Алгоритмы и структуры данных

Сергей Григорян

4 декабря 2024 г.

Содержание

1 Лекция 2 **4**

1.1 Сортировки 4

1.1.1 Merge Sort (Сортировка слиянием) 5

1.1.2 Quick Sort
(Быстрая сортировка; Сортировка Хоара) 6

1.1.3 Quick Select 7

2 Лекция 3 **7**

2.1 Дерандомизация 7

2.1.1 Derandomized Quick Sort 9

2.2 Сортировка чисел 9

2.2.1 Least significant digit sort 10

3 Лекция 4 **10**

3.1 Кучи 10

3.1.1 Бинарная (двоичная) куча 11

3.1.2 HeapSort 12

3.1.3 Удаление из кучи 13

4 Лекция 5 **14**

4.1 Биномиальная куча 14

4.2 Амортизационный анализ 16

4.2.1 Динамический массив
(std::vector) 16

5 Лекция 6 **17**

5.1 Связные списки 18

5.2 Куча Фиббоначи 18

5.2.1 Consolidate (Операция причёсывания кучи) 19

5.2.2 Анализ времени работы 19

6 Лекция 7 **20**

6.1 Стеки и очереди 20

7 Лекция 10 **21**

7.0.1 Метод потенциалов 23

7.0.2 Возвращаемся к *splay*-дерево 23

8 Лекция 11 (Хэш таблицы) 24
8.1 Совершенное хэширование 26

1 Лекция 2

1.1 Сортировки

Задача 1.1. $a_1, a_2, \ldots, a_n$ - дано Найти $b =$sort(a)

Задача 1.2. $a_1, a_2, \ldots, a_n$ - дано. Найти перестановку $G : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$: $a_{G(1)} \leq a_{G(2)} \leq \ldots \leq a_{G(n)}$

Утверждение 1.1. *Пусть $T(n) \geq n$. Тогда, если одна задача решается за $O(T(n))$, то и вторая тоже.*

Доказательство. $2 \Rightarrow 1$: $b_1 = a_{G(1)}, \ldots, b_n = a_{G(n)}$

$1 \Rightarrow 2$: Отсортируем массив пар: $(a_1, 1), (a_2, 2), \ldots, (a_n, n)$

$$(x_1, y_1) \leq (x_2, y_2) \iff \left[\begin{array}{l} x_1 < x_2 \\ x_1 = x_2, y_1 < y_2 \end{array}\right.$$

□

Теорема 1.1 (Оценка кол-во сравнений в сортировке сравнениями)**.** *Если алгоритм может только сравнивать эл-ты, то время сортировки массива $\in \Omega(n \log n)$*

Доказательство. $a_1, a_2, \ldots, a_n$ - все эл-ты попарно различны.

Вопрос алгоса: $a_i?a_j$ (Дерево сравнений)

Рез-т работы алгоритма зависит от n и п-ти получаемых ответов $(<, >)$

Пусть алгоритм всегда совершает $\leq q$ запросов $(a_i?a_j)$. Тогда есть не более $2^0 + 2^1 + \ldots + 2^q = 2^{q+1} - 1 \leq 2^{q+1}$ возм. протоколов работы алгоритма. Должно быть хотя бы $n!$ разл. протоколов. $\Rightarrow$

$$n! \leq 2^{q+1} \Rightarrow q = \Omega(n \log n)?$$

□

Лемма 1.2.

$$\log(n!) = \theta(n \log n)$$

Доказательство. 1)

$$n! = 1 * 2 * \ldots * n \leq n^n$$

$$\log_2(n!) \leq n \log_2(n) \Rightarrow \log(n!) = O(n \log n)$$

2)

$$n = 2k \Rightarrow n! \geq (k+1) * \ldots * (2k) \geq k^{k+1}$$

$$\log_2(n!) \geq \log_2(k^{k+1}) = (k+1)\log_2 k \geq \frac{n}{2}\log_2(\frac{n}{2}) = \frac{1}{2}n(\log n - 1) = \frac{1}{2}n\log n - \frac{1}{2} \geq \frac{1}{2}n\log n$$

$$\Rightarrow \log_2(n!) = \Omega(n\log_2 n)$$

Аналогично, при $n = 2k + 1$

$\square$

1.1.1 Merge Sort (Сортировка слиянием)

Алгоритм:

1) Если массив длины 1, то выходим, иначе делим его на 2 половины;
2) Рекурсивно сортируем половины
3) "Мёрджим"две половины.

Операция merge:

```
merge(a[0..n - 1], b[0..m - 1], to[0..n + m - 1]):
    i = 0, j = 0
    while (i < n || j < m):
        if (j == m || (i < n && a[i] <= b[j])):
            to[i + j] = a[i]
            ++i
        else
            to[i + j] = b[j]
            ++j
```

Листинг 1: Merge

```
MergeSort(a[0..n - 1]):
    if (n <= 1) return
    k = n / 2
    l = a[0..k]
    r = a[k + 1..n - 1]
    MergeSort(l)
    MergeSort(r)
    Merge(l, r, a)
```

Листинг 2: MergeSort

Асимптотика: $T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n \log n)$

Задача 1.3. 1) Сделать потребление памяти $O(n)$

2) Сделать нерекурсивный MergeSort

Задача 1.4 (О числе инверсий в массиве)**.**

$a_1, a_2, \dots, a_n$ - дано
Инверсия $(i, j) := i < j \wedge a_i > a_j$

Решение. *Для решения просто модифицируем merge:*

```
// ans - global variable
merge(a[0..n - 1], b[0..m - 1], to[0..n + m - 1]):
    i = 0, j = 0
    while (i < n || j < m):
        if (j == m || (i < n && a[i] <= b[j])):
            to[i + j] = a[i]
            ++i
            ans += j
        else
            to[i + j] = b[j]
            ++j
            ans += i
```

Листинг 3: Merge for inversions

1.1.2 Quick Sort (Быстрая сортировка; Сортировка Хоара)

```
Partition(a[0..n - 1], x):
    B =>  < x
    C =>  = x
    D =>  > x
    return (|B|, |C|, |D|)

QuickSort(a[0..n - 1]):
    if (n <= 1) return;
    x = a[random(0, n - 1)]
```

```
    l, m, r = Partition(a, x)
    QuickSort(a[1..l - 1])
    QuickSort(a[l + m..n - 1])
```
Листинг 4: Quick Sort

Теорема 1.3. *В среднем асимпт.* $= O(n \log n)$

1.1.3 Quick Select

Определение 1.1. $a_1, a_2, \ldots, a_n$ - массив **k-ая порядковая статистика:** - эл-т на k-ом эл-те после сортировки.

Задача 1.5. Найти k-ую порядковую статистику в массиве a.

Решение.

```
QuickSelect(a[1..n], k):
    if (n == 1) return a[1];
    l, m, r = Partition(a, a[random(1, n)])
    if (k <= l) return QuickSelect(a[1..l], k)
    if (k <= l + m) return x
    return QuickSelect(a[l + m..n], k - l - m)
```
Листинг 5: QuickSelect

2 Лекция 3

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

Решение. *Поддерживаем указатели . . .*

2.1 Дерандомизация

Определение 2.1. $a_1, a_2, \ldots, a_n$ массив. Его **медианой** наз-вом $a_{\lceil \frac{n}{2} \rceil}$ (a - отсортирован)

Алгоритм 2.1 (Медиана медиан)**.** $DQS(a_1, \ldots, a_n, k)$ (Derandomised Quick Select)

Разделим a на блоки по 5 эл-ов. Пусть b_i - медиана i-ого блока. Пусть $x = DQS(b_1, b_2, \ldots, b_{\lfloor \frac{n}{5} \rfloor}, \frac{n}{10})$ - 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 6: Updated DQS

Утверждение 2.1. *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.

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

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

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

ММИ:

База: $n \leq 5$ очев.

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

$\square$

2.1.1 Derandomized Quick Sort

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

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

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

2.2 Сортировка чисел

Задача 2.1. Пусть есть $a_1, \ldots, a_n$ - массив чисел $a_i \in \{0, 1, \ldots, k\}$. 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 7: Counting sort

Асимптотика: $O(n + k)$

Определение 2.2. 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 8: stable counting sort

$cnt[x]$ - кол-во эл-ов $\leq$ x

Асимптотика: $O(n+k)$

Задача 2.2. 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 а

Почему это корректно?

Пусть $(a_i, b_i) < (a_j, b_j) \iff$

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

2.2.1 Least significant digit sort

Задача 2.3. Отсортировать массив чисел, $a_i \in \{0, 1, \ldots, k\}$, 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*:*

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

3 Лекция 4

3.1 Кучи

Определение 3.1. 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

3.1.1 Бинарная (двоичная) куча

Храним S в массиве $a_1, a_2, \ldots, a_n$

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 9: siftUp and siftDown

Остальные методы в heap.cpp

Асимптотика всего: $O(\log n)$

Корректность

Лемма 3.1. *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)

□

3.1.2 HeapSort

Алгоритм:

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

1) $inserta_1, \dots, a_n$

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 10: Heapify

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

Доказательство. База: i - лист

Переход: Picture(4)

$\square$

Асимптотика: $O(n)$
Время работы:

- $\frac{n}{2}$ вершин обраб. за 1 оп.
- $\frac{n}{4}$ вершин обраб. за 2 оп.

$\vdots$

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

3.1.3 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 $= O(n \log n)$

4 Лекция 5

4.1 Биномиальная куча

Хотим следующие операции:

- getMin()
- extractMin()
- insert(x)
- decreaseKey(ptr, $\triangle$)
- merge(heap1, heap2) - обЪединение куч.

Определение 4.1. Биномиальное дерево ранга k:

k = 0) T_0 - одна вершина

k = 1) T_1 - вершина с одним ребёнком

k = 2) T_2 - Дерево T_1, к корню кот. ещё подвешено T_1

k = n) T_n - Дерево T_{n-1}, к корню кот. ещё подвешено T_{n-1}

Кроме того, в вершинах дерева, есть числа, удовл. усл. обыкновенной кучи (значение в родителе $\leq$ значения в сыновьях)

Определение 4.2. Биномиальная куча - это набор биномиальных деревьев, попарно различных рангов.

Пример.
T_0, T_1, T_5 - *OK*
T_3, T_5, T_5 - *NOT OK*

Замечание. *1) Если в куче всего n - эл-ов, то в ней не более $\log_2 n$ - деревьев, т. к. в T_k ровно 2^k вершин.*

Пример. $n = 11 = 1011_2 \Rightarrow T_0 + T_1 + T_3$

2) Дерево ранга k имеет глубину k

$$k \leq \log_2 n$$

Реализация:

- getMin():
  Храним указатель на корень с наим. значением. $\Rightarrow O(1)$
- merge(H_1, H_2):

  1) Если в H_1 и H_2 не содержатся деревья одинаковых рангов, то просто объединяем.
  2) Иначе пусть есть дерево L_k, R_k - два дерева одинакового ранга. Сделаем из них T_{k+1}. Повторяем процедуру, пока у нас есть деревья равных рангов. ($O(\log_2 n)$)
- insert(x):
  Заводим биномиальную кучу из одной вершины с значением x, затем merge новой и старой кучи $\Rightarrow O(\log_2 n)$
- extractMin():
  Пусть min вершина в H_2. На самом деле дерево H_2 - тоже корректная куча. Оставшуюся кучу обозначим за H_1. Удалим из H_2 min, из оставшихся деревьев составим новую кучу H_2' и смёрджим его с H_1
- decreaseKey(ptr, $\triangle$):
  Как в бинарной. ($O(\log_2 n)$) + Проверить, не изменился ли min корень

4.2 Амортизационный анализ

Определение 4.3. Пусть S - какая-то СД, способная обрабатывать m типов запросов. Тогда ф-ции $a_1(n), a_2(n), \ldots, a_m(n)$ наз-ся учётными (амортизационными) асимптотиками ответов на запросы, если $\forall n \forall$ п-ть из n запросов с типами $i_1, i_2, \ldots, i_n$ суммарное время их обработки $=$ $O(\sum_{i=1}^{n} a_{i_j}(n))$

Пример. *В бинарной куче:*

- *insert:* $O(\log n)$
- *extractMin:* $O^*(\log n)$
- *getMin():* $O^*(\log n)$
- *erase: аморт.* $O(\log n)$

Сл-но, любые n *запросов работают за* $O(n \log n)$

Замечание. *Можно даже считать так:*

- *insert:* $O^*(\log n)$
- *extractMin:* $O^*(1) \leq k$
- *getMin:* $O^*(1)$
- *erase:* $O^*(1) \leq k$

На n *запросов.*
Из них k *- insert. Тогда реальное время работы:* $O(k \log k + n - k)$

4.2.1 Динамический массив (std::vector)

Хранит массив: $a_0, a_1, \ldots, a_{n-1}$
Отвечает на запросы:

- []: по i вернуть a_i - $O(1)$
- push-back x: добавить x в конец массива.
- pop-back: удалить последний эл-т.

5 Лекция 6

Утверждение 5.1 (Метод бух. учёта)**.** *Пусть к структуре поступает n запросов, t_i - реальное время выполнения i-ого запроса.*
Пусть d_i - число монет, положенных на счёт; w_i число снятых монет.
Тогда, если баланс на счёте всегда ≥ 0, то:

$$a_i = t_i + d_i - w_i$$

учётная стоимость i-ого запроса.

Доказательство. $\{ a_i \}$ - явл-ся уч. стоимостями, если реальное время $= O(\sum_{i=1}^{n} a_i)$

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} t_i + d_i - w_i = \sum_{i=1}^{n} t_i + \sum_{i=1}^{n} (d_i - w_i), (d_i - w_i \geq 0)$$

$$\Rightarrow \sum_{i=1}^{n} a_i \geq \sum_{i=1}^{n} t_i$$

□

Рассм. динам. массив:

- Лёгкий push-back/pop-back:
  - $t_i = O(1)$
  - $d_i \leq 20$
  - $w_i = 0$
  - $a_i = O(1)$
- Тяжёлый push-back/pop-back:
  - $t_i \leq 4C$
  - $w_i = t_i$
  - $d_i = 0$
  - $a_i = 0$

  $O^*(1)$ - учётное время работы всех операций

5.1 Связные списки

1) За линейное от размера списка время можно просмотреть все его эл-ты

2) В списке можно выполнять удаление по указателю за $O(1)$

Замечание. *Для удобства реализации, в начало и конец списка можно положить некий фиктивный эл-т.*

5.2 Куча Фиббоначи

Умеет:

1) getMin - $O(1)$

2) insert - $O(1)$

3) merge - $O(1)$

4) extractMin - $O^*(\log n)$

5) decreaseKey - $O^*(1)$

Куча - **связный список** деревьев, каждое из кот. удовл. требованиям кучи.

Что храним в вершине?

1) Указатель на левого и правого брата.

2) element x $\in X$

3) **Связный список** детей (А именно, указатель на самого левого сына)

4) Степень вершины (Кол-во детей) (int degree)

5) bool mark: вырезался ли 1 из сыновей.

6) Указатель на родителя.

Разбираем операции:

- getMin: Вместе со списком корней будем хранить указатель на минимальный корень.
- Merge: склеиваем два списка корней и пересчитываем min-root
- insert x: Создаём кучу из одного эл-та + merge.
- extractMin: Удалим вершину min-root, а всех детей merge-ым со старой кучей.
- decreaseKey ptr $\triangle$: у корней можно удалять произв. кол-во сыновей., а у всех остальных не больше одного.

5.2.1 Consolidate (Операция причёсывания кучи)

Будем проходиться по всем корням и объединять деревья одного ранга. (ранг = degree). Объединение:

Из двух деревье одного ранга (H_1, H_2), пусть H_1 - с меньшим числом в корне. Тогда подвесим H_2 к H_1. Теперь ранг H_1 увеличился на 1.

Пусть $D(n)$ - max возможный ранг вершины в куче из n эл-ов. (Позже покажем, что $D(n) = O(\log n)$). Тогда реальное время работы:

$$D(n) + \#(\text{Объединений деревьев})$$

5.2.2 Анализ времени работы

Метод бух. учёта:

На каждом корне лежит по 1 монетке, на каждой вершине с mark = true лежит по 2 монетки. Тогда:

1) decreaseKey работает за O(1)
2) extractMin работает за $O^*(D(n))$

Осталось показать, что $D(n) = O(\log n)$

Пусть $S(k)$ - min кол-во вершин в дереве, ранг кот. равен k
$S(0) = 1, S(1) = 2, S(k) = ?$

$$S(k) \geq 1 + 1 + S(0) + \ldots + S(k-2)$$

Отсюда следует, что $S(k) \geq F_{k+2}$, где F_k - k-ое число Фиббоначи.

$$S(k) = \Omega(\phi^{k+2})$$

Утверждение 5.2.

$$2 + F_2 + F_3 + \ldots + F_k = F_{k+2}$$

Доказательство.

$$k = 2\colon 2 + F_2 = 3 = F_3$$

$$k = k\colon 2 + F_2 + \ldots + F_{k+1} = F_{k+3}$$

□

$$F_n = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2} > 1$$

$$F_n = \Theta(\phi^n) \Rightarrow S(k) = \Omega(\phi^k)$$

Если в дереве n вершин, то макс. степень корня $\leq \log_\phi(n)$

6 Лекция 7

6.1 Стеки и очереди

Определение 6.1. Стек - СД:

- push x - добавить x в начало
- pop x - удалить первый эл-т из начала
- top x - вывести первый эл-т из начала.

Реализация на основе связного списка.
ПСССССССССССП

ОБРАТНАЯ ПОЛЬСКАЯ ЗАПИИИИИИИИИИИИИСЬ
Спарсы

7 Лекция 10

Храним мн-во эл-ов S:

- $find(x)$: сообщить, есть ли x в S
- $insert(x)$: добавить x в S (дубликаты игнорируются)
- $erase(x)$: удалить x из S

optional: перечислить эл-ты S в порядке возрастания

optional: $merge(S_1, S_2) = S_3 = S_1 \cup S_2$

optional: $split(S, k) = \{ S_1, S_2 \}$:

$$S_1 = \{ x \in S \mid x \leq k \}$$

$$S_2 = \{ x \in S \mid x > k \}$$

Определение **7.1. Бинарное дерево поиска** — это дерево, со следующими св-вами:

1) У каждой вершины не более 2-ух сыновей
2) Если в вершине записано число x, то во всех вершинах левого его поддерева значение в вершине $< x$, а в правом поддереве — $> x$.

Реализация операций:

1) $find(x)$. Идём с корня в ту сторону, в кот. имеет смысл идти, пока не найдем x
2) $insert(x)$. Идём вдоль того же пути. Если пытаемся пойти в пустоту, то вставляем туда же наш x.
3) $erase(x)$. Идём по дереву. Находим x. Если имеется только одно поддерево, то очевидно. Иначе, делаем swap() с минимум правого поддерева (или максимум левого поддерева), и удаляем вершину, с кот. свапнули.

Всё работает за $O(h)$, где h - высота дерева. Хотим: $h = O(\log n)$. Начнём с АВЛ.

Определение 7.2. AVL-дерево — бинарное дерево поиска, т. ч. $\forall v\colon\ |h(L_v) - h(R_v)| \leq 1$, где L_v, R_v - левое и правое поддерево v.

Утверждение 7.1. *AVL-дерево на n вершинах имеет глубину $O(\log n)$*

Доказательство. Пусть $S(h)$ - мин. число вершин в AVL-дереве глубины h. Тогда:

$$S(1) = 1$$
$$S(2) = 2$$
$$S(3) = 4$$
$$S(4) = 7$$

Заметим, что это:

$$S(h) = 1 + S(h-1) + S(h-2)$$

Покажем, что:

$$S(h) = F_{h+2} - 1$$

База, очев. Переход:

$$S(h) = 1 + S(h-1) + S(h-2) = 1 + F_{h+1} - 1 + F_h - 1 = F_{h+2} - 1$$

$$\Rightarrow S(h) = \Theta(\phi^h), \phi = \frac{1+\sqrt{5}}{2}$$

□

Значит, если в AVL-дереве n вершин и глубина h, то:

$$n \geq S(h) = \Theta(\phi^h) \Rightarrow h = O(\log_\phi(n)) = O(\log n)$$

Как же сохранять такую структуру? Введём для v:

$$\triangle(v) = h(L) - h(R)$$

Определим малый поворот (и большой поворот) ...

Определение 7.3. Splay-дерево - бинарное дерево поиска, т. ч. при каждом обращении к эл-ту, он поднимается в корень дерева.

Как именно эл-т поднимается в корень?

1) $zigzag(x)$.

2) $zigzig(x)$

3) $zig(x)$

7.0.1 Метод потенциалов

$\Phi, \Phi \geq 0$:

$$\Phi_{start} = 0$$

$$\Phi_i \text{ - числовое состояние системы на } i\text{-ом шаге}$$

$$a_i = t_i + \Phi_i - \Phi_{i-1} \text{ — уч. стоимость}$$

$$t_i \text{ — реальное время работы}$$

7.0.2 Возвращаемся к *splay*-дерево

Обсудим основные операции:

1) $splay(x)$ - комбинация преобразований $zig, zigzag, zigzig$, такая, что в конце, x - корень.

$$S(v) --- \text{ кол-во вершин в поддереве } v$$

$$r(v) = \log_2 S(v)$$

$$\Phi = \sum_{v \in V} r(v)$$

Утверждение 7.2.

$$a(splay(x)) \leq 1 + 3(r'(x) - r(x))$$

Доказательство.

$$a(zig(x)) = 1+r'(x)+r'(p)-r(x)-r(p) \leq 1+r'(x)-r(x) \leq 1+3(r'(x)-r(x))$$

□

где $r(x)$ и $r'(x)$ - старый и новый ранги x.

8 Лекция 11 (Хэш таблицы)

Хотим сделать хэш-таблицу.

Определение 8.1. Ключи $x \neq y$ образуют коллизию отн-но h, если $h(x) = h(y)$

Замечание. *Пусть h — случайная, а $x \neq y$ — ключи. Тогда:*

$$P_h(h(x) = h(y)) = \frac{1}{k}$$

Доказательство. $P_h(h(x) = h(y)) = \sum_{i=1}^{k} \underbrace{P(h(x) = i, h(y) = i)}_{\frac{1}{k^2}} = \frac{1}{k}$ □

Версии:

- Алгоритмы (хэш-таблица цепочками)

  $$h\colon U \to \{\,0, 1, \ldots, m-1\,\}$$

  Заводим m односвязных списков $l_0, \ldots, l_{m-1}$

  - insert x: вычисл. $i = h(x)$ добавл. x в список l_i (добавляем в начало)
  - erase x: вычисл. $i = h(x)$, удалить x из $h(i)$
  - find x: вычисл. $i = h(x)$, проверяем, есть ли x в l_i

Замечание. *Если всего добавлено n ключей, то вообщем размеры всех списков $\approx \frac{n}{m}$. Более того, если x — левый ключ, то мат. ожид. дляны списка $l_{h(x)}$ есть $\frac{n}{m}$*

$$E\left|l_{h(x)}\right| = \sum_{i=1}^{n} P_h(h(y_i) = h(x)) = \frac{n}{m}$$

Определение 8.2. Пусть $H = \{\, h_s\colon U \to \{\,0, 1, \ldots, m-1\,\}\,\}$

Такое сем-во ф-ций наз-ся универ. сем-вом хэш. ф-ций, если $\forall x \neq y \in U$

$$P_s(h_s(x) = h_s(y)) \leq \frac{1}{m}$$

Мораль: вместо идеальной хэш ф-ции можно брать $h_s \in H$

Утверждение 8.1. *Пусть* $U = \mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, *p — простое, тогда при любом* $m < p$ *след. сем-во будет универс.* $H = \{h_{a,b}(x) = ((ax+b)\%p)\%m\}$, $a, b \in \mathbb{Z}_p$

Доказательство. Пусть $x \neq y \in \mathbb{Z}_p$. Хотим оценить $P_{a,b}(h_{a,b}(x) = h_{a,b}(y))$

$$\underbrace{(ax+b)\%p}_{u}\%m \text{ и } \underbrace{(ay+b)\%p}_{v}\%m$$

1) $u = v \Rightarrow (ax+b)\%p = (ay+b)\%p \Rightarrow a(x-y)\%p = 0$

$$\Rightarrow x - y\%p = 0$$

$P(u = v) = \frac{1}{p}$

2) $u \neq v$

$$\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} \text{ в } \mathbb{Z}_p$$

Невыр. $\Rightarrow \exists$ реш. (a, b)

$$P_{a,b}(h(x) = h(y)) \leq \frac{1}{p} + P_{u \neq v}(u\%m = v\%m)$$

Т. к. все пары $(u \neq v)$ равновероятно получаются:

$$(x, y) \to ((ax+b)\%p, (ay+b)\%p)$$

Кол-во коллизий есть v есть $\approx \frac{1}{p}$

□

Новый алгоритм:
Заведем пустую хэш-таблицу с m списками, пусть n кол-во добавленных эл-ов.
$\alpha = \frac{n}{m}$ — Load factor
insert, erase, find — делаются как раньше.
rehash — когда α превышает некий порог (обычно 0.75), то увеличиваем m вдвое, выбираем новое h, перекладываем всё в новую hash table.

Теорема 8.1. *В среднем такая структура работает за* $O^*(1)$ *в среднем.*

8.1 Совершенное хэширование

Теорема 8.2 ((б/д) Парадокс дней рождения). *Пусть $x_1, \dots, x_n$ — случайные числа в $\mathbb{Z}_m$:*

1) *Если $m \geq n^2$, то $P($есть коллизии$) \leq \frac{1}{2}$*

2) *Если $m \leq \frac{1}{5}n^2$, то $P($есть коллизии$) > \frac{1}{2}$*

3) *Более того, если H - сем-во универс. хэш ф-ций, то пункт 1) берём даже, если вместо $x_1, \dots, x_n$ брать $h(x_1), \dots, h(x_n)$*

В структуре лежат только $x_1, \dots, x_n$: insert x $\in \{ x_1, \dots, x_n \}$
erase x $\in \{ x_1, \dots, x_n \}$
find x (x — любой)