуют и более быстрые алгоритмы, например, алгоритм Штрассена). Степень многочлена редко бывает большой. Один из таких случаев — предложенный в 2002 году индийскими математиками тест Агравала — Каяла — Саксены, выясняющий, является ли число n простым, за $O(\log^6 n)$ операций.

Ссылки

Классы сложности

P • NP • co-NP • NP-C • co-NP-C • UP • #P • #P-C • L • NL • NC • P-C • PSPACE • PSPACE-C • EXPTIME • NEXPTIME • EXPSPACE • 2-EXPTIME • PR • RE • Co-RE • RE-C • Co-RE-C • R • BQP • BPP • RP • ZPP • PP • PCP • IP • PH

Класс NP

В теории алгоритмов **классом NP** (от англ. *non-deterministic polynomial*) называют множество задач распознавания (англ.), решение которых при наличии некоторых дополнительных сведений (так называемого *сертификата решения*) можно «быстро» (за время, не превосходящее полинома от размера данных) проверить на машине Тьюринга.

Эквивалентно класс NP можно определить как содержащий задачи, которые можно «быстро» решить на недетерминированной машине Тьюринга.

Определения

Класс сложности NP определяется для множества **языков**, т.е. множеств слов над конечным алфавитом $\sum$. Язык *L* называется принадлежащим классу NP, если существуют двуместный предикат $R(x,\ y)$из класса P (т.е. вычислимый за полиномиальное время) и константа $c > 0$такие, что для всякого слова *x* условие «*x* принадлежит *L*» равносильно условию «найдётся *y* длины меньше $|x|^c$такой, что верно $R(x,\ y)$» (где |x| — длина слова *x*). Слово *y* называется **сертификатом** принадлежности *x* языку *L*. Таким образом, если у нас есть слово, принадлежащее языку, и ещё одно слово-свидетель ограниченной длины (которое бывает трудно найти), то мы быстро сможем удостовериться в том, что *x* действительно принадлежит *L*.

Эквивалентное определение можно получить, используя понятие недетерминированной машины Тьюринга (т. е. такой машины Тьюринга, у программы которой могут существовать разные строки с одинаковой левой частью). Если машина встретила «развилку», т. е. неоднозначность в программе, то дальше возможны разные варианты вычисления. Предикат $R(x)$, который представляет данная недетерминированная машина Тьюринга, считается равным единице, если существует хоть один вариант вычисления, возвращающий 1, и нулю, если все варианты возвращают 0. Если длина вычисления, дающего 1, не превосходит некоторого многочлена от длины *x*, то предикат называется принадлежащим классу NP. Если у языка существует распознающий его предикат из класса NP, то язык называется принадлежащим классу NP. Это определение эквивалентно приведённому выше: в качестве свидетеля можно взять номера нужных веток при развилках в вычислении. Так как для *x* принадлежащему языку длина всего пути вычисления не превосходит многочлена от длины *x*, то и длина свидетеля также будет ограничена многочленом от длины *x*.

Всякую задачу о принадлежности слова *x* языку *L*, лежащую в классе NP, можно решить за экспоненциальное время перебором всех возможных сертификатов длины меньше $|x|^c$.

Соотношение с другими классами

Класс языков, дополнения которых принадлежат NP, называется классом **co-NP**, хотя и не доказано, что этот класс отличен от класса NP. Пересечение классов NP и co-NP содержит класс P. В частности, класс NP включает в себя класс P. Однако ничего не известно о строгости этого включения.

Задача о равенстве классов P и NP является одной из центральных открытых проблем теории алгоритмов. Если они равны, то любую задачу из класса NP можно будет решить быстро (за полиномиальное время). Однако научное сообщество[*кто?*] склоняется в сторону отрицательного ответа на этот вопрос.

Класс NP включается в другие, более широкие классы, например, в класс PH. Существуют также открытые вопросы о строгости его включения в другие классы.

Примеры задач класса NP

Можно привести много задач, про которые на сегодняшний день неизвестно, принадлежат ли они P, но известно, что они принадлежат NP. Среди них:

- Задача выполнимости булевых формул: узнать по данной булевой формуле, существует ли набор входящих в неё переменных, обращающий её в 1. Сертификат — такой набор.
- Задача о клике: по данному графу узнать, есть ли в нём *клики* (полные подграфы) заданного размера. Сертификат — номера вершин, образующих клику.
- Определение наличия в графе гамильтонова цикла. Сертификат — последовательность вершин, образующих гамильтонов цикл.
- Задача о коммивояжёре — расширенный и более приближенный к реальности вариант предыдущей задачи.
- Существование целочисленного решения у заданной системы линейных неравенств. Сертификат — решение.

Среди всех задач класса NP можно выделить «самые сложные» — NP-полные задачи. Если удастся решить любую из них за полиномиальное время, то все задачи класса NP также можно будет решить за полиномиальное время.

Литература

- Задачи поиска. Классы P и NP. Сведения. NP-полные задачи [1]
- "Введение в структурную теорию сложности" [2]. Курс лекций.

Классы сложности

P • NP • co-NP • NP-C • co-NP-C • UP • #P • #P-C • L • NL • NC • P-C • PSPACE • PSPACE-C • EXPTIME • NEXPTIME • EXPSPACE • 2-EXPTIME • PR • RE • Co-RE • RE-C • Co-RE-C • R • BQP • BPP • RP • ZPP • PP • PCP • IP • PH

Примечания

[1] http://logic.pdmi.ras.ru/~hirsch/students/complexity1/lecture1.pdf

[2] http://www.csin.ru/courses/vvedenie-v-strukturnuyu-teoriyu-slozhnosti

Равенство классов P и NP

Задачи тысячелетия
Равенство классов **P** и **NP**
Гипотеза Ходжа
~~Гипотеза Пуанкаре~~
Гипотеза Римана
Квантовая теория Янга — Миллса
Существование и гладкость решений уравнений Навье — Стокса
Гипотеза Берча и Свиннертона-Дайера

В теории алгоритмов вопрос о **равенстве классов сложности *P* и *NP*** является одной из центральных открытых проблем уже более трех десятилетий. Если на него будет дан утвердительный ответ, это будет означать, что теоретически возможно решать многие сложные задачи существенно быстрее, чем сейчас.

Проблема равенства классов *P* и *NP* является одной из семи задач тысячелетия, за решение которой Математический институт Клэя назначил премию в миллион долларов США.

Классы P и NP

В конечном счете проблема $P = NP$ состоит в следующем: если положительный ответ на какой-то вопрос можно быстро *проверить* (за полиномиальное время), то правда ли, что ответ на этот вопрос можно быстро *найти* (за полиномиальное время и используя полиномиальную память)?

Неформально говоря, действительно ли решение задачи легче проверить, нежели отыскать?

Например, верно ли, что среди чисел $\{-2, -3, 15, 14, 7, -10, \ldots\}$ есть такие, что их сумма равна 0 (задача о суммах подмножеств)? Ответ *да*, потому что $-2 -3 + 15 -10 = 0$ легко проверяется несколькими сложениями (информация, необходимая для проверки положительного ответа, называется сертификатом). Следует ли отсюда, что так же легко подобрать эти числа? Проверить сертификат так же легко, как найти его? Кажется, что подобрать числа сложнее (не доказано).

Содержание проблемы

Отношения между классами *P* и *NP* рассматриваются в теории вычислительной сложности (разделе теории вычислений), изучающей ресурсы, необходимые для решения некоторой задачи. Наиболее общие ресурсы — это время (сколько нужно сделать шагов) и память (сколько памяти потребуется для решения задачи).

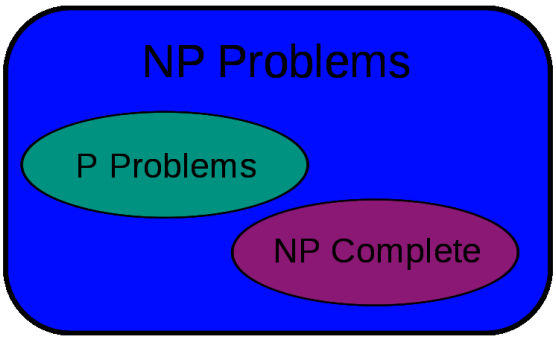


Диаграмма классов сложности при условии *P* ≠ *NP*.

История

Из определения классов *P* и *NP* сразу вытекает следствие: $P \subseteq NP$. Однако до сих пор ничего не известно о строгости этого включения, т. е. существует ли задача, лежащая в *NP*, но не лежащая в *P*. Если такой задачи не существует, то все задачи, принадлежащие классу *NP*, можно будет решать за полиномиальное время, что сулит огромную выгоду с вычислительной точки зрения. Сейчас самые сложные задачи из класса *NP* (так называемые *NP*-полные задачи) можно решить за экспоненциальное время, что почти всегда неприемлемо.

Впервые вопрос о равенстве классов был поставлен Стивеном Куком в 1971 году и независимо Леонидом Левиным в 1973 году.[1]

В настоящее время большинство математиков считают, что эти классы не равны. Согласно опросу, проведённому в 2002 году среди 100 учёных,[2] 61 человек считает, что ответ — «не равны», 9 — «равны», 22 затруднились ответить и 8 считают, что гипотеза не выводима из текущей системы аксиом и, таким образом, не может быть доказана или опровергнута.

Попытки доказательства

6 августа 2010 года[3] сотрудник исследовательской лаборатории Hewlett-Packard в Пало-Альто Винэй Деолаликар (англ.) разослал некоторым ученым на проверку своё доказательство неравенства P и NP.[4] Стивен Кук назвал его препринт «относительно серьезной попыткой решения проблемы P vs NP».[5] Однако уже в том же месяце были найдены недостатки в доказательстве. Деолаликар заявил, что в следующей версии доказательства он постарается учесть все замечания.[3] [6]

На викистранице «Deolalikar P vs NP paper», связанной с проектом Polymath, приводится критический анализ, собраны предполагаемые ошибки и некоторые опечатки в работе Деолаликара.[7] Там же можно проследить за онлайн-реакцией на предложенное доказательство.

См. также

- Открытые математические проблемы

Примечания

[1] Stephen Cook. The Importance of the P versus NP Question (http://www.cs.toronto.edu/~sacook/homepage/JACMpvsnp.ps).

[2] William I. Gasarch (2002). « The P=?NP poll. (http://www.cs.umd.edu/~gasarch/papers/poll.pdf)». *SIGACT News* **33** (2): 34–47. DOI: 10.1145/1052796.1052804 (http://dx.doi.org/10.1145/1052796.1052804).

[3] Lenta.ru — Мимо (http://lenta.ru/articles/2010/08/18/fail/)

[4] Vinay Deolakikar. P ≠ NP (http://www.hpl.hp.com/personal/Vinay_Deolalikar/Papers/pnp12pt.pdf). preprint.

[5] P ≠ NP- Greg and Kat's blog (http://gregbaker.ca/blog/2010/08/07/p-n-np/)

[6] The P≠NP «Proof» Is One Week Old (http://rjlipton.wordpress.com/2010/08/15/the-p≠np-proof-is-one-week-old/)

[7] Deolalikar P vs NP paper (http://michaelnielsen.org/polymath1/index.php?title=Deolalikar_P_vs_NP_paper) (англ.). michaelnielsen.org. Проверено 27 x12 2010.

Ссылки

- С. Кук Официальное описание проблемы (http://www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf) (англ.)
- С. Николенко «P=?NP» (http://www2.computerra.ru/xterra/253871/). Компьютерра, 2006.
- Н. П. Варновский. Проблема P =? NP, классы сложностей и криптография (http://satisfiability.narod.ru/files/www.cryptography.ru-1169817.html). 2005.
- *Притыкин Ю. Л.* Что такое проблема P vs. NP? (http://www.mccme.ru/dubna/2008/material.htm) // *VIII летняя школа «Современная математика»*. — Дубна: 2008.
- А. А. Разборов P ?= NP или проблема перебора: взгляд из 90-х (http://www.mi.ras.ru/~razborov/phasis.ps).
- *Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман* Введение в теорию автоматов, языков и вычислений = Introduction to Automata Theory, Languages, and Computation. — М.: «Вильямс», 2002. — С. 528. — ISBN 0-201-44124-1
- Gerhard J. Woeginger. The P-versus-NP page (http://www.win.tue.nl/~gwoegi/P-versus-NP.htm). (англ.) Список ссылок на предложенные «решения» данной проблемы. Некоторые из них утверждают равенство *P* и *NP*, некоторые — обратное.

NP-полная задача

В теории алгоритмов **NP-полная задача** — это такая задача из класса NP, к которой можно свести любую другую задачу из класса NP за полиномиальное время. Таким образом, NP-полные задачи образуют в некотором смысле подмножество «самых сложных» задач в классе NP; и если для какой-то из них будет найден «быстрый» алгоритм решения, то и любая другая задача из класса NP может быть решена так же «быстро».

Формальное определение

Алфавитом называется всякое конечное множество символов (например, $\{0,1\}$ или $\{a,b,c\}$).

Множество всех возможных **слов** (конечных строк, составленных из символов этого алфавита) над некоторым алфавитом $\sum$обозначается Σ^*. **Языком** L над алфавитом $\sum$называется всякое подмножество множества Σ^*, то есть $L \subset \Sigma^*$.

Задачей распознавания слова для языка L называется определение того, принадлежит ли данное **слово** языку L.

Язык L_1называется сводимым (по Карпу) к языку L_2, если существует функция, $f : \Sigma^* \to \Sigma^*$, вычислимая за полиномиальное время, и обладающая следующим свойством:

- $f(x) \in L_2$тогда и только тогда, когда $x \in L_1$.

Сводимость по Карпу обозначается как $L_1 \leq_p L_2$или $L_1 \propto L_2$.

Язык L_2называется **NP-трудным**, если любой язык из класса NP сводится к нему. Язык называют **NP-полным**, если он NP-труден, и при этом сам лежит в классе NP.

Таким образом, если будет найден алгоритм, решающий некоторую (любую) NP-полную задачу за полиномиальное время, то все NP-задачи окажутся в классе P, то есть будут решаться за полиномиальное время.

Гипотеза P ≠ NP

Совпадают ли классы P и NP уже более 30 лет является открытым вопросом. Научное сообщество склоняется к отрицательному ответу на этот вопрос — в этом случае за полиномиальное время решать NP-полные задачи не удастся.

Примеры NP-полных задач

- Задача о выполнимости булевых формул
- Кратчайшее решение «пятнашек» размера $n \times n$
- Задача коммивояжёра
- Проблема Штейнера
- Проблема раскраски графа
- Задача о вершинном покрытии
- Задача о покрытии множества
- Задача о клике
- Задача о независимом множестве
- Сапер (игра)
- Тетрис[1]

Примечания

[1] Erik D. Demaine, Susan Hohenberger, David Liben-Nowell. Tetris is Hard, Even to Approximate (http://arxiv.org/abs/cs.CC/0210020) (англ.). preprint.

Литература

- *Томас Х. Кормен и др.* **Глава 34. NP-полнота** // Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 0-07-013151-1

Ссылки

- NP-полнота (http://rain.ifmo.ru/cat/view.php/theory/algorithm-analysis/np-completeness-2004)
- Вычислительная сложность игр и головоломок (http://www.ics.uci.edu/~eppstein/cgt/hard.html) (англ.)

Классы сложности

P • NP • co-NP • NP-C • co-NP-C • UP • #P • #P-C • L • NL • NC • P-C • PSPACE • PSPACE-C • EXPTIME • NEXPTIME • EXPSPACE • 2-EXPTIME • PR • RE • Co-RE • RE-C • Co-RE-C • R • BQP • BPP • RP • ZPP • PP • PCP • IP • PH

Венгерский алгоритм

Венгерский алгоритм — алгоритм оптимизации, решающий задачу о назначениях за полиномиальное время (см. исследование операций). Он был разработан и опубликован Харолдом Куном в 1955 году. Автор дал ему имя «венгерский метод» в связи с тем, что алгоритм в значительной степени основан на более ранних работах двух венгерских математиков (Кёнига и Эгервари).

Джеймс Манкрес в 1957 году заметил, что алгоритм является (строго) полиномиальным. С этого времени алгоритм известен также как **алгоритм Куна — Манкреса** или **алгоритм Манкреса решения задачи о назначениях**. Временная сложность оригинального алгоритма была $O(n^4)$, однако Эдмондс и Карп (а также Томидзава независимо от них) показали, что его можно модифицировать так, чтобы достичь времени выполнения $O(n^3)$. Форд и Фалкерсон распространили метод на общие транспортные задачи. В 2006 году было обнаружено, что Якоби нашёл решение задачи о назначениях в XIX веке и опубликовал его в 1890 году на латыни[1].

Объяснение на примере

Предположим, есть три работника: Иван, Пётр и Андрей. Нужно распределить между ними выполнение трёх видов работ (которые мы назовём A, B, C), каждый работник должен выполнять только одну разновидность работ. Как это сделать так, чтобы потратить наименьшую сумму денег на оплату труда рабочих? Сначала необходимо построить матрицу стоимостей работ.

	A	B	C
Иван	1000 руб.	2000 руб.	3000 руб.
Пётр	3000 руб.	3000 руб.	3000 руб.
Андрей	3000 руб.	3000 руб.	2000 руб.

Венгерский алгоритм, применённый к приведённой выше таблице даст нам требуемое распределение: Иван выполняет работу A, Пётр — работу B, Андрей — работу C.

Постановка задачи

Дана неотрицательная матрица размера *n*×*n*, где элемент в *i*-й строке и *j*-ом столбце соответствует стоимости выполнения *j*-ого вида работ *i*-м работником. Нужно найти такое соответствие работ работникам, чтобы расходы на оплату труда были наименьшими. Если цель состоит в нахождении назначения с наибольшей стоимостью, то решение сводится к решению только что сформулированной задачи путём замены каждой стоимости *C* на разность между максимальной стоимостью и *C*.[2]

Основные идеи

Алгоритм основан на двух идеях:

- если из всех элементов некоей строки или столбца вычесть одно и то же число y, общая стоимость уменьшится на y, а оптимальное решение не изменится;
- если есть решение нулевой стоимости, оно оптимально.

Алгоритм ищет значения, которые надо вычесть из всех элементов каждой строки и каждого столбца (разные для разных строк и столбцов), такие что все элементы матрицы останутся неотрицательными, но появится нулевое решение.

Определения

Алгоритм проще описать, если сформулировать задачу, используя двудольный граф. Дан полный двудольный граф *G=(S, T; E)* с *n* вершинами, соответствующими работникам (*S*), и *n* вершинами, соответствующими видам работ (*T*); стоимость каждого ребра *c(i, j)* неотрицательна. Требуется найти совершенное, или полное паросочетание с наименьшей стоимостью.

Будем называть функцию $y : (S \cup T) \mapsto \mathbb{R}$ **потенциалом**, если $y(i) + y(j) \leq c(i,j)$ для каждых $i \in S, j \in T$. **Значение потенциала** *y* равно $\sum_{v \in S \cup T} y(v)$. Нетрудно заметить, что стоимость любого совершенного паросочетания не меньше, чем значение любого потенциала. Венгерский метод находит полное паросочетание и потенциал с одинаковой стоимостью/значением, что доказывает оптимальность обеих величин. Фактически он находит совершенное паросочетание **жёстких** рёбер: ребро *ij* называется жёстким для потенциала *y*, если $y(i) + y(j) = c(i,j)$. Подграф жёстких рёбер будем обозначать как G_y. Стоимость полного паросочетания в G_y(если оно существует) равна значению *y*.

Алгоритм в терминах двудольных графов

Алгоритм хранит в памяти потенциал y и ориентацию (задание направления) каждого жёсткого ребра, обладающую тем свойством, что рёбра, направленные от T к S образуют паросочетание M. Ориентированный граф, состоящий из жёстких рёбер с заданной ориентацией, мы обозначаем $\overrightarrow{G_y}$. Таким образом, в любой момент есть три типа рёбер:

- нежёсткие (и не принадлежащие M)
- жёсткие, но не принадлежащие M
- жёсткие и принадлежащие M

Изначально y везде равно 0, и все рёбра направлены от S к T (таким образом, M пусто). На каждом шаге или модифицируется y так, что увеличивается множество вершин Z, определённое в следующем абзаце, или изменяется ориентация, чтобы получить паросочетание с большим количеством рёбер; всегда остаётся верным, что все рёбра из M являются жёсткими. Процесс заканчивается, если M — совершенное паросочетание.

Пусть на каждом шаге $R_S \subseteq S$ и $R_T \subseteq T$ составляют множество вершин, не инцидентных рёбрам из M (то есть R_S — множество вершин из S, в которые не входит ни одно ребро, а R_T — множество вершин из T, из которых не исходит ни одно ребро). Обозначим через Z множество вершин, достижимых из R_S в $\overrightarrow{G_y}$ (оно может быть найдено поиском в ширину).

Если $R_T \cap Z$ не является пустым, это значит, что есть хотя бы один путь в $\overrightarrow{G_y}$ из R_S в R_T. Выбираем любой из таких путей, и изменяем ориентацию всех его ребёр на обратную. Размер паросочетания увеличится на 1.

Для доказательства отметим два очевидных факта:

- На выбранном пути рёбра из S в T чередуются с рёбрами из T в S. Это следует из двудольности графа.
- Первая вершина пути принадлежит S, а последняя — T. Следовательно, первое и последнее его рёбра направлены из S в T.

По определению $\overrightarrow{G_y}$, отсюда следует, что на выбранном пути рёбра, принадлежащие и не принадлежащие M чередуются, причём первое и последнее рёбра не принадлежат M, то есть путь является повышающим, откуда и следует доказываемое утверждение.

Если $R_T \cap Z$ пусто, положим $\Delta := \min\{c(i,j) - y(i) - y(j) : i \in Z \cap S, j \in T \setminus Z\}$. Δ положительна, потому что нет жёстких рёбер между $Z \cap S$ и $T \setminus Z$.

В самом деле, пускай такое ребро (i, j) есть. Поскольку $i \in Z$, но $j \notin Z$, вершина i достижима из R_S в $\overrightarrow{G_y}$, а вершина j недостижима. Следовательно, $\overrightarrow{G_y}$ не может содержать ребра (i, j). Следовательно, $(i,j) \in M$, откуда $i \notin R_S$. Поскольку i достижима из R_S в $\overrightarrow{G_y}$, существует путь в i из какой-то вершины, принадлежащей R_S. Рассмотрим последнее ребро этого пути. Обозначим его (k, i). Поскольку k достижима из R_S в $\overrightarrow{G_y}$, а j недостижима, $k \neq j$. Поскольку $(k,i) \in \overrightarrow{G_y}$, $(i,k) \in M$. Следовательно, i инцидентна сразу двум рёбрам из M: (i,j) и (i,k), что невозможно, так как M — паросочетание. Противоречие.

Увеличим y на Δ на вершинах из $Z \cap S$ и уменьшим y на Δ на вершинах, входящих в $Z \cap T$. Новый y остаётся потенциалом.

В самом деле, для любого ребра (i, j), $i \in S$, $j \in T$:

- если $i \notin Z$, $j \notin Z$, то c(i, j)-y(i)-y(j) не меняется, потому что не меняются ни y(i), ни y(j)

- если $i \in Z$, $j \in Z$, то c(i, j)-y(i)-y(j) не меняется, потому что y(i) увеличивается на Δ, а y(j) на столько же уменьшается
- если $i \notin Z$, $j \in Z$, разность c(i, j)-y(i)-y(j) увеличивается на Δ, следовательно, остаётся неотрицательной
- если $i \in Z$, $j \notin Z$, разность c(i, j)-y(i)-y(j) уменьшается на Δ, но всё равно остаётся неотрицательной, потому что Δ — наименьшая из таких разностей.

Граф $\overrightarrow{G_y}$ меняется, но, несмотря на это, содержит M.

В самом деле, чтобы исключить из $\overrightarrow{G_y}$ некое ребро (i, j), $i \in S$, $j \in T$, надо сделать его нежёстким, то есть повысить разность c(i, j)-y(i)-y(j). Как мы видели, разность повышается только если $i \notin Z$, $j \in Z$, иными словами, i недостижима из R_S, а j достижима. Но в таком случае ребро (j, i) не может принадлежать $\overrightarrow{G_y}$, следовательно, ребро не принадлежит M.

Ориентируем новые рёбра от S к T. По определению Δ, множество Z вершин, достижимых из R_S, увеличится (при этом число жёстких рёбер вовсе не обязательно возрастёт).

Для доказательства этого утверждения сначала докажем, что ни одна вершина не пропадёт из Z. Пусть $V \in Z$. Тогда существует путь из некоей вершины, принадлежащей R_S, в V. Все вершины на этом пути достижимы из R_S, то есть принадлежат Z. Каждое рёбро на этом пути инцидентно двум вершинам из Z. Как мы видели выше, для таких рёбер разность c(i, j)-y(i)-y(j) не меняется. Значит, все рёбра пути останутся жёсткими, и V по-прежнему будет достижима из R_S. Теперь докажем, что хотя бы одна вершина добавится к Z. Рассмотрим ребро, на котором достигается минимум $\min\{c(i,j) - y(i) - y(j) : i \in Z \cap S, j \in T \setminus Z\}$. Для этого ребра, разность c(i, j)-y(i)-y(j) обнулится, следовательно, оно станет жёстким и будет направлено из S в T, то есть от i к j. Поскольку $i \in Z$, j также станет достижимым из R_S, то есть добавится к Z.

Повторяем эти шаги до тех пор, пока M не станет совершенным паросочетанием; в этом случае оно даёт назначение с наименьшей стоимостью. Время выполнения этой версии алгоритма равно $O(n^4)$: M дополняется n раз, а в стадии, когда M не меняется, может быть не более n изменений потенциала (так как Z увеличивается каждый раз). Время, необходимое на изменение потенциала, равно $O(n^2)$.

Матричная интерпретация

Для n работников и работ, дана матрица *n*×*n*, задающая стоимость выполнения каждой работы каждым работником. Найти минимальную стоимость выполнения работ, такую что каждый работник выполняет ровно одну работу, а каждую работу выполняет ровно один работник.

В дальнейшем мы под **назначением** понимаем соответствие между работниками и работами, имеющее нулевую стоимость, после того как мы произвели трансформации, влияющие лишь на общую стоимость работ.

Прежде всего запишем задачу в матричной форме:

$$\begin{bmatrix} a1 & a2 & a3 & a4 \\ b1 & b2 & b3 & b4 \\ c1 & c2 & c3 & c4 \\ d1 & d2 & d3 & d4 \end{bmatrix}$$

где a, b, c, d — работники, которые должны выполнить работы 1, 2, 3, 4. Коэффициенты a1, a2, a3, a4 обозначают стоимость выполнения работником «a» работ 1, 2, 3, 4 соответственно. Аналогичный смысл имеют остальные символы. Матрица квадратная, поэтому каждый работник может выполнить только одну работу.

Шаг 1

Уменьшаем элементы построчно. Находим наименьший из элементов первой строки (a1, a2, a3, a4), и вычитаем его из всех элементов первой строки. При этом хотя бы один из элементов первой строки обнулится.

То же самое выполняем и для всех остальных строк. Теперь в каждой строке матрицы есть хотя бы один ноль. Иногда нулей уже достаточно, чтобы найти назначение. Пример показан в таблице. Красные нули обозначают назначенные работы.

0	a2'	0	a4'
b1'	b2'	b3'	0
0	c2'	c3'	c4'
d1'	0	d3'	d4'

При большом количестве нулей для поиска назначения (нулевой стоимости) можно использовать алгоритм нахождения максимального паросочетания двудольных графов, например алгоритм Хопкрофта-Карпа. Кроме того, если хотя бы в одном столбце нет нулевых элементов, то назначение невозможно.

Шаг 2

Часто на первом шаге нет назначения, как, например, в следующем случае:

0	a2'	a3'	a4'
b1'	b2'	b3'	0
0	c2'	c3'	c4'
d1'	0	d3'	d4'

Задача 1 может быть эффективно (за нулевую стоимость) выполнена как работником a, так и работником c, зато задача 3 не может быть эффективно выполнена никем.

В таких случаях мы повторяем шаг 1 для столбцов и вновь проверяем, возможно ли назначение.

Шаг 3

Во многих случаях мы достигнем желаемого результата уже после шага 2. Но иногда это не так, например:

0	a2'	a3'	a4'
b1'	b2'	b3'	0
0	c2'	c3'	c4'
d1'	0	0	d4'

Если работник d выполняет работу 2, некому выполнять работу 3, и наоборот.

В таких случаях мы выполняем процедуру, описанную ниже.

Сначала, используя любой алгоритм поиска максимального паросочетания в двудольном графе, назначаем как можно больше работ тем работникам, которые могут их выполнить за нулевую стоимость. Пример показан в таблице, назначенные работы выделены красным.

0	a2'	a3'	a4'
b1'	b2'	b3'	0
0	c2'	c3'	c4'
d1'	0	0	d4'

Отметим все строки без назначений (строка 1). Отметим все столбцы с нулями в этих строках (столбец 1). Отметим все строки с нулями в этих столбцах (строка 3). Продолжаем, пока новые строки и столбцы не перестали отмечаться.

×				
0	a2'	a3'	a4'	×
b1'	b2'	b3'	0	
0	c2'	c3'	c4'	×
d1'	0	0	d4'	

Теперь проводим линии через все отмеченные столбцы и неотмеченные строки.

×				
0	a2'	a3'	a4'	×
b1'	b2'	b3'	0	
0	c2'	c3'	c4'	×
d1'	0	0	d4'	

Все эти действия преследовали лишь одну цель: провести наименьшее количество линий (вертикалей и горизонталей), чтобы покрыть все красные нули. Можно было воспользоваться любым другим методом вместо описанного.

Шаг 4

Из непокрытых линиями элементов матрицы (в данном случае это a2', a3', a4', c2', c3', c4') найти наименьший. Вычесть его из всех не отмеченных строк и прибавить ко всем пересечениям отмеченных строк и столбцов. Так, например, если наименьший элемент из перечисленных равен a2', мы получим

×				
0	0	a3'-a2'	a4'-a2'	×
b1'+a2'	b2'	b3'	0	
0	c2'-a2'	c3'-a2'	c4'-a2'	×
d1'+a2'	0	0	d4'	

Повторять процедуру (шаги 1-4) до тех пор, пока назначение не станет возможным.

Библиография

- R.E. Burkard, M. Dell'Amico, S. Martello: *Assignment Problems*. SIAM, Philadelphia (PA.) 2009. ISBN 978-0-898716-63-4
- Harold W. Kuhn, «The Hungarian Method for the assignment problem», *Naval Research Logistics Quarterly*, **2**:83—97, 1955. Kuhn's original publication.
- Harold W. Kuhn, «Variants of the Hungarian method for assignment problems», *Naval Research Logistics Quarterly*, **3**: 253—258, 1956.
- J. Munkres, «Algorithms for the Assignment and Transportation Problems», *Journal of the Society for Industrial and Applied Mathematics*, **5**(1):32—38, 1957 March.
- M. Fischetti, «Lezioni di Ricerca Operativa», Edizioni Libreria Progetto Padova, Italia, 1995.
- R. Ahuja, T. Magnanti, J. Orlin, «Network Flows», Prentice Hall, 1993.

Примечания

[1] politechnique.fr (http://www.lix.politechnique.fr/~olliver/JACOBI/jacobiEngl.htm)

[2] http://www.ams.jhu.edu/~castello/362/Handouts/hungarian.pdf

Ссылки

Реализации

Не все перечисленные ниже реализации удовлетворяют ограничению времени исполнения $O(n^3)$.

- Реализация на языке Python (http://bmc.github.com/munkres/)
- Короткая реализация на языке Python (http://www.enseignement.polytechnique.fr/informatique/INF441/INF441b/code/kuhnMunkres.py)
- Реализация на Ruby с модульными тестами (http://github.com/evansenter/gene/blob/f515fd73cb9d6a22b4d4b146d70b6c2ec6a5125b/objects/extensions/hungarian.rb)
- Реализация на C# (http://noldorin.com/blog/2009/09/hungarian-algorithm-in-csharp/)
- Интерактивная онлайн-реализация (http://www.ifors.ms.unimelb.edu.au/tutorial/hungarian/welcome_frame.html) Реализует вариант алгоритма.
- Graphical implementation with options (http://web.axelero.hu/szilardandras/gaps.html) (Java-апплет)
- Последовательные and параллельные реализации. (http://www.netlib.org/utk/lsi/pcwLSI/text/node220.html)
- Используются Matlab и C (http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=6543)
- Реализация на Perl (http://search.cpan.org/src/ANAGHAKK/Algorithm-Munkres-0.06/lib/Algorithm/Munkres.pm)
- Реализация на C++ (http://saebyn.info/2007/05/22/munkres-code-v2/)
- Другая реализация на C++ с модульными тестами (http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=hungarianAlgorithm)
- более короткая реализация на C++, метод добавления строк (http://acm.mipt.ru/twiki/bin/view/Algorithms/HungarianAlgorithmCPP)
- Реализация на Java (GPLv3) (http://sites.google.com/site/garybaker/hungarian-algorithm/assignment)
- Другая реализация на Java с тестами JUnit (Apache 2.0) (http://timefinder.svn.sourceforge.net/viewvc/timefinder/trunk/timefinder-algo/src/main/java/de/timefinder/algo/roomassignment/)

Сведение (теория сложности вычислений)

В теории сложности вычислений **сведе́ние** — преобразование одной задачи к другой. В общем случае, если у нас есть алгоритм, преобразующий экземпляры задачи P_1 в экземпляры задачи P_2, которые имеют тот же ответ (да/нет), то говорят, что P_1 **сводится к** P_2. Таким образом, **сводимость** — это отношение между двумя задачами. С помощью такой связи могут быть доказаны вычислимость задачи или ее принадлежность тому или иному классу сложности.

Некоторые виды сведений

- Сведение по Куку
- Сведение по Карпу
- Сведение по Левину
- Сведение по Тьюрингу

Самым общим типом сведения является Сведение по Тьюрингу. В данном случае некоторый алгоритм (вычислимый на машине Тьюринга) может быть вызван любое количество раз, при этом каждый вызов будет считаться за один шаг алгоритма. Для формального определения сводимости по Тьюрингу используется понятие Тьюринг-машины с оракулом.

Ссылки

- Курс «Введение в структурную теорию сложности» [2]

Литература

- *Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман* Введение в теорию автоматов, языков и вычислений = Introduction to Automata Theory, Languages, and Computation. — М.: «Вильямс», 2002. — С. 528. — ISBN 0-201-44124-1

Задача выполнимости булевых формул

Зада́ча выполни́мости бу́левых фо́рмул (SAT или ВЫП) — важная для теории вычислительной сложности алгоритмическая задача.

Экземпляром задачи **SAT** является булева формула, состоящая только из имен переменных, скобок и операций $\wedge$ (*И*), $\vee$ (*ИЛИ*) и $\neg$ (*НЕ*). Задача заключается в следующем: можно ли назначить всем переменным, встречающимся в формуле, значения *ложь* и *истина* так, чтобы формула стала истинной.

Согласно теореме Кука, доказанной Стивеном Куком в 1971-м году, задача SAT для булевых формул, записанных в конъюнктивной нормальной форме, является NP-полной. Требование о записи в конъюнктивной форме существенно, так как, например, задача SAT для формул, представленных в дизъюнктивной нормальной форме, тривиально решается за линейное время от размера записи формулы.

Точная формулировка

Чтобы четко сформулировать задачу распознавания, необходимо условиться об алфавите, с помощью которого задаются экземпляры языка. Этот алфавит должен быть фиксирован и конечен. В своей книге Хопкрофт, Мотвани и Ульман предлагают использовать следующий алфавит: {« $\wedge$ », « $\vee$ », « $\neg$ », « $($ », « $)$ », « x », « 0 », « 1 »}.

При использовании такого алфавита скобки и операторы записываются естественным образом, а переменные получают следующие имена: x1, x10, x11, x100 и т. д., согласно их номерам, записанным в двоичной системе счисления.

Пусть некоторая булева формула, записанная в обычной математической нотации, имела длину N символов. В ней каждое вхождение каждой переменной было описано хотя бы одним символом, следовательно, всего в данной формуле не более N переменных. Значит, в предложенной выше нотации каждая переменная будет записана с помощью $O(\log N)$ символов. В таком случае, вся формула в новой нотации будет иметь длину $O(N \log N)$ символов, то есть длина строки возрастет в полиномиальное число раз.

Например, формула $a \wedge \neg(b \vee c)$ примет вид $x1 \wedge \neg(x10 \vee x11)$.

Вычислительная сложность

В 1971-м году в статье Стивена Кука был впервые введен термин «NP-полная задача», и задача SAT была первой задачей, для которой доказывалось это свойство.

В доказательстве теоремы Кука каждая задача из класса NP в явном виде сводится к SAT. После появления результатов Кука была доказана NP-полнота для множества других задач. При этом чаще всего для доказательства NP-полноты некоторой задачи приводится полиномиальное сведение задачи SAT к данной задаче, возможно в несколько шагов, то есть с использованием нескольких промежуточных задач.

Частные случаи задачи SAT

Интересными важными частными случаями задачи SAT являются:

- Задача выполнимости булевых формул в конъюнктивной нормальной форме (SATCNF или ВКНФ) — аналогичная задача, с наложенной на формулу условием: она должна быть записана в конъюнктивной нормальной форме. Задача ВКНФ также NP-полна.
- Задача выполнимости булевых формул в k-конъюнктивной нормальной форме (k-SAT или k-ВЫП) — задача выполнимости при условии, что формула записана в k-конъюнктивной нормальной форме. Эта задача является NP-полной при $k \geq 3$.
- Задача выполнимости булевых формул в 2-конъюнктивной нормальной форме имеет полиномиальное решение, то есть принадлежит классу P.

См. также

- Решатель
- Satisfiability Modulo Theories

Ссылки

- 2.5-ВЫПОЛНИМОСТЬ (вопросы сведения 3-SAT к 2-SAT) [1]
- The international SAT Competitions web page [2]
- SATLIB - The Satisfiability Library [3]
- Sat Live [4] - общий сайт о SAT.

Примечания

[1] http://report.vtsnet.ru
[2] http://www.satcompetition.org/
[3] http://www.satlib.org/
[4] http://www.satlive.org/

Задача коммивояжёра

Задача коммивояжёра (англ. *Travelling salesman problem*) (коммивояжёр — разъездной сбытовой посредник) является одной из самых известных задач комбинаторной оптимизации. Задача заключается в отыскании самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и т. п.) и соответствующие матрицы расстояний, стоимости и т. п. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов.

Существует несколько частных случаев общей постановки задачи, в частности геометрическая задача коммивояжёра (также называемая планарной или евклидовой, когда матрица расстояний отражает расстояния между точками на плоскости), треугольная задача коммивояжёра (когда на матрице стоимостей выполняется неравенство треугольника), симметричная и асимметричная задачи коммивояжёра. Также существует обобщение задачи, так называемая обобщённая задача коммивояжёра.

Общая постановка задачи, впрочем как и большинство её частных случаев, относится к классу NP-сложных задач.

Методы решения

Простейшие

- полный перебор
- случайный перебор
- жадные алгоритмы
  - метод ближайшего соседа
  - метод включения ближайшего города
  - метод самого дешёвого включения
- метод минимального остовного дерева
- метод имитации отжига

Все эффективные (сокращающие полный перебор) методы решения задачи коммивояжёра — методы эвристические. В большинстве эвристических методов находится не самый эффективный маршрут, а приближённое решение. Зачастую востребованы так называемые any-time алгоритмы, то есть постепенно улучшающие некоторое текущее приближенное решение.

Существует также постановки, в которых задача коммивояжера становится NP-полной. Обычно такие постановки выглядят следующим образом: существует ли на заданном графе G такой обход, что его стоимость не превышает x. Часто на ней проводят обкатку новых подходов к эвристическому сокращению полного перебора.

На практике применяются различные модификации более эффективных методов: метод ветвей и границ и метод генетических алгоритмов, а также алгоритм муравьиной колонии.

Метод ветвей и границ

Для решения задачи коммивояжёра предложен в 1963 году группой авторов (Дж. Литл, К. Мурти, Д. Суини, К.Кэрол). [1]

Литература

- *Ананий В. Левитин* **Глава 3. Метод грубой силы: Задача коммивояжера //** Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Algorithms. — М.: «Вильямс», 2006. — С. 159-160. — ISBN 0-201-74395-7
- *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
- *В.И. Мудров* **Задача о коммивояжере**. — М.: «Знание», 1969. — С. 62.

Примечания

[1] Костевич Л.С. Математическое программирование: Информ. технологии оптимальных решений: Учеб. пособие / Л.С. Костевич. — Мн.: Новое знание, 2003. ил., стр. 150, ISBN 985-6516-83-8

См. также

- Граф (математика)
- Оптимизация
- Комбинаторика
- Исследование операций
- Математическое программирование
- Дискретная математика
- Транспортная логистика
- Проблема Штейнера

Ссылки

- Решение задачи коммивояжёра на Java (http://sites.google.com/site/indy256/algo/dynamic_tsp)

Источники и основные авторы

Формальный язык *Источник*: http://ru.wikipedia.org/w/index.php?oldid=27632380 *Редакторы*: A.I., Abyr, Algo, Atr2006, BoBaH, D06osipov, Dmitru, Loveless, Mackseem, Myer, Peni, PlatypeanArchcow, Sbos, Tosha, Zzerox, РобоСтася, Хацкер, 7 анонимных правок

Конечный автомат *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30773605 *Редакторы*: A5b, Av.starlink.ru, Avosco, EdwardHades, Evgeny.berkovich, Fractaler, Ilana, Kemm, Lazil, Leseul, Maximaximax, Mospehraict, Nzeemin, Pk boot, PlatypeanArchcow, Ps ttf, Shamin Roman, Sivanov87, Structor, Suisui, Uieoa, Veikia, Vertaler, Viz, Vizu, Wasu64, Zenixan, Адель Чепкунов, Алеко, Виктор Штонда, Иван Поздеев, Игоревич, Конст. Карасёв, Хацкер, 41 анонимных правок

Регулярное множество *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29020836 *Редакторы*: Avosco, Ayvengo21, Dims, G0rn, Lazil, Nzeemin, Рунова Ольга, 7 анонимных правок

Теорема Клини *Источник*: http://ru.wikipedia.org/w/index.php?oldid=22577828 *Редакторы*: Aleksandrit, Mhym, Obersachse, Ps ttf, Rasim, SergeyRyzhikov, Посторонний, Рунова Ольга, 2 анонимных правок

Граф (математика) *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30800900 *Редакторы*: Abyr, Acrobat, Agor153, Aldanur, Alex Kapranoff, Alexandrov, Amoses, Andronniy, AntonR, Arachn0, Art-Roman, Avosco, Bepa, BeteTest, Captain, DaKick, Dark-oneof, Dims, EmbargEr, Enzet, FBI, Fdsc, Fibonachi, Fractaler, Incnis Mrsi, Infovarius, Irina Gelbukh, Krassotkin, LaserJet, Les, Leshabirukov, Lew Wadoo, Lvova, MRids, Maestro, ManN, Mashiah Davidson, Maxim Razin, NovaCxarmulo, Ob ivan, Obakeneko, Oleg R, Patro.macro, Phoenix, Prijutme4ty, Qkowlew, Ramir, Rmn, Sir sigurd, Snch, Sovetnik, Swappp, Tosha, Unomano, Viz, Volodymyr Obrizan, Wesha, Woo, X7q, Ysoldak, Андрей Куликов, Сорокин, 89 анонимных правок

Теория графов *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30771538 *Редакторы*: A.I., Abyr, Agor153, AlexAvramenko, Alexandrov, Amoses, Andrew Stankevich, AntonR, Artemka373, Avosco, Baz-Wiki, Baz.77.243.99.32, Cveto4ek001, D1gital love, Evatutin, Exlex, Fractaler, Incnis Mrsi, Jaro.p, Jinma, Kitaets, Kobanov, Kuzmaka, Leshabirukov, Loveless, Marcuch, Maximaximax, Mhym, Mir76, Mkp, NickShaforostoff, Nickst, RainbowUnicorn, Ramir, Rogalik, Sergey539, Shlakoblock, Singa, Splux, Tim2, Toto, Tyahti, UNV, Vasiliev Mihail, VladimirReshetnikov, Woo, X7q, XJamRastafire, Ysoldak, ВиКо, Денис Рыбкин, Хацкер, 58 анонимных правок

Словарь терминов теории графов *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30772055 *Редакторы*: AKA MBG, Ahtoh, Amoses, Anton Khorev, Avosco, Azh7, Bandar Lego, Bepa, BeteTest, Buenasdiaz, Csman, Dims, Drugov, ESSch, Evatutin, FHen, Fat763, Flym, Fractaler, George142, Glaue2dk, Gribozavr, Incnis Mrsi, LEMeZza, Les, Leshabirukov, Mashiah Davidson, Mikhail Dvorkin, Niketas, NovaCxarmulo, Nutus, Pascal65536, Peni, Pnzrusher, Prijutme4ty, Ramir, SashokA, Sim3331621, Tauraloke, Termar, Tim2, Torin, Tosha, Vguzev, Woo, X7q, Андрей Матвеякин, Дмитрий Джус, Кыс, Николай Васильев, Цуканов Кирилл, 79 анонимных правок

Ориентированный граф *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30263655 *Редакторы*: Amoses, Artem Korzhimanov, Bosik GN, DonAvero, Greck, Halyavin, Leshabirukov, Mackseem, Maestro, Michaello, Ramir, Voropz, X7q, Ботильда, 12 анонимных правок

Полный граф *Источник*: http://ru.wikipedia.org/w/index.php?oldid=27704373 *Редакторы*: Anton Khorev, Infovarius, 1 анонимных правок

Задача о клике *Источник*: http://ru.wikipedia.org/w/index.php?oldid=27565396 *Редакторы*: Andrew Stankevich, FelixTheRat, Fractaler, Maxal, Mousy, Prijutme4ty, Tyahti, X7q, Юлия Безрукова, 6 анонимных правок

Разрез графа *Источник*: http://ru.wikipedia.org/w/index.php?oldid=26776575 *Редакторы*: Abyr, Melancholic, Mhym, Oyster, Peni, Siriuz, Голем, 6 анонимных правок

Шарнир (теория графов) *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30210002 *Редакторы*: Nutus, 1 анонимных правок

Дерево (теория графов) *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30785786 *Редакторы*: AVB, Burivykh, Consigliere, D1gital love, Demdim, Dimanovs, Evatutin, FHen, Fractaler, Greck, Halyavin, Ivanhoe, Maksim-e, Maximaximax, Pk boot, Rokur, Serebr, Termar, Tigga ru, Tookser, Toothless, Tosha, Tutaishy, Volkov, X7q, Алеко, Даматрий, Дмитрий Джус, Леонтьев Владимир (тот самый), РобоСтася, Сорокин, Цуканов Кирилл, 47 анонимных правок

Теорема Кэли о числе деревьев *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30164442 *Редакторы*: Burivykh

Остовы графа *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30547151 *Редакторы*: ЕкатеринаРумия, Сергей Мальков

Алгоритм Шеннона — Фано *Источник*: http://ru.wikipedia.org/w/index.php?oldid=23315760 *Редакторы*: AVB, Andre Engels, BPK, Berillium, Dstary, Infovarius, Kcmamu, Korvin2050, Nickst, OckhamTheFox, Structor, Анна Белобратова, 19 анонимных правок

Алгоритм Флойда — Уоршелла *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29691119 *Редакторы*: Andre Engels, AntonR, Evatutin, KleverI, Leper Messiah, Loveless, Mercury, Peni, Pozd, Sergey539, VoidLess, Wacok, X7q, Всеволод Шорин, Дмитрий Джус, 21 анонимных правок

Паросочетание *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30355299 *Редакторы*: Demdim

Двудольный граф *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29386831 *Редакторы*: Fractaler, Koterpillar, Peni, X7q, РобоСтася, 5 анонимных правок

Трансверсаль *Источник*: http://ru.wikipedia.org/w/index.php?oldid=17964375 *Редакторы*: LEMeZza

Эйлеров цикл *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30700655 *Редакторы*: AVB, Claymore, Evatutin, Exprmntr, FearChild, Fractaler, G0rn, Ilya Voyager, Infovarius, Leper Messiah, Loveless, NBS, NRozina, Nickst, NovaCxarmulo, Termar, X7q, YuSh, Выползень, Голем, 21 анонимных правок

Гамильтонов граф *Источник*: http://ru.wikipedia.org/w/index.php?oldid=24843546 *Редакторы*: Aikr, Fractaler, Holop, Infovarius, Lockal, Loveless, Maxal, NBS, NovaCxarmulo, Sivanov87, SkliF, Toto, User№101, YuSh, Zorgit, Выползень, Хацкер, Четыре тильды, 5 анонимных правок

Поиск с возвратом *Источник*: http://ru.wikipedia.org/w/index.php?oldid=28460263 *Редакторы*: Dnikitin, Pvshnik, 1 анонимных правок

Гиперкуб *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30523990 *Редакторы*: Anyart, Crazy-EyE, Dark Bob, Dipsy, Errandir, Freezeman, Hzioulquoigmnzhah, In digma, Infovarius, Insuranze, Kalan, Knight Of Math, Laim, Lipedia, Lopk, Maksim-e, Maximaximax, Mmv, Pentusha, Rathori, RussianSpy, Stephen dedalus, Tosha, Wassily, Zorgit, ·1e0nid·, Акутагава, Александр Сигачёв, Викарий, Землемер, Канопус Киля, Переход Артур, Юкатан, 26 анонимных правок

Код Грея *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30320099 *Редакторы*: 4RESTER, AdmiralHood, Alatar, Dstary, INSAR, Jommer, Keeper B, Loveless, ManN, Manicon, Mercury, Peni, Suic, Голем, 23 анонимных правок

Теория сложности вычислений *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29903719 *Редакторы*: AVB, Alik Kirillovich, AsixSPb, Avnichkasov, BSoD, Clockwork, Dimka77777, Dstary, Exceeder, Futurum26, Golddim, Grebenkov, Ilana, MDA, Malek, Maxal, Mercury, Peni, Qwertic, Sergey539, Shooter0313, Sigwald, Tim2, VictorAnyakin, Yurylifshits, Хацкер, 19 анонимных правок

Класс сложности *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29522254 *Редакторы*: 4th-otaku, A.I., Abyr, Algo, AntonR, AsixSPb, Denton, Ilya (uk), Maxal, Maximaximax, Peni, Siriuz, 4 анонимных правок

Класс P *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29905032 *Редакторы*: Abyr, Algo, Fractaler, Halyavin, Kulagin, Maxal, Moudrick, Nashev, Stassats, The Wrong Man, Voroninv, КОСТАС, 7 анонимных правок

Класс NP *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29901930 *Редакторы*: Abyr, Kulagin, Loveless, Maxal, Prijutme4ty, UNV, X7q, Конст. Карасёв, Юлия Безрукова, 4 анонимных правок

Равенство классов P и NP *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30609329 *Редакторы*: Abyr, AntonR, AsixSPb, Badger M., Black Gold, DarkCherry, Denis BSU, Dmalinovsky, Dnikitin, EvgenyGenkin, Fractaler, Hance, Lexicon, Loveless, Maksim-e, Malek, Maxal, Nikov, Nxx, Peni, Piatlin-Pel, Rasim, Tosha, Toyota Dream House, V1adis1av, Vald, Vanuan, Volodia Spb, X7q, Zy, Всезнайка, Цуканов Кирилл, 52 анонимных правок

NP-полная задача *Источник*: http://ru.wikipedia.org/w/index.php?oldid=29853223 *Редакторы*: 4th-otaku, A.I., Abyr, Alexeytuzhilin, AndyVolykhov, AntonR, DarkCherry, Fractaler, Hwangroman, Maxal, Maximaximax, Nashev, Sasha l, Sizikov Anton, Stassats, VladimirReshetnikov, Хацкер, 12 анонимных правок

Венгерский алгоритм *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30379899 *Редакторы*: Fractaler, Halyavin, NatashaMilkova, Sergey539, Tetromino, Urod, Wikiwide, Четыре тильды, ﾑ, 1 анонимных правок

Сведение (теория сложности вычислений) *Источник*: http://ru.wikipedia.org/w/index.php?oldid=27639132 *Редакторы*: A.I., AntonR, AsixSPb, Malek, Rasim, Юлия Безрукова, 1 анонимных правок

Задача выполнимости булевых формул *Источник*: http://ru.wikipedia.org/w/index.php?oldid=24618191 *Редакторы*: BSoD, Brunner, Fractaler, Loveless, Malek, Mikhail Dvorkin, Mutilin, Peni, Sl, Structor, Urod, X7q, Юлия Безрукова, 2 анонимных правок

Задача коммивояжёра *Источник*: http://ru.wikipedia.org/w/index.php?oldid=30777193 *Редакторы*: Alex Kapranoff, Alexeytuzhilin, Amoses, Andre Engels, AntonR, Cryptomancer, Daniel Karapetyan, Drakosh, Fractaler, Halyavin, Indy256, Maximaximax, Metkere, Sergey539, Wald, Zoom82, Александр Сигачёв, Оркрист, Юлия Безрукова, 18 анонимных правок

Источники, лицензии и редакторы изображений

Файл:ДКА.jpg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:ДКА.jpg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Участник:Абдурашитова Роза Наильевна

Файл:НКА_c_e.jpg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:НКА_c_e.jpg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Panther, Абдурашитова Роза Наильевна

Файл:НКА_без_e.jpg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:НКА_без_e.jpg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Участник:Абдурашитова Роза Наильевна

Файл:редукция ребра.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Редукция_ребра.png *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Рунова Ольга

Файл:редукция вершины.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Редукция_вершины.png *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Рунова Ольга

Файл:теор2.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Теор2.png *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Рунова Ольга

Файл:Sample graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Sample_graph.svg *Лицензия*: GNU Free Documentation License *Редакторы*: User:Acrobat

Файл:Undirected.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Undirected.svg *Лицензия*: Public Domain *Редакторы*: Josette, Kilom691, 1 анонимных правок

Файл:Directed.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Directed.svg *Лицензия*: Public Domain *Редакторы*: Grafite, Josette, 1 анонимных правок

Файл:6n-graf.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:6n-graf.svg *Лицензия*: Public Domain *Редакторы*: User:AzaToth

Файл:3node digraph0.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph0.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph1.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph1.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph21.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph21.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph31.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph31.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph41.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph41.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph5.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph5.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph6.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph6.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph22.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph22.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph32.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph32.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph42.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph42.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph23.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph23.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph33.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph33.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph43.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph43.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph24.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph24.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph34.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph34.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:3node digraph44.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:3node_digraph44.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Leshabirukov

Файл:Divisors 12.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Divisors_12.svg *Лицензия*: GNU Free Documentation License *Редакторы*: User:KleinKlio

Изображение:Complete graph K1.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K1.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:Complete graph K2.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K2.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:Complete graph K3.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K3.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:Complete graph K4.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K4.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:Complete graph K5.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K5.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:Complete graph K6.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K6.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:Complete graph K7.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K7.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:Complete graph K8.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_graph_K8.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Image:6n-graf-clique.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:6n-graf-clique.svg *Лицензия*: Public Domain *Редакторы*: Dcoetzee, Erin Silversmith, GeorgHH, 1 анонимных правок

Image:Sample_graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Sample_graph.svg *Лицензия*: GNU Free Documentation License *Редакторы*: User:Acrobat

Image:Cayley's formula 2-4.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Cayley's_formula_2-4.svg *Лицензия*: неизвестно *Редакторы*: User:Rocchini

Файл:Дерево Шеннона.PNG *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Дерево_Шеннона.PNG *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Анна Белобратова

Изображение:Maximal-matching.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Maximal-matching.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: User:Miym

Изображение:Simple-bipartite-graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Simple-bipartite-graph.svg *Лицензия*: Public Domain *Редакторы*: MistWiz

Изображение:Complete bipartite graph K3,2.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complete_bipartite_graph_K3,2.svg *Лицензия*: Public Domain *Редакторы*: User:Dbenbenn

Изображение:RecursiveEvenBipartite.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:RecursiveEvenBipartite.svg *Лицензия*: GNU Free Documentation License *Редакторы*: User:Melchoir

Файл:konigsburg graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Konigsburg_graph.svg *Лицензия*: GNU Free Documentation License *Редакторы*: AnonMoos, Riojajar, Squizzz

Файл:Labelled Eulergraph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Labelled_Eulergraph.svg *Лицензия*: Public Domain *Редакторы*: User:S Sepp

Файл:Hamiltonian_Dodecahedron_Graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Hamiltonian_Dodecahedron_Graph.svg *Лицензия*: Creative Commons Attribution-Sharealike 3.0 *Редакторы*: Zorgit

Файл:Точка.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Точка.svg *Лицензия*: Public Domain *Редакторы*: Alex Spade, Insuranze

Файл:Отрезок.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Отрезок.png *Лицензия*: неизвестно *Редакторы*: Alex Spade, Dims, KR

Файл:Квадрат.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Квадрат.png *Лицензия*: неизвестно *Редакторы*: Alex Spade, Dims, KR

Файл:Куб.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Куб.png *Лицензия*: неизвестно *Редакторы*: Alex Spade, Dims, KR

Файл:Hypercubecentral.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Hypercubecentral.svg *Лицензия*: Public Domain *Редакторы*: User:Lipedia

Файл:Penteract.jpg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Penteract.jpg *Лицензия*: Creative Commons Attribution 2.5 *Редакторы*: Hzioulquoigmnzhah. Original uploader was Hzioulquoigmnzhah at ru.wikipedia

Файл:Hexeract graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Hexeract_graph.svg *Лицензия*: Creative Commons Attribution 2.5 *Редакторы*: Claudio Rocchini

Файл:Hepteract graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Hepteract_graph.svg *Лицензия*: Creative Commons Attribution 2.5 *Редакторы*: Claudio Rocchini

Файл:Octeract Petrie polygon.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Octeract_Petrie_polygon.svg *Лицензия*: Creative Commons Attribution 3.0 *Редакторы*: User:Lipedia

Файл:Enneract graph.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Enneract_graph.svg *Лицензия*: Creative Commons Attribution 2.5 *Редакторы*: Claudio Rocchini

Файл:10cube ortho polygon.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:10cube_ortho_polygon.svg *Лицензия*: Creative Commons Attribution 3.0 *Редакторы*: User:Rocchini

Файл:US02632058 Gray.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:US02632058_Gray.png *Лицензия*: Public Domain *Редакторы*: Original uploader was Dicklyon at en.wikipedia

Файл:Encoder Disc (3-Bit).svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Encoder_Disc_(3-Bit).svg *Лицензия*: Public Domain *Редакторы*: jjbeard

Файл:Aquote1.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Aquote1.png *Лицензия*: Public Domain *Редакторы*: Paul Pogonyshev, Rhanyeia, Wst

Файл:Aquote2.png *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Aquote2.png *Лицензия*: Public Domain *Редакторы*: Berillium, Paul Pogonyshev, Rhanyeia, Schekinov Alexey Victorovich, Wst, 5 анонимных правок

Изображение:Complexity classes tree.JPG *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complexity_classes_tree.JPG *Лицензия*: Public Domain *Редакторы*: Участник:Abyr

Файл:Complexity classes.svg *Источник*: http://ru.wikipedia.org/w/index.php?title=Файл:Complexity_classes.svg *Лицензия*: Public Domain *Редакторы*: Booyabazooka, Mdd, Mike1024, 1 анонимных правок

Лицензия

Creative Commons Attribution-Share Alike 3.0 Unported
http://creativecommons.org/licenses/by-sa/3.0/