Теория Графов

Alexander Lazarev

Institute of Control Sciences of Russian Academy of Sciences

2009-2010 учебный год

Outline

Элементы теории графов

Задача о кёнигсбергских мостах. Однажды великому математику Леонарду Эйлеру был задан вопрос: можно ли обойти все семь мостов, стоявших тогда в городе Кёнигсберге (современный Калининград, Россия), побывав на каждом по одному разу?

Рассмотрев эту задачу, в 1736 году Эйлер доказал, что это невозможно, причем он рассмотрел более общую задачу: какие местности, разделенные рукавами рек и соединенные мостами, возможно обойти, побывав на каждом мосту ровно один раз, а какие невозможно.

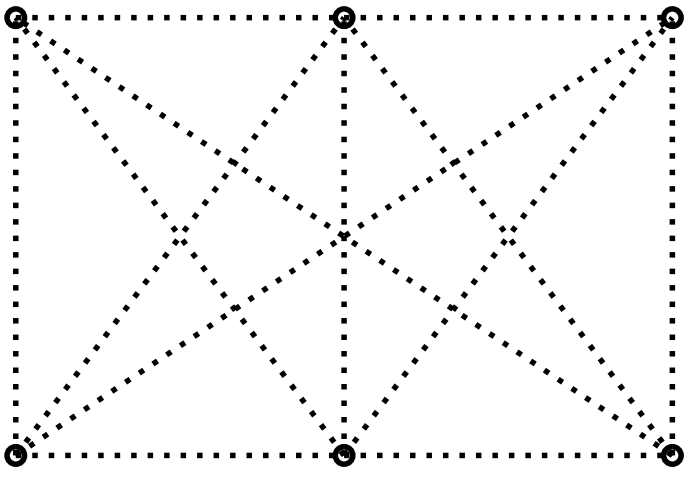
Проблема четырех красок, впервые поставленная перед математиками Де Морганом около 1850 г.
Выяснить, можно ли всякую расположенную на сфере карту раскрасить четырьмя красками так, чтобы любые две области, имеющие общий участок границы, были раскрашены в разные цвета.

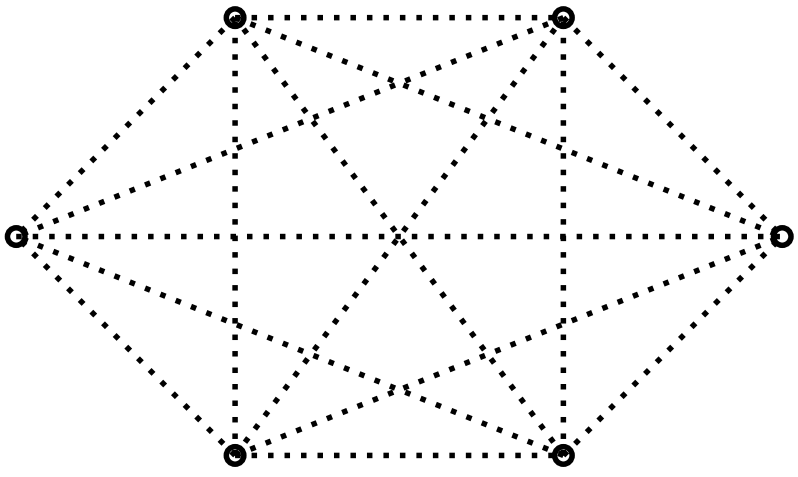
К. Аппель и В. Хакен доказали в 1976 г., что так можно раскрасить любую карту. Это была первая крупная математическая теорема, для доказательства которой был применён компьютер. Несмотря на последующие упрощения, доказательство практически невозможно проверить, не используя компьютер. Поэтому некоторые математики отнеслись к этому доказательству с недоверием, что объяснялось не только использованием компьютера, но и громоздкостью описания алгоритма первых доказательств (741 страница), впоследствии были предложены более компактные алгоритмы и скорректирован ряд ошибок. Проблема четырех красок является одним из известнейших прецедентов неклассического доказательства в современной математике.

Граф $G =< V, E >$ есть совокупность множества вершин V и множества рёбер (дуг) , причем $E \subseteq V \times V$ есть множество упорядоченных пар $< x, y >$ для ориентированного графа и $E \subseteq \{\{x, y\} : (x, y \in V)\&(x \neq y)\}$ для неориентированного графа (при этом для неориентированного графа считаем, что ребра $\{a, b\}$ и $\{b, a\}$ совпадают $\{a, b\} = \{b, a\}$).
Граф неориентированный, если все его ребра не ориентированы, и граф ориентированный, если все его ребра ориентированы.

Ребро ориентированного графа мы обозначаем символом $< x, y >$; ребро неориентированного графа обозначается символом $\{x, y\}$. В случае, когда несущественно, о каком ребре идет речь, или когда это ясно из контекста, будем произвольное ребро графа обозначать символом (x, y).
Говорят, что ребро $e \in E$ инцидентно вершинам x и $y, x, y \in V$, а так же, что x и y инцидентны ребру e.
Вершины x и y графа смежны, если (x, y) является ребром.
Два ребра смежны, если они имеют общую вершину.
Вершина, не инцидентная никакому ребру, называется изолированной.

Два графа $G =< V, E >$ и $G' =< V', E' >$ изоморфны, если существует такое взаимно-однозначное соответствие между множествами их вершин V и V', что вершины соединены ребрами в одном графе тогда и только тогда, когда соответствующие им вершины соединены в другом графе. Если $V \Leftrightarrow V'$ и при этом соответствии $x \Leftrightarrow x', y \Leftrightarrow y'$, то $(x, y) \in E$ тогда и только тогда, когда $(x', y') \in E' \Leftrightarrow ((x, y) \in E \Leftrightarrow (x', y') \in E')$.





Степени вершин

Пусть G – неориентированный граф. Число $\rho(x)$ ребер, инцидентных вершине x, называется локальной степенью, или просто степенью вершины x графа G. В каждом случае должно быть указано, считается петля однократной или двойной. Пусть $\rho(x, y) = \rho(y, x)$ – число ребер, соединяющих вершины x и y.

$$\rho(x) = \sum_{y \in V} \rho(x, y)$$

Обозначим через $n_e = n_e(G)$ число ребер в неориентированном графе G. Так как каждое ребро учитывается в двух степенях в вершинах x и y, то

$$2n_e = \sum_{x \in V} \rho(x).$$

Формула остается справедливой и при наличии петель, если только в локальных степенях вершин считать их дважды.

$$2n_e = \sum_{x,y \in V} \rho(x, y).$$

Теорема 3.1. В конечном графе число вершин нечетной степени четно.

О машинном представлении графов

В теории графов классическим способом представления графа служит матрица инциденций. Для графа $G =< V, E >$ – это матрица $I(G)$ с n строками, соответствующими вершинам, $|V| = n$, и с m столбцами, $|E| = m$, соответствующими ребрам. Для ориентированного графа столбец, соответствующий ребру $< x, y >$ содержит -1 в строке, соответствующей вершине x, $+1$ в строке, соответствующей вершине y, и нули во всех остальных строках (петлю $< x, x >$ иногда представляют значением 2). В случае неориентированного графа столбец, соответствующий ребру $\{x, y\}$, содержит 1 в строках, соответствующих x и y, и нули в остальных строках.

С алгоритмической точки зрения матрица инциденций является, вероятно, самым худшим способом представления графа, который только можно представить.
Требуется $m \times n$ ячеек памяти. Доступ неудобен. Ответ на элементарные вопросы типа:
"существует ли ребро $< x, y >$?";
"к каким вершинам ведут ребра из x?"
требует перебора всех столбцов матрицы.

Более удобным способом представление графа является матрица смежности (вершин), определяемая как матрица $B = ||b_{ij}||$ размера $n \times n$, где $b_{ij} = 1$, если существует ребро, ведущее из вершины i в вершину j, $b_{ij} = 0$ — в противном случае. Здесь мы подразумеваем, что ребро $\{i, j\}$ неориентированного графа идет как от i к j, так и от j к i, так что матрица смежности такого графа всегда является симметричной. Ответ на вопрос типа:
"$\{x, y\} \in E$?"может быть получен за один шаг.
Недостаток такого способа представления графа — $n \times n$ ячеек занятой памяти независимо от числа ребер.

Более экономным в отношении требуемого объема памяти (особенно для неплотных графов $m \ll n$) является метод представления графа с помощью списка пар. Пара (x, y) соответствует ребру $< x, y >$, если граф ориентированный, и ребру $\{x, y\}$, если граф неориентированный. Очевидно, что объем памяти в этом случае составляет $2m$ ячеек.
Неудобство — большое число шагов, порядка m в худшем случае, необходимое для получения множества вершин, к которым ведут ребра из данной вершины.

Ситуацию можно значительно улучшить, упорядочив множество пар лексикографически и применяя двоичный поиск, но лучшим решением во многих случаях оказывается структура данных, которая называется списками инцидентности.
Она содержит для каждой вершины $v \in V$ список вершин u, таких что $< v, u > \in E$ (или $\{v, u\} \in E$ в случае неориентированного графа).

Поиск в графе

Будем рассматривать ориентированные и неориентированные графы без петель и кратных ребер, которые будем называть простыми. Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина просматривается в точности один раз. Поэтому важной задачей является нахождение хороших методов поиска в графе. Вообще говоря, метод поиска "хороший":

1) он позволяет легко применить этот метод в алгоритме решения задачи ("погрузить" алгоритм решения нашей задачи в этот метод);

2) каждое ребро графа анализируется не более одного раза (или, что существенно не меняет ситуации, число раз, ограниченное константой, не зависящей от $|V|$ и $|E|$).

Поиск в глубину в графе

Мы начинаем поиск с некоторой фиксированной вершины v_0. Затем выбираем произвольную вершину u, смежную с $v_0, (v_0, u) \in E$, и повторяем процесс от u. В общем случае предположим, что мы находимся в вершине v. Если существует новая (еще "непросмотренная") вершина $u, (v, u) \in E$, то мы рассматриваем эту вершину (она перестает быть новой) и, начиная с нее, продолжаем поиск. Если же не существует ни одной новой вершины, смежной с v, то мы говорим, что вершина v использована, возвращаемся в вершину, из которой мы попали в v, и продолжаем процесс (если $v = v_0$, то поиск закончен). Таким образом, поиск в глубину из вершины v основывается на поиске в глубину из всех новых вершин, смежных с v.

```
1 procedure ПОИСК В ГЛУБИНУ В ГРАФЕ G(v)
  {поиск в глубину из вершины v; переменные НОВЫЙ,
  ЗАПИСЬ – глобальные}
2 begin рассмотреть v; НОВЫЙ[v] := ложь;
3    for u ∈ ЗАПИСЬ[v] do
4       if НОВЫЙ[u] then ПОИСК В ГЛУБИНУ В ГРАФЕ G(u)
5 end. {вершина v использована}
```

Покажем теперь, что этот алгоритм просматривает каждую вершину в точности один раз и его сложность порядка $O(n+m)$. Вызов ПОИСК В ГЛУБИНУ В ГРАФЕ $G(v)$ влечет за собой просмотр всех вершин связной компоненты графа, содержащей v (если НОВЫЙ$[u]$ = истина для каждой вершины u этой компоненты). Это непосредственно следует из структуры процедуры ПОИСК В ГЛУБИНУ В ГРАФЕ $G(v)$: после посещения вершины (строка 2) следует вызов процедуры для всех ее новых соседей. Каждая вершина графа просматривается не более одного раза, так как просматриваться может только вершина v, для которой НОВЫЙ$[v]$ = истина, сразу же после посещения этой вершины исполняется присваивание НОВЫЙ$[v]$:= ложь (строка 2 процедуры).

Алгоритм начинает поиск поочередно от каждой еще не просмотренной вершины, следовательно, просматриваются все вершины графа (необязательно связного).
Чтобы оценить сложность алгоритма, отметим сначала, что число шагов в обоих циклах (строки 2 и 3 алгоритма) порядка n, не считая шагов, выполнение которых инициировано вызовом процедуры ПОИСК В ГЛУБИНУ В ГРАФЕ G. Эта процедура выполняется не более n раз во втором цикле сразу после посещения каждой вершины для каждого из ее новых соседей, итого суммарно $O(n+m)$ раз. Полное число шагов, выполняемое циклом в строке 3 процедуры ПОИСК В ГЛУБИНУ В ГРАФЕ G, для всех вызовов этой процедуры будет порядка m, где m – число ребер. Это дает общую сложность алгоритма $O(n+m)$.

В связи с тем, что поиск в глубину играет важную роль в проектировании алгоритмов на графах, представляет интерес нерекурсивная версия процедуры ПОИСК В ГЛУБИНУ В ГРАФЕ G. Рекурсия устраняется стандартным способом при помощи стека. Каждая просмотренная вершина помещается в стек и удаляется из стека после ее использования.

Поиск в ширину в графе

Отметим, что при поиске в глубину чем позднее будет посещена вершина, тем раньше она будет использована. Это прямое следствие того факта, что просмотренные, но еще не использованные вершины накапливаются в стеке. Поиск в ширину основывается на замене стека очередью. После такой модификации чем раньше посещается вершина (помещается в очередь), тем раньше она используется (удаляется из очереди). Использование вершины происходит с помощью просмотра всех еще непросмотренных соседей этой вершины.

```
1  procedure ПОИСК В ШИРИНУ В ГРАФЕ G(v);
   {поиск в ширину в графе с началом в вершине v;
   переменные НОВЫЙ, СПИСОК – глобальные}
2  begin
3     ОЧЕРЕДЬ := ∅; ОЧЕРЕДЬ ⇐ v; НОВЫЙ [v] := ложь;
4     while ОЧЕРЕДЬ ≠ ∅ do
5        begin p ⇐ ОЧЕРЕДЬ; посетить p;
6           for u ∈ ЗАПИСЬ [p] do
7              if НОВЫЙ [u] then
8                 begin ОЧЕРЕДЬ ⇐ u; НОВЫЙ [ u ] := ложь
9                 end
10       end
11 end. {вершина v использована}
```

Способом, аналогичным тому, который был применен для поиска в глубину, можно легко показать, что вызов процедуры ПОИСК В ШИРИНУ В ГРАФЕ $G(v)$ приводит к посещению всех вершин связной компоненты графа, содержащей вершину v, причем каждая вершина просматривается в точности один раз. Вычислительная сложность алгоритма поиска в ширину также имеет порядок $O(m+n)$, так как каждая вершина (всего n вершин) помещается в очередь и удаляется из очереди в точности один раз, а число итераций цикла 6, очевидно, будет иметь порядок числа ребер (всего m ребер) графа.

Пути и циклы

Путем в ориентированном или неориентированном графе $G =< V, E >$ называют последовательность ребер вида $< (v_1, v_2), (v_2, v_3), ...(v_{n-1}, v_n) >= S =< e_1, ..., e_{n-1} >$, где $e_i = (v_i, v_{i+1}) \in E$, e_i и e_{i+1} инцидентны одной вершине, $i = \overline{1, n-1}$. Говорят, что этот путь идет из v_1 в v_n и имеет длину $n-1$. Часто такой путь представляют последовательностью вершин $< v_1, ..., v_n >, < v_i, v_{i+1} > \in E$, лежащих на нем. В вырожденном случае одна вершина обозначает путь длины 0, идущий из этой вершины в нее же. Путь называется простым, если все ребра и все вершины на нем, кроме быть может первой и последней, различны.

Цикл — это простой путь длины не менее 1, который начинается и заканчивается в одной и той же вершине. В неориентированном простом графе длина цикла должна быть не менее 3.

Оба вида поиска в графе – в глубину и в ширину – могут быть использованы для нахождения пути между фиксированными вершинами v и u. Достаточно начать поиск в графе с вершины v и вести его до момента посещения вершины u.

Преимуществом поиска в глубину является тот факт, что в момент посещения вершины u стек содержит последовательность вершин, определяющих путь из v в u. Каждая вершина, помещаемая в стек, является смежной с верхним элементом стека. Недостатком поиска в глубину является то, что полученный таким образом путь в общем случае не будет кратчайшим путем из v в u.

```
1  procedure ПОИСК В ШИРИНУ В ГРАФЕ G(v);
   {поиск в ширину в графе с началом в вершине v;
   переменные НОВЫЙ, СПИСОК – глобальные}
2  begin
3     ОЧЕРЕДЬ := ∅; ОЧЕРЕДЬ ⇐ v; НОВЫЙ [v] := ложь;
4     while ОЧЕРЕДЬ ≠ ∅ do
5        begin p ⇐ ОЧЕРЕДЬ; посетить p;
6           for u ∈ ЗАПИСЬ [p] do
7              if НОВЫЙ [u] then
8                 begin ОЧЕРЕДЬ ⇐ u; НОВЫЙ [ u ] := ложь;
8'                   ПРЕДЫДУЩИЙ [u ] :=p
9                 end
10       end
11 end. {вершина v использована}
```

По окончании работы модифицированной процедуры ПОИСК В ШИРИНУ В ГРАФЕ массив ПРЕДЫДУЩИЙ содержит для каждой просмотренной вершины u вершину ПРЕДЫДУЩИЙ[u], из которой мы попали в u. Кратчайший путь из исходной вершины v до каждой вершины из массива ПРЕДЫДУЩИЙ.

Связность

Пусть граф G — неориентированный. Две вершины a и b называются связанными, если существует путь S с начальной вершиной a и конечной вершиной $b, S =< a, a_1, ..., a_n, b >$. Если S проходит через какую-нибудь вершину a_i более одного раза, то можно, очевидно, удалить его циклический участок и при этом остающиеся ребра будут составлять путь S' из a в b. Отсюда следует, что связанные путем вершины связаны и простым путем. Граф называется связным, если любая его пара вершин связана.

Для всякого графа существует такое разбиение множества его вершин

$$V = \bigcup_{i=1}^{k} V_i$$

на попарно непересекающиеся подмножества вершин V_i, что вершины в каждом V_i связаны, а вершины из различных V_i не связаны. Граф состоит из k непересекающихся связных подграфов — k компонентов связанности.

Теорема 3.2. Если в конечном неориентированном простом графе G ровно две вершины a и b имеют нечетную локальную степень, то они связаны.

Доказательство. По теореме 3.1, каждый конечный граф имеет четное число вершин нечетной степени. Так как это условие выполняется и для той компоненты G, которой принадлежит a, то b принадлежит той же компоненте связности.

Теорема 3.3. Если неориентированный простой граф G имеет n вершин и k связных компонент, то максимальное число ребер в G равно

$$N(n,k) = \frac{1}{2}(n-k)(n-k+1).$$

Доказательство. Пусть в графе G связная компонента G_i имеет n_i вершин. Тогда максимальное число ребер в G равно

$$N(n,k) = \frac{1}{2}\sum_{i=1}^{k}(n_i)(n_i-1).$$

Это число достигается, когда каждый из графов G_i полный и имеет n_i вершин. Допустим, что среди графов G_i найдутся хотя бы два графа G_1, G_2, которые имеют более одной вершины, например $n_2 \geq n_1 > 1$. Построим вместо G другой граф G' с тем же числом вершин и компонент, заменяя G_1 и G_2 полными графами G_1' и G_2' соответственно с $n_1 - 1$ и $n_2 + 1$ вершинами. Общее количество вершин не изменилось.

$$\frac{1}{2}[(n_1)(n_1-1)+(n_2)(n_2-1)] \qquad G$$

$$\frac{1}{2}[(n_1-1)(n_1-2)+(n_2+1)(n_2)] \qquad G'$$

$$\frac{1}{2}[n_1^2-n_1+n_2^2-n_2] \qquad G$$

$$\frac{1}{2}[n_1^2-3n_1+2+n_2^2+n_2] \qquad G'$$

Число ребер увеличилось на $(n_2-n_1)+1>0$.

Таким образом максимальное число ребер должен иметь граф, состоящий из $k-1$ изолированных вершин и одного полного графа с $n-k+1$ вершинами.

$$N(n,k) = \frac{1}{2}(n-k+1)(n-k).$$

Теорема-следствие 3.4. Простой неориентированный граф с n вершинами и с числом ребер, большим, чем

$$N(n, 2) = \frac{1}{2}(n-1)(n-2)$$

связан.

Деревья

Связный неориентированный граф называется деревом, если он не имеет циклов. Дерево не имеет петель и кратных ребер.
Граф без циклов есть граф, связные компоненты которого являются деревьями — лес.
Любая связанная компонента дерева будет также деревом.

Теорема 3.5. В дереве любые две вершины связаны единственным простым путем.

Доказательство. Любой путь в графе без циклов является простым. Если бы было два связывающих вершины простых пути, то был бы и цикл в дереве, что невозможно.

Наглядное представление для произвольного дерева $T =< V, E >$ можно получить при помощи следующей конструкции. Выберем произвольную вершину a_0 и будем рассматривать ее как корень дерева или вершину нулевого уровня. Затем "потрясём" дерево.
Будем называть вершину a дерева концевой, если $\rho(a) = 1$.
Будем называть ребро концевым, если хотя бы одна инцидентная ему вершина является концевой.

Утверждение 3.6. Любое нетривиальное конечное дерево имеет хотя бы две концевые вершины и хотя бы одно концевое ребро.

Теорема 3.7. Каждое дерево с n вершинами имеет $n-1$ ребро.

Доказательство. Доказательство легко проводится индукцией по числу вершин. Для $n = 1$ утверждение, очевидно, справедливо. Пусть $n > 1$. Тогда в дереве существует концевая вершина v. Удаляя из дерева v и ребро (u, v), инцидентное v, получим дерево с $n-1$ вершиной, которое в силу индуктивного предположения имеет $n-2$ ребра. Следовательно, первоначальное дерево имеет $n-2+1 = n-1$ ребро.

Теорема 3.8. Число различных деревьев, которые можно построить на заданном множестве V, содержащем n вершин, равно

$$t_n = n^{n-2}.$$

Доказательство. Обозначим элементы данного множества V, расположенные в некотором фиксированном порядке, числами $V = \{1, 2, ..., n\}$.
Для любого дерева T, определенного на V, введем некоторый символ, характеризующий его однозначно. В T существуют концевые вершины. Обозначим через b_1 первую концевую вершину в последовательности $V = \{1, 2, ..., n\}$, а через $e_1 = \{a_1, b_1\}$ – соответствующее концевое ребро. Удалив из T ребро e_1 и вершину b_1, мы получим новое дерево T_1 . Для T_1 найдется первая в списке $V = \{1, 2, ..., n\}$ концевая вершина b_2 с ребром $e_2 = \{a_2, b_2\}$. Эта редукция повторяется до тех пор, пока после удаления ребра $e_{n-2} = \{a_{n-2}, b_{n-2}\}$ не останется единственное ребро $e_{n-1} = \{a_{n-1}, b_{n-1}\}$, соединяющее две оставшиеся вершины.

Список $\sigma(T) = [a_1, a_2, ..., a_{n-2}]$ однозначно определяются деревом T и двум различным деревьям T и T', соответствуют разные символы такого вида. Каждая вершина a_i появляется в $\sigma(T)$ $\rho(a_i) - 1$ раз.

Последовательность $\sigma(T)$ однозначно определяет дерево T с помощью обратного построения. Если дана последовательность $\sigma(T)$, то находится первая вершина b_1 в списке $V = \{1, 2, ..., n\}$, которая не содержится в $\sigma(T)$. Это определяет ребро $e_1 = \{a_1, b_1\}$. Далее удаляем вершины a_1 из последовательности $\sigma(T)$ и b_1 из списка $V = \{1, 2, ..., n\}$ и продолжаем построение для оставшихся элементов. Получающийся в результате построения граф является деревом, что может быть установлено, например, по индукции. После удаления e_1 последовательность $\sigma(T)$ будет содержать $n - 3$ элемента. Если она соответствует дереву T_1, то граф T, получаемый из него добавлением e_1, также есть дерево, так как вершина b_1 не принадлежит T_1. В $\sigma(T)$ каждая вершина может принимать все n возможных значений. Все они соответствуют различным деревьям, откуда и получается формула $t_n = n^{n-2}$.