Алгоритмы и структуры данных

Сергей Григорян

2 октября 2024 г.

Содержание

1 Лекция 4

1.1 Кучи

Определение 1.1. 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

1.1.1 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])
```

```
15      siftDown(u)
16    }
```

Листинг 1: siftUp and siftDown

Остальные методы в heap.cpp

Асимптотика всего: $O(\log n)$

Корректность

Лемма 1.1. *Пусть $a_1, \ldots, a_n$ - 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)

□

1.1.2 HeapSort

Алгоритм:

$$a_1, a_2, \ldots, a_n$$

1) $insert a_1, \ldots, a_n$

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 2: 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.

  ⋮

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

1.1.3 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 $= O(n \log n)$

2 Лекция 5

2.1 Биномиальная куча

Хотим следующие операции:

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

Определение 2.1. 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 2.2. 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

2.2 Амортизационный анализ

Определение 2.3. 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: $O(k \log k + n - k)$

2.2.1 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.