Алгоритмы и структуры данных

① Введение

Сложность алгоритма

$M(n)$ – память

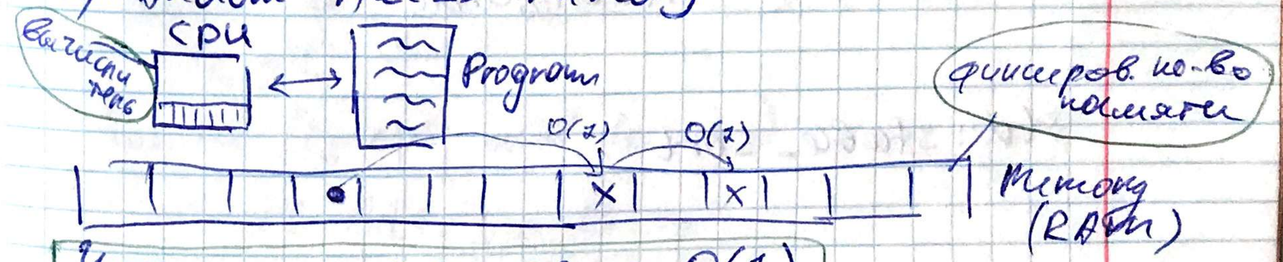
$T(n)$ – время, n – размер входа

Модель вычислений

Какие операции за сколько работают:
за линейное время, квадратичное, ...

1) Машина Тьюринга
2) ..... Поста
3) Большие данные
4) Параллельные вычисления

5) Random Access Memory



Чтение и запись – $O(1)$
Арифметика $O(1)$

P.S. $f(n) = O(g(n)) \Leftrightarrow f(n) \le c \cdot g(n)$
для некоторого $c > 0$
и $n > N$

Merge Sort

Стабильность: $i < j$ и $a_i = a_j$, то $\pi(i) < \pi(j)$
в отсортиров. массиве одинаковые эл-ты будут идти в том же порядке, что и в неотсортир.

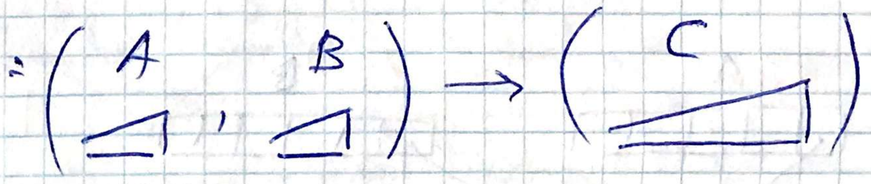
QuickSort и HeapSort - НЕ стабильные

2 1' 3 1''

1' 1'' 2 3 - стабильная

1'' 1' 2 3 - Нестабильн.

std::stable_sort

Merge: 

Merge ($begin_A$, end_A, $begin_B$, end_B, b_C)

b_C ↑ начало нового массива

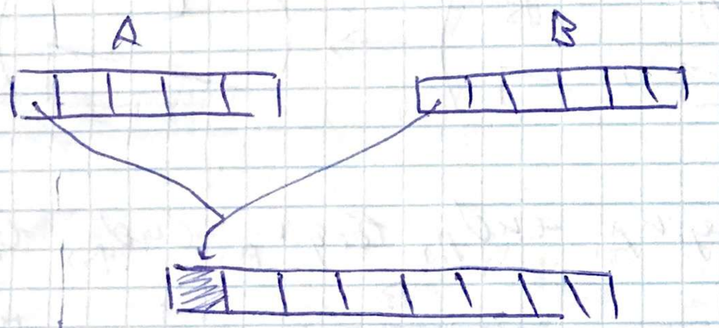
```
while (|A| > 0 and |B| > 0):
    if (A[b_A] < B[b_B]):
        C[b_C] = A[b_A]
```

```
Merge(A, B, C) {
    i_A = 0
    i_B = 0
    i_C = 0
    while (i_A < |A| and i_B < |B|):
        if A[i_A] ≤ B[i_B]:
            C[i_C] = A[i_A], i_C++; i_A++
```

else:

$C[i_C] = B[i_B]$

$i_C{}^{++}$; $i_B{}^{++}$



while ($i_A < |A|$):

$C[i_C] = A[i_A]$

$i_C{}^{++}$; $i_A{}^{++}$

while ($i_B < |B|$):

$C[i_C] = B[i_B]$

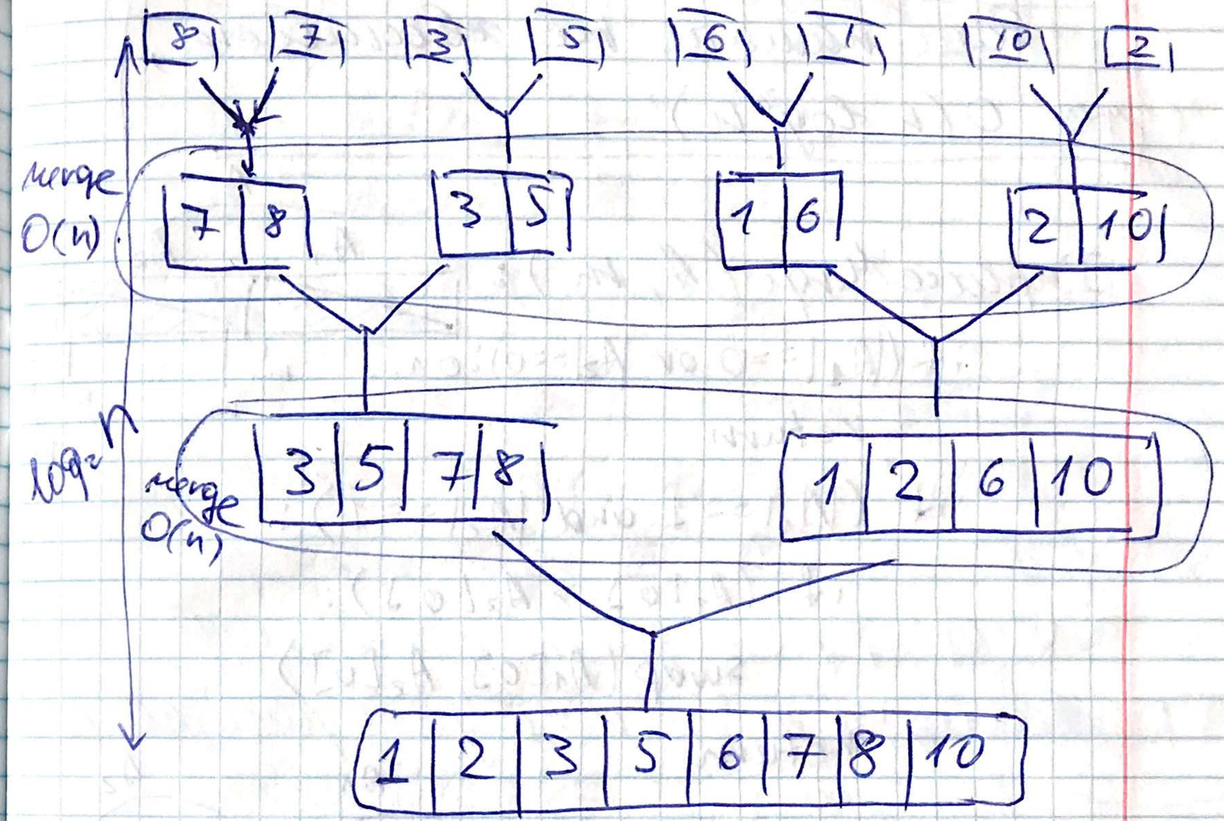
$i_C{}^{++}$; $i_B{}^{++}$

return C

}

~~MergeSort(A):~~

~~C = tmp_buffer~~



<u>Merge</u>: ~~$O(|A|+|B|)$~~

на каждом уровне n эл-ов $\Rightarrow$

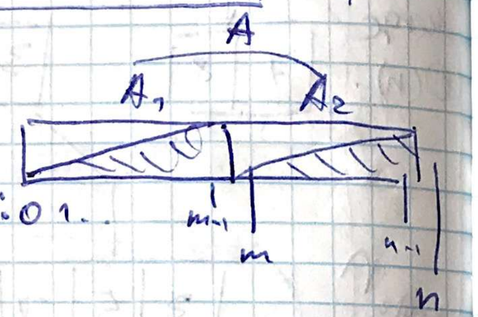
$\Rightarrow$ merge ~ $O(n)$

<u>Уровней</u>: $\log_2 n$

$T(n) = O(n \log n)$

$M(n) = O(n)$

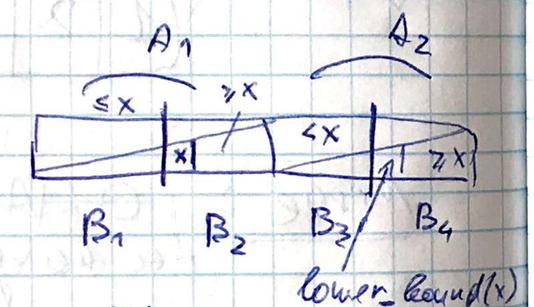
Если памяти не хватает, то $O(n \log^2 n)$

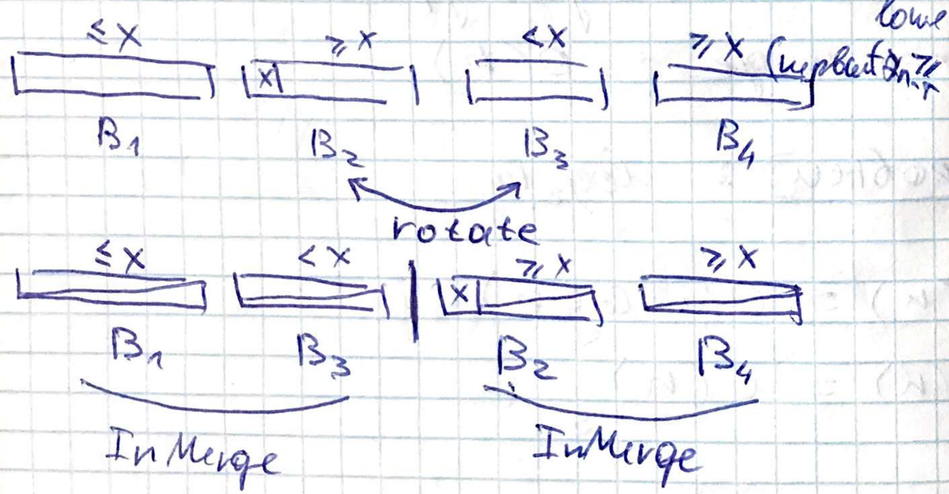



```
InplaceMerge(A, m):
    if (|A1|==0 or |A2|==0): 0 1..
        return
    if (|A1|==1 and |A2|==1):
        if (A1[0] > A2[0]):
            swap(A1[0], A2[0])
        return
```

(Пусть $|A_1| > |A_2|$):

$m_1 = m/2$





$B_1 = A_1[0 \dots \frac{m}{2}]$ $\quad x = B_2[0]$

$B_2 = A_1[m_1 \dots m)$

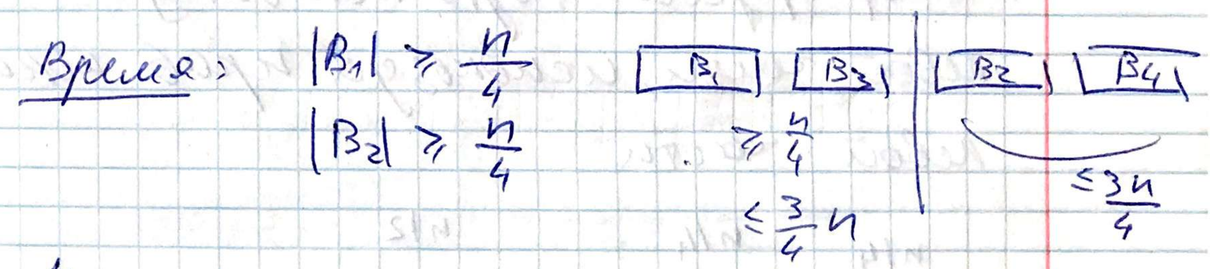
$B_3 = A_2[m \dots lb(x))$

$B_4 = A_2[lb(x) \dots n)$

rotate (B_2, B_3) (цикл. сдвиг)

InplaceMerge (B_1, B_3)

InplaceMerge (B_2, B_4)



Каждая из половин уменьш. в $\frac{4}{3}$ раза минимум

$\log_{\frac{4}{3}} n$ раз уменьшаем

InMerge: $O(n \log n)$

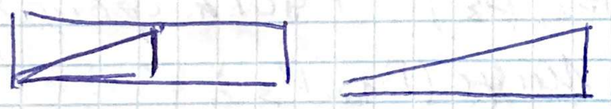
MergeSort: $T(n) = O(n \log^2 n) = O(n \log n) \cdot O(\log n)$

$M(n) = O(1)$ // $O(\log n)$ стек вызовов

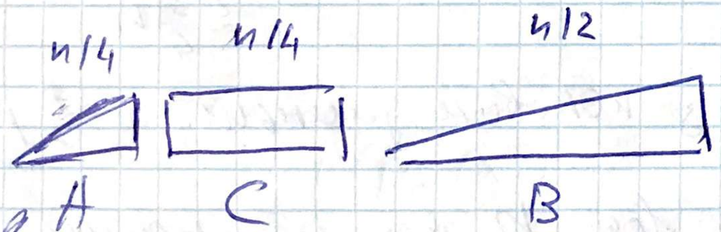
Inplace Mergesort за $O(n \log n)$

не требует доп. памяти

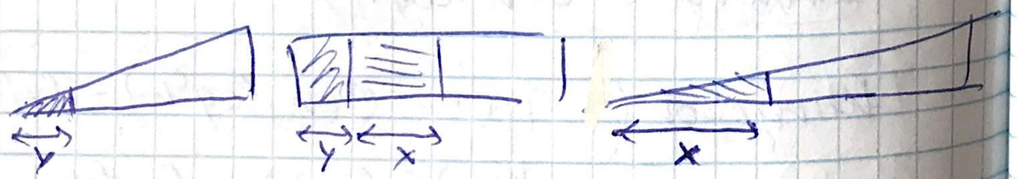
1) Отсортируем правую часть, используя левую

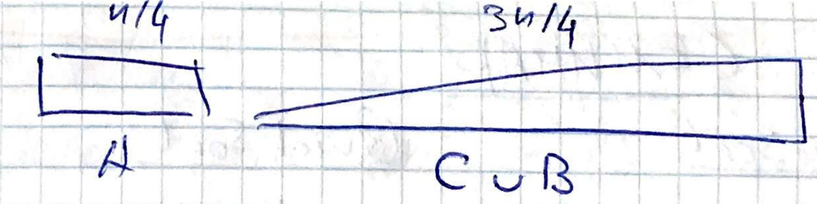


2) отсортируем левую половину левой части, используя прав. полов. левой части

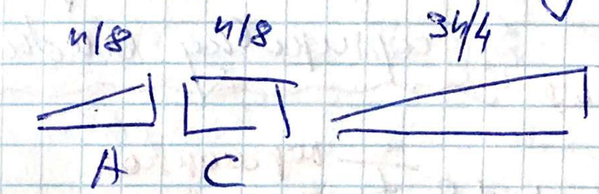


3) Merge (A, B, C)

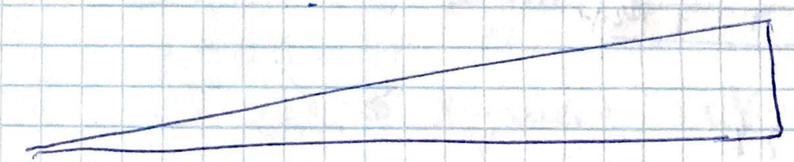




4) Сортируем половину A



5) Merge (A, B, C)



Замечание: чтобы это работало Merge вместо присваиваний должен делать swap

$(C[i_C] = A[i_A] \leftrightarrow swap(A[i_A], C[i_C])$

Это реализация НЕ стабильна

TimSort, WikiSort - стабильны, не требует доп. памяти, быстрые

05.09.20

СЕМИНАР

MergeSort

- Стабильность

QuickSort

- Не-стабильность

Как сделать любую сортировку стабильной?

для целых чисел

разрядность числа

$O(n \cdot d)$

сортировка по разрядам

$O(n+k)$ Radix Sort, Bucket Sort — вероятная для действ. чисел $\in [0;1]$

Counting Sort (Сортировать массив из небольшого диапазона)

```
dict = {}
for x in A:                          nlogn
    dict[x].push(pos(x))
S = dict.keys() //sorted             O(n)
for s in S:                          O(k)
    for k in dict[s]:                O(k)
        res.push-back(A[k])
```

$O(n \log n)$

$a_0 \quad a_1 \quad a_2 \quad \dots \quad a_{n-1}$

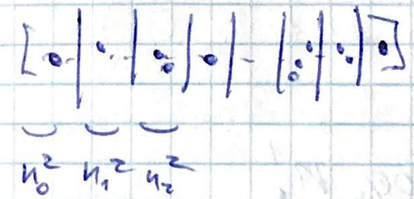
$(a_0, 0)(a_1, 1) \quad \dots \quad (a_{n-1}, n-1)$

$(a_i, i) < (a_j, j)$

$\Updownarrow$

$a_i < a_j$ или $((a_i = a_j)$ и $(i < j))$

Bucket Sort



в среднем $\sum_{i=0}^{n-1} n_i^2 = O(n)$

в худшем $O(n^2)$

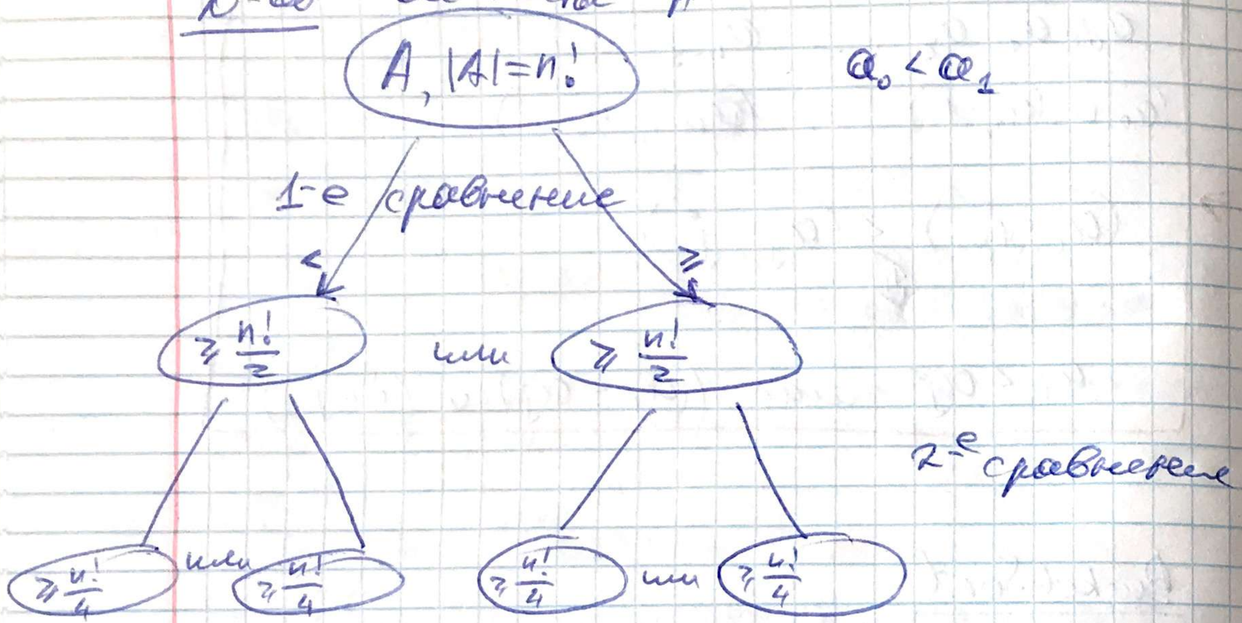
Нижняя граница на скорость сортировки

Th В общем случае (без доп. информации) время работы любого алгоритма сортировки $\Omega(n \log n)$

$f(n) = O(g(n)) \Leftrightarrow f(n) \le c \cdot g(n)$

$f(n) = \Omega(g(n)) \Leftrightarrow f(n) \ge c\, g(n),\ c > 0$

Д-во: все ответы A



Всего сравнений: $\geq \log_2 n!$

$$\log_2 n! = \sum_{k=1}^{n} \log_2 k \geq \sum_{k=\frac{n}{2}}^{n} \log_2 k \geq$$

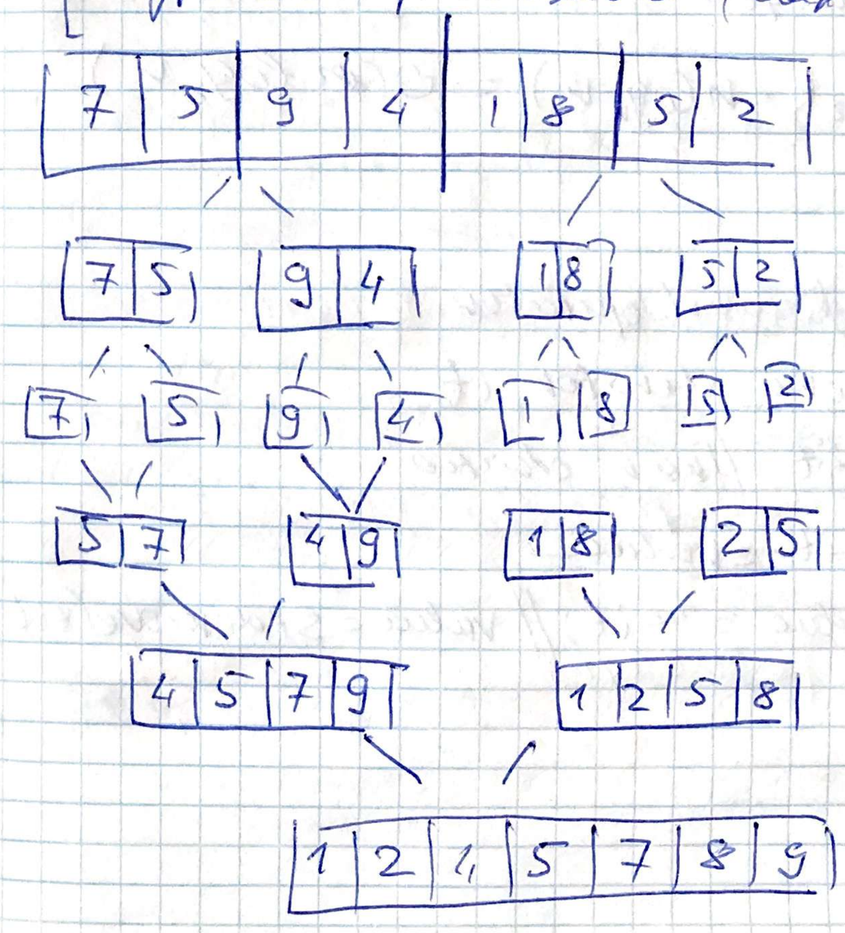
$$\geq \sum_{k=\frac{n}{2}}^{n} \log_2 \frac{n}{2} = \frac{n}{2} \log \frac{n}{2}$$

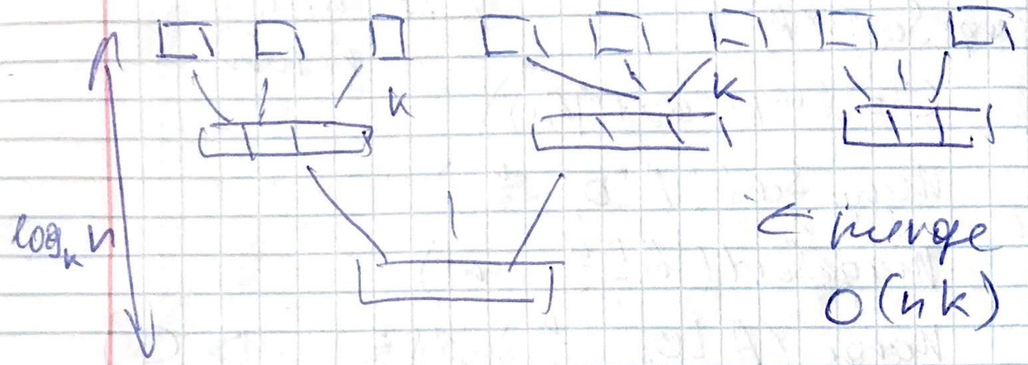
```
MergeSort(A):
    C = tmp_buf(|A|)
    MergeSort(A[0,.., n/2))
    MergeSort(A[n/2,.. n))
    Merge(A[0,.., n/2), A[n/2,.. n), C)

    copy(C, A)
```

if $n \leq 1$: return

[Рекурсивная реализация (сверху вниз)]





$O(k n \log_k n)$

с пирамидой

$O(\log_2 k \cdot n \log_k n) = O(n \log_2 n)$

std::back_insert_operator

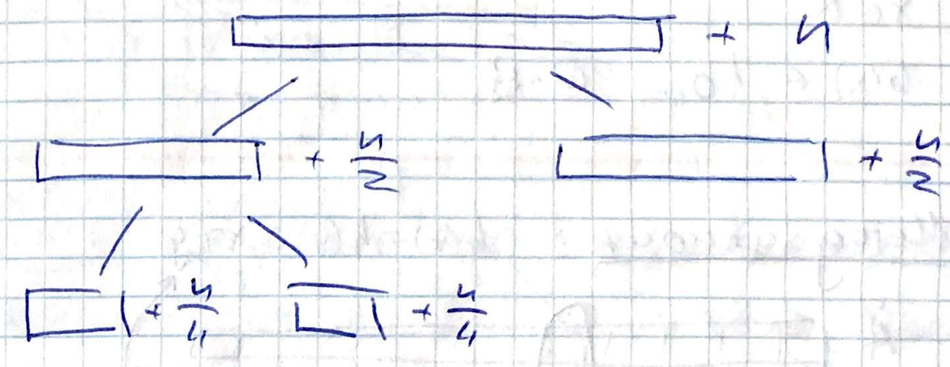
std::move_iterator it

++it; // как обычно

~~*it value;~~

value = *it; // value = std::move(*it)

Память:

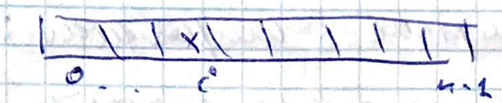


Хеширование

K - мно-во ключей

Insert(key), Erase(key), Find(key)

Если $K = \{0, 1, \dots, n\}$, где n мало, то



Введем $h: K \to \{0, 1, \dots, m-1\}$, m - размер таблицы

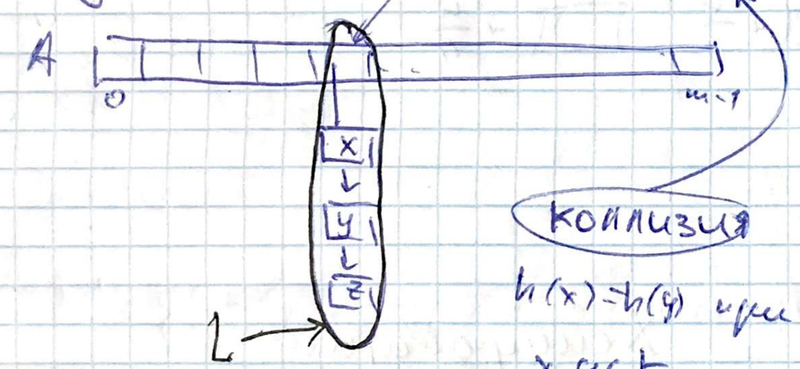
↑ большое мно-во

$A[h(x)]$

$x \in K$

$h(x) \in \{0, \dots, m-1\}$

<u>Метод цепочек:</u> $h(x)=h(y),\ x \neq y$



коллизия

$h(x)=h(y)$ при $x \neq y$

$x, y \in K$

<u>Время работы метода цепочек?</u>

$h(x)=0$ - плохо, т.к. Search за линию

Простое равномерное хеширование:

h выбирается случайно и равномерно из $\{0, 1, \dots, m-1\}^K$

$(K \to \{0, 1, \dots, m-1\})$

Пусть $x \neq y$ $\quad P(h(x)=h(y)) = \dfrac{\#(i,i)}{\#(i,j)} = \dfrac{m}{m^2} = \dfrac{1}{m}$

$$\boxed{P(\text{коллизии}) = \frac{1}{m}}$$

Th1 В случае простого равномерного хеширования время работы Insert, Erase, Find $O(\alpha)$, где $\alpha = \frac{n}{m}$ (load factor)

n - кол-во вставленных элементов

m - размер таблицы

<u>Д-во:</u> Пусть вставл. эл-ты $X = \{x_1, x_2, \dots, x_n\}$

Пусть ищем эл-т x и ищем его в корзине длины L

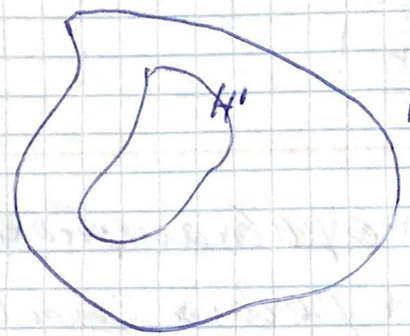
$$\mathbb{E}L = \mathbb{E}\sum_{i=1}^{n} I(h(x)=h(x_i)) = \sum_{i=1}^{n} \mathbb{E}I(h(x)=h(x_i)) =$$

(Expectation)

$$= \sum_{i=1}^{n} P(h(x)=h(y)) = \sum_{i=1}^{n} \begin{cases} 1, & \text{если } x = x_i \\ \frac{1}{m}, & \text{else} \end{cases} = \begin{cases} \frac{n-1}{m}+1, & \text{если } x \in X \\ \frac{n}{m}, & \text{else} \end{cases}$$

$$= O(\alpha) = O\left(\frac{n}{m}\right)$$

$|H| = |\{0, \dots, m-1\}|^{|K|} \gg 2^{64}$



H – мно-во всевозм. хеш-функций

H' – подмно-во ф., которое легко представить ЭВМ

$h(x) = ax^2$ – достаточно сохранить a

<u>Опр</u> H' – универсальное семейство хеш-ф., если $\forall\ x, y \in K\ (x \neq y)$

$P(h(x) = h(y)) \le \frac{1}{m}$ при условии случайного выбора h из H'

<u>Замечание</u>: Ясно, что универсальные семейства удовл. th1

<u>Пример универсального семейства</u>:

$K = \{0, 1, \dots, p-1\}$, p – большое простое число

$\mathbb{Z}_p$ – системы вычетов по mod p

$h_{a,b}(x) = [(a \cdot x + b) \% p] \% m$ – универс. семейство

$a \neq 0,\ a \in \mathbb{Z}_p$

$b \in \mathbb{Z}_p$ $\quad m < p$ — else: $\Delta[x] = true$ ....

<u>Д-во</u>: Зафиксируем $(x, y) \in K(\mathbb{Z}_p)$ $x \neq y$

Докажем, что $P(h(x) = h(y)) \le \frac{1}{m}$

(при случайном выборе (a, b))

$\Delta\ (a, b) \xrightarrow{\text{е}} (\hat{x}, \hat{y})$

$$\begin{cases} (ax + b) \% p = \hat{x} \\ (ay + b) \% p = \hat{y} \end{cases} \longrightarrow$$

1) $\hat{x} \neq \hat{y}$

до-во: $\hat{x} - \hat{y} \overset{p}{\equiv} (ax + b - ay - b) \overset{p}{\equiv}$

$\equiv \underbrace{a}_{\text{не делится на } p}\underbrace{(x - y)}_{\text{не делится на } p} \overset{p}{\not\equiv} 0$

2) $\#(a, b) = (p-1)p$

$\#(\hat{x}, \hat{y}) = p(p-1)$

$\hat{x} \neq \hat{y}$

3) φ – ~~биекция~~ инъекция

<u>Д-во</u>: $\forall (\hat{x}, \hat{y})\ \hat{x} \neq \hat{y}\ \exists!\ (a, b)$:

имеет единств. решение

$$\begin{cases} (ax + b) \% p = \hat{x} \\ (ay + b) \% p = \hat{y} \end{cases}$$

4) из г) и 3) $(a,b) \xrightarrow{\text{ц}} (\hat{x},\hat{y})$ – фиксиров. $\Rightarrow$

$\Rightarrow$ сгенерировать (a,b) то же самое, что сгенерировать $(\hat{x},\hat{y})$

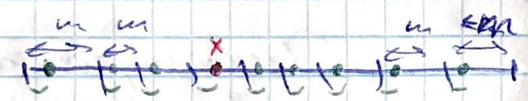
5) $((ax+b)\%p)\%m = \hat{x}\%m$

$((ay+b)\%p)\%m = \hat{y}\%m$

Сколько $(\hat{x},\hat{y})$: $\hat{x}\%m = \hat{y}\%m$?

$$\le \underbrace{p}_{\text{кол-во способов для } \hat{x}} \Big(\underbrace{\left\lceil \frac{p}{m} \right\rceil - 1}_{\text{кол-во способов для } \hat{y}}\Big)$$

кол-во отрезков длины m



$$p\left(\left\lceil \frac{p}{m} \right\rceil - 1\right) < p\left(\frac{p+m-1}{m} - 1\right) = \frac{p(p-1)}{m}$$

6) $P(h(x)=h(y)) = P(((ax+b)\%p)\%m = ((ay+b)\%p)\%m) =$

$$= P(\hat{x}\%m = \hat{y}\%m,\ \hat{x}\ne\hat{y}) \overset{5)}{\le} \frac{p(p-1)/m}{p(p-1)} = \frac{1}{m}$$

чтд

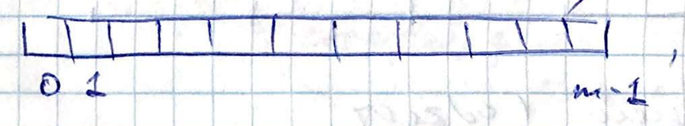
Статическое идеальное хеширование

X – известно и фиксировано заранее (нет Insert, Erase)

Хотим хеш-таблицу без коллизий (Find за $O(1)$ в худшем случае)

$|X| = n$

размер. табл. m



, h – из универ. семейства

$m = n$

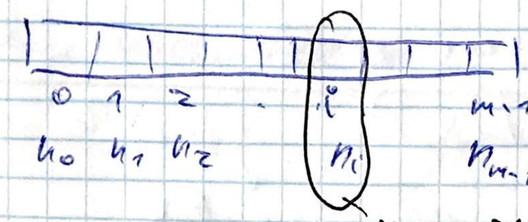
Каким нужно взять m, чтобы без коллизий?

Th 2 Если $m = n^2$, то $P(\text{есть коллизии}) < \frac{1}{2}$

$n^{2+\varepsilon}$ $P \to 0$

$n^{2-\varepsilon}$ $P \to 1$

Хочется таблицу размера $O(n)$



$m = n$

n_{m-1} – кол-во эл-ов в i-й корзине

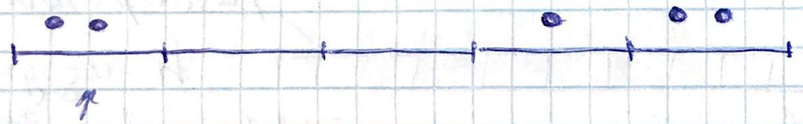
хеш-таблица размера $n_i^2 = m_i$

по Th 2 в i-й таблице не будет коллизий

$$n^2 > \sum_{i=1}^{n} n_i^2$$

Th3 Если $m=n$, то $\mathbb{E}\sum_{i=1}^{n} n_i^2 < 2n$

(n_i - кол-во эл-ов в $i^{\text{й}}$ хеш-таблице)



в среднем 1 объект

$O(1)^2 = O(1)$

$\sum_{i=1}^{n} (O(1))^2 = O(n)$

Сл-е из Th3 $P\left(\sum_{i=1}^{n} n_i^2 \geq 4n\right) \leq \frac{1}{2}$

АЛГОРИТМ построения статич. хеш-таблицы

1) Строим хеш-таблицу $F^{\text{го}}$ уровня так, чтобы $\sum_{i=1}^{n} n_i^2 \leq 4n$ (для этого генерируем h)

~2 итерации пок-но из Th 3

2) Для каждого i строим хеш-табл размера n_i^2 так, чтобы не было коллизий (генерируем h_i)

~2 итерации по Th 2

Таким образом получили отсутствие коллизий за $O(n)$ времени в среднем и $O(n)$ доп. памяти

Такая таблица работает в 2 раза быстрее, чем unordered set

Док-во:

Лемма (Нер-во Маркова)

Пусть $X \geq 0$ случайная величина

Тогда $P(X \geq k) \leq \frac{\mathbb{E}X}{k} \quad \forall k > 0$

Д-во $\mathbb{E}X = \sum_{x \in X} x \cdot P(x) \geq \sum_{k=0}^{\infty} k P(k \leq x < k+1) =$

$= 0 \cdot P(0 \leq x < 1) + 1 \cdot P(1 \leq x < 2) + \ldots + k P(k \leq x < k+1) +$

$+ (k+1) P(k+1 \leq x < k+2) + \ldots \geq$

$\geq k P(k \leq x \leq k+1) + k P(k+1 < x < k+2) + \ldots =$

$= k(P([k;k+1]) + P([k+1;k+2]) + \ldots) = k \cdot P(x \geq k)$

Th2 $m = n^2$, $P(\text{коллизий}) < \frac{1}{2}$

Д-во $X = \{x_1, \ldots, x_n\}$

C - число коллизий

$C = \sum_{i<j} I(h(x_i) = h(x_j))$

$P(\text{есть коллизии}) = P(C \geq 1) \leq \frac{\mathbb{E}C}{1} =$ (нер-во Маркова)

$= \mathbb{E} \sum_{i<j} I(h(x_i) = h(x_j)) = \sum_{i<j} \mathbb{E} I(h(x_i) = h(x_j)) =$

$= \sum_{i<j} P(h(x_i) = h(x_j)) \leq \sum_{i<j} \frac{1}{m} = \frac{n(n-1)}{2} \frac{1}{m} < \frac{1}{2}$

h из сем. ун. $\qquad m = n^2$

Th3 $m = n$, то $\mathbb{E} \sum_{i=1}^{n} n_i^2 < 2n$

Д-во: $\mathbb{E} \sum_{i=1}^{n} n_i^2 = \mathbb{E} \sum_{i=1}^{n} \left(2 \frac{n_i(n_i-1)}{2} + n_i\right) =$

$= 2\left(\underbrace{\mathbb{E} \sum_{i=1}^{n} \frac{n_i(n_i-1)}{2}}_{=C}\right) + \underbrace{\mathbb{E} \sum_{i=1}^{n} n_i}_{=n} = 2\mathbb{E}C + n$

C (кол-во всех коллизий)

$= \leq 2 \cdot \frac{n(n-1)}{2} \cdot \frac{1}{m} + n \overset{m=n}{=} 2n - 1 < 2n$

$C_{n_i}^2$ - число различных пар чисел в $i^{\text{й}}$ ячейке $)$ = число коллизий в ячейке

Следствие: $P\left(\sum_{i=1}^{n} n_i^2 \geq 4n\right) < \frac{1}{2}$

$(k := 4n)$ по неравенству Маркова

12.09.20

СЕМИНАР №2

Хеширование

K – ключи

$x_1, x_2, \dots, x_n \in K$

$n \in |K|$

$+x_i$

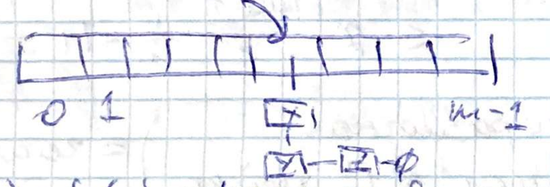
$-x_i$

$?x_i$

$$\boxed{T = O(1+\alpha), \quad \alpha = \frac{n}{m}}$$

$h: K \to \{0, 1, \dots, m-1\}$

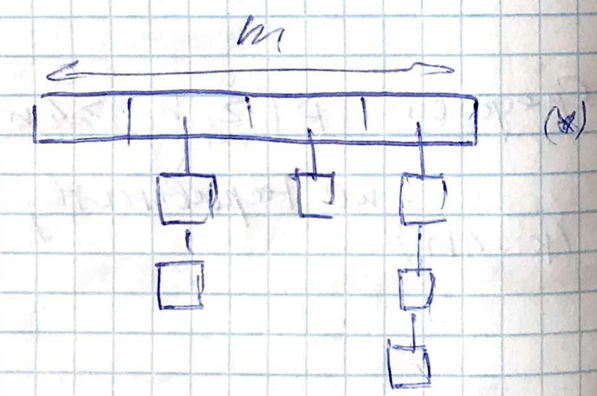
$h(x) = i$



$h(x) = h(y)$ – коллизия

std::hash

$h(x) \% m$

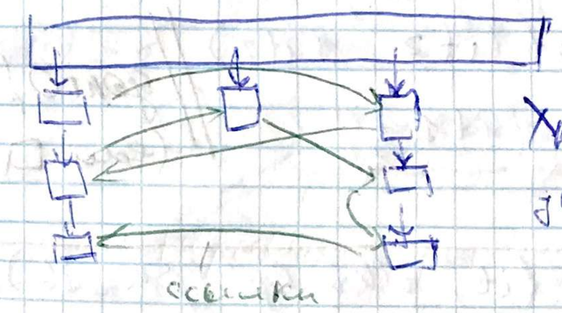


если $\frac{n}{m} > C$: „проделаемся по (*) по всем элементам и находим новое место в (**)"

```
template <class T, class Hash = std::hash<T>,
class HashTable {                    class Equal = std::equal<T> >
    vector <list<T>> Buckets_;
    size_t size_;
    Hash hash_;
    Equal eq_;
    Insert, Erase, Find
```

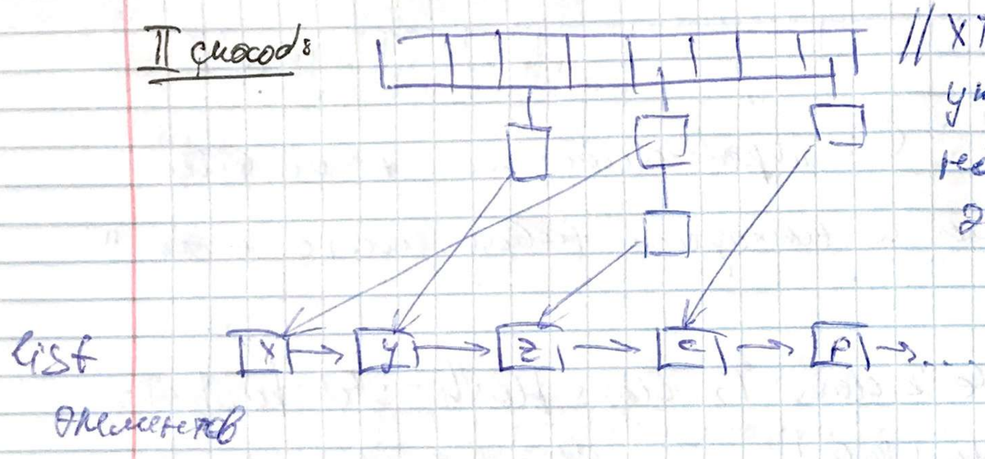
Вывести эл-ты ХТ в порядке добавления.

Способ:

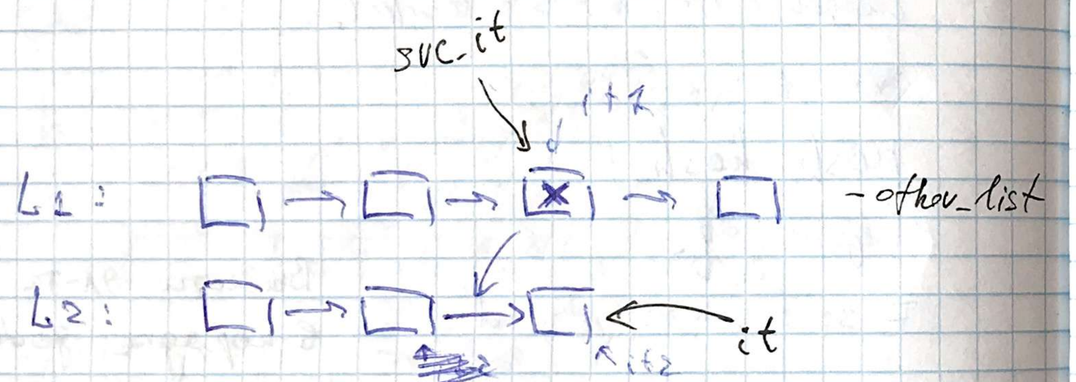


Хранить со значением указателем на след.

II способ:



// XT хранит указатели на свои эл-т



L2.insert(it2, *it2)

L1.erase(it1)

// долго! (если [ ] - вектор большой)

L2.splice(it, other_list, suc_it)

куда, откуда

просто перефрасьвам указатели

При перехешировании

Универсальное семейство хеш функций

H - мно.во всех функций

h - случайная ф.

$\forall x,y \quad P(h(x) == h(y)) = \frac{1}{m}$
$x \neq y$

Пример: $K = \{'a', 'b', 'c', 'd'\}$
$m = 2$

$h('a') = 1$
$h('b') = 1$
$h('c') = 0$
$h('d') = 1$

случайная ф.

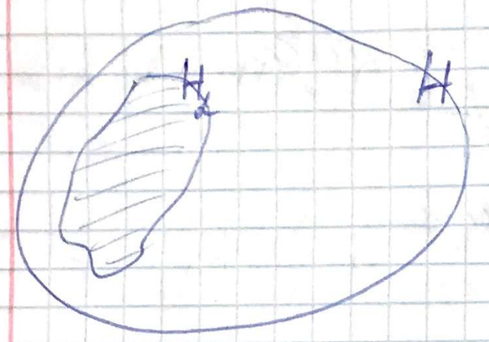
чтобы запомнить значения этой случайной ф. нужна хеш-таблица

нужно по-другому

$h(x) = f_\alpha(x)$

$\forall \alpha \quad f_\alpha$ - детерминированная

α - случайная



Нужно достаточно/много коллизий, чтобы сопоставлялся (a) $\Rightarrow$ нужна ф. $f(x)$

При переполнении нужна другая ф. $\Rightarrow$

$\Rightarrow$ лучше взять $f_\alpha(x)$

α-случайная

Опр $\{f_\alpha(x)\}_{\alpha\text{-случайное}}$ — УНИВЕРСАЛЬНОЕ СЕМЕЙСТВО,

если $\forall x, y : x \neq y \quad P(f_\alpha(x) = f_\alpha(y)) \leq \frac{1}{m}$

Пример: $K = \{0, \dots, p-1\}$, p — простое

$$f_{a,b}(x) = ((ax+b)\,\%\,p)\,\%\,m$$

Теоретическая задача

Пусть $P(f_\alpha(x) = f_\alpha(y)) \leq \varepsilon$

д-ть: $\varepsilon \geq \frac{1}{m} - \frac{1}{k}$

Д-во: от противного

Пусть $\forall x, y \quad P(f_\alpha(x) = f_\alpha(y)) < \frac{1}{m} - \frac{1}{k}$

$$P = \frac{(\text{кол-во } \alpha : f_\alpha(x) = f_\alpha(y))}{\text{кол-во } \alpha} < \frac{1}{m} - \frac{1}{k}$$

$$\frac{1}{m} - \frac{1}{k} > \frac{|A_{xy}|}{|H_\alpha|} \Rightarrow \underbrace{A_{xy} < H_\alpha\left(\frac{1}{m} - \frac{1}{k}\right)}$$

Сколько всего коллизий во всех ф-ях f_α и при $\forall x, y$? $\Big) < C_k^2 \cdot H_\alpha\left(\frac{1}{m} - \frac{1}{k}\right)$

выбираем x, y

А самую лучшую в мире

размещаю k объектов

в каждой ячейке $\frac{k}{m}$ объектов. Всего коллизий: $m \cdot C_{\frac{k}{m}}^2$

для α-fix

кол-во $\alpha \to |H_\alpha|$

Всего коллизий коллизий $\sum C_k^2 |H_\alpha| \left(\frac{1}{m} - \frac{1}{k}\right) =$

$$= |H_\alpha| \left(\frac{k(k-1)}{2m} - \frac{k-1}{2} \right) \quad (1)$$

для лучшей мерсх. ф-и число коллизий $m C_{\frac{k}{m}}^2 |H_\alpha|$

$$= |H_\alpha| \cdot \frac{\frac{k}{m}\left(\frac{k}{m}-1\right)}{2} \cdot m =$$

$$= |H| \left(\frac{k^2}{2m} - \frac{k}{2} \right) \quad (2)$$

$(1) < (2) \Rightarrow$

Статическая хеш таблица

Insert

Erase

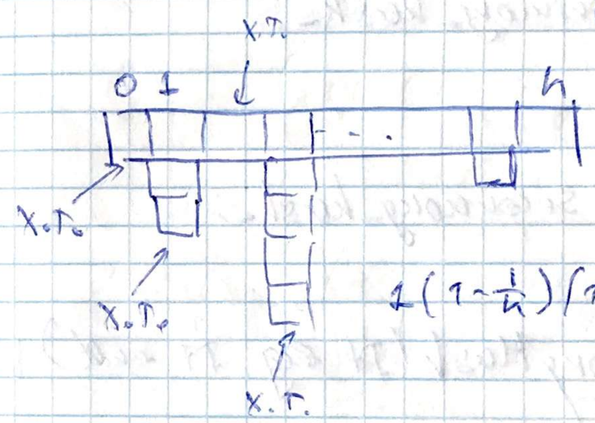
Find ← хотим за $O(1)$ в худшем случае

$X = \{x_1, x_2, \dots, x_n\} \subseteq K$

фиксированно

$O(n)$ памяти

$O(n)$ - построение



$m = n$

вероятность без коллизии

$$1\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\dots\left(1-\frac{n-1}{n}\right)$$

$$= \frac{1}{n^n} n! \to 0$$

Пусть в $i^{й}$ ячейке n_i элементов

построим $i^{го}$ х.т. без коллизий!

(th) Если $m = n^2$, то $P_{коллизий} < \frac{1}{2}$, т.е. $m_i = n_i^2$, то с большой вероятностью без коллизий

(th) $P(\sum n_i^2 > 4n) < \frac{1}{2}$

1) ~2 итерации

2) $\forall i$ подбираю h_i: не было коллизий ~2 итерации

КОД

```
template < ... >
class StaticHashTable {
    vector< vector<T> > table_;
    const size_t size_;
    Hash primary_hash_;
    Equal equal_;
    Vector < Hash > secondary_hash_;
public:
    void BuildPrimaryHash(It beg, It end);
        фотка в телефоне
}
```

класс: operator()1..3

n_i^2 занят

```
Hash BuildSecondaryHash(It beg, It end);
    Hash tmp_hash;
    bool has_built;
    while (!has_built) {
        tmp_hash = hash(rand());
        vector< > v(size^2, 0);
    for (auto i = beg; i != end; ++i) {
        if (*i) {
            has_built = True
        } else {
            v[tmp_hash(*i) % size^2] = true;
    return tmp_hash
}
```

n_i

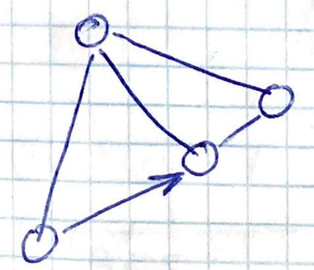
size^2 - size

False

Графы

$G = (V, E)$, V - мно-во вершин

$E \subseteq V \times V$ мно-во ребер



Ориентированные графы

неориент. графы

Представление графов в памяти

1) Список ребер $[(u_1, u_2), (u_2, u_4), (u_5, u_2) \ldots]$

$M = O(|E|)$

- Проверка наличия ребра $O(|E|)$
- Узнать соседей V $O(|E|)$

2) Матрица смежности

$A =$

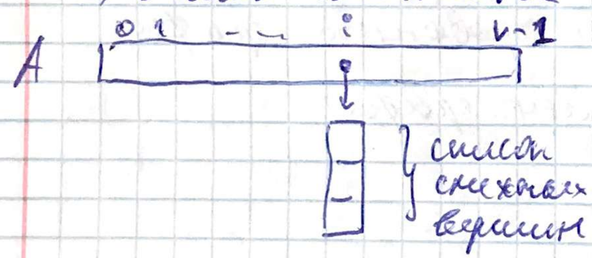
$a_{ij} = I((i,j) \in E)$

либо индикатор наличия ребра, либо кол-во таких ребер

$M = O(V^2)$

- Проверка ~~за~~ наличия $O(1)$
- Узнать соседей $O(V)$

3) Списки смежности



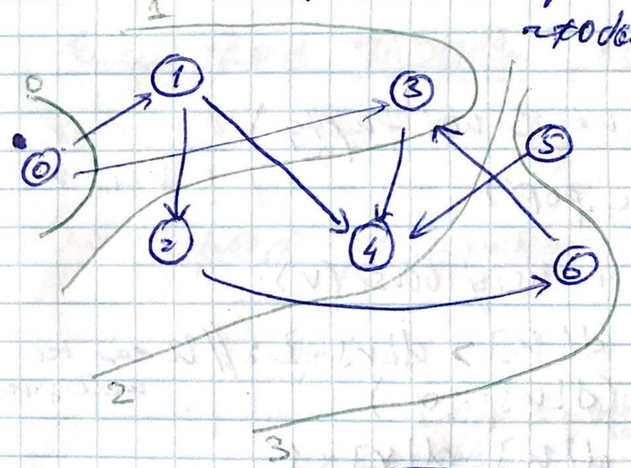
$M = O(\sum_{v \in V} \deg v) = O(E)$

- Проверка наличия ребра $O(\deg v)$
- Узнать соседей $O(\deg v)$

Обход в ширину (Breadth-First Search) (поиск)

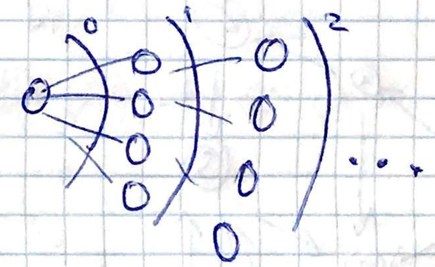
BFS

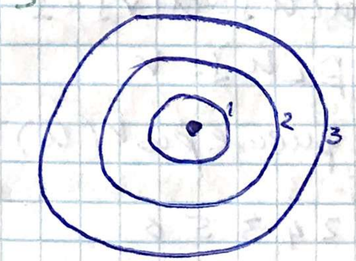
«Посетить вершины графа в опред. порядке, чтобы было хорошее свойство»



~~0~~ ~~1~~ ~~3~~ ~~2~~ ~~4~~ ~~6~~

0	1 3	2 4	6



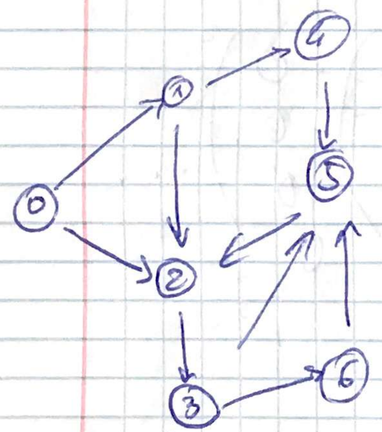


> BFS для min расстояния до вершины
> или
> найти ребра, до которых можно добраться из стартовой вершины

стартовая вершина

```
BFS(G, S):
    queue = {} //очередь
    d[0 .. v-1] = ∞ //расстояние от S до вершин
    p[0 .. v-1] = 1 //предки (из какой вершины обнар. v = p[v])

    q.push(S)
    d[S] = 0
    while (queue is not empty):
        v = queue.pop()
        for u in neighbors(v):
            if d[u] > d[v] + 1 : //u еще не посещена (d[u] == ∞)
                d[u] = d[v] + 1
                p[u] = v
                queue.push(u)
```



0 1 2 4 3 5 6

	0	1	2	3	4	5	6
d	0	1	1	2	2	3	3
p	⊥	0	0	2	1	4	3

Если матрица смежности:

$V(\text{while}) \cdot V(\text{for}) \Rightarrow O(V^2)$

Если списки смежности:

$V(\text{while}) \cdot \deg V(\text{for}) \Rightarrow V + \Sigma \deg V$

$\Downarrow$

$O(V+E)$

Если граф <u>плотный</u>, то $E \sim V^2$ $O(V^2)$

$(C_V^2) = O(V^2)$ в каком случае

Если граф <u>разреженный</u>, то $E \sim V$

$O(V^2)$ – МС

$O(V)$ – СС

th (корректность алг. BFS)

По завершении BFS $\forall v \in V$ $d[v] = \rho(S, v)$

оценка

истинное кратч. расстояние

Д-во База: S $d[S] = 0 = \rho(S, S)$

ПИ: верно нашли и построили все кратчайшие пути длина $\le n$

(*) $(d[v] \le n$ или $\rho(S, v) \le n) \Rightarrow d[v] = \rho(S, v)$

если нашли, то хороший

если хороший то найдем

Переход:

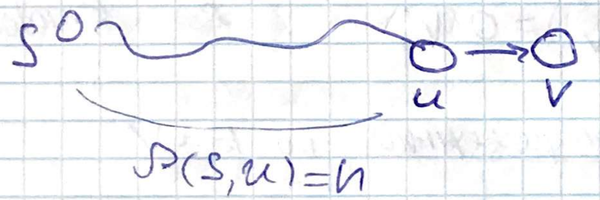
1) Пусть $d[v] = n+1$ (нашли какой-то путь длины $n+1$)

$\overset{(*)}{\Rightarrow} \rho(S,v) \geq n+1 \Rightarrow$

$\Rightarrow d[v] = \rho(S,v) = n+1$

2) Пусть $\rho(S,v) = n+1$

Рассмотрим КП из S в V



(u - вершина перед V в КП)

$\overset{(*)}{\Rightarrow}$

$d[u] = n \Rightarrow d[v] \leq n+1$ (т.к. когда происходила обработка u, мы встретили V)

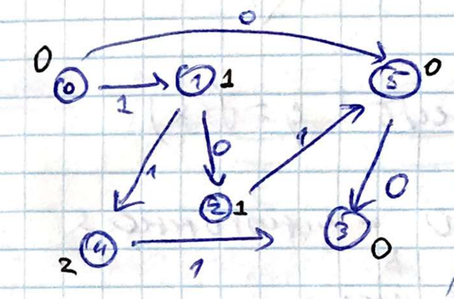
$\rho(S,v) = d[v] = n+1$ $(d[v] \geq \rho(S,v))$

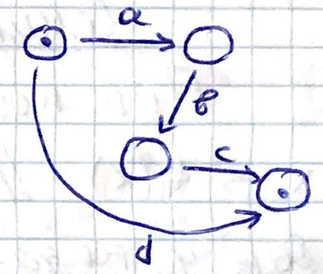
НЕКОТОРЫЕ МОДИФ. BFS

КП во взвешенных графах

1) 0-1 Граф

(вес веса, либо 0, либо 1)

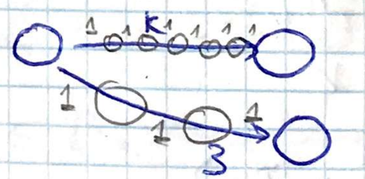




"У ребра есть вес"

Решение: Вместо очереди использовать деque и вершины до которых ведут ребра 0 класть в начало, а остальные - в конец

2) 0-K Граф (веса лежат $\{0, 1, .., k\}$)



"Добавим фиктивные вершины, в ребра с весом > 1"

$V' \leq V + (k-1)E$ // в худшем случае

$E' \leq k \cdot E$

$O(V' + E') = O(V + k \cdot E)$

3) 0-к Граф (за $O(kV+E)$)

19.09.20

Семинар

Бин поиск по ответу

$x \to f(x)$ // сложно

Дан y надо проверить $y = f(x)$

Пример: $e^x \ln x = 1000$ – монотонная

$x \in [\underset{A}{\frac{\ln 1000}{2}}; \underset{B}{\ln 1000}]$

x_1 = середина X;

$g(x_1) \geq 1000$ → $[A; x_1]$

$g(x_1) < 1000$ → $[x_1; B]$

Пример: $\boxed{\text{за } x \text{ сек}}$ $\boxed{\text{за } y \text{ сек}}$ min T – ?

Успеваем ли за T_0 расчитать N чисел?

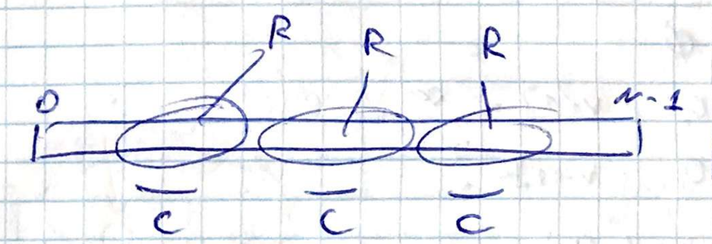
$\left\lceil \frac{T_0}{x} \right\rceil + \left\lceil \frac{T_0}{y} \right\rceil \geq N$

$T_0 \in [-1; N_x]$

$2 \cdot 10^9$

Делим на 2 $\log_2 (N_x + 1) = 9 \log_2 10 \leq 36$

Пример:



// отсортир. порядку

T – число неудобства

1) монотонность по T

2) Проверка, явл. ли T оптимальная

130 160 170 190 205 225 225 260

можно ли за $T \leq 30$?

~~130~~ 160 170 190 205 225 225 ~~260~~

$T \in [-1; \max - \min]$

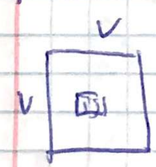
BFS

V - кол-во вершин

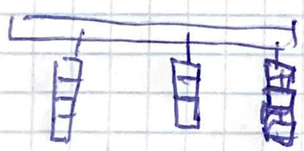
E - кол-во ребер

```
BFS(G, S):
    q = ∅
    d[0, .., v-1] = ∞
    p[0, .., v-1] = ?
    q.push(S)
    while (q is not Empty):
        v = q.pop()
        for u in neighbours(v):
            if d[u] = ∞:
                q.push(u);
                d[u] = d[v] + 1
                p[u] = v
```

1) Матрица смежности



2) Списки смежности

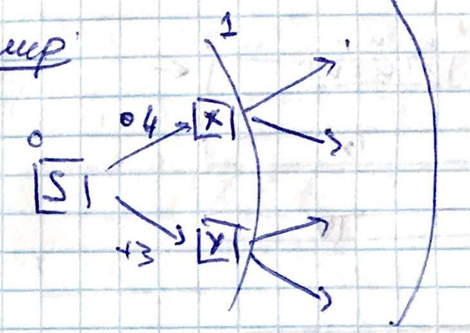


Задача

```
std::vector<int> GetShortestPath(const Graph& g, int from, int to){
```

```
using Graph = std::vector<std::vector<int>>;
```

Пример:



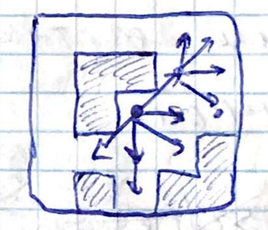
for u in neighbors(v):

$\Updownarrow$

$u_1 = v + 3$

$u_2 = v \cdot 4$

Пример

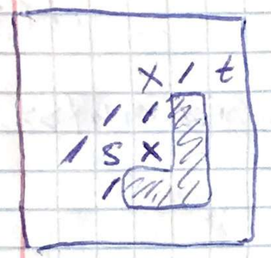


Решение:

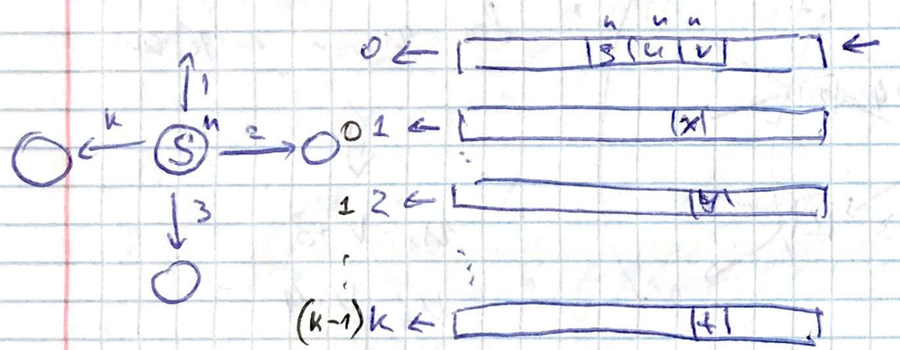
```
size_t GetNumOfCommands(int a, b){
    queue q;
    unordered_map<int, int> d;   // i -> расстояние
    (unordered_map<int, int> p;)
    q.push(a)
    d[a] = 0;
    while (not q.empty()){
        v = q.pop();
        u1 = q + 3
        u2 = q * 4
```

```
if d[u1] not exist and u1 ≤ b:
    q.push(u1)
    d[u1] = d[v] + 1
    p[u1] = v;
if d[u2] not exist: u2 ≤ b
```

Задача 2003



~~visited[x] = 6~~

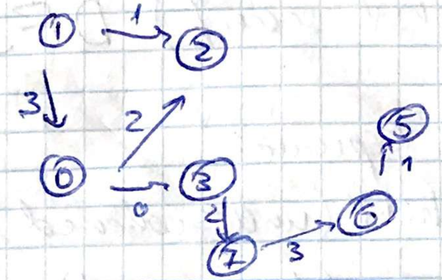


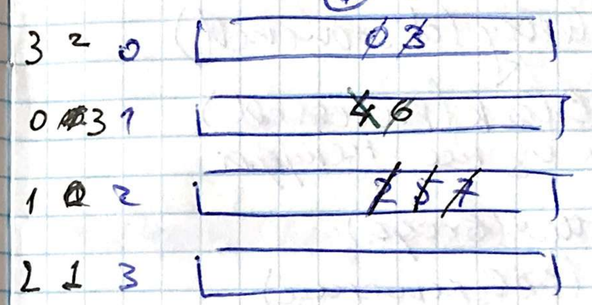
Номер очереди - расстояние от текущей вершины до рассматриваемой

~~if d[u] = ∞ → d[u] = d[x] + w~~ ~~размер ребра~~

```
x = q.pop();
if visited[x]:
    continue
```

	0	1	2	3	4	5	6	7
d	0	∞	2	0	5	2	5	7





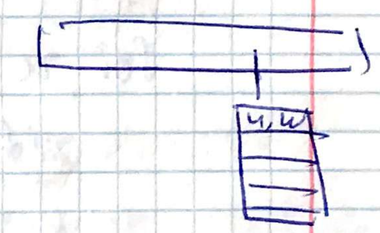
Каждая вершина максимум k раз появляется в очередях

$O(kV)$

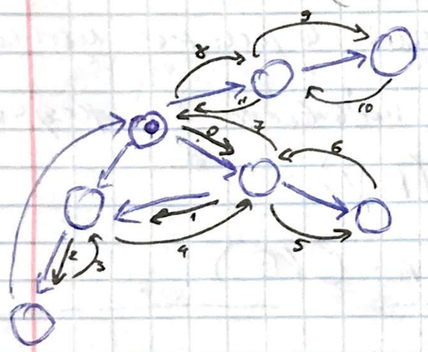
Цикл: $O(E)$

$O(kV+E)$

```
BFS(G, s, k):
    visited[0,..,v-1] = false
    queues[0... k] = ∅;
    d[0...  v-1] = ∞
номер текущей очереди → id = 0
    queues[id].push(s);
    d[s] = 0;
    while True:
        while(queues[id] not empty):
            v = queues[id].pop()
            if (visited[v]) continue;
            for u, w, in neighbours(v):
                if d[u] > d[v]+w : queues[(id+w)%(k+1)].push(u)
                    ~~push~~ d[u] = d[v]+w
            visited[v] = True
id = (id+1)%(k+1)
```



Обход (Поиск) в глубину (Depth-first search) DFS



Цвет вершин:

- white (undiscovered)
- Gray (discovered)
- Black (processed) идти из нее некуда

t_in - время обнаружения (входа)

t_out - время выхода (завершение)

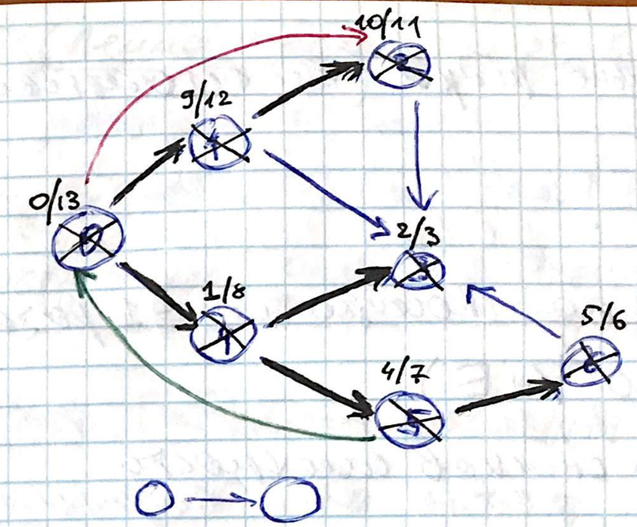
DFSVisit(G, v):

```
color[v] = Gray
t_in = ++time
for u in G.neighbors(v):
    if color[u] == white:
        DFSVisit(G, u)
color[u] = Black
t_out[v] = ++time
```

} при обнаружении

} рекурсивный запуск от соседей

} при выходе



стек вызовов

- Дерево (неор. граф) - связный граф без циклов
- Дерево (ор. граф) - слабосвязный граф, у которого степень захода каждой вершины = 1, (кроме одной, у которой = 0) — корень

Классификация ребер при DFS

↑ - ребра дерева (ребра, по которым откр. новые вершины)

↑ - обратные ребра (ребра, которые ведут от потомков к предкам)

↑ - прямые ребра (ребро, которое ведет от предков к потомкам, не считая ребер дерева)

↑-перекрестные ребра (все остальные)

<u>Время работы</u>:

Каждая вершина посещается ≤ 1 раза

$O(\sum_{v \in V} 1 + \deg v) = O(V+E)$

↑ для списков смежности

Как классифицировать во время работы алгоритма?

Ребро дерево - ведет в белую вершину

Обратное - ведет в серую вершину

Прямые, перекрестные - ведет в черные

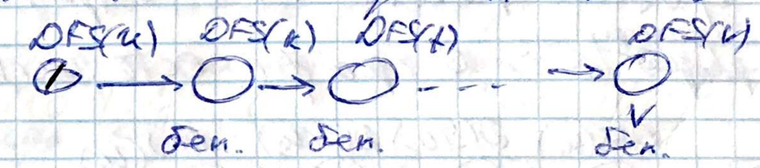
(u,v) - прямое, если $t_in[u] < t_in[v]$

(u,v) - перекр, $t_in[u] > t_in[v]$

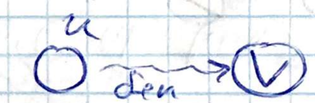
Th (Лемма о белых путях, о корректности DFS)

v - потомок вершины u. (v будет посещена при вызове DFS(u)) $\Leftrightarrow$ ($\exists$ путь из u в v, проходящий only по белым вершинам)

<u>Д-во</u>: $\boxed{\Rightarrow}$ Достаточно рассмотреть момент запусков DFS на пути от u к v



$\boxed{\Leftarrow}$



доказали не обнаружили v при запуске DFS(u)

1)

2)

серая?

т.к. u - черная $\Rightarrow$ все вызовы DFS от соседей завершены $\Rightarrow$ соседи черные

Аналогично из черной вершины нет ребер в белые, т.к. обязательно посещены белого соседа

все вершины от u к v - черные