Алгоритмы и структуры данных

Сергей Григорян

9 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.

2 Лекция 6

Утверждение 2.1 (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

2.1 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.*

2.2 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.

2.2.1 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:

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

2.2.2 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.

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