Первый курс, осенний семестр 2018/19
Практика по алгоритмам #7

Кучи

20 октября

Собрано 28 октября 2018 г. в 11:39

Содержание

Кучи

1. **Добавление в биномиальную кучу**
   a) Докажите, что добавление в биномиальную кучу через `Merge` с одной вершиной работает за амортизированное $\mathcal{O}(1)$.
   b) Теперь сделаем так, чтобы добавление происходило за $\mathcal{O}(1)$ в худшем случае.
   Рассмотрим избыточный двоичный счетчик, в каждом разряде которого можно хранить 0, 1 или 2. Например, число 12 можно хранить в нем как 1100, а можно как 1020. Мы хотим научиться прибавлять 1 к такому счетчику.
   Заметим, что 2 всегда можно раскрыть как 10. Но если у нас много двоек подряд, в худшем случае все равно прибавление может работать за $\Omega(\log n)$.
   Придумайте инвариант и способ его поддерживания, чтобы прибавление 1 работало за $\mathcal{O}(1)$ даже в худшем случае.

2. **Оптимизации куч**
   a) Есть куча, которая умеет `Add`, `Merge`, `ExtractMin` за $\mathcal{O}(\log n)$, `Build` за $\mathcal{O}(n)$. На ее основе построить кучу, которая умеет все то же, но `Add` за $\mathcal{O}(1)$.
   b) Есть куча, которая умеет `Add` за $\mathcal{O}(1)$, `Merge`, `ExtractMin` за $\mathcal{O}(\log n)$, `Build` за $\mathcal{O}(n)$. На ее основе построить кучу, которая умеет все то же, но `Merge` за $\mathcal{O}(1)$.

3. **Куча Фибоначчи**
   a) Что будет, если в куче Фибоначчи делать `DecreaseKey` просто через `SiftUp`?
   b) Получить $\mathcal{O}(n)$ операциями над кучей бамбук из n вершин.
   c) Оценить сложность худшего случая операций `Add`, `Merge`, `ExtractMin`, `DecreaseKey` в куче Фибоначчи.
   d) Что будет, если разрешить отрезать сколько угодно детей?
   e) Что будет, если разрешить отрезать двух детей?
   f) Придумайте, как сделать так, чтобы `ExtractMin` работал за амортизированное $\mathcal{O}(\log C)$, где C – число различных значений в куче.
   g) Разрешим удалять только самого большого ребенка. Придумайте, как теперь делать `DecreaseKey` за амортизированное $\mathcal{O}(1)$. Докажите, что теперь в худшем случае он работает за $\mathcal{O}(\log n)$.

4. **(*) Счетчик Кнута**
   Избыточный двоичный счетчик – счетчик, где в каждом разряде может стоять $0, 1, 2$. Значение числа – сумма $d_i 2^i$. Нам нужны операции:

   - `get(i)` – посмотреть `i`-й разряд счетчика.
   - `inc(i)` – прибавть к числу в счетчике 2^i.

   Придумайте структуру данных, которая позволяет выполнять эти операции за $\mathcal{O}(1)$ в худшем случае.

Разбор задач практики

1. **Добавление в биномиальную кучу**
   a) Добавление в биномиальную кучу – это прибавление 1 к двоичному счетчику. А `+1` в работает за амортизированную $\mathcal{O}(1)$.
   b) Алгоритм: при добавлении единички берем самую младшую двойку и раскрываем ее в 10, затем прибавляем единичку в младший разряд. Можем, так как в младшем уже не 2.
   Инвариант: между любыми двумя двойками есть ноль.
   Раскрытие младшей двойки создало ноль. Этот ноль разделяет следующую за ней двойку и новую двойкой в 0 разряде, если та появилась.
   Варианты (младшие разряды справа):
   $\ldots 2 \ldots 02 \ldots \to \ldots 2 \ldots 10 \ldots$,
   $\ldots 2 \ldots 0 \ldots 12 \ldots \to \ldots 2 \ldots 0 \ldots 20 \ldots$.
   Как получить младшую двойку? Хранить список позиций с двойками.

2. **Оптимизации куч**
   a) `Add` за $\mathcal{O}(1)$. Добавляет новые элементы в отдельный список `freshItems`.
   Если пришел `ExtractMin` или `Merge`, делаем `Merge(q, Build(freshItems))`.
   Потенциал $\varphi = |$`freshItems`$|$.
   b) `Merge` за $\mathcal{O}(1)$. У нас была структура `Q`, строим структуру `B`.
   ```
   struct<T> B {
     T min;
     Q<B<T>> q;
   };
   ```
   Если держать кучу `Q` структур `B`, то `Merge` двух структур `B` – это `Add` в структуру `Q` за $\mathcal{O}(1)$.
   `ExtractMin`: `bmin = q.ExtractMin(), min = bmin.min, q = Merge(q, bmin.q);`
   Вышло $\mathcal{O}(\log n)$.

3. **Куча Фибоначчи**
   a) Если делать `DecreaseKey` просто через `SiftUp`, то есть не резать детей, то выйдет просто биномиальная куча. `SiftUp` за высоту биномиального дерева $= \mathcal{O}(\log n)$.
   b) Можем получить бамбук из двух вершин. Пусть есть k вершин.
   Ранг бамбука равен 1, подвесим его к другому дереву ранга 1 и отрежем у того другого ребенка. Стало $k + 1$.
   c) `Add` и `Merge` всегда $\mathcal{O}(1)$.
   `ExtractMin` в худшем случае за $\Omega(n)$: добавили n элементов, потом `ExtractMin`.
   `DecreaseKey` в худшем случае за $\Omega(n)$. Сделали бамбук с отростками на каждом уровне. Вызвали `DecreaseKey` от каждого отростка, теперь все вершины бамбука помечены. Теперь `DecreaseKey` от его нижней вершины.
   d) Если отрезать сколько угодно детей, то `DecreaseKey` работает за честное $\mathcal{O}(1)$, ведь нам не надо рекурсивно резать предков.
   Но теперь дерево ранга k может быть размера $(k + 1)$: корень и k детей.
   Тогда max ранг может быть $\Omega(n)$ вместо $\mathcal{O}(\log n) \Rightarrow$ после `ExctractMin` останется не $\mathcal{O}(\log n)$ корней, амортизированная оценка сломалась.
   Но! Ранг равен числу детей $\Rightarrow$ size$(r) \geqslant r + 1 \Rightarrow$ может быть не более $\mathcal{O}(\sqrt{n})$ разных рангов.

Итого амортизированное время `ExctractMin` равно $\mathcal{O}(\sqrt{n})$.

Можно построить пример, где будет достигаться $\Omega(\sqrt{n})$.
Есть одно дерево ранга k. Обрежем у него всех правнуков, ранги детей не изменились.
Режем всех детей. Теперь у нас деревья рангов $0, 1, \ldots, k-1$, в них вершин $1, 2, \ldots, k$. Суммарно $\Theta(k^2)$.
Если все время делать `ExtractMin` из самого маленького дерева, поиск нового будет все время $\Omega(k)$.

e) Если резать двух детей, то будут те же оценки, что в обиычное куче Фибоначчи.
Для этого надо показать, что размер дерева экспоненциальный от ранга.
Покажем $\text{siẓe}(r) \geqslant C^{r-2}$. База $r = 0, 1$.
Переход: $\text{siẓe}(r) \geqslant 1 + C^{0-2} + C^{1-2} + \ldots + C^{(r-3)-2} = 1 + \frac{C^{-2}(C^{r-2}-1)}{C-1} = 1 + \frac{C^{r-4}-C^{-2}}{C-1}$.
Например, для $C = \sqrt{2}$ верно $1 + \frac{C^{r-4}-C^{-2}}{C-1} \geqslant C^{r-2}$ при $r > 1$.

f) `ExtractMin` за $\mathcal{O}(\log C)$. Храним в куче пары $\langle x, \text{coụnt}(x)\rangle$.
Также понадобится хеш-таблица, которая по значению дает узел с этим значением.
С `Add` всё просто. `Merge` не получится, и ладно.
`DecreaseKey`. Уменьшили $x \to y$. Уменьшим счетчик x, увеличим счетчик y. Если y еще не было, добавим.
Проблема: если счетчик x занулился, то у нас есть лишний узел в куче, портит асимптотику. Чтобы ее решить, нужен хороший потенциал...

g) Если можно удалять только самого большого ребенка, то у вершины ранга k должны быть дети рангов $0, 1, \ldots, k-2, k-1$. Если вершина помечена, то старшего нет. Назовем такую вершину *тонкой*.
Делая `DecreaseKey`, таки отрезаем ребенка. Его родитель v, старший брат u. Сейчас ранг u больше, чем надо для того, чтобы ранги детей v шли подряд.
Если u не тонкий, делаем его старшего ребенка его младшим братом, теперь всё хорошо. u стал тонким.
Если u тонкий, просто уменьшаем его ранг на один, исправляем его старшего брата.
Теперь может оказаться, что у вершины v ранга k детей $k-2$. Вырежем и ее, решаем проблемы ее родителя.
При исправлениях движемся по дереву либо влево, либо вверх. При этом растет *исходный* ранг обрабатываемой вершины, так что в худшем случае $\mathcal{O}(\log n)$.
Это называется **тонкая куча**.

4. **(*) Счетчик Кнута**
Хочется поддержать тот же инвариант: между любыми двумя двойками есть ноль.
Будем так же раскрывать ближайшую к нам слева двойку. И если у нас вышла двойка, тоже раскроем ее.
Нужно поддерживать ближайшую слева двойку к позиции. Трудно.