Билеты Теория Графов - 4

Тимур Адиатуллин | telegram, github

Содержание

1 Специальные бинарные отношения. Связь между понятием отношения и понятием графа. Орграфы и бинарные отношения.

Бинарные отношения

Пусть A — непустое множество. **Бинарным отношением** на A называется любое подмножество

$$R \subseteq A \times B.$$

Пару $(a, b) \in R$ принято обозначать как $a\,R\,b$.

Специальные свойства бинарных отношений

- **Рефлексивность:**

$$\forall a \in A:\ (a, a) \in R.$$

- **Антирефлексивность:**

$$\forall a \in A:\ (a, a) \notin R.$$

- **Симметричность:**

$$\forall a, b \in A:\ (a, b) \in R \Rightarrow (b, a) \in R.$$

- **Антисимметричность:**

$$(a, b) \in R \text{ и } (b, a) \in R \Rightarrow a = b.$$

- **Транзитивность:**

$$(a, b) \in R \text{ и } (b, c) \in R \Rightarrow (a, c) \in R.$$

Основное определение

Графом $G(V, E)$ называется совокупность двух множеств — непустого множества V (множества вершин) и множества E двухэлементных подмножеств множества V (E — множество рёбер),

$$G(V, E) \stackrel{\text{def}}{=} \langle V; E\rangle, \quad V \neq \emptyset, \quad E \subseteq 2^V \ \&\ \forall e \in E\ (|e| = 2).$$

ЗАМЕЧАНИЕ. Легко видеть, что любое множество E двухэлементных подмножеств множества V определяет симметричное бинарное отношение на множестве V. Поэтому можно считать, что

$$E \subseteq V \times V, \quad E = E^{-1}$$

и трактовать ребро не только как множество $\{v_1, v_2\}$, но и как пару (v_1, v_2).

Число вершин графа G обозначим p, а число рёбер — q:

$$p \stackrel{\text{def}}{=} p(G) \stackrel{\text{def}}{=} |V|, \quad q \stackrel{\text{def}}{=} q(G) \stackrel{\text{def}}{=} |E|.$$

Если хотят явно упомянуть числовые характеристики графа, то говорят: (p, q)-граф.

Ориентированный граф (орграф)

Если элементами множества E являются упорядоченные пары (т. е. $E \subseteq V \times V$), то граф называется **ориентированным** (или **орграфом**). В этом случае элементы множества V называются **узлами**, а элементы множества E — **дугами**.

Орграфы и свойства отношений

Свойства отношения естественно интерпретируются в терминах орграфов:

- **Рефлексивность** соответствует наличию петли в каждой вершине.
- **Антирефлексивность** означает отсутствие петель.
- **Симметричность** означает, что каждая дуга имеет дугу в обратном направлении.
- **Антисимметричность** означает отсутствие пар противоположных дуг между различными вершинами.
- **Транзитивность** означает: если есть дуги $a \to b$ и $b \to c$, то должна быть дуга $a \to c$.

Соответствие графов и отношений

- Полный граф — универсальное отношение.
- Неорграф — симметричное отношение.
- Дополнение графов — дополнение отношений.
- Изменение всех направлений дуг — обратное отношение.

Итог

Бинарное отношение на множестве A полностью эквивалентно ориентированному графу на том же множестве вершин. Свойства отношения имеют естественные графовые интерпретации, что делает орграфы удобным инструментом для визуализации и анализа отношений.

2 Определение графа. Смежность. Диаграммы. Псевдо-, гипер- и мультиграфы. Виды графов. Связность

Основное определение

Графом $G(V, E)$ называется совокупность двух множеств — непустого множества V (множества вершин) и множества E двухэлементных подмножеств множества V (E — множество рёбер),

$$G(V, E) \stackrel{\text{def}}{=} \langle V; E\rangle, \quad V \neq \emptyset, \quad E \subseteq 2^V \ \& \ \forall e \in E \ (|e| = 2).$$

Смежность

Пусть v_1, v_2 — вершины, $e = (v_1, v_2)$ — соединяющее их ребро. Тогда вершина v_1 и ребро e

7.1.3. Смежность

Пусть v_1, v_2 — вершины, $e = (v_1, v_2)$ — соединяющее их ребро. Тогда вершина v_1 и ребро e **инцидентны**, ребро e и вершина v_2 также инцидентны. Два ребра, инцидентные одной вершине, называются смежными; две вершины, инцидентные одному ребру, также называются смежными.

Множество вершин, смежных с вершиной v, называется **множеством смежности** (или **окрестностью**) вершины v и обозначается $\Gamma^+(v)$:

$$\Gamma^+(v) \stackrel{\text{Def}}{=} \{u \in V \mid (u, v) \in E\}, \quad \Gamma^*(v) \stackrel{\text{Def}}{=} \Gamma^+(v) + v.$$

ЗАМЕЧАНИЕ. Если не оговорено противное, то символ Γ без индекса подразумевает Γ^+, то есть саму вершину в окрестность не включают.

Очевидно, что $u \in \Gamma(v) \iff v \in \Gamma(u)$. Если $A \subset V$ — множество вершин, то $\Gamma(A)$ — множество всех вершин, смежных с вершинами из A:

$$\Gamma(A) \stackrel{\text{Def}}{=} \{u \in V \mid \exists\, v \in A \ (u \in \Gamma(v))\} = \bigcup_{v \in A} \Gamma(v).$$

Диаграммы

Обычно граф изображают диаграммой: вершины — точками (или кружками), рёбра — линиями.

Пример. На рис. 7.4 приведён пример диаграммы графа, имеющего четыре вершины и пять рёбер. В этом графе вершины v_1 и v_2, v_2 и v_3, v_3 и v_4, v_4 и v_1, v_2 и v_4 смежны, а вершины v_1 и v_3 не смежны. Смежные рёбра: e_1 и e_2, e_2 и e_3, e_3 и e_4, e_4 и e_1, e_1 и e_5, e_2 и e_5, e_3 и e_5, e_4 и e_5. Несмежные рёбра: e_1 и e_3, e_2 и e_4.

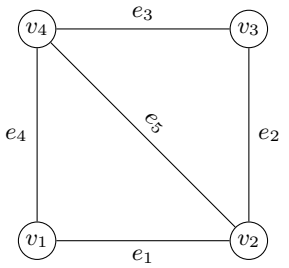

Рис. 1: Диаграмма графа

Типы графов

1. **Псевдограф** — граф с петлями.
2. **Мультиграф** — граф с кратными рёбрами.
3. **Гиперграф** — дуги являются множествами с одним и более элементами.
4. **Нумерованный граф** — если существует функция, отображающая множество вершин или рёбер в множество чисел или символов.

- Если элементом множества E может быть пара одинаковых (не различных) элементов V, то такой элемент называется **петлёй**, а граф — **графом с петлями** (или **псевдографом**).
- Если E является не множеством, а мультимножеством, содержащим некоторые элементы по несколько раз, то такие элементы называются **кратными рёбрами**, а граф — **мультиграфом**.

- Если элементами множества E являются не обязательно двузначные, а любые (непустые) подмножества множества V, то такие элементы называются **гипердугами**, а граф — **гиперграфом**.
- Если задана функция $F : V \to M$ и/или $F : E \to M$, то множество M называется **множеством пометок**, а граф — **помеченным** (или **нагруженным**). В качестве множества пометок обычно используются буквы или целые числа. Если функция F инъективна, то есть разные вершины (рёбра) имеют разные пометки, то граф называют **нумерованным**.

Специальные графы

- **Тривиальный граф** — состоит из одной вершины.
- **Циклический граф с k вершинами** — обозначается C_k.
- **Полный граф** — содержит все возможные рёбра между вершинами, обозначается K_p.

Число рёбер в полном графе:

$$q(K_p) = \frac{p(p-1)}{2}.$$

Колёса и двудольные графы

- **Колесо** — граф, обозначаемый W_n.
- **Двудольный граф** — граф, множество вершин V которого можно разбить на два непересекающихся множества V_1 и V_2 так, что каждое ребро из E инцидентно вершинам из V_1 и V_2. Множества V_1 и V_2 называются **долями графа**.

Связность графа

Говорят, что две вершины в графе **связаны**, если существует соединяющая их (простая) цепь. Граф, в котором все вершины связны, называется **связным**.

Связность является **эквивалентностью**. Классы эквивалентности по отношению связности — это **компоненты связности** графа:

$$k(G).$$

Граф, состоящий только из вершин, называется **вполне несвязным**.

3 Понятие изоморфизма. Изоморфизм графов (+2 теоремы). Инварианты графа.

Гомоморфизм

Пусть $\mathcal{A} = \langle A; \varphi_1, \ldots, \varphi_m \rangle$ и $\mathcal{B} = \langle B; \psi_1, \ldots, \psi_m \rangle$ — две алгебры одного типа (одинаковые векторы арностей). Если существует функция $f : A \to B$, такая, что

$$\forall i \in \{1, \ldots, m\} : \quad f(\varphi_i(a_1, \ldots, a_n)) = \psi_i(f(a_1), \ldots, f(a_n)),$$

то говорят, что f — **гомоморфизм** из $\mathcal{A}$ в $\mathcal{B}$.

Действие гомоморфизма можно изобразить с помощью диаграммы:

$$\begin{array}{ccc} A & \longrightarrow & A \\ \downarrow & & \downarrow \\ B & \longrightarrow & B \end{array}$$

Рис. 2: Коммутативная диаграмма: $f \circ \varphi = \psi \circ f$

Диаграмма называется **коммутативной**, потому что условие гомоморфизма можно переписать с помощью суперпозиции функций:

$$f \circ \varphi = \psi \circ f.$$

Изоморфизм

Пусть $\mathcal{A} = \langle A; \varphi_1, \ldots, \varphi_m \rangle$ и $\mathcal{B} = \langle B; \psi_1, \ldots, \psi_m \rangle$ — две алгебры одного типа, и $f : A \to B$ — изоморфизм или гоморфизм с биекцией. Тогда алгебры $\mathcal{A}$ и $\mathcal{B}$ изоморфны:

$$\mathcal{A}^f \sim \mathcal{B}.$$

Теорема 1. Если $f : A \to B$ — изоморфизм, то $f^{-1} : B \to A$ тоже является изоморфизмом.

Теорема 2. Отношение изоморфизма на множестве однотипных алгебр является эквивалентностью.

Изоморфизм графов

Говорят, что два графа $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$ **изоморфны** (обозначается $G_1 \sim G_2$ или $G_1 = G_2$), если существует биекция $h : V_1 \to V_2$, сохраняющая смежность:

$$e_1 = (u, v) \in E_1 \iff e_2 = (h(u), h(v)) \in E_2.$$

Теорема

Изоморфизм графов есть отношение эквивалентности

Теорема 1.

Графы изоморфны тогда и только тогда, когда их матрицы смежности вершин получаются друг из друга одновременными перестановками строк и столбцов.

Теорема 2.

Графы (орграфы) изоморфны тогда и только тогда, когда их матрицы инцидентности получаются друг из друга произвольными перестановками строк и столбцов.

Инварианты

Ну найдете сами

4 Элементы графов: подграфы, валентность, маршруты, цепи, циклы. Метрические характеристики графа. Особенности алгоритмов теории графов

Подграфы

Граф $G'(V', E')$ называется **подграфом** (или **частью**) графа $G(V, E)$ (обозначается $G' \subseteq G$), если

$$V' \subseteq V \quad \& \quad E' \subseteq E.$$

Если $V' = V$, то G' называется **остовным подграфом** графа G.

Если $V' \subset V$, $E' \subset E$ и $(V' \neq V \vee E' \neq E)$, то граф G' называется **собственным подграфом** графа G.

Подграф $G'(V', E')$ называется **правильным подграфом** графа $G(V, E)$, если он содержит все возможные рёбра графа G между вершинами из V':

$$\forall\, u, v \in V' \quad ((u, v) \in E \Rightarrow (u, v) \in E').$$

Правильный подграф $G'(V', E')$ графа $G(V, E)$ определяется подмножеством вершин V'.

Замечание.

Иногда подграфами называют только правильные подграфы, а неправильные подграфы называют **изграфами**.

Степень вершины

Количество рёбер, инцидентных вершине v, называется **степенью** (или **валентностью**) вершины v и обозначается $d(v)$:

$$\forall v \in V \quad 0 \leq d(v) \leq p - 1, \qquad d(v) = |\Gamma^+(v)|.$$

Таким образом, степень $d(v)$ вершины v совпадает с количеством смежных с ней вершин. Количество вершин, не смежных с v, обозначается $\overline{d}(v)$. Ясно, что:

$$\forall v \in V \quad d(v) + \overline{d}(v) = p - 1.$$

Обозначим минимальную степень вершины графа G через $\delta(G)$, а максимальную — через $\Delta(G)$:

$$\delta(G(V, E)) \stackrel{\text{def}}{=} \min_{v \in V} d(v), \qquad \Delta(G(V, E)) \stackrel{\text{def}}{=} \max_{v \in V} d(v).$$

Очевидно, что $\delta(G)$ и $\Delta(G)$ являются **инвариантами** графа.

Если степени всех вершин равны k, то граф называется **регулярным степени** k:

$$\delta(G) = \Delta(G) = k, \qquad \forall v \in V \quad d(v) = k.$$

Степень регулярности обозначается $r(G)$. Для нерегулярных графов $r(G)$ не определено.

Маршруты, цепи, циклы

Маршрутом в графе называется чередующаяся последовательность вершин и рёбер, начинающаяся и заканчивающаяся вершиной:

$$v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k,$$

в которой любые два соседних элемента инцидентны, причём однородные элементы (вершины, рёбра) через один — смежны или совпадают.

Маршруты, цепи, циклы

Если $v_0 = v_k$, то маршрут называется **замкнутым**, иначе — **открытым**.

Если все рёбра различны, то маршрут называется **цепью**. Если все вершины (а значит, и рёбра) различны, то маршрут называется **простой цепью**.

В цепи $v_0, e_1, \ldots, e_k, v_k$ вершины v_0 и v_k называются **концами цепи**. Говорят, что цепь с концами u и v **соединяет вершины** u и v.

Цепь, соединяющая вершины u и v, обозначается $\langle u, v \rangle$. Если нужно указать граф G, которому принадлежит цепь, то добавляют индекс: $\langle u, v \rangle_G$.

Нетрудно показать, что если существует какая-либо цепь, соединяющая вершины u и v, то существует и **простая цепь**, соединяющая эти вершины.

Замкнутая цепь называется **циклом**; **замкнутая простая цепь** называется **простым циклом**.

Число циклов в графе G обозначается $z(G)$. Граф без циклов называется **ациклическим**.

Для орграфов **цепь** называется **путём**, а **цикл — контуром**.

Путь в орграфе из узла u в узел v обозначается:

$$\langle \vec{u}, v \rangle.$$

Метрические характеристики графа

Длина маршрута — количество рёбер в нём. Маршрут M имеет длину k тогда и только тогда, когда $|M| = k$.

Расстоянием между вершинами u и v, обозначаемым $d(u, v)$, называется длина кратчайшей цепи $\langle u, v \rangle$, а сама кратчайшая цепь называется **геодезической**:

$$d(u, v) = \min_{\{\langle u,v \rangle\}} |\langle u, v \rangle|.$$

Если цепи нет, расстояние считается бесконечным.

Ярус — множество вершин на расстоянии n от вершины v.

Диаметр графа — длина самой длинной геодезической цепи:

$$D(G) = \max_{u,v \in V} d(u, v).$$

Эксцентриситет $e(v)$ вершины v в связном графе — максимальное расстояние от v до других вершин.

Радиус графа $R(G)$ — наименьший эксцентриситет:

$$R(G) = \min_{v \in V} e(v).$$

Вершина называется **центральной**, если её эксцентриситет совпадает с радиусом. Множество центральных вершин называется **центром графа**.

5 Способы задания графа. Представление графов в ЭВМ. Обходы графов.

Представление графов в программах

Следует ещё раз подчеркнуть, что конструирование структур данных для представления в программе объектов математической модели — это основа искусства практического программирования.

Мы приводим четыре различных базовых представления графов. Выбор наилучшего представления определяется требованиями конкретной задачи. Более того, на практике используются, как правило, некоторые комбинации или модификации указанных представлений, общее число которых необозримо. Но все они так или иначе основаны на тех базовых идеях, которые описаны в этом разделе.

Требования к представлению графов

Известны различные способы представления графов в памяти компьютера, которые различаются объёмом занимаемой памяти и скоростью выполнения операций над графами. Представление выбирается, исходя из потребностей конкретной задачи.

Далее приведены четыре наиболее часто используемых представления с указанием характеристики $\eta(p, q)$ — объёма памяти для каждого представления, где p — число вершин, а q — число рёбер.

7.4.2. Матрица смежности

Представление графа с помощью квадратной булевой матрицы

$$M : \text{array } [1..p, 1..p] \text{ of } \{0, 1\},$$

отражающей смежность вершин, называется **матрицей смежности**, где

$$M[i, j] = \begin{cases} 1, & \text{если вершина } v_i \text{ смежна с вершиной } v_j, \\ 0, & \text{если вершины } v_i \text{ и } v_j \text{ не смежны.} \end{cases}$$

Для матрицы смежности объём памяти:

$$\eta(p, q) = \mathcal{O}(p^2).$$

Пример. Матрицы смежности графов G и D:

$$G = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Замечание. Матрица смежности графа симметрична относительно главной диагонали, поэтому достаточно хранить только верхнюю (или нижнюю) треугольную часть.

Матрица инцидентций

Представление графа с помощью матрицы

$$H : \text{array } [1..p, 1..q] \text{ of } \{0, 1\},$$

а для орграфов:

$$H : \text{array } [1..p, 1..q] \text{ of } \{-1, 0, 1\},$$

отражающей инцидентность вершин и рёбер, называется **матрицей инцидентций**, где для неориентированного графа:

$$H[i, j] = \begin{cases} 1, & \text{если вершина } v_i \text{ инцидентна ребру } e_j, \\ 0, & \text{в противном случае.} \end{cases}$$

А для ориентированного графа:

$$H[i, j] = \begin{cases} 1, & \text{если узел } v_i \text{ инцидентен дуге } e_j \text{ и является её концом,} \\ -1, & \text{если узел } v_i \text{ инцидентен дуге } e_j \text{ и является её началом,} \\ 0, & \text{если узел } v_i \text{ и дуга } e_j \text{ не инцидентны.} \end{cases}$$

Для матрицы инцидентций объём памяти:

$$\eta(p, q) = \mathcal{O}(pq).$$

Пример. Матрицы инцидентций графов G и D:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} -1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 \end{pmatrix}$$

Замечание. Для связных графов $q > p$, поэтому матрица смежности несколько компактнее матрицы инцидентций.

7.4.4. Списки смежности

Представление графа с помощью списочной структуры, отражающей смежность вершин и состоящей из массива указателей

$$G : \text{array } [1..p] \text{ of } \uparrow N,$$

на списки смежных вершин, где элемент списка описывается структурой:

$$N = \text{record } \{v : 1..p; \quad n :\uparrow N\} \text{ end record},$$

называется **списком смежности**.

В случае представления неориентированных графов:

$$\eta(p, q) = \mathcal{O}(p + 2q),$$

а в случае ориентированных графов:

$$\eta(p, q) = \mathcal{O}(p + q).$$

Замечание. Массив G также можно представить списком.

Пример. Списки смежности для графа G (слева) и орграфа D (справа) представлены на рисунке ниже.

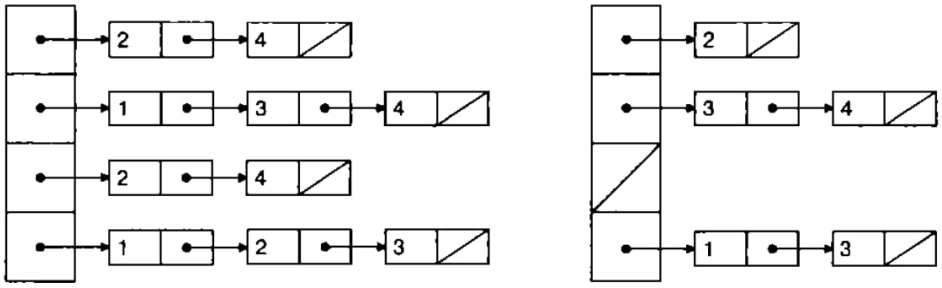

Рис. 3: Списки смежности для графа G и орграфа D

7.4.5. Массив дуг

Представление графа с помощью массива структур

$$E : \text{array } [1..q] \text{ of record } \{b, e : 1..p\} \text{ end record},$$

отражающего список пар смежных вершин (или, для орграфов, узлов), называется **массивом рёбер** (или **массивом дуг**).

Для массива рёбер (или дуг) объём памяти:

$$\eta(p, q) = \mathcal{O}(2q).$$

Замечание. Для представления графов с изолированными вершинами может понадобиться хранить также число p, если только система программирования не позволяет извлечь это число из массива структур E.

Пример. Представление с помощью массива рёбер (дуг) показано в следующей таблице: слева — для графа G, справа — для орграфа D.

Граф G		Орграф D	
b	e	b	e
1	2	1	2
1	4	2	3
2	3	2	4
2	4	4	1
3	4	4	3

Замечание. Указанные представления пригодны для графов и орграфов, а после некоторой модификации — также и для псевдографов, мультиграфов и гиперграфов.

Обходы графов

Обход графа — систематическое перечисление его вершин.

Теорема.

Если граф G связан и конечен, то поиск в ширину и поиск в глубину обходят все вершины по одному разу.

Суть методов:

- **Стек** — используется в поиске в глубину (DFS).
- **Очередь** — используется в поиске в ширину (BFS).

Алгоритм обхода графа:

1. Дан граф. Выбираем начальную вершину случайно или задаём вручную.
2. Создаём массив, где отмечаем пройденные вершины (изначально все равны 0).
3. Помещаем начальную вершину в структуру (стек или очередь) и отмечаем её.
4. Заходим в цикл:
   (a) Извлекаем вершину из структуры.
   (b) Просматриваем все смежные вершины:
      - Если вершина не отмечена — помещаем её в структуру и отмечаем.
   (c) Повторяем, пока структура не опустеет.

6 Операции над графами: локальные, алгебраические.

7.3.4. Операции над графами (продолжение)

4. **Удаление вершины** v из графа $G_1(V_1, E_1)$ (обозначение: $G_1(V_1, E_1) - v$, при условии $v \in V_1$) даёт граф $G_2(V_2, E_2)$, где

$$V_2 = V_1 \setminus \{v\}, \quad E_2 = E_1 \setminus \{e = (v_1, v_2) \mid v_1 = v \vee v_2 = v\}.$$

   Пример: $C_3 - v = K_2$.

5. **Удаление ребра** e из графа $G_1(V_1, E_1)$ (обозначение: $G_1(V_1, E_1) - e$, при условии $e \in E_1$) даёт граф $G_2(V_2, E_2)$, где

$$V_2 = V_1, \quad E_2 = E_1 \setminus \{e\}.$$

   Пример: $K_2 - e = K_2$.

6. **Добавление вершины** v в граф $G_1(V_1, E_1)$ (обозначение: $G_1(V_1, E_1) + v$, при условии $v \notin V_1$) даёт граф $G_2(V_2, E_2)$, где

$$V_2 = V_1 \cup \{v\}, \quad E_2 = E_1.$$

   Пример: $K_2 + v = K_2 \cup K_1$.

7. **Добавление ребра** e в граф $G_1(V_1, E_1)$ (обозначение: $G_1(V_1, E_1) + e$, при условии $e \notin E_1$) даёт граф $G_2(V_2, E_2)$, где

$$V_2 = V_1, \quad E_2 = E_1 \cup \{e\}.$$

8. **Стягивание (правильного) подграфа** A графа $G_1(V_1, E_1)$ (обозначение: $G_1(V_1, E_1)/A$, при условии $A \subset V_1, v \notin V_1$) даёт граф $G_2(V_2, E_2)$, где

$$V_2 = (V_1 \setminus A) \cup \{v\},$$

$$E_2 = E_1 \setminus \{e = (u, w) \mid u \in A \vee w \in A\} \cup \{e = (u, v) \mid u \in \Gamma(A) \setminus A\}.$$

   Пример: $K_4/C_3 = K_2$.

9. **Размножение вершины** v графа $G_1(V_1, E_1)$ (обозначение: $G_1(V_1, E_1) \uparrow v$, при условии $v \in V_1, v' \notin V_1$) даёт граф $G_2(V_2, E_2)$, где

$$V_2 = V_1 \cup \{v'\},$$

$$E_2 = E_1 \cup \{(v, v')\} \cup \{e = (u, v') \mid u \in \Gamma^+(v)\}.$$

   Пример: $K_2 \uparrow v = C_3$.

7 Упорядочение дуг и вершин орграфа. Алгоритм Фалкерсона.

Упорядочивание дуг и вершин графа

Под **упорядочиванием ациклического орграфа** понимается такое разбиение его вершин на группы, при котором:

1. Вершины первой группы не имеют предшествующих, а последней — последующих.
2. Вершины любой другой группы не имеют предшествующих в следующей группе.
3. Вершины одной и той же группы дугами не соединяются.

Такое разбиение всегда возможно.

Алгоритм Фалкерсона

1. Находим истоки — они образуют первую группу, нумеруем их в произвольном порядке.
2. Вычёркиваем все пронумерованные вершины и дуги.
3. Повторяем первый шаг для оставшегося графа — новая группа, новая нумерация.
4. Продолжаем, пока не будут упорядочены все вершины.

Аналогично можно упорядочить и дуги.

Матричный способ

1. Берём матрицу смежности.
2. Находим столбцы, состоящие из нулей — это первая группа.
3. Вычёркиваем найденные столбцы и соответствующие строки.
4. Повторяем, пока не упорядочим все вершины.

8 Выявление маршрутов с заданным количеством ребер. Определение экстремальных путей на графах. Метод Шимбелла. Волновые алгоритмы. (составлял гпт)

Маршруты с заданным количеством рёбер

Пусть задан граф $G(V, E)$ и две вершины $s, t \in V$. Требуется определить, существует ли маршрут длины ровно k рёбер.

- Используется матрица смежности A графа.
- Элемент $(A^k)_{st}$ показывает количество маршрутов длины k между s и t.
- Если $(A^k)_{st} > 0$, то существует хотя бы один маршрут длины k.

Таким образом, задача сводится к возведению матрицы смежности в степень.

Экстремальные пути

Экстремальные пути — это пути, обладающие некоторым оптимальным свойством:

- **Кратчайший путь** — минимальная суммарная длина.
- **Длиннейший простой путь** — максимальная длина без повторения вершин (NP-трудная задача).
- **Минимальный по числу рёбер путь**.
- **Максимальный потоковый путь** (в сетях).

Для кратчайших путей применяются алгоритмы Дейкстры, Беллмана–Форда, Флойда–Уоршелла.

Метод Шимбелла

Метод Шимбелла — это итерационный метод динамического программирования для нахождения кратчайших путей.

$$d^{(k)}(i,j) = \min\left(d^{(k-1)}(i,j),\ \min_{v \in V}\left(d^{(k-1)}(i,v) + w(v,j)\right)\right)$$

- $d^{(0)}(i,j)$ — исходная матрица весов.
- На каждом шаге учитываются пути длиной не более k рёбер.
- После n итераций получаем матрицу кратчайших расстояний.

Метод аналогичен алгоритму Флойда–Уоршелла, но формулируется как последовательное уточнение расстояний.

Волновые алгоритмы

Волновые алгоритмы — это алгоритмы поиска в ширину (BFS-подобные), основанные на распространении «волны» по графу.

- На первом шаге волна исходит из стартовой вершины.
- На каждом следующем шаге волна распространяется на все ещё не посещённые соседние вершины.
- Вершины помечаются номерами слоёв (расстоянием в рёбрах).

Применения:

- Поиск кратчайшего пути в невзвешенном графе.
- Проверка связности.
- Построение уровневых графов (например, в алгоритме Диница).

Волновые алгоритмы являются основой для многих методов маршрутизации и анализа графов.

9 Связность: компоненты связности, точки сочленения. Вершинная и реберная связность (мосты и блоки, меры связности).

Связность: компоненты связности, точки сочленения

Теорема

Граф связан тогда и только тогда, когда его нельзя представить в виде объединения двух графов.

Компоненты связности

Классы эквивалентности по отношению связности называются **компонентами связности** графа. Число компонент связности обозначается $k(\mathcal{G})$.

Точка сочленения

Вершина графа называется **точкой сочленения**, если её удаление увеличивает число компонент связности.

Мост

Мостом называется ребро, удаление которого увеличивает число компонент связности.

Замечание В любом нетривиальном графе существует по крайней мере две вершины, которые не являются точками сочленения.

Вершинная и реберная связность (мосты и блоки, меры связности)

Теорема 1

Пусть $G(V, E)$ — связный граф и $v \in V$. Тогда следующие утверждения эквивалентны:

1. v — точка сочленения.
2. $\exists\, u, w \in V$ такие, что $u \neq w$ и $v \in \langle u, w\rangle_G$.
3. $\exists\, U, W \subseteq V \setminus \{v\}$ такие, что $U \cap W = \emptyset$, $U \cup W = V \setminus \{v\}$ и для всех $u \in U$, $w \in W$ любые пути $\langle u, w\rangle_G$ проходят через v.

Следствие

Если вершина инцидентна мосту и не является висячей, то она является точкой сочленения.

Теорема 2

Пусть $G(V, E)$ — связный граф и $x \in E$. Тогда следующие утверждения эквивалентны:

1. x — мост.
2. x не принадлежит ни одному простому циклу.
3. $\exists\, u, w \in V$ такие, что все пути $\langle u, w\rangle_G$ содержат x.
4. $\exists\, U, W \subseteq V$ такие, что $U \cap W = \emptyset$, $U \cup W = V$ и для всех $u \in U$, $w \in W$ любые пути $\langle u, w\rangle_G$ содержат x.

Вершинная связность

Вершинной связностью графа называется наименьшее число вершин, удаление которых приводит к несвязному или тривиальному графу. Обозначается $\chi(G)$.

Рёберная связность

Рёберной связностью графа называется наименьшее число рёбер, удаление которых приводит к несвязному или тривиальному графу. Обозначается $\lambda(G)$.

10 Теорема Менгера (с доказательством). Непересекающиеся цепи и разделяющие множества. Варианты теоремы Менгера. Теорема Холла.

Теорема Менгера

Пусть u и v — несмежные вершины в графе G. Наименьшее число вершин в множестве, разделяющем u и v, равно наибольшему числу вершинно непересекающихся простых цепей $\langle u, v\rangle$:

$$\max |P(u,v)| = \min |S(u,v)|$$

где:

- $P(u,v)$ — множество вершинно непересекающихся цепей $\langle u, v\rangle$;
- $S(u,v)$ — разделяющее множество вершин (удаление которых приводит к разбиению графа так, что u и v оказываются в разных компонентах связности).

я ебал это доказывать

Разрез

Разделяющее множество рёбер называется **разрезом**.

Варианты теоремы Менгера

Теорема

Для любых двух несмежных вершин u и v графа G наибольшее число рёберно-непересекающихся цепей $\langle u, v\rangle$ равно наименьшему числу рёбер в $\langle u, v\rangle$-разрезе.

Теорема

Граф G является n-связным тогда и только тогда, когда любые две несмежные вершины соединены не менее чем n вершинами, образующими вершинно-непересекающиеся простые цепи.

Теорема Холла

Паросочетание

Паросочетанием (или независимым множеством рёбер) называется множество рёбер, никакие два из которых не смежны. Паросочетание называется **максимальным**, если никакое его надмножество не является независимым.

Пусть $G(V_1, V_2, E)$ — двудольный граф. **Совершенным паросочетанием** из V_1 в V_2 называется паросочетание, покрывающее все вершины множества V_1.

Теорема Холла

Совершенное паросочетание существует тогда и только тогда, когда

$$\forall A \subseteq V_1 \quad (|A| \leq |\Gamma(A)|),$$

11 Нахождение кратчайших путей: алгоритм Дейкстры, алгоритм Беллмана-Форда.

Алгоритм Дейкстры

Алгоритм Дейкстры используется для поиска кратчайших путей от заданной вершины-истока s до всех остальных вершин графа.

Сложность

$$\mathcal{O}(p^2)$$

Результат

Алгоритм возвращает вектор расстояний $d[v]$ от истока s до каждой вершины $v \in V$.

Шаги алгоритма

1. **Инициализация:**
   - Для всех вершин $v \in V$ задать $d[v] := \infty$.
   - Для истока s задать $d[s] := 0$.
   - Создать множество необработанных вершин $Q := V$.
2. **Основной цикл:** Пока $Q \neq \emptyset$:
   (a) Выбрать вершину $u \in Q$ с минимальным значением $d[u]$.
   (b) Удалить u из Q.
   (c) Для каждого соседа v вершины u, где $v \in Q$:
      - Вычислить новое расстояние: temp $:= d[u] + w(u, v)$.
      - Если temp $< d[v]$, то обновить: $d[v] :=$ temp.

Алгоритм Беллмана–Форда

Алгоритм Беллмана–Форда используется для поиска кратчайших путей от заданной вершины-истока s до всех остальных вершин графа, допускающего отрицательные веса рёбер.

Сложность

$$\mathcal{O}(pq)$$

Результат

Алгоритм возвращает вектор расстояний $d[v]$ от истока s до каждой вершины $v \in V$.

Инициализация

- Для всех вершин $v \in V$ задать $d[v] := \infty$.
- Для истока s задать $d[s] := 0$.

Основной алгоритм

1. Повторить следующие шаги $|V| - 1$ раз:
   - Для каждого ребра $(u, v) \in E$:
     - Вычислить новое расстояние: temp $:= d[u] + w(u, v)$.
     - Если temp $< d[v]$, то обновить: $d[v] :=$ temp.

Проверка на наличие отрицательных циклов

- Для каждого ребра $(u, v) \in E$:
  - Если $d[u] + w(u, v) < d[v]$, то в графе существует отрицательный цикл.

12 Нахождение кратчайших путей: алгоритм Флойда-Уоршалла. Алгоритм нахождения максимального пути.

Алгоритм Флойда–Уоршалла

Алгоритм Флойда–Уоршалла используется для нахождения кратчайших расстояний между всеми парами вершин во взвешенном графе (возможно с отрицательными весами, но без отрицательных циклов).

Сложность

$$\mathcal{O}(p^3)$$

Результат

Алгоритм возвращает матрицу расстояний размером $p \times p$.

Инициализация

- Для всех пар вершин (i, j) задать $d[i][j] := \infty$.
- Для всех i задать $d[i][i] := 0$.
- Для каждого ребра (u, v) с весом w задать $d[u][v] := w$.

Основной алгоритм

- Для каждой вершины k от 1 до n:
  - Для каждой пары вершин (i, j):
    - Вычислить: temp $:= d[i][k] + d[k][j]$.
    - Если temp $< d[i][j]$, то обновить: $d[i][j] :=$ temp.

Проверка на отрицательные циклы (опционально)

Если после выполнения алгоритма существует i такое, что $d[i][i] < 0$, то в графе присутствует отрицательный цикл.

Алгоритм нахождения максимального пути

Для нахождения максимального пути в графе можно использовать полный перебор всех возможных путей от текущей вершины ко всем достижимым из неё вершинам.

Идея

Перебрать все возможные пути от текущей вершины до всех последующих, достижимых из неё, и выбрать путь с максимальной суммарной длиной (весом).

Применение

Подходит для ориентированных ациклических графов (DAG), где отсутствуют циклы, и можно использовать динамическое программирование или топологическую сортировку для оптимизации.

13 Потоки в сетях: определение потока, разрезы. Теорема Форда и Фалкерсона. Алгоритм Форда-Фалкерсона. Коммуникационные сети.

Сеть

Пусть $G(V, E)$ — орграф с одним источником $s \in V$ и одним стоком $t \in V$, где $s \neq t$. **Сетью** $S = (G; c)$ называется орграф G с заданной функцией $c : E \to R_+$, сопоставляющей каждой дуге $e \in E$ неотрицательное действительное число $c(e)$, называемое **пропускной способностью**.

Поток

Потоком в сети $S = (G; c)$, где $G(V, E)$, называется функция $f : E \to R_+$, приписывающая каждой дуге e неотрицательное число $f(e)$ — **поток по дуге** e, при выполнении следующих условий:

1. $f(e) \leq c(e)$ — поток не превышает пропускную способность;
2. Для любой вершины $v \in V \setminus \{s, t\}$ сумма входящих потоков равна сумме исходящих:

$$\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w).$$

Разрезы

Пусть $S = (G; c)$ — сеть, где $G = (V, E)$. Если $X \subset V$, $s \in X$, $t \notin X$, и $Y = V \setminus X$, то множество

$$R = \{e \in E \mid e = (v, w),\ v \in X,\ w \in Y\}$$

называется **разрезом** сети S и обозначается $R = (X, Y)$.

Пропускная способность разреза

Пропускной способностью разреза $R(X, Y)$ называется неотрицательное число

$$c(R) = \sum_{e\in R} c(e).$$

Теорема Форда–Фалкерсона

Формулировка

Пусть $S = (G; c)$ — сеть, где $G = (V, E)$. Величина максимального потока $p_{\max}$ в сети S совпадает с минимальной пропускной способностью $r_{\min}$ её разрезов:

$$p_{\max} = r_{\min}.$$

Алгоритм Форда–Фалкерсона

Трудоёмкость

$$\mathcal{O}(q \cdot p_{\max})$$

Шаги алгоритма

1. **Инициализация:**
   - Для всех рёбер (u, v) установить $f(u, v) := 0$.
   - Задать исток s и сток t.
2. **Поиск увеличивающего пути:** Пока существует путь p из s в t в остаточной сети (где $c_f(u, v) > 0$):
   (a) Найти минимальную остаточную пропускную способность:

$$c_f(p) := \min\{c_f(u, v) \mid (u, v) \in p\}.$$

   (b) Для каждого ребра $(u, v) \in p$:

- Увеличить поток: $f(u, v) := f(u, v) + c_f(p)$.
- Уменьшить обратный поток: $f(v, u) := f(v, u) - c_f(p)$.

3. **Завершение:** Когда увеличивающих путей больше нет, максимальный поток равен сумме потоков из s в смежные вершины (или в t).

Коммуникационные сети

- Моделью компьютерной сети может служить ориентированный граф, чьи узлы представляют компьютерные компоненты, а дуги — коммуникационные линии связи. Каждая дуга снабжена весом, обозначающим пропускную способность этой линии.
- Процедура **статической маршрутизации** учитывает информацию о пропускной способности линии для определения фиксированного пути передачи между узлами. В целях оптимизации таких путей применяют алгоритм, близкий к алгоритму Дейкстры. Однако задержки могут возникать при сбоях и превышении пропускной способности сети.
- Процедура **динамической маршрутизации** постоянно корректирует пропускную способность линий с учётом текущих потребностей. Набор правил или протокол позволяет узлам решать, когда и куда передавать новую информацию.
- Каждый узел поддерживает свою таблицу путей — задача оптимизации рассредоточена по всей сети.

14 Эвристические алгоритмы. Алгоритм A*. Метод ветвей и границ

Эвристика

Эвристикой называется алгоритм решения задачи, включающий практический метод, не являющийся гарантированно точным или оптимальным, но достаточный для получения решения и позволяющий ускорить процесс вычислений.

- Не гарантирует лучшее решение.
- Не гарантирует наличие решения.
- Может дать неверное решение.

Алгоритм A*

Порядок обхода вершин определяется **эвристической функцией** $f(x)$, которая представляет собой сумму двух компонент:

$$f(x) = g(x) + h(x),$$

где:

- $g(x)$ — функция стоимости достижения вершины x из начальной вершины (может быть эвристической или точной);
- $h(x)$ — эвристическая оценка расстояния от вершины x до целевой вершины.

Функция $h(x)$ должна быть **допустимой эвристикой**, то есть не должна переоценивать расстояние до цели. Например, в задаче маршрутизации $h(x)$ может представлять собой расстояние до цели по прямой линии.

Интуиция

На каждом шаге алгоритм оценивает, насколько текущая вершина приближает к цели, используя функцию $f(x)$, и выбирает путь, минимизирующий эту оценку.

Ну короче типо мы смотрим на каждом шаге насколько мы вообще приблизились к цели или отдалились с помощью какой-либо эвристической функции и это учитываем при выборе пути

Метод ветвей и границ

Метод ветвей и границ является развитием метода полного перебора, но с отсевом подмножеств, заведомо не содержащих оптимальных решений.

Идея

На каждом шаге элементы разбиения анализируются: содержит ли подмножество оптимальное решение или нет. Если решается задача минимизации, то проверка осуществляется сравнением нижней оценки значения целевой функции с верхней оценкой функционала.

Рекорд

Допустимое решение, дающее наименьшую верхнюю оценку, называется **рекордом**. Если нижняя оценка целевой функции на данном подмножестве не меньше текущего рекорда, то это подмножество не содержит лучшего решения и может быть отброшено. Если значение целевой функции меньше рекорда, рекорд обновляется.

Завершение

Если все элементы разбиения просмотрены, алгоритм завершает работу, и текущий рекорд является оптимальным решением. Если нет — выбирается перспективное множество, которое подвергается дальнейшему разбиению. Процесс продолжается, пока не будут просмотрены все элементы.

15 Поток минимальной стоимости. Алгоритм определения потока минимальной стоимости

Поток минимальной стоимости

Пусть задана транспортная сеть $G(V, E)$, где каждому ребру $e \in E$ сопоставлены:

- пропускная способность $c(e)$;
- стоимость единицы потока $w(e)$.

Требуется найти поток максимальной величины или фиксированного объёма F, **минимизирующий суммарную стоимость**:

$$\text{Cost}(f) = \sum_{e \in E} f(e) \cdot w(e)$$

при выполнении условий:

- ограничения пропускной способности: $0 \le f(e) \le c(e)$;
- закон сохранения потока во всех вершинах, кроме истока и стока.

Такой поток называется **потоком минимальной стоимости**.

Основная идея

Если рассматривать стоимость как вес рёбер, то задача сводится к поиску путей минимальной стоимости в остаточной сети. Пока существует путь, уменьшающий стоимость потока, поток можно улучшать.

Алгоритм нахождения потока минимальной стоимости

Один из классических методов — **алгоритм наименьшей стоимости увеличивающего пути** (Successive Shortest Path).

1. Инициализировать поток $f = 0$.
2. Построить остаточную сеть G_f:
   - прямые рёбра имеют стоимость $w(e)$;
   - обратные рёбра имеют стоимость $-w(e)$.
3. Найти кратчайший путь по стоимости из истока s в сток t в остаточной сети.
4. Если пути нет — текущий поток является потоком минимальной стоимости.
5. Иначе:
   - определить возможное увеличение Δ по минимальной пропускной способности на пути;
   - увеличить поток вдоль пути на Δ;
   - обновить остаточную сеть.
6. Перейти к шагу 3.

Свойства

- Алгоритм всегда завершается, если все стоимости целые.
- При использовании алгоритма Дейкстры с потенциалами (метод Джонсона) достигается полиномиальная сложность.
- Метод применяется в логистике, распределении ресурсов, оптимизации расписаний.

Альтернативные методы

- Метод потенциалов (алгоритм Беллмана–Форда + корректировка весов).
- Алгоритм цикла отрицательной стоимости (canceling negative cycles).
- Линейное программирование (решение через симплекс-метод).

16 Транспортная задача. Алгоритмы Диница и Кинга. Задачи многокритериальной оптимизации.

Транспортная задача

Пусть имеется m пунктов производства однородного продукта: $A_1, A_2, \ldots, A_m$ и n пунктов его потребления: $B_1, B_2, \ldots, B_n$. В каждом пункте A_i производится a_i единиц продукта, а в каждом пункте B_j потребляется b_j единиц. Необходимо составить оптимальный план перевозок.

Математическая модель

Пусть x_{ij} — объём продукта, перевозимого из A_i в B_j. Тогда общая стоимость перевозок, которую требуется минимизировать:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \to \min$$

Задачи многокритериальной оптимизации

Пусть имеется несколько целевых функций, которые необходимо максимизировать. Основная сложность заключается в том, что максимум одного критерия может сопровождаться неудовлетворительными значениями других.

Методы сведения к однокритериальной задаче

- Выделение главного критерия.
- Линейная свёртка.
- Максминная свёртка.
- Метод идеальной точки.
- Оптимальность по Парето, по Слейтору, по Никоретте.

Метод последовательных уступок

1. Находим максимальное значение самого важного критерия путём решения однокритериальной задачи.
2. Назначаем допустимое отклонение для первого критерия.
3. В ограниченной области ищем наилучшее значение второго критерия.
4. Повторяем процедуру для всех критериев по убыванию важности.

Алгоритмы Диница и Кинга

Алгоритмы Диница и Кинга используются для поиска максимального потока в сети.

Алгоритм Диница

Сложность: $\mathcal{O}(p^2q)$

Инициализация

- Для всех рёбер (u, v) установить поток $f(u, v) := 0$.
- Построить остаточную сеть G_f с пропускными способностями $c_f(u, v) := c(u, v) - f(u, v)$.

Фаза построения слоистой сети (BFS)

Пока существует путь из s в t в остаточной сети:

- Построить слоистую сеть (level graph) обходом в ширину из s.
- Для каждой вершины v вычислить $level[v]$ — расстояние от s до v.
- Ребро (u, v) включается, если $level[v] = level[u] + 1$ и $c_f(u, v) > 0$.

Фаза блокирующего потока (DFS)

Пока в слоистой сети существует путь из s в t:

- Найти блокирующий поток с помощью обхода в глубину.
- Для каждого найденного пути p:
  - Найти минимальную остаточную пропускную способность:

$$c_f(p) := \min\{c_f(u,v) \mid (u,v) \in p\}.$$

  - Увеличить поток вдоль пути: $f(u,v) := f(u,v) + c_f(p)$.
  - Обновить обратные рёбра: $f(v,u) := f(v,u) - c_f(p)$.

Завершение

Если BFS не достигает t, алгоритм завершает работу. Максимальный поток равен сумме потоков из s в смежные вершины.

Алгоритм Кинга

Сложность: $\mathcal{O}(nm)$

Игра на двудольном графе

Рассмотрим неориентированный двудольный граф $G = (U, V, E)$, где $|U| = |V| = n$, $|E| = m$.

Правила игры

Инициализация: Алгоритм-игрок назначает дугу каждой вершине из множества U.
Ходы соперника:

- Удалить любую дугу из графа (**edge-kill**) — очки не начисляются.
- Удалить любую вершину из множества V вместе с инцидентными ей дугами — начисляется по одному очку за каждую удалённую дугу.

Ходы алгоритма-игрока:

- Назначить дугу вершине $u \in U$, если дуга ещё не назначена.
- Назначить новую инцидентную дугу вершине u — старая дуга теряет специальный статус, а соперник получает дополнительное очко.

Стратегия алгоритма-игрока

Пусть l — минимальный коэффициент (степень) вершин u. Обозначим $U' \subseteq U$ — множество вершин с коэффициентом больше l.

Для каждой вершины $v \in V$ определим оценку $r(v)$ — число назначенных дуг (u, v), где u имеет минимальный коэффициент.

Разделим диапазон значений $r(v)$ на уровни: Уровень i содержит вершины v, для которых $r(v) \in [r_i, r_{i+1})$, где $r_i = 2^i$.

При назначении дуги вершине $u \in U'$, алгоритм выбирает дугу, инцидентную вершине v с минимальным уровнем.

17 Связность в орграфах (сильная, односторонняя и слабая связность, компоненты сильной связности). Алгоритм выделения компонент сильной связности.

Связность в ориентированных графах

В неориентированном графе две вершины либо связаны (если существует соединяющая их цепь), либо не связаны. В ориентированном графе отношение связности узлов несимметрично, поэтому определения связности различаются.

Пусть $G(V, E)$ — орграф, v_1 и v_2 — его вершины.

- **Сильная связность:** Вершины v_1 и v_2 **сильно связаны** в орграфе G, если существует путь из v_1 в v_2 и путь из v_2 в v_1.
- **Односторонняя связность:** Вершины v_1 и v_2 **односторонне связаны**, если существует путь либо из v_1 в v_2, либо из v_2 в v_1.
- **Слабая связность:** Вершины v_1 и v_2 **слабо связаны**, если они связаны в неориентированном графе G', полученном из G забыванием ориентации дуг.

Если все вершины орграфа G сильно (односторонне, слабо) связаны между собой, то орграф называется соответственно **сильно**, **односторонне** или **слабо связным**.

Иерархия связности

Сильная связность $\Rightarrow$ односторонняя связность $\Rightarrow$ слабая связность. Обратные импликации, как правило, неверны.

Компонента сильной связности

Компонентой сильной связности (КСС) орграфа G называется его максимальный сильно связный подграф. Каждая вершина орграфа принадлежит ровно одной КСС. Если вершина не сильно связана ни с одной другой, она сама образует КСС.

Конденсация орграфа

Конденсацией орграфа G (обозначается G^*), также называемой **графом Герца** или **фактор-графом**, называется орграф, полученный путём стягивания каждой компоненты сильной связности в один узел.

В результате конденсации получается ациклический орграф, отражающий структуру связности исходного графа на уровне компонент.

Алгоритм 8.2 Выделение компонент сильной связности

Вход: орграф $G(V, E)$, заданный списками смежности $\Gamma(v)$.
Выход: список C компонент сильной связности, каждый элемент которого есть список узлов, входящих в компоненту сильной связности.

```
C := ∅
for v ∈ V do
  M[v] := {v} { M[v] список узлов, входящих в ту же КСС, что и v }
  e[v] := 0 { все узлы не рассмотрены }
end for
while V ≠ ∅ do
  select v ∈ V { взять v из V }
  v → T { положить v в стек }
  e[v] := 1 { отметить v }
  КСС { вызов процедуры КСС }
end while
```

Рекурсивная процедура КСС

Основная работа выполняется рекурсивной процедурой КСС, не принимающей параметров. Процедура использует стек T для хранения просматриваемых узлов и выделяет все компоненты сильной связности, достижимые из узла, выбранного в основном алгоритме.

Интуиция

Любой контур принадлежит ровно одной компоненте сильной связности (КСС). Если КСС содержит несколько узлов, то они обязательно входят в один или несколько контуров.

Обработка при обходе

Если при обходе в глубину мы попадаем в уже отмеченный узел w, это означает, что обнаружен контур. Предшествующие узлы этого контура находятся на стеке — от вершины стека до узла w, который также присутствует в стеке.

Склеивание

Все узлы найденного контура можно «склеить»:

- Список смежности узла w объединяется со списком смежности узла i.
- Список уже найденных узлов КСС, к которой принадлежит w, объединяется с соответствующим списком узла i.

После склеивания поиск в глубину продолжается от узла i, который остаётся в стеке.

18 Деревья. Свободные деревья. Основные свойства деревьев (с доказательствами). Код Прюфера.

Деревья и их свойства

Определения

- **Свободное дерево** — связный ациклический граф.
- **Древочисленный граф** — граф, для которого выполняется равенство:

$$q(G) = p(G) - 1,$$

где $p(G)$ — число вершин, $q(G)$ — число рёбер.

Теорема

Пусть $G(V, E)$ — граф с p вершинами, q рёбрами, k компонентами связности и z простыми циклами. Тогда любые два из следующих четырёх свойств:

1. связность;
2. ацикличность;
3. древочисленность ($q = p - 1$);
4. субцикличность ($z = 0$),

в совокупности характеризуют граф как дерево.

Характеризации графов

1. G — дерево, то есть связный граф без циклов:

$$k(G) = 1, \quad z(G) = 0$$

2. Любые две вершины соединены в G единственной простой цепью:

$$\forall u, v \in V, \quad |P(u, v)| = 1$$

3. G — связный граф, и каждое ребро является мостом:

$$k(G) = 1, \quad \forall e \in E, \quad k(G - e) > 1$$

4. G — связный и древочисленный граф:

$$k(G) = 1, \quad q(G) = p(G) - 1$$

5. G — ациклический и древочисленный граф:

$$z(G) = 0, \quad q(G) = p(G) - 1$$

6. G — ациклический и субциклический граф:

$$z(G) = 0, \quad z(G + x) = 1$$

7. G — связный, субциклический и неполный граф:

$$k(G) = 1, \quad G \neq K_p, \quad p \geq 3, \quad z(G + x) = 1$$

8. G — древочисленный и субциклический граф (за двумя исключениями):

$$q(G) = p(G) - 1 \wedge G \neq K_1 \cup K_3 \wedge G \neq K_2 \cup K_3 \wedge z(G + x) = 1$$

Следствия из свойств графов

Следствие 1. В любом нетривиальном дереве имеются по крайней мере две висячие вершины. Например, это могут быть концы диаметра дерева.

Следствие 2. Каждая невисячая вершина свободного дерева является точкой сочленения.

Следствие 3. Если в связном графе нет висячих вершин, то в нём обязательно существует цикл.

Код Прюфера

Алгоритм 9.1 Построение кода Прюфера свободного дерева

Вход: Дерево $T(V, E)$ в любом представлении, вершины дерева занумерованы числами $1..p$ произвольным образом.
Выход: Массив A : **array** $[1..p-1]$ **of** $1..p$ — код Прюфера дерева T.

```
for i from 1 to p − 1 do
  v := min {k ∈ V | d(k) = 1} { выбираем вершину v — висячую вершину с наименьшим номером }
  A[i] := Γ(v) { заносим в код номер единственной вершины, смежной с v }
  V := V − v { удаляем вершину v из дерева }
end for
```

Алгоритм 9.2 Распаковка кода Прюфера свободного дерева

Вход: Массив A : **array** $[1..p-1]$ **of** $1..p$ — код Прюфера дерева T.
Выход: Дерево $T(V, E)$, заданное множеством рёбер E, вершины дерева занумерованы числами $1..p$.

```
E := ∅ { в начале множество рёбер пусто }
B := 1..p { множество неиспользованных номеров вершин }
for i from 1 to p − 1 do
  v := min {k ∈ B | ∀j ⩾ i (k ≠ A[j])} { выбираем вершину v — неиспользованную вершину с наименьшим номером, который не встречается в остатке кода Прюфера }
  E := E + (v, A[i]) { добавляем ребро (v, A[i]) }
  B := B − v { удаляем вершину v из списка неиспользованных }
end for
```

19 Ориентированные, упорядоченные и бинарные деревья. Свойства ордерева. Эквивалентное определение ордерева. Упорядоченные деревья.

Ориентированные, упорядоченные и бинарные деревья. Свойства ор- дерева

Ориентированным деревом (или **ордеревом**, или **корневым деревом**) называется орграф $G = (V, E)$, обладающий следующими свойствами:

1. Существует единственный узел r, полустепень захода которого равна нулю:

$$d^+(r) = 0.$$

   Узел r называется **корнем** ордерева.

2. Полустепень захода всех остальных узлов равна единице:

$$\forall v \in V \setminus \{r\}, \quad d^+(v) = 1.$$

3. Каждый узел достижим из корня:

$$\forall v \in V \setminus \{r\}, \quad \exists \text{ ориентированный путь } (rv).$$

Теорема

Ориентированное дерево обладает следующими свойствами:

1. Число рёбер:

$$q = p - 1.$$

2. Если забыть ориентацию дуг, то получится свободное (неориентированное) дерево.
3. В ордереве нет контуров.
4. Для каждого узла существует единственный путь от корня к этому узлу.
5. Подграф, состоящий из узлов, достижимых из узла v, является ордеревом с корнем v (такое ордерево называется **поддеревом** узла v).
6. Если в свободном дереве любую вершину назначить корнем, то получится ордерево.

Эквивалентное определение ордерева. Упорядоченные деревья

Ордерево T — это непустое конечное множество узлов, на котором задано разбиение, обладающее следующими свойствами:

1. Существует один выделенный одноэлементный блок $\{r\}$, называемый **корнем** данного ордерева.
2. Остальные узлы (исключая корень) содержатся в k блоках $T_1, T_2, \ldots, T_k$, где $k \geq 0$. Каждый из блоков T_i является ордеревом и называется **поддеревом**.

Таким образом, ордерево можно записать как:

$$T \stackrel{\text{def}}{=} \{\{r\}, T_1, T_2, \ldots, T_k\}.$$

Если относительный порядок поддеревьев $T_1, T_2, \ldots, T_k$ фиксирован, то ордерево называется **упорядоченным**.

20 Представление деревьев в ЭВМ. Обходы бинарных деревьев. Алгоритм симметричного обхода бинарного дерева

Представление бинарных деревьев в ЭВМ

Существует несколько способов представления бинарных деревьев в памяти ЭВМ:

1. Списочные структуры

Каждый узел представляется записью типа N, содержащей:

- поле i — информация, связанная с узлом;
- указатели l и r — на левый и правый дочерние узлы соответственно.

Тип данных:

$$N = \text{record } i : \text{info};\ l, r :\uparrow N \text{ end record}$$

Дерево представляется указателем на корень. Объём памяти:

$$n(p) = 3p$$

2. Упакованные массивы

Все узлы размещаются в массиве так, чтобы узлы поддерева следовали за текущим узлом. Каждый узел хранит индекс первого узла правого поддерева.

Тип данных:

$$T : \text{array } [1..p] \text{ of record } i : \text{info},\ k :\uparrow \text{ end record}$$

Объём памяти:

$$n(p) = 2p$$

3. Польская запись

Вместо указателей фиксируется «степень» узла:

- 0 — лист;
- 1 — только левый потомок;
- 2 — только правый потомок;
- 3 — оба потомка.

Тип данных:

$$T : \text{array } [1..p] \text{ of record } i : \text{info},\ d : 0..3 \text{ end record}$$

Объём памяти:

$$n(p) = 2p$$

Обходы бинарных деревьев

Существует три основных способа обхода бинарного дерева:

1. Прямой обход (префиксный, левый) preorder

- Посетить корень.
- Обойти левое поддерево.
- Обойти правое поддерево.

2. Внутренний обход (инфиксный, симметричный) inorder

- Обойти левое поддерево.
- Посетить корень.
- Обойти правое поддерево.

3. Концевой обход (постфиксный, правый) postorder

- Обойти левое поддерево.
- Обойти правое поддерево.
- Посетить корень.

Кроме этих трёх, возможны ещё три варианта обхода, отличающиеся порядком рассмотрения левого и правого поддеревьев. Все возможные обходы исчерпываются, если в представлении фиксированы только дуги, ведущие от отцов к сыновьям.

Алгоритм симметричного обхода бинарного дерева

Вход: бинарное дерево, представленное списочной структурой, r — указатель на корень.
Выход: последовательность узлов бинарного дерева в порядке симметричного обхода.

$T := \varnothing; p := r$ { вначале стек пуст и p указывает на корень дерева }
M : { анализируем узел, на который указывает p }
if $p =$ **nil then**
  if $T = \varnothing$ **then**
    stop { обход закончен }
  end if
  $p \leftarrow T$ { левое поддерево обойдено }
  yield p { очередной узел при симметричном обходе }
  $p := p.r$ { начинаем обход правого поддерева }
else
  $p \rightarrow T$ { запоминаем текущий узел... }
  $p := p.l$ { ... и начинаем обход левого поддерева }
end if
goto M

21 Деревья сортировки. Ассоциативная память, способы реализации ассоциативной памяти. Алгоритм поиска в дереве сортировки.

Деревья сортировки

В данном разделе рассматривается одно конкретное применение деревьев в программировании — **деревья сортировки** (также называемые **деревьями упорядочивания**).

Анализ охватывает как теоретические аспекты, например оценку высоты деревьев, так и практическую реализацию соответствующих алгоритмов. Кроме того, обсуждаются различные прагматические аспекты использования деревьев сортировки и затрагиваются некоторые смежные вопросы.

Ассоциативная память

В практическом программировании часто используется механизм организации хранения и доступа к данным, называемый **ассоциативной памятью**.

При использовании ассоциативной памяти данные разбиваются на **записи**, каждая из которых ассоциирована с **ключом**. Ключ — это значение из упорядоченного множества, а записи могут иметь произвольную природу и различные размеры. Доступ к данным осуществляется по значению ключа, которое обычно выбирается простым, компактным и удобным для работы.

Примеры использования

Ассоциативная память применяется во многих сферах:

- **Словарь или энциклопедия:** Ключ — заголовок статьи (обычно выделен жирным), запись — текст статьи.
- **Адресная книга:** Ключ — имя контакта, запись — адресная информация (телефон, почтовый адрес и т.д.).
- **Банковские счета:** Ключ — номер счёта, запись — финансовая информация (возможно, весьма сложная).

Основные операции

Ассоциативная память должна поддерживать как минимум три базовые операции:

1. `Add(key, record)` — добавление записи по ключу;
2. `Find(key)` — поиск записи по ключу;
3. `Delete(key)` — удаление записи по ключу.

Эффективность каждой операции зависит от структуры данных, используемой для представления ассоциативной памяти. Общая эффективность определяется соотношением частот различных операций в конкретной программе.

Способы реализации ассоциативной памяти

Для представления ассоциативной памяти используются следующие основные структуры данных:

1. **Неупорядоченный массив**
2. **Упорядоченный массив**
3. **Дерево сортировки** Бинарное дерево, каждый узел которого содержит ключ и указатель на запись. Свойство: значения ключей в узлах левого поддерева меньше, а в узлах правого — больше, чем в текущем узле.
4. **Таблица расстановки (хэш-таблица)** Использует хеш-функцию для быстрого доступа к записям по ключу.

Основное внимание в этом разделе уделено алгоритмам выполнения операций с деревом сортировки.

22 Выровненные, заполненные и полные деревья. Сбалансированные деревья. Алгоритм бинарного (двоичного) поиска.

Типы бинарных деревьев

- **Выровненное бинарное дерево** — все листья находятся на одном (последнем) уровне.
- **Заполненное бинарное дерево** — все узлы, степень которых меньше максимальной, располагаются на одном или двух последних ярусах. То есть все ярусы, кроме последнего, полностью заполнены.
- **Полное бинарное дерево** — все ярусы заполнены, все узлы имеют полустепень исхода 2, листья находятся на последнем уровне. Такое дерево содержит 2^{h-1} узлов, где h — высота дерева.
- **Сбалансированное дерево** — для любого узла высота левого и правого поддеревьев отличается не более чем на единицу. Возможна также балансировка по весу. Сбалансированные деревья менее эффективны по времени поиска, чем полные, но проще в реализации операций вставки и удаления.

Алгоритм бинарного поиска

В отсортированном массиве:

1. Выбирается середина.
2. Сравнивается искомое значение с серединным.
3. В зависимости от результата поиск продолжается в левой или правой половине массива.

Алгоритм 9.7 Бинарный поиск

Вход: упорядоченный массив A : **array** $[1..n]$ **of record** $k : key; i : info$ **end record**; ключ $a : key$.
Выход: индекс записи с искомым ключом a в массиве A или 0, если записи с таким ключом нет.

```
b := 1 { начальный индекс части массива для поиска }
e := n { конечный индекс части массива для поиска }
while b ⩽ e do
  c := (b + e)/2 { индекс проверяемого элемента (округленный до целого) }
  if A[c].k > a then
    e := c − 1 { продолжаем поиск в первой половине }
  else if A[c].k < a then
    b := c + 1 { продолжаем поиск во второй половине }
  else
    return c { нашли искомый ключ }
  end if
end while
return 0 { искомого ключа нет в массиве }
```

23 Информационные деревья. A- и B-деревья. Красно-черные деревья.

A-дерево

A-дерево — это разновидность сбалансированного дерева поиска, разработанная как альтернатива B-дереву для эффективной работы с большими объёмами данных в системах хранения.

Основные особенности

- A-дерево оптимизировано для хранения на внешних носителях (например, дисках), минимизируя количество операций ввода-вывода.
- В отличие от B-дерева, A-дерево использует **буферизацию** в узлах, позволяя накапливать операции и выполнять их пакетно.
- Каждый узел содержит буфер, в котором временно хранятся вставки, удаления и обновления, прежде чем они будут распространены вниз по дереву.
- Это позволяет значительно сократить количество обращений к диску при массовых операциях.

Структура узла

Каждый узел A-дерева содержит:

- Упорядоченный список ключей.
- Буфер операций (вставка, удаление, обновление).
- Указатели на дочерние узлы.

Преимущества

- Эффективность при пакетной обработке данных.
- Подходит для систем с ограниченным доступом к памяти и высокой стоимостью операций ввода-вывода.
- Хорошо масштабируется при работе с большими индексами.

Недостатки

- Более сложная реализация по сравнению с классическим B-деревом.
- Задержка при выполнении отдельных операций из-за буферизации.

B-дерево

Общие сведения

B-дерево — это сбалансированное, сильно ветвистое дерево, широко применяемое для организации индексов в системах управления базами данных (СУБД).

Разновидности

- **B-дерево:** данные хранятся в любом узле.
- **B*-дерево:** данные хранятся в любом узле, но узлы заполняются на две трети.
- **B+-дерево:** данные хранятся только в листьях, во внутренних узлах — копии ключей.

Свойства B-дерева

B-деревом называется дерево, удовлетворяющее следующим условиям:

1. Ключи в каждом узле упорядочены для быстрого доступа. Корень содержит от 1 до $2t - 1$ ключей. Любой другой узел содержит от $t - 1$ до $2t - 1$ ключей. Листья не являются исключением. Здесь $t \geq 2$ — параметр дерева.
2. Листья не имеют потомков. Узел с ключами $K_1, \ldots, K_n$ имеет $n + 1$ потомков. Потомки содержат ключи, попадающие в интервалы между ключами узла. Иначе говоря, каждый внутренний узел можно представить как упорядоченный список, в котором чередуются ключи и указатели на потомков.
3. Все листья находятся на одном уровне, то есть имеют одинаковую глубину.

Красно-чёрное дерево

Красно-чёрным деревом называется бинарное поисковое дерево, в котором каждому узлу сопоставлен дополнительный атрибут — цвет (красный или чёрный), и выполняются следующие свойства:

1. Каждый узел промаркирован красным или чёрным цветом.
2. Корень дерева и все листья (конечные узлы) — чёрные.
3. У красного узла родитель обязательно чёрный.
4. Все простые пути от любого узла x до листьев содержат одинаковое количество чёрных узлов.
5. Чёрный узел может иметь чёрного родителя.

24 Кратчайший остов. Алгоритм построения остова экстремального веса. Алгоритм Краскала. Алгоритм Прима. Алгоритм Борувки. Число остовов в связном обыкновенном графе. Задача Штейнера

Кратчайший остов

Остовный подграф — это подграф, содержащий все вершины исходного графа. Остов задаётся множеством рёбер, поскольку вершины те же. Если рёбрам заданы длины (веса), то возникает задача нахождения *кратчайшего остова* — остовного дерева минимального веса.

Алгоритмы построения кратчайшего остова

Существуют следующие классические алгоритмы:

1. **Алгоритм Краскала** Сложность: $O(q \cdot \log q)$
   - Берутся все вершины, но при этом множество рёбер берётся пустым.
   - Рёбра графа сортируются по их весу.
   - Выбирается ребро с минимальным весом и если его добавление не вызовет появление цикла, то оно добавляется. Далее берётся следующее ребро.
   - Повторяется пока есть рёбра не вставленные рёбра, которые не вызовут появление цикла.
2. **Алгоритм Прима** Сложность: $O(p^2)$ или $O(q + p \cdot \log p)$
   - Выбираем произвольную вершину.
   - Находим минимальное ребро, инцидентное этой вершине, и добавляем его в дерево.
   - Далее выбираем минимальное ребро, соединяющее дерево с новой вершиной.
   - Продолжаем, пока все вершины не включены в дерево.
3. **Алгоритм Борувки** Сложность: $O(q \cdot \log p)$
   - Каждая вершина представляется как отдельное дерево.
   - Для каждого дерева находим самое дешёвое ребро, связывающее это дерево с другим деревом.
   - Соединяем деревья найденным ребром.
   - Повторяем шаги 23 пока не останется только одно дерево.

Примечание: Алгоритмы Прима и Краскала можно использовать также для нахождения *максимального остова*, если изменить критерий выбора рёбер.

Число остовов в связном графе

Теорема Кирхгофа. Число остовных деревьев в связном графе G порядка $n \geq 2$ равно алгебраическому дополнению любого элемента матрицы Кирхгофа $B(G)$.

Матрица Кирхгофа

Матрица Кирхгофа определяется как разность матрицы степеней и матрицы смежности:

$$B(G) = \begin{pmatrix} \deg(v_1) & 0 & \cdots & 0 \\ 0 & \deg(v_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \deg(v_n) \end{pmatrix} - A(G)$$

где $A(G)$ — матрица смежности графа G.

Число остовов

Число остовных деревьев графа G равно любому *алгебраическому дополнению* элемента матрицы $B(G)$, то есть:

$$t(G) = \mathrm{cof}_{ij}(B(G)), \quad \text{для любых } i, j \in \{1, \ldots, n\}$$

Задача Штейнера

Формулировка задачи

Пусть задано множество городов на плоскости. Требуется определить минимальный набор железнодорожных линий (по суммарной длине), который обеспечит возможность переезда из любого города в любой другой.

Важно: кратчайший остов полного графа расстояний между городами не является решением задачи Штейнера, поскольку допускается введение дополнительных промежуточных точек.

Евклидова задача Штейнера

Пусть задано произвольное множество U точек на евклидовой плоскости. Необходимо соединить их непрерывными линиями так, чтобы любые две точки были связаны либо напрямую, либо через другие точки и соединяющие их линии, а суммарная длина всех линий была минимальной.

Допускается введение дополнительных точек (вершин), не входящих в исходное множество U, что отличает задачу Штейнера от задачи построения остовного дерева.

Графовая формулировка

Задача Штейнера эквивалентна задаче нахождения остова минимального веса в порождённых подграфах графа G, множество вершин которых содержит U.

Сложность

Задача Штейнера является NP-трудной и до конца не решена в общем случае. Для её решения применяются эвристики, аппроксимации и специальные алгоритмы для ограниченных классов графов.

25 Фундаментальные циклы и разрезы. Фундаментальная система циклов и циклический ранг. Фундаментальная система разрезов и коциклический ранг. Подпространства циклов и коциклов

Рассматриваются только **связные графы**.

Циклы и разрезы

Циклы

Цикл может входить только в одну компоненту связности графа $G(V, E)$. В несвязном графе понятие разреза является вырожденным, поэтому без ограничения общности в этом разделе граф $G(V, E)$ считается связным.

Цикл не может содержать одно и то же ребро более одного раза, поэтому в данном контексте цикл рассматривается как *множество рёбер*. В связи с этим можно дать эквивалентное определение:

Простым циклом называется такой цикл, никакое собственное подмножество которого не является циклом.

Разрез

Разрезом называется множество рёбер, удаление которых делает граф несвязным. Любое разбиение множества вершин V на два непустых подмножества V_1 и V_2 определяет разрез:

$$S := \{(v_1, v_2) \in E \mid v_1 \in V_1,\ v_2 \in V_2\}$$

Очевидно, что множества V_1 и V_2 определяют друг друга:

$$V_1 = V \setminus V_2, \quad V_2 = V \setminus V_1$$

Поэтому достаточно задать одно из них:

$$\forall U \subset V, \quad \overline{U} \stackrel{\text{def}}{=} V \setminus U$$

Обозначение

Обозначим через $E(V_1, V_2)$ множество рёбер, соединяющих два дизъюнктивных непустых подмножества вершин графа $G = (V, E)$:

$$E(V_1, V_2) \stackrel{\text{def}}{=} \{(v_1, v_2) \in E \mid v_1 \in V_1,\ v_2 \in V_2\}$$

Правильный разрез

Разрез связного графа $G(V, E)$, определяемый непустым подмножеством $U \subset V$, называется **правильным разрезом** и обозначается:

$$S(U) \stackrel{\text{def}}{=} E(U, \overline{U})$$

Правильный разрез не содержит лишних рёбер и соответствует минимальному разделению графа по вершинам.

Система циклов и ранг

Пусть $T(V, E_T)$ — остов графа $G(V, E)$. **Кодеревом** $T^*(V, E_T^*)$ остова T называется остовный подграф, такой что:

$$E_T^* = E \setminus E_T$$

Кодерево не является деревом. Рёбра кодерева называются *хордами* остова.

Каждая хорда $e \in E_T^*$ порождает ровно один простой цикл, обозначаемый Z_e. Таким образом, множество таких циклов образует **фундаментальную систему циклов**:

$$\mathcal{Z} \stackrel{\text{def}}{=} \{Z_e\}_{e \in T^*}$$

Количество циклов в фундаментальной системе называется **циклическим рангом** графа и обозначается $m(G)$.

Теорема

Любой цикл в связном графе можно представить как *симметрическую разность* нескольких фундаментальных циклов, определяемых произвольным остовом T.

Циклический ранг графа равен числу хорд остова:

$$m(G) = q - p + 1$$

где $q = |E|$ — число рёбер, $p = |V|$ — число вершин графа G.

Фундаментальная система разрезов и коциклический ранг

Пусть $T(V, E_T)$ — остов графа $G(V, E)$. Рассмотрим ребро $e \in E_T$ и определим разрез S_e следующим образом.

Ребро e является мостом в дереве T, поэтому его удаление разбивает множество вершин V на два непустых подмножества V_1 и V_2. Включим в разрез S_e само ребро e и все хорды остова T, соединяющие вершины из V_1 с вершинами из V_2:

$$S_e \stackrel{\text{def}}{=} \{e\} \cup \{(v_1, v_2) \in T^* \mid v_1 \in V_1,\ v_2 \in V_2\} = E(V_1, V_2)$$

Такой разрез S_e называется **простым разрезом**.

Множество всех таких разрезов, порождённых рёбрами остова, образует **фундаментальную систему разрезов**:

$$\mathcal{S} \stackrel{\text{def}}{=} \{S_e\}_{e \in T}$$

Количество разрезов в фундаментальной системе называется **коциклическим рангом** графа и обозначается $m^*(G)$.

Теорема

Любой *правильный разрез* в связном графе можно представить как *симметрическую разность* нескольких фундаментальных разрезов, определяемых произвольным остовом T.

Формула ранга

Число разрезов в фундаментальной системе равно числу рёбер остова:

$$m^*(G) = p - 1$$

где $p = |V|$ — число вершин графа G.

Подпространства циклов и коциклов

Рассмотрим граф $G(V, E)$, где множество рёбер E интерпретируется как векторное пространство над бинарной арифметикой (по модулю 2). Элементы этого пространства можно отождествить с подмножествами рёбер: ребро входит в подмножество тогда и только тогда, когда его коэффициент в линейной комбинации равен 1.

Операции

Сложение векторов определяется как **симметрическая разность** множеств рёбер:

$$A \oplus B := (A \cup B) \setminus (A \cap B)$$

Циклическое и коциклическое подпространства

Множества циклов и правильных разрезов замкнуты относительно симметрической разности, поэтому они образуют подпространства в пространстве всех подмножеств рёбер графа.

- Циклы рассматриваются как *циклические векторы*.
- Правильные разрезы — как *коциклические векторы*.

Базисы и ранги

- **Независимая система циклов** — такая система $\{Z_i\}$, в которой ни один цикл не выражается как линейная комбинация остальных.
- Максимальная независимая система циклов образует **базис** циклического подпространства.
- **Фундаментальная система циклов**, порождённая остовом, является таким базисом.
- **Циклический ранг** графа $m(G)$ — это размерность циклического подпространства.
- Аналогично, максимальная независимая система правильных разрезов образует **базис** коциклического подпространства.
- **Фундаментальная система разрезов**, порождённая остовом, является таким базисом.
- **Коциклический ранг** графа $m^*(G)$ — это размерность коциклического подпространства.

26 Эйлеровы циклы. Эйлеровы графы. Алгоритм Флери. Оценка числа эйлеровых графов.

Эйлеровы циклы. Эйлеровы графы

Если граф содержит цикл (не обязательно простой), проходящий по всем рёбрам графа, то такой цикл называется **эйлеровым циклом**, а сам граф — **эйлеровым графом**.

Теорема

Пусть граф G связен и нетривиален. Тогда следующие утверждения эквивалентны:

1. G — эйлеров граф.
2. Каждая вершина графа G имеет чётную степень.
3. Множество рёбер графа G можно разбить на простые циклы.

Алгоритм построения эйлерова цикла (Флери)

Пусть граф G — эйлеров, то есть связен и все вершины имеют чётную степень. Алгоритм Флери позволяет построить эйлеров цикл следующим образом:

1. Инициализируем стек и помещаем в него произвольную вершину v.
2. Пока стек не пуст:
   (a) Берём вершину v с вершины стека.
   (b) Если у v нет смежных рёбер:
   - Удаляем v из стека и добавляем её в выходной массив (последовательность цикла).

   (c) Иначе:
   - Выбираем смежную вершину u.
   - Добавляем u в стек.
   - Удаляем ребро (u, v) из графа.

В результате получаем последовательность вершин, задающую эйлеров цикл.

Оценка числа эйлеровых графов

Обозначим:

- $\mathcal{G}(p)$ — множество всех графов с p вершинами;
- $\mathcal{E}(p)$ — множество эйлеровых графов с p вершинами.

Теорема

Эйлеровых графов *почти нет*, то есть:

$$\lim_{p \to \infty} \frac{|\mathcal{E}(p)|}{|\mathcal{G}(p)|} = 0$$

Таким образом, при большом числе вершин доля эйлеровых графов среди всех возможных стремится к нулю.

27 Гамильтоновы циклы. Теорема Дирака. Задача коммивояжера.

Гамильтоновы циклы. Теорема Дирака

Граф называется **гамильтоновым**, если существует простой цикл, проходящий через все вершины графа ровно один раз. Такой цикл называется **гамильтоновым циклом**.

Теорема Дирака

Пусть граф G имеет p вершин и минимальную степень $\delta(G)$. Если выполнено условие:

$$\delta(G) \geq \frac{p}{2}$$

то граф G является гамильтоновым.

Примечание: Простых необходимых и достаточных условий гамильтоновости не существует, однако теорема Дирака даёт полезный критерий.

Задача коммивояжера

Имеется p городов, расстояния между которыми известны. Коммивояжер должен посетить каждый город ровно один раз и вернуться в исходный. Требуется найти маршрут, минимизирующий суммарное расстояние.

Это задача нахождения **кратчайшего гамильтонова цикла** в нагруженном полном графе.

Сложность: Задача является NP-трудной. Точный алгоритм возможен только методом полного перебора. На практике применяются эвристические и аппроксимационные методы.

28 Гиперграфы. Двойственные гиперграфы. Циклы и реализации.

Гиперграфы. Двойственные гиперграфы

Определение гиперграфа

Гиперграф — это обобщение графа, в котором рёбра могут соединять произвольные подмножества вершин, а не только пары.

Пусть:

- V — конечное непустое множество вершин;
- E — семейство непустых подмножеств множества V;

Тогда пара (V, E) называется **гиперграфом**.

Обозначения:

- VH — множество вершин гиперграфа H;
- EH — множество рёбер гиперграфа H;
- $|H| = n$ — число вершин;
- $|EH| = m$ — число рёбер;
- H называется (n, m)-гиперграфом.

Двойственный гиперграф

Пусть $H(V, E)$ — гиперграф без изолированных вершин. Его **двойственным гиперграфом** называется гиперграф $H^*(V^*, E^*)$, где:

- $V^* = E$;
- $E^* = \{E(v) \mid v \in V\}$ — семейство множеств рёбер, содержащих вершину v.

Циклы и реализация

Циклы в гиперграфе

Гиперграф не содержит циклов тогда и только тогда, когда для любого непустого подмножества $V' \subseteq V$ выполнено:

$$\left| \bigcup_{v \in V'} \delta(v) \right| > \sum_{v \in V'} (\delta(v) - 1)$$

где $\delta(v)$ — множество рёбер, содержащих вершину v.

Реализация гиперграфа

Граф G называется **реализацией** гиперграфа $H(V, E)$, если выполняются следующие условия:

1. $VG = VH$;
2. Каждое ребро графа G содержится в некотором ребре гиперграфа H;
3. Для любого ребра $e \in E$ подграф G_e является связным и $VG_e = e$.

Любая реализация гиперграфа является объединением реализаций его рёбер.

29 Задачи маршрутизации. VRP. Классификация. Методы решения задач VRP.

Задачи маршрутизации (VRP)

Задачи маршрутизации актуальны в следующих областях:

1. Сервисное обслуживание;
2. Перевозка товаров;
3. Общественный транспорт.

Транспортная задача (Vehicle Routing Problem, VRP) формулируется на графе $G(V, E, C)$, где:

- V — множество вершин (точек доставки или клиентов);
- E — множество рёбер (маршрутов между точками);
- C — матрица стоимостей, определённая на множестве рёбер E.

Цель задачи VRP — минимизация общей стоимости при построении множества маршрутов, удовлетворяющих заданным ограничениям.

Классификация задач VRP

- Транспортная задача с учетом грузоподъемности (Capacitated VRP): транспортное средство имеет ограниченную грузоподъемность; Разновидностями задачи CVRP являются задачи 2L-CVRP и 3L-CVRP, в которых помимо грузоподъемности ТС учитывается размещение груза внутри ТС. В задаче 2L-CVRP — двумерное размещение, в задаче 3L-CVRP — трехмерное размещение.
- Транспортная задача с временными окнами (VRP with Time Windows, VRPTW): каждый клиент или заявка должны быть обслужены в определенный временной промежуток;
- Задача маршрутизации ТС с множеством депо (Multiple Depot Vehicle Routing Problem, MDVRP): имеется несколько депо для обслуживания клиентов.
- Задача маршрутизации ТС с раздельной доставкой (Split Delivery Vehicle Routing Problem, SDVRP): каждый клиент может быть обслужен одновременно несколькими ТС.
- Транспортная задача с пополнением и доставкой (VRP with Pick-up and Delivery, PDP): товары должны быть пополнены и доставлены в определенные точки;

Методы решения задач VRP

1. Перебор всех допустимых (пример: Метод ветвей и границ)
2. Эвристические (пример: алгоритм заметания — группируем в кластеры и для каждого кластера решаем)

Три вида эвристик:

1. Конструктивный (сначала для одной, потом для двух, потом т.д.)
2. Двухфазный (группируем в кластеры)
3. Улучшающие (берем любое и улучшаем)

30 Независимые и покрывающие множества вершин и ребер. Теорема о связи чисел независимости и покрытий

Независимые и покрывающие множества вершин и рёбер

- Множество вершин называется **независимым**, если никакие две вершины из него не смежны. Максимальное число вершин в независимом множестве называется **числом независимости по вершинам** и обозначается β_0.
- Множество рёбер называется **независимым** (или **паросочетанием**), если никакие два ребра не смежны. Максимальное число рёбер в независимом множестве называется **числом независимости по рёбрам** и обозначается β_1.
- Вершина **покрывает** свои инцидентные рёбра, а ребро — свои инцидентные вершины.
- Множество вершин, покрывающее все рёбра графа, называется **покрывающим множеством вершин**. Минимальное число таких вершин называется **числом покрытия по вершинам** и обозначается α_0.
- Множество рёбер, покрывающее все вершины графа, называется **покрывающим множеством рёбер**. Минимальное число таких рёбер называется **числом покрытия по рёбрам** и обозначается α_1.

Теорема о связи чисел независимости и покрытий

Для любого нетривиального графа выполняется равенство:

$$\alpha_0 + \beta_0 = \alpha_1 + \beta_1$$

31 Построение независимых множеств вершин. Поиск с возвратами. Улучшенный перебор. Доминирующие множества. Доминирование и независимость. Задача о наименьшем покрытии.

Построение независимых множеств вершин. Поиск с возвратами

Пусть задан граф $G(V, E)$. Требуется найти такое множество вершин $X \subset V$, что:

$$w(X) = \max_{Y \in \mathcal{E}} w(Y)$$

где:

$$w(Y) \stackrel{\text{def}}{=} |Y|, \quad \mathcal{E} \stackrel{\text{def}}{=} \{Y \subset V \mid \forall u, v \in Y,\ (u, v) \notin E\}$$

То есть X — максимальное по мощности независимое множество вершин.

Методы решения

- Полный перебор всех подмножеств 2^V с проверкой условия независимости.
- Поиск с возвратами (backtracking) — перебор с отсечениями, позволяющий избежать рассмотрения заведомо неподходящих конфигураций.

Алгоритм 10.2 Поиск с возвратами

Вход: граф $G(V, E)$.
Выход: наибольшее независимое множество X.

```
m := 0 { наилучшее известное значение β0 }
X := ∅ { наибольшее известное независимое множество X }
BT(∅, V) { вызов рекурсивной процедуры BT }
```

Основная работа выполняется рекурсивной процедурой BT.

Вход: S — текущее независимое множество вершин, T — оставшиеся вершины графа.
Выход: изменение глобальной переменной X, если текущее множество не может быть расширено (является максимальным).

```
for v ∈ T do
  if S + v ∈ 𝓔 then
    if |S| > m then
      X := S; m := |S| { наибольшее известное независимое множество }
    end if
    BT(S + v, T \ Γ*(v)) { пробуем добавить v }
  end if
end for
```

Обоснование По построению вершина v добавляется в множество S только при сохранении независимости расширенного множества. В алгоритме это обстоятельство указано в форме условия $S + v \in \mathcal{E}$. Проверить сохранение условия независимости нетрудно, например, с помощью следующей функции.

Вход: независимое множество S и проверяемая вершина v.
Выход: **true**, если множество $S + v$ независимое, **false** — в противном случае.

```
for u ∈ S do
  if (u, v) ∈ E then
    return false { множество S + v зависимое }
  end if
end for
return true { множество S + v независимое }
```

Этот цикл не включен в явном виде в рекурсивную процедуру BT, чтобы не загромождать основной текст и не затуманивать идею поиска с возвратами. Таким образом, множество S, а следовательно, и множество X — независимые. В тот момент, когда множество S нельзя расширить, оно максимально по определению. Переменная m глобальна, поэтому среди всех максимальных независимых множеств в конце работы алгоритма построенное множество X является наибольшим независимым множеством вершин. □

Улучшенный перебор

Идея: начинаем с пустого множества и пополняем его вершинами с сохранением независимости (пока возможно). За подробностями пойдёт нахуй.

Доминирующие множества

Множество вершин S графа $G(V, E)$ называется **доминирующим**, если:

$$S \cup \Gamma(S) = V$$

Очевидно, что множество S доминирует тогда и только тогда, когда:

$$\forall v \notin S, \quad \Gamma(v) \cap S \neq \emptyset$$

Доминирующее множество называется:

- **Минимальным**, если никакое его подмножество не является доминирующим;
- **Наименьшим**, если его мощность минимальна среди всех доминирующих множеств.

Доминирование и независимость

Теорема. Независимое множество вершин является **максимальным** тогда и только тогда, когда оно является **доминирующим**.

Задача о наименьшем покрытии

Каждой вершине сопоставлена некоторая цена. Требуется выбрать доминирующее множество с наименьшей суммарной ценой — это **задача о наименьшем покрытии**.

Для её решения применяются переборные алгоритмы с теми или иными улучшениями.

32 Ядро графа. Алгоритм Магу. Спектры графов.

32.1. Ядро графа

Ядро графа — это независимое доминирующее множество.

- В полном графе каждая вершина является ядром.
- Любой граф имеет ядро.

Свойства ядра

1. Не может содержать петель или смежных вершин.
2. Максимальное внутренне устойчивое подмножество.
3. Одновременно максимально внутренне устойчивое и минимально внешне устойчивое.
4. Граф без контуров всегда обладает ядром.
5. Каждый орграф, не имеющий контуров нечётной длины, обладает ядром.
6. Граф, не имеющий контуров нечётной длины, допускает ядро.

Независимое множество — внутренне устойчивое. **Доминирующее** множество — внешне устойчивое.

32.2. Алгоритм Магу

Для нахождения множества внутренней устойчивости (независимого)

1. Составляется матрица смежности.
2. По таблице смежности выписываются парные дизъюнкции.
3. Выражение приводится к ДНФ.
4. Для любой элементарной конъюнкции выписываются недостающие элементы — они образуют множество внутренней устойчивости.

Для нахождения множества внешней устойчивости (доминирующего)

1. Составляется матрица смежности.
2. Дополняется единицами по главной диагонали.
3. Для каждой строки выписываются дизъюнкции.
4. Выражение приводится к ДНФ.
5. Все вершины, входящие в элементарную конъюнкцию, образуют множество внешней устойчивости.

Спектры графов

Спектр графа — это множество собственных значений матрицы смежности графа.

Графы, не являющиеся изоморфными, но имеющие одинаковые характеристические многочлены, называются **коспектральными**.

Спектральные свойства графов находят применение в **квантовой химии**.

33 Разметка графа. Грациозная, счастливая разметки. Раскраска графа. Примеры задач. Хроматическое число. Алгоритмы раскрашивания. Двойственный граф

Разметка графа. Грациозная, счастливая разметки

Разметка — это функция из множества вершин/рёбер графа в множество меток.

Грациозная разметка

Грациозная разметка вершин: Если все вершины графа помечены числами от 0 до q, и эта разметка порождает рёберную разметку от 1 до q.

Иначе говоря, для любого ребра между двумя вершинами модуль разности их меток должен быть равен метке ребра:

$$\text{метка ребра} = |\text{метка вершины}_1 - \text{метка вершины}_2|$$

Счастливая разметка

Счастливая разметка: Назначаем положительные целые числа вершинам так, чтобы сумма меток соседних вершин соответствовала раскраске графа.

Раскраска графа

Цель раскраски — назначить цвета вершинам графа так, чтобы никакие две смежные вершины не имели одинаковый цвет.

- **Хроматическое число** графа $\chi(G)$ — минимальное количество цветов, необходимое для корректной раскраски вершин.
- **Хроматический индекс** графа $\chi'(G)$ — минимальное количество цветов, необходимое для корректной раскраски рёбер.

Примеры задач

1. Составление расписаний: лекции, которые нельзя проводить одновременно, соединяются рёбрами.
2. Распределение оборудования: аналогично, конфликты моделируются рёбрами.
3. Раскраска карты: соседние страны должны иметь разные цвета.

Алгоритмы раскрашивания

- Выбираем максимальное независимое множество — раскрашиваем его, затем удаляем эти вершины и повторяем.
- Задача раскраски является NP-полной.
- Возможна эвристика: начинать с вершин с наибольшей степенью.

Двойственный граф

В двойственном графе:

- Вершины соответствуют граням исходного графа;
- Рёбра проводятся между вершинами, если соответствующие грани имеют общее ребро.

34 Планарность. Укладка графов. Эйлерова характеристика. Теорема о пяти красках.

Укладка графов

Уложить граф на поверхности — значит изобразить его так, чтобы рёбра не пересекались (кроме в вершинах). Если это возможно на плоскости, граф называется **планарным**.

- Планарность — возможность укладки графа на плоскости без пересечений рёбер.
- Любой граф можно вложить в трёхмерное пространство без пересечений.

Эйлерова характеристика

Для связного планарного графа справедливо соотношение Эйлера:

$$p - q + f = 2$$

где:

- p — число вершин;
- q — число рёбер;
- f — число граней (областей, на которые разбивается плоскость).

Это соотношение используется для проверки планарности и анализа топологических свойств графа.

Теорема о пяти красках

Теорема: Всякий планарный граф можно раскрасить не более чем в 5 цветов так, чтобы никакие две смежные вершины не имели одинаковый цвет.

Дополнение:

- Теорема о четырёх красках утверждает, что достаточно 4 цветов — доказана с помощью компьютерной проверки.
- Теорема о пяти красках имеет классическое доказательство и используется как надёжная оценка сверху.

Применения

- Раскраска карт: соседние регионы должны иметь разные цвета.
- Составление расписаний: конфликты моделируются рёбрами.
- Планирование частот в сетях: соседние узлы не должны использовать одинаковые частоты.

35 Элементы сетевого планирования: критические пути, работы, резервы. Линейные графики. Алгоритм сетевого планирования (составлял гпт)

Элементы сетевого планирования

Критические пути

Критический путь — это последовательность работ, определяющая минимальную длительность выполнения всего проекта. Любое изменение длительности работ на критическом пути напрямую влияет на срок завершения проекта.

- Критический путь — путь с нулевым полным резервом.
- Может быть не единственным.
- Определяется по графу проекта.

Работы

Работа — элемент проекта, имеющий продолжительность и связывающий две события (вехи). В сетевом графе представляется ориентированным ребром.

- Каждая работа имеет раннее и позднее время начала и окончания.
- Работы могут быть реальными (физическими) и фиктивными (логическими связями).

Резервы времени

- **Полный резерв** — максимальное время, на которое можно задержать выполнение работы без изменения срока завершения проекта.
- **Свободный резерв** — время, на которое можно задержать работу без влияния на раннее начало последующих работ.
- **Нулевой резерв** — признак принадлежности работы к критическому пути.

Линейные графики

Линейный график — способ визуализации выполнения работ во времени. Оси: горизонтальная — время, вертикальная — работы.

- Позволяет оценить загрузку ресурсов.
- Удобен для строительных и производственных процессов.

Алгоритм сетевого планирования

1. Построение сетевого графа: определение событий и работ.
2. Расчёт ранних сроков начала и окончания работ.
3. Расчёт поздних сроков начала и окончания работ.
4. Вычисление резервов времени.
5. Определение критического пути.
6. Построение календарного плана и линейного графика.