Алгоритмы и структуры данных

Сергей Григорян

7 октября 2024 г.

Содержание

1 Лекция 5

1.1 Биномиальная куча

Хотим следующие операции:

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

Определение 1.1. 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 1.2. 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

1.2 Амортизационный анализ

Определение 1.3. 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: $O(k \log k + n - k)$

1.2.1 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.