Лекция по алгоритмам №6
Куча Фибоначчи и Динамика

21 октября 2014

---

1 Куча Фибоначчи

Определения:

Фибоначчиево дерево — биномиальное дерево, для каждой вершины которой удалили не более одного сына.

Ранг фибоначчиева дерева — ранг биномиального дерева, из которого его получили.

Ранг вершины фибоначчиева дерева — степень вершины в последний момент, когда она была корнем в своем фибоначчиевом дереве. Заметим, что так как удаляется не более одного сына, то этот ранг равен или степени вершины, или степени вершины, увеличенной на 1.

Куча Фибоначчи — это двусвязный список фибоначчиевых деревьев.

1.1 Лемма $Size_{rank} \geq F_{rank}$

Где $Size_{rank}$ — размер фибоначчиева дерева ранга $rank$, F_n — n-ое число Фибоначчи.

Докажем по индукции:

База: при $n = 0$, $Size_0 = 1 > F_0 = 0$. При $n = 1$, $Size_1 \geq 1 = F_1$.

Переход: $Size_n \geq 1 + \sum_{i=0}^{n-1} Size_i - Size_{n-1} = 1 + \sum_{i=0}^{n-2} Size_i$. Тогда, по индукционному предположению, $Size_n \geq 1 + \sum_{i=0}^{n-2} F_i$. Факт из жизни чисел Фибоначчи: $F_n = 1 + \sum_{i=0}^{n-2} F_i$. (Его просто доказать по индукции) Значит, $Size_n \geq F_n$, чтд.

Следствие 1: максимальный ранг дерева в куче Фибоначчи размера n это $O(\log n)$. Это так, потому что F_n растет экспоненциально от n. $F_n \approx \phi^n > 1.6^n$.

Следствие 2: если в куче Фибоначчи размера n нет деревьев одинакового ранга, то их количество это $O(\log n)$.

1.2 Потенциал

$\Phi_i = Roots_i + 2 \cdot marked\_count_i$, где Φ_i — потенциал в i-ый момент времени, $Roots_i$ — количество деревьев в списке, а $marked\_count_i$ — количество вершин, у которых удалили сына.

1.3 Add, Merge, GetMin, DelMin

1. Add реализован через Merge: создаем кучу Фибоначчи из одной вершины и Merge-им две кучи. Время работы $O(1)$.

2. Merge реализован через Merge списков. Просто связываем два списка и пересчитываем указатель на новый минимум, как указатель на меньшую вершину из двух бывших минимумов. Реальное время работы $O(1)$. Амортизированное время работы тоже $O(1)$, потому что потенциал до слияния мы считали как сумму потенциалов для сливаемых куч, которай после слияния стал потенциалом текущей кучи.

3. GetMin, как и в Pairing Heap, получаем, храня указатель на дерево, ключ корня которого минимален. Время работы $O(1)$.

4. DelMin: найдем минимальную вершину; удалим её, склеив список деревьев кучи и список детей-подеревьев удаленной вершины; вызовем процедуру объединения деревьев, которая применит Merge для некоторых деревьев так, чтобы осталось $O(\log n)$ деревьев в списке.

Процедура объединения: будем добавлять деверья в ответ по одному за раз. Зарание создадим массив $res[i]$ — указатель на дерево в результирующей кучи ранга i. При добавлении очередного дерева ранга i, если дерево такого же ранга уже есть в ответе, то есть $res[i] \neq \text{NULL}$, то Merge-им дерево из $res[i]$ и добавляемое дерево, и переходим к его добавлению. Если же $res[i] = \text{NULL}$, то просто кладем указатель на корень этого дерева в res. Процесс напоминает переносы в сложении числа со степенью двойки в двоичном виде. В конце, превращаем массив res в двусвязный список.

Реальное время работы: $t_i \leq Roots + Deg_v - 1$. $Deg_v = O(\log n)$, $\Delta\Phi = -Roots + O(\log n)$. Значит, амортизированное время работы это $Roots + O(\log n) - Roots + O(\log n) = O(\log n)$.

1.4 DecreaseKey

Будем хранить $marked[v]$ — был ли удален сын вершины v.

Пусть у вершины v уменьшили значение ключа. Тогда, если у отца v теперь стал больший ключ, чем у v, то отрубим себя от отца и, соответственно, припишем себя в список деревьев, присваевая $marked[v] \leftarrow 0$. Если у отца ещё не было отрублиных сыновей, то $marked[parent[v]] \leftarrow 1$, иначе отрубаем и его. И так далее, пока мы не дойдем до $marked = 0$ или до корня.

Амортизированное время работы: $t_i^* = t_i + \Delta\Phi$. Заметим, что за каждое действие в t_i мы увеличивали $Roots$ на 1 и уменьшали $marked\_count$ на 1. Но, $marked\_count$ умножается на 2 в Φ, так что суммарное $\Delta\Phi = t_i - 2 \cdot t_i = -t_i$, значит $t_i^* = O(1)$.

2 Динамика

Coming soon...?