# Алгоритмы и структуры данных

Сергей Григорян

25 сентября 2024 г.

# Содержание

# 1 Лекция 3

**Замечание.** *QuickSelect можно реализовать с привлечением $O(1)$ доп. памяти. (время $O(n)$ в среднем)*

**Решение.** *Поддерживаем указатели ...*

## 1.1 Дерандомизация

**Определение 1.1.** $a_1, a_2, \dots, a_n$ массив. Его **медианой** наз-вом $a_{\lceil \frac{n}{2} \rceil}$ ($a$ - отсортирован)

**Алгоритм 1.1** (Медиана медиан). $DQS(a_1, \dots, a_n, k)$ (Derandomised Quick Select)

Разделим $a$ на блоки по 5 эл-ов. Пусть $b_i$ - медиана $i$-ого блока. Пусть $x = DQS(b_1, b_2, \dots, b_{\lfloor \frac{n}{5} \rfloor}, \frac{n}{10})$ - медиана массива $b$. Тогда $x$ - наш pivot.

```
DQS(A, k):
  x = DQS(b_1, b_2, ..., b_(n/5), n/10)
  Parition(A, X)
  if (k <= l) => return DQS(B, k)
  if (k <= l + m) => return x
  return DQS(D, k - l - m)
```

Листинг 1: Updated DQS

**Утверждение 1.1.** *Пусть $T(n)$ - время работы алгоритма $DQS$ на массива длины $n \Rightarrow$*

$$T(n) = O(n)$$

*Доказательство.* $x$ - явл. порядковой статистикой массива $A$ с номером $\in [\frac{3}{10}n, \frac{7}{10}n]$

Почему? Представим $b$ как табл. так, что $Mediana(b_i) < Mediana(b_j) (i < j)$

$$\begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{\frac{n}{10}1} & b_{\frac{n}{10}2} & b_{\frac{n}{10}3} & b_{\frac{n}{10}4} & b_{\frac{n}{10}5} \end{pmatrix}$$

Тогда $x$ в центре табл., Ч. Т. Д.

$T(n) = O(n)$(поиск $b_1, \dots, b_n$)$+T(\dfrac{n}{5})$(Нахождение $x$)$+O(n)$(Partition)$+T(\dfrac{7n}{10})$

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7n}{10}) + Cn$$

Докажем, что $T(n) \leq 10Cn, \forall n$

ММM:

База: $n \leq 5$ очев.

Переход: $T(n) \leq T(\frac{n}{5}) + T(\frac{7n}{10}) + Cn \leq \frac{10Cn}{5} + 7Cn + Cn == 10Cn$

□

### 1.1.1 Derandomized Quick Sort

Используя $DQS(A, \frac{n}{2})$ в качестве pivot, получаем новую асимптотику:

$$T(n) \leq 2T(\frac{n}{2}) + O(n)$$

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

## 1.2 Сортировка чисел

**Задача 1.1.** Пусть есть $a_1, \dots, a_n$ - массив чисел $a_i \in \{0, 1, \dots, k\}$. Отсортировать $a$.

**Решение.**

```
cnt[k + 1] = {0, ..., 0}
for i=1..n
  ++cnt[a[i]]
for x=0..k:
  for i=1..cnt[x]
    print(x)

```

Листинг 2: Counting sort

*Асимптотика:* $O(n+k)$

**Определение 1.2. Стабильная сортировка:**
$a_1, a_2, \ldots, a_n \Rightarrow a_{\sigma(1)}, a_{\sigma(2)}, \ldots, a_{\sigma(n)}$, при усл. что равные эл-ты сохраняют свой отн. порядок, т. е., если $a_{\sigma(i)} = a_{\sigma(j)}$ и $i < j$, то $\sigma(i) < \sigma(j)$

**Стабильная сортировка подсчётом**:

```
cnt[k + 1] = {0, .., 0};
for i=1..n
  ++cnt[a[i]]
for x = 1..k:
  cnt[x] += cnt[x - 1]
b = {-1, .., -1}
for i=1..k:
 b[cnt[a[i]]] = a[i] // sigma[cnt[a[i]]] = i
 --cnt[a[i]]
```

Листинг 3: stable counting sort

$cnt[x]$ - кол-во эл-ов $\leq$ x
Асимптотика: $O(n+k)$

**Задача 1.2.** Дан массив пар чисел:

$$(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$$

$$a_i, b_i \in \{0, 1, \ldots, k\}$$

Отсортировать!

**Решение.** 1) *Отсортируем массив пар по b*

2) *Результат стабильно отсортируем по a*
*Почему это корректно?*
*Пусть* $(a_i, b_i) < (a_j, b_j) \iff$

$$\left[\begin{array}{l} a_i < a_j \\ a_i = a_j, b_i < b_j (\textit{стабильность!}) \end{array}\right.$$

### 1.2.1 Least significant digit sort

**Задача 1.3.** Отсортировать массив чисел, $a_i \in \{0, 1, \dots, k\}$, $k$ очень большое

**Решение.** *Сортируем от младшего разряда к старшему стабильно*
*Асимптотика:* $O(\log k(n + 10))$ *(десятич. система)*
*Вместо 10 - СС можно использовать СС с основанием* $2^b$*:*

$$x \mod 2^b = x \wedge ((1 << b) - 1)$$

# 2 Лекция 4

## 2.1 Кучи

**Определение 2.1.** Куча - СД, умеющая:

- Хранить мультимн-во эл-ов $S$
- $insert(x)$ - добавить $x$ в $S$
- $getMin()$ - вернуть $min(S)$
- $extractMin()$ - найти $min(S)$ и удалить его
- $decreaseKey(ptr, \triangle), \triangle > 0$ - уменьшить число по адресу $ptr$ на $\triangle$

**Применения**: алгоритмы Дейкстры, Прима

### 2.1.1 Бинарная (двоичная) куча

Хранил $S$ в массиве $a_1, a_2, \dots, a_n$
Picture(1)
**Требование кучи**: эл-т, записанный в каждой вершине, $\leq$ всех эл-ов своего поддерева
Тогда $getMin()$: return $a_1$;

```
siftUp(u):
  if (v == 1) return
  p = (v / 2)
  if (a[p] > a[v]) {
```

```
    swap(a[p], a[v])
    siftUp(p)
  }

siftDwon(v)
  if (2 * v > n) return
  u = 2v
  if (2 * v + 1 <= n && a[2 * v + 1] < a[u]): u = 2 * v + 1
  if (a[u] < a[v]) {
    swap(a[u], a[v])
    siftDown(u)
  }
```

Листинг 4: siftUp and siftDown

Остальные методы в heap.cpp

Асимптотика всего: $O(\log n)$

**Корректность**

**<u>Лемма</u> 2.1.** *Пусть $a_1, \dots, a_n$ - корректная куча*
*Пусть $a_v \leftarrow x$*

1) *Если $a_v$ уменьшилось, то после $siftUp(v)$ куча вновь станет корректной*

2) *Если $a_v$ увеличилось, то после $siftDown(v)$ куча вновь станет корректной*

*Доказательство.* 1) Индукция по $v$:

База: $v = 1$: куча остаётся корректной, $siftUp$ при уменьшении корня ничего не делает

Переход: $a_v \leftarrow x$

a) $x \geq a_p$ - родитель; нер-во сохраняется, $siftUp$ ничего не делает, куча остаётся корректной

b) $x < a_p$. Тогда сделаем $swap(a_p, a_v)$, тогда нер-во снова сохр., и, по предположению индукции, после $siftUp(p)$ - куча становится корректной.
Picture(2)

2) Индукция от листьев к корню

База: $v$ - лист, куча корректна, $siftDown$ ничего не делает

Переход: Пусть $a_u$ - наименьший из детей $v$
Picture(3)

□

### 2.1.2 HeapSort

Алгоритм:

$$a_1, a_2, \dots, a_n$$

1) $insert a_1, \dots, a_n$

2) $extractMin$ n раз

Асимптотика: $O(n \log n)$

**Замечание.** *Heapsort основан на сравнениях*

**Следствие.** $\neg\exists$ *реализации кучи осн. на сравнениях, в кот.* $insert$ *и* $extractMin$ *работают за* $O(n)$

Процедура $heapify$: строит корректную кучу по $n$ эл-ам без доп. памяти за $O(n)$

```
for i = n...1:
  siftDown(i)
```

Листинг 5: Heapify

Корректность? Индукцией по $i$: после вызова $siftDown(i)$, поддерево с корнем $i$ станет корректной кучей.

*Доказательство.* База: $i$ - лист

Переход: Picture(4)

□

Асимптотика: $O(n)$
Время работы:

- $\frac{n}{2}$ вершин обраб. за 1 оп.

- $\frac{n}{4}$ вершин обраб. за 2 оп.

  $\vdots$

$$Sum = \frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 2 + \ldots = \frac{n}{2} + \frac{n}{4} + \ldots + (\frac{n}{4} \cdot 1 + \frac{n}{8} \cdot 2 + \frac{n}{16} \cdot 3 + \ldots) =$$
$$= n + \frac{n}{2} + (\frac{n}{8} + \frac{n}{16} \cdot 2 + \frac{n}{32} \cdot 3 + \ldots) \leq 2 \cdot n$$

### 2.1.3 Удаление из кучи

$erasex$:

a) По указателю на $x$

   1) $a_v \leftarrow -\infty$
   2) $siftUp(v)$
   3) $extractMin()$

b) По значению $x$
   У нас нет способа найти $x$ в куче, поэтому:

   1) Заведём кучи $A$ - то, что добавили, $D$ - то, что хотим удалить. При запросе удаления $x$, добавляем его в $D$
   2) Если при запросе $getMin(), A.getMin() == D.getMin()$, то удаляем $min$ в обоих кучах и смотрим далее.

Итого: $n$ запросов $= O(n \log n)$