

Алгоритмы и структуры данных

Сергей Григорян

6 ноября 2024 г.

Содержание

1	Лекция 2	3
1.1	Сортировки	3
1.1.1	Merge Sort (Сортировка слиянием)	4
1.1.2	Quick Sort (Быстрая сортировка; Сортировка Хоара)	5
1.1.3	Quick Select	6
2	Лекция 3	6
2.1	Дерандомизация	6
2.1.1	Derandomized Quick Sort	8
2.2	Сортировка чисел	8
2.2.1	Least significant digit sort	9
3	Лекция 4	9
3.1	Кучи	9
3.1.1	Бинарная (двоичная) куча	10
3.1.2	HeapSort	11
3.1.3	Удаление из кучи	12
4	Лекция 5	13
4.1	Биномиальная куча	13
4.2	Амортизационный анализ	15
4.2.1	Динамический массив (std::vector)	15
5	Лекция 6	16
5.1	Связные списки	17
5.2	Куча Фибоначчи	17
5.2.1	Consolidate (Операция причёсывания кучи)	18
5.2.2	Анализ времени работы	18
6	Лекция 7	19
6.1	Стеки и очереди	19
7	Лекция 10	20
7.0.1	Метод потенциалов	22
7.0.2	Возвращаемся к <i>splay</i> -дерево	22

1 Лекция 2

1.1 Сортировки

Задача 1.1. a_1, a_2, \dots, a_n - дано Найти $b = \text{sort}(a)$

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

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

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

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

$$(x_1, y_1) \leq (x_2, y_2) \iff \begin{cases} x_1 < x_2 \\ x_1 = x_2, y_1 < y_2 \end{cases}$$

□

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, \dots, a_n - все эл-ты попарно различны.

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

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

Пусть алгоритм всегда совершает $\leq q$ запросов $(a_i ? a_j)$. Тогда есть не более $2^0 + 2^1 + \dots + 2^q = 2^{q+1} - 1 \leq 2^{q+1}$ 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)

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

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

2)

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

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

Аналогично, при $n = 2k + 1$

□

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:

```
1 merge(a[0..n - 1], b[0..m - 1], to[0..n + m - 1]):
2     i = 0, j = 0
3     while (i < n || j < m):
4         if (j == m || (i < n && a[i] <= b[j])):
5             to[i + j] = a[i]
6             ++i
7         else
8             to[i + j] = b[j]
9             ++j
```

Листинг 1: Merge

```
1 MergeSort(a[0..n - 1]):
2     if (n <= 1) return
3     k = n / 2
4     l = a[0..k]
5     r = a[k + 1..n - 1]
6     MergeSort(l)
7     MergeSort(r)
8     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:

```
1 // ans - global variable
2 merge(a[0..n - 1], b[0..m - 1], to[0..n + m - 1]):
3     i = 0, j = 0
4     while (i < n || j < m):
5         if (j == m || (i < n && a[i] <= b[j])):
6             to[i + j] = a[i]
7             ++i
8             ans += j
9         else
10            to[i + j] = b[j]
11            ++j
12            ans += i
```

Листинг 3: Merge for inversions

1.1.2 Quick Sort

(Быстрая сортировка; Сортировка Хоара)

```
1 Partition(a[0..n - 1], x):
2     B => < x
3     C => = x
4     D => > x
5     return (|B|, |C|, |D|)
6
7 QuickSort(a[0..n - 1]):
8     if (n <= 1) return;
9     x = a[random(0, n - 1)]
```

```

10  l, m, r = Partition(a, x)
11  QuickSort(a[1..l - 1])
12  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, \dots, 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.

```

1  QuickSelect(a[1..n], k):
2      if (n == 1) return a[1];
3      l, m, r = Partition(a, a[random(1, n)])
4      if (k <= l) return QuickSelect(a[1..l], k)
5      if (k <= l + m) return x
6      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, \dots, a_n массив. Его **медианой** наз-вом $a_{\lceil \frac{n}{2} \rceil}$ (a - отсортирован)

Алгоритм 2.1 (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.

```

1 DQS(A, k):
2   x = DQS(b_1, b_2, ..., b_{(n/5)}, n/10)
3   Partition(A, x)
4   if (k <= l) => return DQS(B, k)
5   if (k <= l + m) => return x
6   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.

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

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

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

МММ:

База: $n \leq 5$ очев.

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

□

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, \dots, a_n - массив чисел $a_i \in \{0, 1, \dots, k\}$. Отсортировать a .

Решение.

```
1 cnt[k + 1] = {0, ..., 0}
2 for i=1..n
3     ++cnt[a[i]]
4 for x=0..k:
5     for i=1..cnt[x]
6         print(x)
7
```

Листинг 7: Counting sort

Асимптотика: $O(n + k)$

Определение 2.2. Стабильная сортировка:

$a_1, a_2, \dots, a_n \Rightarrow a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(n)}$, 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:

```
1 cnt[k + 1] = {0, ..., 0};
2 for i=1..n
3     ++cnt[a[i]]
4 for x = 1..k:
5     cnt[x] += cnt[x - 1]
6 b = {-1, ..., -1}
7 for i=1..k:
8     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:

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

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

Отсортировать!

Решение. 1) Отсортируем массив пар по b

2) Результат стабильно отсортируем по a

Почему это корректно?

Пусть $(a_i, b_i) < (a_j, b_j) \iff$

$$\begin{cases} a_i < a_j \\ a_i = a_j, b_i < b_j \end{cases} \text{ (стабильность!)}$$

2.2.1 Least significant digit sort

Задача 2.3. 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 :

$$x \bmod 2^b = x \wedge ((1 \ll b) - 1)$$

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
- $decreaseKey(ptr, \Delta), \Delta > 0$ - уменьшить число по адресу ptr на Δ

Применения: алгоритмы Дейкстры, Прима

3.1.1 Бинарная (двоичная) куча

Храним 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 ;

```

1 siftUp(u):
2     if (v == 1) return
3     p = (v / 2)
4     if (a[p] > a[v]) {
5         swap(a[p], a[v])
6         siftUp(p)
7     }
8
9 siftDown(v)
10    if (2 * v > n) return
11    u = 2v
12    if (2 * v + 1 <= n && a[2 * v + 1] < a[u]): u = 2
13        * v + 1
14    if (a[u] < a[v]) {
15        swap(a[u], a[v])
16        siftDown(u)
17    }

```

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: $a_v \leftarrow x$

- а) $x \geq a_p$ - родитель; нер-во сохраняется, $siftUp$ ничего не делает, куча остаётся корректной
- б) $x < a_p$. Тогда сделаем $swar(a_p, a_v)$, 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) $insert a_1, \dots, a_n$
- 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 $O(n)$

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

Листинг 10: Heapify

Корректность? Индукцией по i : после вызова *siftDown*(i), поддереву 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

$$\begin{aligned} Sum &= \frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 2 + \dots = \frac{n}{2} + \frac{n}{4} + \dots + \left(\frac{n}{4} \cdot 1 + \frac{n}{8} \cdot 2 + \frac{n}{16} \cdot 3 + \dots \right) = \\ &= n + \frac{n}{2} + \left(\frac{n}{8} + \frac{n}{16} \cdot 2 + \frac{n}{32} \cdot 3 + \dots \right) \leq 2 \cdot n \end{aligned}$$

3.1.3 Удаление из кучи

erase:

а) По указателю на x

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

б) 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:

- $getMin()$
- $extractMin()$
- $insert(x)$
- $decreaseKey(ptr, \Delta)$
- $merge(heap1, heap2)$ - 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, Δ):**
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 $a_1(n), a_2(n), \dots, a_m(n)$ наз-ся учётными (амортизационными) асимптотиками ответов на запросы, если $\forall n \forall$ п-ть из n запросов с типами i_1, i_2, \dots, 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*: аморти. $O(\log n)$

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, \dots, 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 Δ : 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 деревьев 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
 $S(0) = 1, S(1) = 2, S(k) = ?$

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

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.

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

Доказательство.

$$k = 2: 2 + F_2 = 3 = F_3$$

$$k = k: 2 + F_2 + \dots + F_{k+1} = F_{k+3}$$

☐

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

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

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 связанного списка.

ПСССССССССССП

ОБРАТНАЯ ПОЛЬСКАЯ ЗАПИИИИИИИИИИИСЬ

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 $swar()$ с минимум правого поддерева (или максимум левого поддерева), и удаляем вершину, с кот. свагнули.

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. $\forall v: |h(L_v) - h(R_v)| \leq 1$, где L_v, R_v - левое и правое поддереву 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 :

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

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

$\Phi, \Phi \geq 0$:

$$\Phi_{start} = 0$$

Φ_i - числовое состояние системы на i -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*-дереву

Обсудим основные операции:

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

$S(v)$ — — — 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 .