

# Алгоритмы и структуры данных.

Андрей Тищенко @AndrewTGk

2024/2025

## Содержание

<b>1</b>	<b>Структуры данных</b>	<b>1</b>
1.1	Линейные структуры данных . . . . .	1
1.2	Список . . . . .	1
1.3	Стек . . . . .	1
1.4	Очередь . . . . .	2
1.5	Устройство вектора . . . . .	2
<b>2</b>	<b>Метод потенциалов анализа сложности</b>	<b>2</b>
<b>3</b>	<b>Символы Ландау</b>	<b>2</b>
<b>4</b>	<b>Мастер теорема</b>	<b>3</b>
<b>5</b>	<b>Алгоритмы быстрого умножения</b>	<b>4</b>
5.1	Алгоритм Карацуба . . . . .	4
5.2	Алгоритм Штрассена . . . . .	5
	Улучшения . . . . .	5
	Гипотеза Штрассена . . . . .	5
5.3	Быстрые преобразования Фурье . . . . .	5
<b>6</b>	<b>Вероятностные алгоритмы</b>	<b>5</b>
	Алгоритм Фреймвальдса . . . . .	6
	Лемма Шварца-Зиппеля . . . . .	6
	Быстрая сортировка . . . . .	7
	SkipList . . . . .	7
	Алгоритм имитации отжига . . . . .	8
<b>7</b>	<b>Численное интегрирование</b>	<b>8</b>
	Метод Монте-Карло . . . . .	8
	Метод прямоугольников . . . . .	8
	Метод трапеций . . . . .	8
	Формула Симпсона . . . . .	8
<b>8</b>	<b>Декартово дерево</b>	<b>9</b>
8.1	История декартовых деревьев . . . . .	9
	8.1.1 Vuillemin (1980) Cartesian Tree . . . . .	9
	8.1.2 Seidel, Aragon (1996) treap (tree + heap) . . . . .	10
8.2	Теорема 1 . . . . .	10
8.3	Следствие . . . . .	10
	8.3.1 Лемма . . . . .	10
	8.3.2 Следствие . . . . .	10
	8.3.3 Теорема 2 . . . . .	10

<b>9</b>	<b>Zip tree</b>	<b>11</b>
9.1	Утверждение . . . . .	11
9.2	Лемма . . . . .	11
9.3	Теорема 1 . . . . .	11
9.4	Лемма 1 . . . . .	12
9.5	Лемма 2 . . . . .	12
9.6	Теорема 2 . . . . .	12
9.7	Теорема 3 . . . . .	12
<b>10</b>	<b>Графы</b>	<b>12</b>
10.1	Определение . . . . .	12
10.2	Способы хранения . . . . .	13
10.3	BFS . . . . .	13
10.4	DFS . . . . .	13
10.5	Приливания . . . . .	15
10.6	CHM (DSU) . . . . .	15
10.7	short . . . . .	17
10.7.1	Остовное дерево . . . . .	17
10.7.2	Минимальное остовное дерево (MST) . . . . .	17
10.7.3	Лемма о безопасном ребре . . . . .	18
10.7.4	Алгоритм Кр(а/у)скала (1956) . . . . .	18
10.7.5	Алгоритм Борувки (1926) . . . . .	19
10.8	Алгоритм Прима (1957) . . . . .	19
10.9	Алгоритм Дейкстры (1959) . . . . .	20

Лекция 3 сентября.

## Выставление оценок

Накоп =  $0.25\text{Коллок} + 0.25\text{КР} + 0.4\text{ДЗ} + 0.1\text{РС}$ .

Коллоквиум между 1 и 2 модулем. После коллока письменная контрольная работа (примерно в конце второго модуля).

ДЗ — констест.

РС — работа на семинаре.

Итог = Накоп или  $0.5\text{Накоп} + 0.5\text{Экз}$  (экзамен можно не сдавать).

В первом случае накоп округляется, во втором — нет.

## 1 Структуры данных

*Абстрактный тип данных* — определяется набор операций, но умалчивается реализация.

*Структура данных* — реализация абстаркного типа данных.

### 1.1 Линейные структуры данных

#### Массив

---

```
int a[20];
array<int, 20> a;
vector<int> a(20); // O(1) amortized
// amortized means average O(1), but O(n) is possible
```

---

## 1.2 Список

### Виды списков

1. Односвязный (храним указатель на начало и конец, указатель на следующий элемент).
2. Двусвязный (аналогично односвязному, но также указатель на предыдущий).

## 1.3 Стек

Свойства: LIFO (last in, first out)

### Реализация

Массив: простая реализация, так как переполнение невозможно.

Список: возвращаем head, добавление в head (список двусвязный).

deque: добавление и взятие элементов из начала или конца (покрывает функционал).

### Виды стеков

1. Стек с минимумом (дополнительный стек, который хранит минимумы на префиксах).

## 1.4 Очередь

Свойства: FIFO (first in, first out)

### Реализация

Массив: кладём элементы по очереди, после переполнения массива мы должны класть элементы в начало (храним указатель на начало и конец очереди).

Список: Возвращаем tail, добавление в head (список двусвязный).

deque: добавление и взятие элементов из начала или конца (покрывает функционал).

Два стека: кладём элементы в первый стек, если нужно взять элемент, то берём из второго стека. Если второй стек пустой, перекладываем все элементы во второй стек. Амортизированное  $O(1)$ .

## 1.5 Устройство вектора

Выделяет какое-то базовое количество памяти по умолчанию. Хранится указатель на начало, конец используемой пользователем памяти и конец аллоцированной памяти.

Когда конец используемой пользователем памяти совпадает с концом аллоцированной памяти, аллоцируется кусок памяти в 1.5 или 2 (зависит от реализации) раза больше. Получается, что при выполнении  $n$  пушбеков, вектор перезапишет себя не более  $\log n$  раз. Всего будет переписано не более  $1 + \dots + \frac{n}{2} + n \approx 2n = O(n)$ .

## 2 Метод потенциалов анализа сложности

$\varphi$  — функция подсчёта потенциала (зависит от параметров структуры данных).

$\varphi_0 \rightarrow \varphi_1 \rightarrow \varphi_2 \rightarrow \dots \rightarrow \varphi_n$

Определение: амортизированное время работы:

$$a_i = t_i + \Delta\varphi, \Delta\varphi = \varphi_{i+1} - \varphi_i$$

$$\sum a_i = \sum t_i + (\varphi_n - \varphi_0) \Rightarrow \frac{\sum t_i}{n} = \frac{\varphi_0 - \varphi_n}{n} + \frac{\sum a_i}{n} \leq \frac{\varphi_0 - \varphi_n}{n} + \max(a_i)$$

$$\varphi_i = 2n_1$$

$$\text{push: } t_i = 1, a_i = 1 + 2 = 3$$

$$\text{pop: } t_i = 1 \text{ или } t_i = 2n_1 + 1, a_i = 1 \text{ или } a_i = 2n_1 + 1 + (0 - 2n_1) = 1$$

Значит  $\max(a_i) \leq 3$ , при этом  $\frac{\varphi_0 - \varphi_n}{n} \leq 0$ , то есть амортизированное время работы:

$$\frac{\sum t_i}{n} \leq 3$$

### 3 Символы Ландау

Оценка сверху:

$$f(x) = O(g(x)) \Leftrightarrow \exists C > 0 \exists x_0 \geq 0 \forall x \geq x_0 : |f(x)| \leq C|g(x)|$$

Оценка снизу:

$$f(x) = o(g(x)) \Leftrightarrow \forall \varepsilon > 0 \exists x_0 \forall x \geq x_0 : |f(x)| \leq \varepsilon |g(x)|$$

Равенство функций:

$$f(x) = \Theta(g(x)) \Leftrightarrow \exists 0 < C_1 < C_2, \exists x_0 \forall x \geq x_0 : C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

Примеры:

1.  $3n + 5\sqrt{n} = O(n)$
2.  $n = O(n^2)$ . Оценка грубая, но правильная, потому что  $n \leq n^2$ . Лучше было бы понять, что  $n = O(n)$
3.  $n! = O(n^n)$
4.  $\log n^2 = O(\log n)$
5. Пусть мы в задаче ввели параметр  $k$ , при этом оптимально, чтобы выполнялось соотношение  $k \log k = n$ .  
Как можно оценить  $k$ ?  
 $k = O(?)$ , обсудим на семинаре.

### Задача

Найти асимптотику сортировки слиянием.

Пусть  $T(n)$  — время, используемое для сортировки массива длины  $n$ .

Зная принцип работы этой сортировки можно сказать, что

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$$

### 4 Мастер теорема

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + O(n^c), & a \in \mathbb{N}, b \in \mathbb{R}, b > 1, c \in \mathbb{R}, c \geq 0 \\ O(1), & n \leq n_0 \end{cases}$$

Разберём три случая:

1.  $c > \log_b a : T(n) = O(n^c)$
2.  $c = \log_b a : T(n) = O(n^c \log n)$
3.  $c < \log_b a : T(n) = O(n^{\log_b a})$

На  $i$ -ом слое:  $a^i \left(\frac{n}{b^i}\right)^c$

Листья (слой  $\log_b n$ ):  $a^{\log_b n}$  задач, сложность каждой равна 1

$$T(n) \leq \sum_{i=0}^{\log_b n} O\left(a^i \left(\frac{n}{b^i}\right)^c\right) = O\left(\sum_{i=0}^{\log_b n} a^i \left(\frac{n}{b^i}\right)^c\right) = O\left(n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i\right)$$

Положим  $q = \frac{a}{b^c}$ , тогда:

$$q < 1 \Leftrightarrow a < b^c \Leftrightarrow c > \log_b a, \quad O\left(n^c \sum_{i=0}^{\log_b n} q^i\right) = O\left(n^c \frac{1}{1-q}\right) = O(n^c)$$

$$q = 1 \Leftrightarrow O(n^c \log_b n)$$

$$q > 1:$$

## Обобщение

$$\text{Случай } \begin{cases} \log_b a = c \\ T(n) = aT(\frac{n}{b}) + O(n^c \log^k n) \end{cases}$$

$$T(n) = O(n^c \log^{k+1} n)$$

Докажем лемму:

$$\forall q > 1 : 1 + q + \dots + q^n = O(q^n)$$

$$\frac{q^{n+1}-1}{q-1} < \frac{q^{n+1}}{q-1} = \frac{q}{q-1} q^n = O(q^n)$$

$$\begin{aligned} \text{Тогда для } q > 1 : O\left(n^c \left(\frac{a}{b^c}\right)^{\log_b n}\right) &= O\left(n^c \frac{a^{\log_b n}}{b^{c \log_b n}}\right) = O\left(n^c \frac{a^{\log_b n}}{n^c}\right) = \\ &= O(a^{\log_b n}) = O(n^{\log_b a}) \end{aligned}$$

## Примеры

Сортировка слиянием:

$$T(n) = 2T(\frac{n}{2}) + O(n^1) \Rightarrow$$

$$\Rightarrow a = 2, b = 2, c = 1 \Rightarrow \log_b a = c \wedge T(n) = O(n^c \log n) = O(n \log n)$$

Бинарный поиск:

$$T(n) = T(\frac{n}{2}) + O(1) \Rightarrow$$

$$\Rightarrow a = 1, b = 2, c = 0 \Rightarrow \log_b a = c \Rightarrow T(n) = O(n^c \log n) = O(\log n)$$

Обход полного двоичного дерева с  $n$  вершинами:

$$T(n) = 2T(\frac{n}{2}) + O(1) \Rightarrow$$

$$\Rightarrow a = b = 2, c = 0 \Rightarrow \log_2 2 > 0 \Rightarrow T(n) = O(n^{\log_2 2}) = O(n^1) = O(n)$$

Лекция 17 сентября

## 5 Алгоритмы быстрого умножения

### 5.1 Алгоритм Карацуба

Придумали в 1960 году. Этот алгоритм мотивировал людей искать более быстрые способы решения известных задач. (На коллоквиуме может пригодиться базовое знание алгоритма Фурье).

**brute**<sup>1</sup>

$A(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ . Будем называть это многочленом с  $n$  коэффициентами ( $(n)$ -член в дальнейшем).

$B(x) = b_0 + b_1x + \dots + b_{m-1}x^{m-1}$  -  $(m)$ -член

$C(x) = A(x) \cdot B(x) = c_0 + c_1x + \dots + c_{n+m-2}x^{n+m-2}$  -  $(n+m-1)$ -член

$$c_k = \sum_{i=0}^k a_i \cdot b_{k-i} \text{ - } k\text{-ый коэффициент в } C(x)$$

Такое решение имеет асимптотику  $O(n \cdot m)$ . Мы будем писать алгоритм для перемножения многочленов одинаковых степеней, поэтому асимптотика будет  $O(\max(n, m)^2)$ , где  $\max(n, m)$  - степень двойки.

$$\begin{aligned} A(x) &= a_0 + a_1x + \dots + a_{n-1}x^{n-1} = \\ &= \underbrace{(a_0 + a_1x + \dots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1})}_{A_0(x)} + \underbrace{(a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + \dots + a_{n-1}x^{\frac{n}{2}-1})}_{A_1(x)} x^{\frac{n}{2}} \end{aligned}$$

Аналогично разбиваем  $B(x) = B_0(x) + B_1(x)x^{\frac{n}{2}}$ , тогда:

$$A(x) \cdot B(x) = (A_0 + A_1x^{\frac{n}{2}})(B_0 + B_1x^{\frac{n}{2}}) = A_0B_0 + (A_1B_0 + A_0B_1)x^{\frac{n}{2}} + A_1B_1x^n$$

Однако если это тупо перемножить, то мы ничего не выиграем:

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n). \text{ По мастер теореме } a = 4, b = 2, c = 1 \Rightarrow$$

$$\Rightarrow 1 < \log_2 4 = 2 \Rightarrow T(n) = O(n^2)$$

<sup>1</sup>Далее так будут называться решения в лоб или переборные решения

Методом подстановки получаем такую же асимптотику (опускаем  $O(n)$ , так как здесь оно не сильно влияет).

$$T(n) = 4T\left(\frac{n}{2}\right) = 4^2T\left(\frac{n}{2^2}\right) = \dots = 4^kT\left(\frac{n}{2^k}\right).$$

$$\text{В какой-то момент } 2^k = n \Rightarrow 4^k = n^2 \Rightarrow T(n) = n^2T(1) = n^2$$

Идея Карацубы заключается в выполнении трёх умножений вместо четырёх.

Сделаем два умножения:  $A_0B_0, A_1B_1$ , далее алгебраические фокусы:

$$\text{Делаем умножение } (A_0 + A_1)(B_0 + B_1) = A_0B_0 + A_1B_1 + A_0B_1 + A_1B_0, \text{ тогда: } (A_1B_0 + A_0B_1) = (A_0 + A_1)(B_0 + B_1) - A_0B_0 - A_1B_1.$$

Этот метод работает быстрее, так как сложение и вычитание мы выполняем за линию, то есть за 2 сложения и два вычитания (4 линии) экономим одно умножение (квадрат).

$$\text{Теперь } T(n) = 3T\left(\frac{n}{2}\right) + O(n) \xrightarrow{\text{M Th}} 1 < \log_2 3 \Rightarrow T(n) = O(n^{\log_2 3})$$

$$\text{Проверим методом подстановки: } n = 2^k : O(3^k) = O(3^{\log_2 n}) = O(n^{\log_2 3})$$

## 5.2 Алгоритм Штрассена

Придуман в 1969 для умножения матриц.

Математически:  $C = \underset{n \times n}{A} \cdot \underset{n \times n}{B}$ , асимптотика  $O(n^3)$

$$C_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}.$$

Алгоритм заключается в следующем преобразовании:

$$\begin{pmatrix} a_{11} & \vdots & a_{12} \\ \dots & \cdot & \dots \\ a_{21} & \vdots & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & \vdots & b_{12} \\ \dots & \cdot & \dots \\ b_{21} & \vdots & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & \vdots & a_{11}b_{12} + a_{12}b_{22} \\ \dots & \cdot & \dots \\ a_{21}b_{11} + a_{22}b_{21} & \vdots & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

$$d = (a_{11} + a_{22})(b_{11} + b_{22})$$

$$d_1 = (a_{12} - a_{22})(b_{21} + b_{22})$$

$$d_2 = (a_{21} - a_{11})(b_{11} + b_{12})$$

$$h_1 = (a_{11} + a_{12})b_{22}$$

$$h_2 = (a_{21} + a_{22})b_{11}$$

$$v_1 = a_{22}(b_{21} - b_{11})$$

$$v_2 = a_{11}(b_{12} - b_{22})$$

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7 \approx 2,81})$$

## Алгоритм Копперсмита-Виноградова (1990)

Работает за  $O(n^{2,3755})$ . Является улучшением алгоритма Штрассена.

## Алгоритм Алмана Вильямса (2020)

За  $O(n^{2,3728})$ .

## Гипотеза Штрассена

$$\forall \varepsilon > 0 \exists \text{ алгоритм } \exists N \forall n \geq N : O(n^{2+\varepsilon})$$

## 5.3 Быстрые преобразования Фурье

$O(n \log n)$ . Переменяем  $n$ -члены, где  $n$  - степень двойки.

Храним многочлен в виде  $n$  его специально подобранных точек (по ним можно восстановить коэффициенты).

На коллоквиуме понадобится знать основную идею работы и асимптотику.

---

<sup>0</sup>M Th = Мастер Теорема

## 6 Вероятностные алгоритмы

Для детерминированных алгоритмов есть понятия:

Сложность — количество операций на данных размера  $n$ .

Сложность в среднем — математическое ожидание количества действий. Вероятностный алгоритм — использует генератор случайных чисел. Могут быть алгоритмы:

- Работающие без ошибок
- С односторонней ошибкой
- С двусторонней ошибкой

Ожидаемое время — математическое ожидание времени работы (на одном наборе данных)

Ожидаемая сложность — максимальное время работы на данных размера  $n$ .

### Поиск $k$ -ой порядковой статистики

Выбираем опорный элемент  $i$ . Пусть меньше него в массиве  $m - 1$  элемент, а больше него  $n - m$  элементов.

$$\begin{aligned} \text{Тогда } E(T(n)) &= \sum_{m=1}^n P(m) E(T(\max(m-1, n-m))) + O(n) = \\ &= \sum_{m=\frac{n}{2}}^n \frac{1}{n} 2T(m) + O(n) = \frac{2}{n} \sum_{m=\frac{n}{2}}^{n-1} E(T(n)) + O(n) = \\ &= \frac{2}{n} (O(\frac{n}{2}) + O(n-1)) + O(n) = \frac{2}{n} O(\frac{3n^2}{8}) + O(n) = O(\frac{3}{4}n) + O(n) = O(n) \end{aligned}$$

Получаем  $E(T(n)) = O(n)$  по индукции.

Детерминированный поиск медианы.

Делим массив на группы по 5 чисел, сортируем их за 7 сравнений (или за 10 действий пузырьком), тратим на это  $7\frac{n}{5}$  действий.

Получаем  $\frac{n}{5}$  медиан:  $m_1, \dots, m_{\frac{n}{5}}$  рекуррентно ищем медиану в них, пусть это  $m_s$ . Тогда:

$\frac{n}{10}$  медиан  $\leq m_s \leq \frac{n}{10}$  медиан. Для каждой медианы слева справедливо, что есть два числа, меньше неё. Справа — есть два числа больше неё. Так как это были медианы в пятёрках чисел.

$$\begin{aligned} T(n) &= T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + O(n) \leq T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + C(n) \Rightarrow \\ \Rightarrow T(n) &\leq 10C \cdot n \Rightarrow T(n) = O(n) \end{aligned}$$

### Алгоритм Фреймвальдса

Умножаем  $A \cdot B = C$ ,  $n \times n$

$v = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & \dots & 1 \end{pmatrix}$  случайные числа 0 или 1

$$A \cdot B \cdot v = C \cdot v \Rightarrow O(n^2)$$

Если получили равенство, то вероятность  $A \cdot B \neq C$ :  $P_{\text{неудачи}} \leq \frac{1}{2}$

### Доказательство

Положим  $D = AB - C = \begin{pmatrix} d_{11} & \dots & d_{1n} \\ \dots & \dots & \dots \\ d_{n1} & \dots & d_{nn} \end{pmatrix}$ , без ограничения общности в нём  $d_{11} \neq 0$

Тогда при домножении матрицы  $D$  на вектор  $v$ . Вероятность наличия на нужной позиции в  $v$  нуля есть  $\frac{1}{2}$  (так

как вектор состоит из 0 и 1).  $Dv = \begin{pmatrix} d_{11}v_1 + (d_{12}v_2 + \dots + d_{1n}v_n) \\ \dots \\ \dots \end{pmatrix}$

$$\left(\frac{m}{2^n}\right)^k \leq \left(\frac{1}{2}\right)^k$$

## Лемма Шварца-Зиппеля

$f(x_1, \dots, x_k)$  — многочлен степени  $n$  от  $k$  переменных.

Возьмём точку  $Y = (y_1, \dots, y_k)$  равномерно из множества  $S$  (множество значений).

$$P(f(y_1, \dots, y_k)) \leq \frac{n}{|S|}$$

Дерандомизация: положим  $k = 1$  и смотрим  $n + 1$  точку.

Лекция 1 октября.

## Быстрая сортировка

Пусть изначальный массив  $a = \{a_1, \dots, a_n\}$

Хотим произвести операцию  $a \rightsquigarrow b = \{b_1, \dots, b_n\}$ , где  $b_1 \leq \dots \leq b_n$ .

$$E(T(n)) = E\left(\sum_{i=0}^{n-1} \sum_{j=i+1}^n \delta_{ij}\right) = \sum_{i=0}^{n-1} \sum_{j=i+1}^n E(\delta_{ij}) = \sum_{i=0}^{n-1} \sum_{j=i+1}^n P(b_i \text{ сравнивается с } b_j)$$

$$\delta_{ij} = \text{bool}(b_i \text{ сравним с } b_j)$$

Рассмотрим массив  $b$ , посмотрим на  $b_i$  и  $b_j$ :

$$b_1 \leq b_2 \leq \dots \leq \overbrace{b_i \leq \dots \leq b_j}^{j-i+1 \text{ элемент}} \leq \dots \leq b_n$$

Нас интересует выбор элемента из этих  $j - i + 1$ . Если выберем что-то между, то интересующие нас два сравниваться не будут. Если выберем один из них, то придётся сравнивать.

Вероятность выбора  $b_i$  или  $b_j$  из  $j - i + 1$  равна  $\frac{2}{j-i+1}$

$$\text{Тогда } E(T(n)) = \sum_{i=0}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} = \sum_{i=0}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} = \sum_{i=0}^{n-1} O(\log n) = O(n \log n)$$

## SkipList

Является структурой данных, придумал William Pugh в 1989 году.

Сначала представлял собой отсортированный список, первым элементом которого является  $-\infty$ , имеющий большое количество указателей в вершинах (у каждого второго был указатель для прыжка на 2, у каждого 4 — на 4 и т.д.), но возникали проблемы с удалением.

Появилась новая идея. Создать список “второго уровня”, в который с вероятностью  $\frac{1}{2}$  попадают элементы из начального списка ( $-\infty$  и элемент сразу после неё попадают во все уровни).

У каждого элемента получившегося списка есть указатель на следующий элемент и на его копию на более низком уровне.

По получившемуся списку мы проходим в два раза быстрее, поэтому мы создаём аналогичный список, базирясь на втором уровне, продолжая это до тех пор, пока размер получившегося уровня не станет меньше либо равен 2.

В среднем на каждом уровне количество элементов уменьшается вдвое.

## Поиск

В такой структуре данных поиск можно осуществлять бинарным поиском.

## Удаление

В случае удаления элемента, он должен удаляться во всех уровнях. Пусть мы удаляем элемент  $x$ , тогда при рекуррентном бинарном поиске этого элемента мы получим указатели  $l < x \leq r$  для каждого уровня. Если  $r$  совпадает с  $x$ , то  $r$  удаляется как в обычно списке, иначе на этом и на всех уровнях выше  $x$  уже не будет, удаление завершено. Перебалансировку мы не производим, так как вероятность существенно уменьшить количество уровней крайне мала.



## Вставка

Ищем позицию для вставки, после чего вставляем его на более высокий уровень с вероятностью  $\frac{1}{2}$ . В результате этой операции количество уровней могло увеличиться (в теории оно могло увеличиться более чем на 1, но на практике более чем на 1 уровень увеличивать смысла нет).

## Оценка количества уровней

$P(\text{есть } i\text{-й уровень}) = 1 - \left(1 - \frac{1}{2^i}\right)^n \leq 1 - \left(1 - \frac{n}{2^i}\right) = \frac{n}{2^i}$   
 $i = 4 \log_2 n \quad P(i) \leq \frac{n}{4 \log_2 n} = \frac{n}{n^4} = \frac{1}{n^3}$ , то есть вероятность сильно превзойти  $\log_2 n$  очень мала.

## Оценка оптимальной вероятности повышения

Пусть  $p \neq \frac{1}{2}$ , тогда  $n, pn, p^2n, \dots, 1$  — размеры слоёв.

$\log_{\frac{1}{p}} n$  — количество слоёв.

Асимптотика:  $O\left(\frac{1}{p} \log_{\frac{1}{p}} n\right)$

## Алгоритм имитации отжига

Введём функционал качества  $Q_0 \rightsquigarrow Q_1 \rightsquigarrow Q_2 \rightsquigarrow \dots$ .

Это некая функция, которую стоит минимизировать. Программа производит случайные изменения, после чего смотрит на изменение функционала.

$Q_{i+1} < Q_i \Rightarrow$  делаем изменение.  $Q_{i+1} > Q_i \Rightarrow$  делаем изменение с вероятностью  $p = e^{-\frac{Q_{i+1}-Q_i}{T_i}}$   
 $T_i$  - некая убывающая функция (гипербола, линейная, что лучше подходит).

Лекция 8 октября.

# 7 Численное интегрирование

Подсчёт площади функции  $f(x)$  на отрезке  $[a, b]$ .

## Метод Монте-Карло

Составляем описанный прямоугольник для этой функции. Случайным образом помещаем в него точки. Далее составляем пропорцию: площадь прямоугольника относится к площади фигуры так же как количество помещённых точек относится к количеству точек, попавших в нашу фигуру.

## Метод прямоугольников

Поделим отрезок  $[a, b]$  на  $n$  равных частей, далее складываем площади  $S$  получившихся трапеций.  $a = x_0 < x_1 < \dots < x_n = b$ ,  $h = \frac{b-a}{n}$ ,

$$\begin{cases} S = h \cdot f(x_i) & \text{— метод левых прямоугольников} \\ S = h \cdot f(x_{i+1}) & \text{— метод правых прямоугольников} \\ S = h \cdot f\left(\frac{x_i + x_{i+1}}{2}\right) & \text{— метод центральных прямоугольников} \end{cases}$$

## Метод трапеций

Разбиваем аналогично, но площадь считаем по формуле

$$S = \frac{h}{2} (f(x_i) + f(x_{i+1}))$$

## Формула Симпсона

Разбиваем на равные отрезки, для каждого отрезка считаем:

$$S = \frac{h}{6} \left( f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1}) \right)$$

## Подсчёт площади круга

Пусть окружность задаётся уравнением  $(x - 1)^2 + (y - 5)^2 = 9$

Разобьём окружность на полуокружности:  $y = 5 \pm \sqrt{9 - (x - 1)^2}$

Пусть  $f_1(x) = 5 + \sqrt{9 - (x - 1)^2}$ ,  $f_2(x) = 5 - \sqrt{9 - (x - 1)^2}$

Тогда искомая площадь:  $\int_{-2}^4 f_1(x) - f_2(x) dx$

Для подсчёта этих интегралов стоит написать рекуррентную функцию. Она должна на отрезках с недостаточной точностью добавлять новые точки, пока точность нас не устроит. (так называемая переменная сетка).

## Сетка переменной плотности

Рассматриваем полученную на отрезке площадь, обозначим её  $S_1$ . Всё-таки поделим пополам и посчитаем полученную площадь, обозначим её  $S_2$ . Выбираем необходимую точность  $\varepsilon$ , если выполняется  $|S_1 - S_2| < \varepsilon$ , то точку мы оставляем.

## Задача 2 идея

Даётся экран, на нём расположено  $n$  фигур. Нужно найти площадь объединения этих фигур.

### Возможные решения

#### 0. Монте-Карло

1. Прямоугольники. Если центр прямоугольника лежит в фигуре, считаем, что весь прямоугольник лежит.
2. Квадродерево. Изначально разбиваем на большие прямоугольники. Каждый большой по необходимости разбиваем на маленькие (если квадрат частично лежит в какой-то фигуре). Тогда маленькие квадраты будут появляться на границах фигур, общая площадь граничных фигур будет  $S \approx \sqrt{\varepsilon} \cdot Length$ , где  $\varepsilon$  — площадь одного квадрата,  $Length$  — длина всех фигур.
3. Вертикальные полосы переменной плотности + сканирующая прямая. Нарезаем вертикально, считаем площадь прямоугольников, полученных методом центральных прямоугольников. Делим отрезок пополам, считаем площадь по-новому, если разность площадей нас устраивает, прекращаем деление.

Лекция 15 октября

## 8 Декартово дерево

Декартово дерево хранит пары {ключ, приоритет}, в дальнейшем будет называться Vertex.

Можно визуализировать в декартовой системе координат, сопоставив каждой вершине {key, priority} точку {x, y}. При таком отображении увеличение ключа будет смещать вершину вправо, а увеличение приоритета будет поднимать вершину вверх.

Поиск по ключу — аналогичен поиску в двоичном дереве поиска (Binary Search Tree).

Поиск по приоритетам — поиск по куче (Heap).

### 8.1 История декартовых деревьев

#### 8.1.1 Vuillemin (1980) Cartesian Tree

Вставка до листа в порядке случайной перестановки. Использует повороты.

Операции в структуре данных:

- Insert(Vertex) — вставляем как лист, затем поднимаем поворотами.
- Delete(Vertex) — спускаем нужную вершину поворотами, удаление листа.
- Split(key a) — выполняется Insert(Vertex(a,  $+\infty$ )), затем удаляется корень.
- Join(tree, tree, key a) — подвешивает деревья к Vertex(a,  $+\infty$ ), потом эта вершина удаляется.

### 8.1.2 Seidel, Aragon (1996) treap (tree + heap)

Вставка в произвольном порядке (декартово дерево однозначно задаётся своими вершинами). Используются функции Split и Merge.

$(x, y)$ ,  $y \in R[0, 1]$  (равномерное распределение)

Можем посчитать математическое ожидание некоторых величин дерева:

$$E(\text{depth}[v]) = O(\log n)$$

$$E(\text{height}[v]) = O(1)$$

$$E(\text{size}[v]) = O(\log n)$$

- Insert(Vertex  $v$ ) — выполняем Split(Vertex  $v$ ), два дерева  $tl$  (ключи меньше  $v.\text{key}$ ),  $tr$  (ключи больше  $v.\text{key}$ ), выполняем Merge( $tl$ ,  $tr$ ,  $v$ )
- Delete(Vertex  $v$ ) — Split(Vertex  $v$ ), получаем  $tl$ ,  $tr$ , после чего выполняем Merge( $tl$ ,  $tr$ ), не включая  $v$  в новое дерево.

## 8.2 Теорема 1

$$\text{depth}[x_l] = \sum_{i=1}^n a_{i,l}, \text{ где } a_{i,l} = \begin{cases} 1, & \text{если } x_i - \text{предок } x_l \\ 0, & \text{иначе} \end{cases} \quad (\text{количество предков})$$

$$\text{size}[x_l] = \sum_{j=1}^n a_{l,j}, \text{ где } a_{l,j} = \begin{cases} 1, & \text{если } x_l - \text{предок } x_j \\ 0, & \text{иначе} \end{cases} \quad (\text{количество детей})$$

## 8.3 Следствие

$$E(\text{depth}[x_l]) = \sum_{i=1}^n p_{i,l}, \quad E(\text{size}[x_l]) = \sum_{j=1}^n p_{l,j}, \quad p_{i,j} = P(a_{i,j} = 1), \quad (P \text{ это вероятность})$$

### 8.3.1 Лемма

Пусть  $T$  — декартово дерево (множество вершин), тогда:

$$\forall i, j \in \underline{|T|} \quad (v_i.\text{prior} = v_j.\text{prior} \Rightarrow i = j) \wedge a_{i,j} = 1 \Rightarrow v_i.\text{prior} = \max_{\min(i,j) \leq k \leq \max(i,j)} (v_k.\text{prior})$$

То есть при отсутствии вершин с равными приоритетами, приоритет вершины с номером  $i$ , являющейся предком для вершины  $j$ , будет максимальным среди приоритетов всех вершин на отрезке  $\begin{cases} [i, j], & i \leq j \\ [j, i], & j \leq i \end{cases}$

### 8.3.2 Следствие

$$p_{i,j} = \frac{1}{(|i - j| + 1)}$$

Вероятность того, что одна вершина (с номером  $i$ ) является максимальной среди всех вершин на отрезке с концами  $i$  и  $j$ .

### 8.3.3 Теорема 2

$$E(\text{depth}[x_l]) = \frac{1}{|l-1|+1} + \frac{1}{|l-2|+1} + \dots + \frac{1}{1+1} + \frac{1}{1} + \frac{1}{1+1} + \dots + \frac{1}{|l-n|+1} =$$

$$= H_l + H_{n-l+1} - 1 < 1 + 2 \ln n$$

За  $H_n$  обозначена сумма первых  $n$  членов гармонического ряда. Первое слагаемое — уменьшение знаменателя от  $l$  до 1, второе — увеличение знаменателя от 1 до  $l - n + 1$ , единицу учли и там, и там, поэтому надо вычесть (третье слагаемое). Аналогично можем посчитать математическое ожидание  $\text{size}[x_l]$ :

$$E(\text{size}[x_l]) < 1 + 2 \ln n$$

Однако распределение глубины плотное, а распределение размера — всегда неравномерное.

Интересный факт из математики (Devroye):  $\frac{H_n}{\ln n} \xrightarrow{n \rightarrow +\infty} \gamma \approx 4.311$

## 9 Zip tree

*Создатели:* Tarjan, Levy, Timmel (2017)

Задумка — улучшение декартова дерева путём уменьшения количества памяти, хранимого для приоритета. Теперь приоритеты лежат на отрезке  $[1, \log_2 n]$  (вместо  $[1, n^3]$ ), то есть на хранение можно выделять  $\log_2 \log_2 n$  бит.

Аналог ДД, но вместо  $R[0, 1]$  будем пользоваться геометрическим распределением приоритетов (в дальнейшем будем называть рангом):

$$P(\text{rank} = 0) = \frac{1}{2}, P(\text{rank} = 1) = \frac{1}{4}, \dots, P(\text{rank} = k) = \frac{1}{2^{k+1}}$$

Операции со структурой данных:

- Upzip = Split
- Zip = Merge
- Insert
- Delete

### 9.1 Утверждение

Zip Tree имеет единственный изоморфизм со Skip List.

### 9.2 Лемма

Zip Tree единственно (однозначно задаётся множеством вершин).

### Свойства

$$P(v_i.\text{rank} = k) = \frac{1}{2^{k+1}} \Rightarrow P(v_i.\text{rank} < k) = \sum_{i=1}^k \frac{1}{2^i} = 1 - \frac{1}{2^k}$$

$$P(v_i.\text{rank} > k) = 1 - P(v_i.\text{rank} \leq k) = 1 - \left( \frac{1}{2^{k+1}} + 1 - \frac{1}{2^k} \right) = \frac{1}{2^{k+1}} \Rightarrow$$

$$\Rightarrow P(\max_i(v_i.\text{rank}) < k) = (P(v_i.\text{rank} < k))^n = \left( 1 - \frac{1}{2^k} \right)^n \geq 1 - \frac{n}{2^k} \Rightarrow$$

$$\Rightarrow P(\max_i(v_i.\text{rank}) \geq k) \leq \frac{n}{2^k}$$

$$P(v_{\text{root}}.\text{rank} \geq \log_2 n + C) \leq \frac{n}{2^{\log_2 n + C}} = \frac{1}{2^C}$$

$$P(v_{\text{root}}.\text{rank} \geq (C+1) \log_2 n) \leq \frac{n}{2^{(C+1) \log_2 n}} = \frac{n}{n^{C+1}} = \frac{1}{n^C}$$

### 9.3 Теорема 1

$$\begin{aligned} E(v_{\text{root}}.\text{rank}) &= 0 \cdot P(v_{\text{root}}.\text{rank} = 0) + 1 \cdot P(v_{\text{root}}.\text{rank} = 1) + \dots = \\ &= 0 \cdot P_0 + 1 \cdot P_1 + \dots + \lceil \log_2 n \rceil P_{\lceil \log_2 n \rceil} + (\lceil \log_2 n \rceil + 1) P_{\lceil \log_2 n \rceil + 1} + \dots \leq \\ &\leq \lceil \log_2 n \rceil \cdot \sum_{i=1}^{\infty} P_i + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + \dots = \\ &= \lceil \log_2 n \rceil + 2 < \log_2 n + 3 \end{aligned}$$

## 9.4 Лемма 1

low предок — вершина, имеющая больший ранг, но меньший ключ (находятся слева сверху от вершины).  
Для вершины  $x$  и её low предка  $y_l$  с наибольшим рангом справедливо:

$$E(\# \text{low предков}) = 1 + (y_l.rank - x.rank) \leq 1 + y_l.rank$$

## 9.5 Лемма 2

$$E(\# \text{low предков}) \leq \frac{1 + y_h.rank}{2}$$

## 9.6 Теорема 2

$$E(depth[v]) = \frac{3}{2} \cdot \log_2 n + O(1)$$

## 9.7 Теорема 3

$$E(size[v], v.rank == k) \leq 3 \cdot 2^k - 1$$

$$E(size[v]) \leq \frac{3}{2} \cdot \log_2 n + 2$$

Доказательства теорем не были разобраны на лекции (рекомендовали ознакомиться с фрагментом оригинальной статьи про Zip Tree).

## Сравнение глубины ДД и Zip дерева

ДД -  $z \log_2 n \approx 1,38 \log_2 n$

Zip дерево -  $1,5 \log_2 n$

Лекция 5 ноября

# 10 Графы

## 10.1 Определение

Граф  $G = (V, E)$ , множество вершин и рёбер.

Графы бывают

- ориентированные
- неориентированные

Рёбра в графе

- Взвешенные
- Невзвешенные

Могут быть

- Петли (из вершины можно попасть в неё же)
- Кратные рёбра (несколько рёбер из одной вершины в другую)

## 10.2 Способы хранения

### 1. Матрица смежности

```
int g[n][n];
```

$$g[i][j] = \begin{cases} 0, & \text{между вершинами } i, j \text{ нет ребра} \\ 1, & \text{иначе} \end{cases}$$

### 2. Список смежности

```
std::vector<std::vector<int>> g;
```

$g[i] = \{\text{Массив соседей вершины } i\}$

### 3. Множество смежности

```
std::vector<std::set<int>> g;
```

$g[i] = \{\text{Множество соседей вершины } i\}$

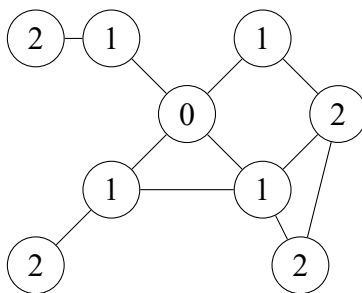
### 4. Список рёбер

```
std::vector<std::pair<int, int>> g;
```

$g[i] = \{\text{Пара вершин, соединённых ребром}\}$

Неориентированный	Ориентированный	Взвешенные
Компонента связности	Компоненты сильной связности	Минимальный остов
Двудольность	DAG (directed acyclic graph), Топ. сорт., ДП	Кратчайшее расстояние
Остов	Кратчайшее расстояние	
Кратчайшее расстояние		

## 10.3 BFS



### Через queue

Вершина, в которой начинаем, имеет номер 0, далее заносим в очередь всех её соседей с номером, увеличенным на 1.

### 0-1 BFS

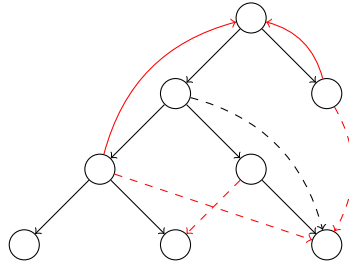
Используется deque, если у вершины номер  $x$ , то заносим его в начало, если номер  $x + 1$ , то заносим в конец.

## 10.4 DFS

Храним массив посещённых вершин

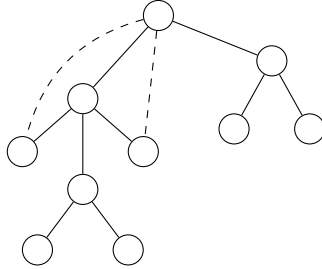
```
used[v];
```

### Ориентированный:



Термины: ребро обхода, прямое ребро, обратное ребро, перекрёстное ребро

### Неориентированный граф:



Не имеет перекрёстных рёбер.

Напишем реализацию DFS с глобальным графом  $g$ .

```
void dfs(int vertex_idx) {
    used[vertex_idx] = 1;
    for (auto dest_idx : g[vertex_idx]) {
        if (!used[dest_idx]) {
            dfs(dest_idx);
        }
    }
}
```

### Примеры задач на dfs:

1. Подсчёт количества компонент связности. (В цикле запускаем dfs от всех вершин. Количество запусков dfs совпадёт с количеством компонент связности)
2. Серия запусков dfs.
3. Поиск остовного леса.
4. Двудольность.
5. Поиск цикла.
6. Поиск Эйлера пути (поиск пути, проходящего все рёбра один раз).

### Обход дерева

Вместо `used[v]` храним `parent[v]`. То есть храним предков.

ДП снизу: `size[v]`;

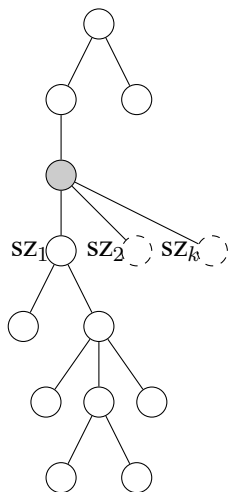
ДП сверху: `dep[v]`;

## 10.5 Приливания

Не путать с переливаниями и т.д.

Предположим мы реализуем динамическое программирование на дереве (далее будем называть *динамикой на дереве*).

Пусть нам дано несбалансированное дерево ( $sz_i$  — какая-то метрика поддеревя):



В сбалансированном дереве может быть  $sz_1 > sz_2 > \dots > sz_k$

$\forall i \in [2, k]$  с конца приливаем  $sz_i$  к  $sz_{i-1}$  (приливаем из меньшего в большее), при необходимости добавляем какую-то константу.

### Утверждение

В таком случае произойдёт  $O(n \log n)$  добавлений.

### Доказательство

Возьмём произвольный элемент  $x$ , пусть у него  $sz = 1$ . Тогда после приливания к следующему элементу, полученный  $sz$  будет не менее 2, далее — не менее 4, и т.д. Постоянно получаем  $sz$  не менее чем в два раза больше, пока не получим  $n$ . Тогда будет сделано  $O(\log n)$  приливаний.

## 10.6 CHM (DSU)

Расшифровка: “система непересекающихся множеств”  $n$  покрашенных элементов,  $m$  запросов вида:

- $\text{unite}(a, b)$ . Присваиваем множествам  $a, b$  одинаковый цвет.
- $\text{col}(a)$ . Получить цвет множества.
- $\text{col}(a) \stackrel{?}{=} \text{col}(b)$ . Сравниваем цвета множеств.
- $\text{sz}(a)$ . Размер множества, содержащего элемент  $a$ .

### Наивное решение

Хотим иметь  $\text{col}[a]$  — массив цветов.

Старт: заполняем  $\text{col}[i] = \text{color}_i$ , заполняем массив  $\text{sz}[\text{color}]$ .

$\text{unite}(a, b)$ :

Пусть  $ca = \text{col}[a]$ ,  $cb = \text{col}[b]$ . Базовая реализация такого алгоритма:

```
for (int i = 0; i < n; i++) {
    if (col[i] == ca) {
        col[i] = cb;
        sz[cb]++;
    }
}
```



Работает за  $O(n^2 + m)$ .

### Возможное улучшение (приливания)

Для каждого цвета храним (допустим в векторе  $\text{ind}[\text{color}]$ ) позиции с элементами этого цвета. Положим  $\text{col}[a] \neq \text{col}[b]$ . Тогда нужно сравнить  $\underbrace{\text{ind}[\text{col}[a]].\text{size()}}_{\text{num a}}$  и  $\underbrace{\text{ind}[\text{col}[b]].\text{size()}}_{\text{num b}}$ .

Положим  $\text{num a} > \text{num b}$  больше, тогда

1. все элементы  $\text{col}[b]$  красим в цвет  $\text{col}[a]$  за  $O(\text{num b})$ .
2. В  $\text{ind}[\text{col}[a]]$  добавим все индексы из  $\text{ind}[\text{col}[b]]$  за  $O(\text{num b})$ .

Всего получим  $O(n \log n)$  операция (асимптотика приливаний). Приливаем меньшую часть (её размер не более  $\frac{1}{2^{k+1}}$ , где  $k$  — количество совершённых операций unite) к большей.

### Возможное улучшение (лес)

Храним массив предков каждой вершины (предком корня является корень). Изначально каждую вершину считаем корнем дерева, содержащего только эту вершину. Далее все вершины, попавшие в поддерево какой-то вершины считаются покрашенными в цвет корня.

Старт: массив  $\text{parent}[i] = i$ .

Цвет — номер корня дерева.

$\text{unite}(a, b): \begin{cases} a = \text{col}(a) \\ b = \text{col}(b) \end{cases} \rightarrow a \neq b \Rightarrow \text{подвешиваем "меньшего" к "большому"}.$

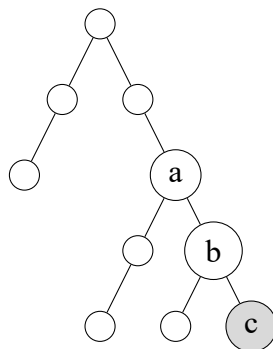
Сравнивать вершины можно

- По высоте
- По размеру

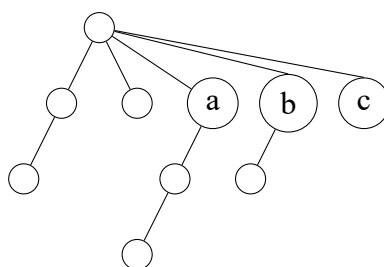
Работает за  $O(n \log n + m \log n)$  первое слагаемое — выполнение всех операций unite, второе — поиск предка (цвета) вершины.

### Эвристика сжатия пути

Пусть дано дерево:



Пусть мы хотим узнать цвет вершины c, тогда дерево перейдёт:



Это ~~честно-честно~~ работает за  $O(\alpha(n) \cdot n + m)$ .  $\alpha(n)$  — некая функция (обратная функции Аккермана), которая во всех реальных задачах не более 4.

## СНМ с откатами

Операции:

- `unite( $a$ ,  $b$ )`
- $a \sim b$
- `sz[ $a$ ]`
- `rollback()`

Не делаем сжатие

Лекция 19 ноября

## 10.7 Остовы

Пусть  $G = (V, E)$  — связный, взвешенный и неориентированный граф.  
Взвешенный значит есть некая функция  $w : E \rightarrow \mathbb{R}$ .

### Первый способ хранения

---

```
std::vector<std::vector<int>> g, w;
```

---

Здесь `g[v]` — список соседей вершины `v`.  
`w[v]` — синхронный список весов рёбёр.

### Второй способ хранения

---

```
std::vector<std::vector<std::pair<int, int>>> g;
```

---

Здесь `g[i][j]` — пара `<вес, вершина>`.

### Третий способ хранения

---

```
std::vector<std::pair<int, std::pair<int, int>>> g;
```

---

Здесь хранятся пары `<вес <начало ребра, конец ребра>`

### 10.7.1 Остовное дерево

Остовным деревом связного неориентированного графа  $G = (V, E)$  называется связный ациклический подграф  $G' = (V, E')$ , где  $E' \subseteq E$ .

### 10.7.2 Минимальное остовное дерево (MST)

MST = Minimal Span Tree. Минимальным остовным деревом связного неориентированного взвешенного графа  $G = (V, E)$  называется остовное дерево с минимальной суммой весов рёбёр.

## Важные термины

Назовём  $G_b = (V, E_b)$  безопасным подграфом  $G = (V, E)$ , если выполняется:

$$\exists MST\ G' = (V, E') \text{ графа } G : G_b \subseteq G'$$

Безопасным ребром  $e$  безопасного подграфа  $G_b$  назовём ребро, удовлетворяющее условию:

$$G_b \cup e \text{ — тоже безопасный подграф}$$

Разрез — разделение множества вершин  $V$  на два непересекающихся подмножества  $V_1, V_2$ .

Обозначать можно двумя способами:

$$V = V_1 \sqcup V_2$$
$$\langle V_1, V_2 \rangle \text{ — разрез}$$

Ребро  $e = (a, b)$  пересекает разрез, если:

$$\begin{cases} a \in V_1 \wedge b \in V_2 \\ a \in V_2 \wedge b \in V_1 \end{cases}$$

То есть концы этого ребра лежат в разных подмножествах.

Граф  $G$  совместим с разрезом  $V = V_1 \sqcup V_2$ , если:

$$\forall e \in E \quad e \text{ не пересекает разрез}$$

### 10.7.3 Лемма о безопасном ребре

$G_b = (V, E_b)$  — безопасный подграф  $G = (V, E)$ .

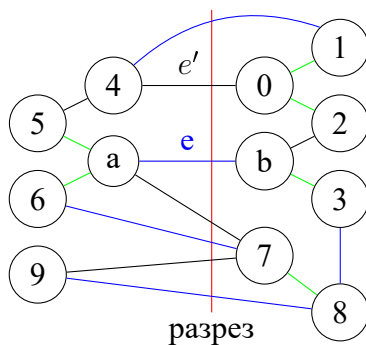
Разрез  $V = V_1 \sqcup V_2$ , совместимый с  $G_b$ .

Пусть  $E_{\text{cross}}$  — все рёбра из  $E$ , пересекающие разрез.

Пусть  $e$  — ребро минимального веса в  $E_{\text{cross}}$ , тогда  $e$  — безопасное ребро для  $G_b$ .

#### Доказательство:

Пусть  $e = (a, b)$



Зелёные рёбра лежат в  $G_b$ .

Чёрные дополняют зелёные до MST.

Синие — оставшиеся рёбра.  $T$  — MST, которое дополняет  $G_b$ ,  $e'$ ,  $w(e) \leq w(e')$

$T' = T \cup \{e\} \setminus e'$  — это MST, ч.т.д.

### 10.7.4 Алгоритм Кр(а/у)скала (1956)

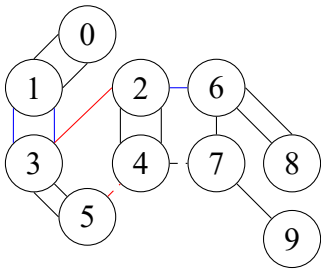
1. Сортируем все рёбра по весу.
2. Идём в порядке сортировки и берём ребро  $e$  в остов  $\Leftrightarrow e$  — соединяет разные компоненты связности (СНМ)

#### Асимптотика

$O(n \log n + m + \alpha(n) \cdot n)$ , то есть сумма асимптотик сортировки и СНМ.

10.7.5 Алгоритм Борувки (1926)

Начинаем с  $n$  множествами.  
Стадии: для каждого множества находим минимальное ребро, у которого ровно один конец в этом множестве, все этим рёбра добавляем в СНМ.  
Рассматриваем вместо весов рёбер пару <вес, номер ребра>.



Двойные рёбра — значит минимумы соединённых вершин совпали.  
Пунктирные — отвергнутые алгоритмом СНМ, так как давали цикл. Стадия 0:  $n$  — множеств. Чёрные рёбра.  
Стадия 1:  $\leq \frac{n}{2}$  множеств. Синие рёбра.  
Стадия 2:  $\leq \frac{n}{4}$  множеств. Красные рёбра.  
Стадия  $\log n$ :  $\leq 1$  множества.

Доказательство

Если тут нет графика, пишите

$V_1 = \text{компонента связности } a, V_2 = V \setminus V_1$   
 $e_1, e_k \rightarrow \text{sort}, w(e_1) \leq \dots \leq w(e_k).$

10.8 Алгоритм Прима (1957)

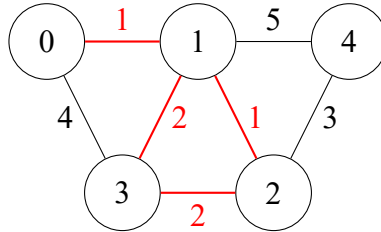
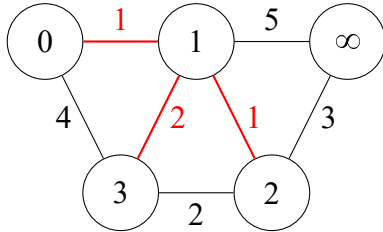
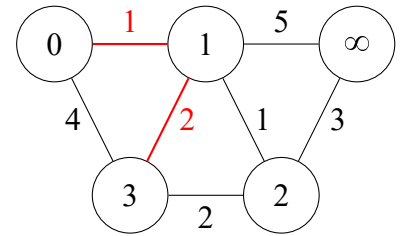
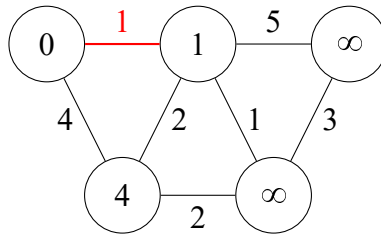
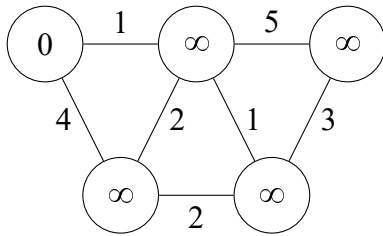
На самом деле алгоритм был опубликован в 1930 году малоизвестным математиком.  
Алгоритм ищет минимальный остов в графе.

P.S. граф анимированный, однако может не проигрываться в браузере или других базовых средствах просмотра. Рекомендую использовать Adobe Acrobat Reader, в нём анимации проигрываются нормально.  
Далее  $w$  — вершина, только что попавшая в множество обработанных вершин.  $v$  — элемент из множества вершин, имеющих ребро в  $w$ , но не лежащих в множестве обработанных. Асимптотика работы:  $O(n \cdot T(\text{get\_min}) + m \cdot T(\text{recalc}))$  в общем виде.  
Формула recalc:  $d[v] = \min(d[v], \text{len}(w, v));$

Реализация	Асимптотика
В лоб	$O(n \cdot n + m) = O(n^2)$
set или heap	$O(n \log n + m \log n) = O(m \log n)$
sqrt decomposition	$O(n\sqrt{c} + m \cdot 1)$
Фибоначчиева куча	$O(n \log n + m \cdot 1)$

Если использовать set для хранения пар  $\{d[v], v\}$ , где  $d[v]$  — вес минимального ребра из  $v$  в множество построенных вершин, то можно добиться асимптотики.  
Фибоначчиева куча — структура данных, придуманная специально для алгоритмов Прима и Дейкстры.  
Замечание: корневая декомпозиция работает только для целых неотрицательных весов, не более некоторого  $C$ .

## 10.9 Алгоритм Дейкстры (1959)



---

```
d.assign(n, +infty);  
d[s] = 0;
```

---

### Лемма

Вершина  $v$ , у которой  $d[v]$  минимально среди всех вершин до которых расстояние не посчитано, может быть причислена к множеству вершин, до которых расстояние посчитано.

Формула recalc:  $d[v] = \min(d[v], d[w] + \text{len}(v, w))$ .

Асимптотики аналогичны алгоритму Прима.

*Замечание:* если менять recalc, можно получать алгоритмы, решающие другие задачи.33