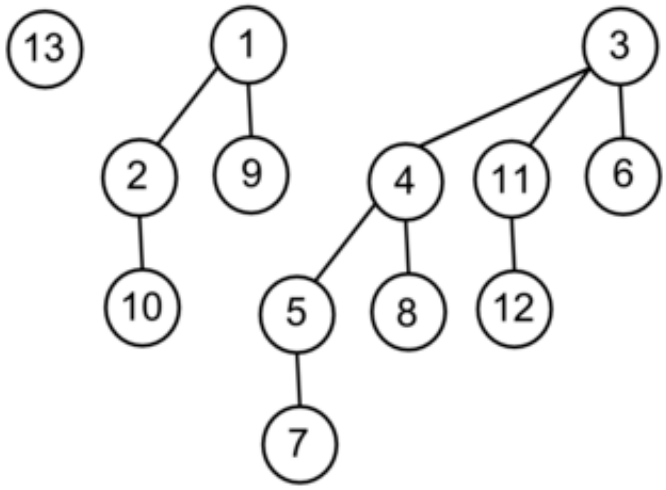
Биномиальная куча

Содержание



Пример биномиальных деревьев B_0, B_2, B_3

Биномиальное дерево

Определение:

Биномиальное дерево B_k (англ. *binomial tree*) — дерево, определяемое для каждого $k = 0, 1, 2, \ldots$ следующим образом: B_0 — дерево, состоящее из одного узла; B_k состоит из двух биномиальных деревьев B_{k-1}, связанных вместе таким образом, что корень одного из них является дочерним узлом корня второго дерева.

Свойства биномиальных деревьев

Утверждение:

Биномиальное дерево B_k с n вершинами имеет 2^k узлов.

▷

Докажем по индукции:

База $k = 1$ — верно. Пусть для некоторого k условие верно, то докажем, что для $k + 1$ это также верно:

Так как в дереве порядка $k + 1$ вдвое больше узлов, чем в дереве порядка k, то дерево порядка $k + 1$ имеет $2^k \cdot 2 = 2^{k+1}$ узлов. Переход доказан, то биномиальное дерево B_k с n вершинами имеет 2^k узлов.

◁

Утверждение:

> Биномиальное дерево B_k с n вершинами имеет высоту k.

▷

Докажем по индукции:

База $k = 1$ — верно. Пусть для некоторого k условие верно, то докажем, что для $k + 1$ это также верно:

Так как в дереве порядка $k + 1$ высота больше на 1 (так как мы подвешиваем к текущему дереву дерево того же порядка), чем в дереве порядка k, то дерево порядка $k + 1$ имеет высоту $k + 1$. Переход доказан, то биномиальное дерево B_k с n вершинами имеет высоту k.

◁

Утверждение:

> Биномиальное дерево B_k с n вершинами имеет ровно $\binom{k}{i}$ узлов на высоте i.

▷

Докажем по индукции:

База $k = 1$ — верно. Пусть для некоторого k условие верно, то докажем, что для $k + 1$ это также верно:

Рассмотрим i уровень дерева B_{k+1}. Дерево B_{k+1} было получено подвешиванием одного дерева порядка k к другому. Тогда на i уровне дерева B_{k+1} всего узлов $\binom{k}{i} + \binom{k}{i-1}$, так как от подвешенного дерева в дерево порядка $k + 1$ нам пришли узлы глубины $i - 1$. То для i-го уровня дерева B_{k+1} количество узлов $\binom{k}{i} + \binom{k}{i-1} = \binom{k+1}{i}$. Переход доказан, то биномиальное дерево B_k с n вершинами имеет ровно $\binom{k}{i}$ узлов на высоте i.

◁

Утверждение:

> Биномиальное дерево B_k с n вершинами имеет корень степени k; степень всех остальных

вершин меньше степени корня биномиального дерева;

▷

Так как в дереве порядка $k+1$ степень корня больше на 1, чем в дереве порядка k, а в дереве нулевого порядка степень корня 0, то дерево порядка k имеет корень степени k. И так как при таком увеличении порядка (при переходе от дерева порядка k к $k+1$) в полученном дереве лишь степень корня возрастает, то доказываемый инвариант, то есть степень корня больше степени остальных вершин, не будет нарушаться.

◁

Утверждение:

В биномиальном дереве B_k с n вершинами максимальная степень произвольного узла равна $\log n$.

▷

Докажем это утверждение для корня. Степень остальных вершин меньше по предыдущему свойству. Так как степень корня дерева порядка k равна k, а узлов в этом дереве $n = 2^k$, то прологарифмировав обе части получаем, что $k = O(\log n)$, то степень произвольного узла не более $\log n$.

◁

Биномиальная куча

Определение:

Биномиальная куча (англ. *binomial heap*) представляет собой множество биномиальных деревьев, которые удовлетворяют следующим свойствам:

- каждое биномиальное дерево в куче подчиняется свойству **неубывающей кучи**: ключ узла не меньше ключа его родительского узла (упорядоченное в соответствии со свойством неубывающей кучи дерево),
- для любого неотрицательного целого k найдется не более одного биномиального дерева, чей корень имеет степень k.

Представление биномиальных куч

Поскольку количество детей у узлов варьируется в широких пределах, ссылка на детей осуществляется через левого ребенка, а остальные дети образуют односвязный список. Каждый узел в биномиальной куче представляется набором полей:

- key — ключ (вес) элемента,
- $parent$ — указатель на родителя узла,

- $child$ — указатель на левого ребенка узла,
- $sibling$ — указатель на правого брата узла,
- $degree$ — степень узла (количество дочерних узлов данного узла).

Корни деревьев, из которых состоит куча, содержатся в так называемом **списке корней**, при проходе по которому степени соответствующих корней находятся в возрастающем порядке. Доступ к куче осуществляется ссылкой на первый корень в списке корней.

Операции над биномиальными кучами

Рассмотрим операции, которые можно производить с биномиальной кучей. Время работы указано в таблице:

Операция	Время работы
insert	$O(\log n)$
getMinimum	$O(\log n)$
extractMin	$\Theta(\log n)$
merge	$\Omega(\log n)$
decreaseKey	$\Theta(\log n)$
delete	$\Theta(\log n)$

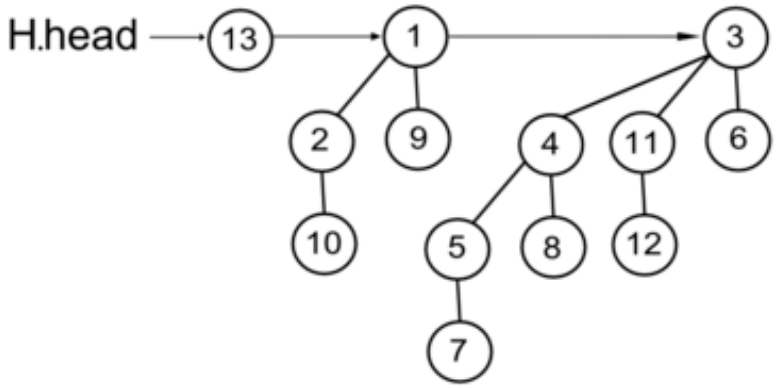
Обозначим нашу кучу за H. То пусть $H.\,head$ — указатель на корень биномиального дерева минимального порядка этой кучи. Изначально $H.\,head = null$, то есть куча не содержит элементов.

getMinimum

Для нахождения минимального элемента надо найти элемент в списке корней с минимальным значением (предполагается, что ключей, равных ∞, нет).

Так как корней в этом списке не более $\lfloor \log n \rfloor + 1$, то операция выполняется за $O(\log n)$.

При вызове этой процедуры для кучи, изображенной на картинке ниже, будет возвращен указатель на вершину с ключом 1.

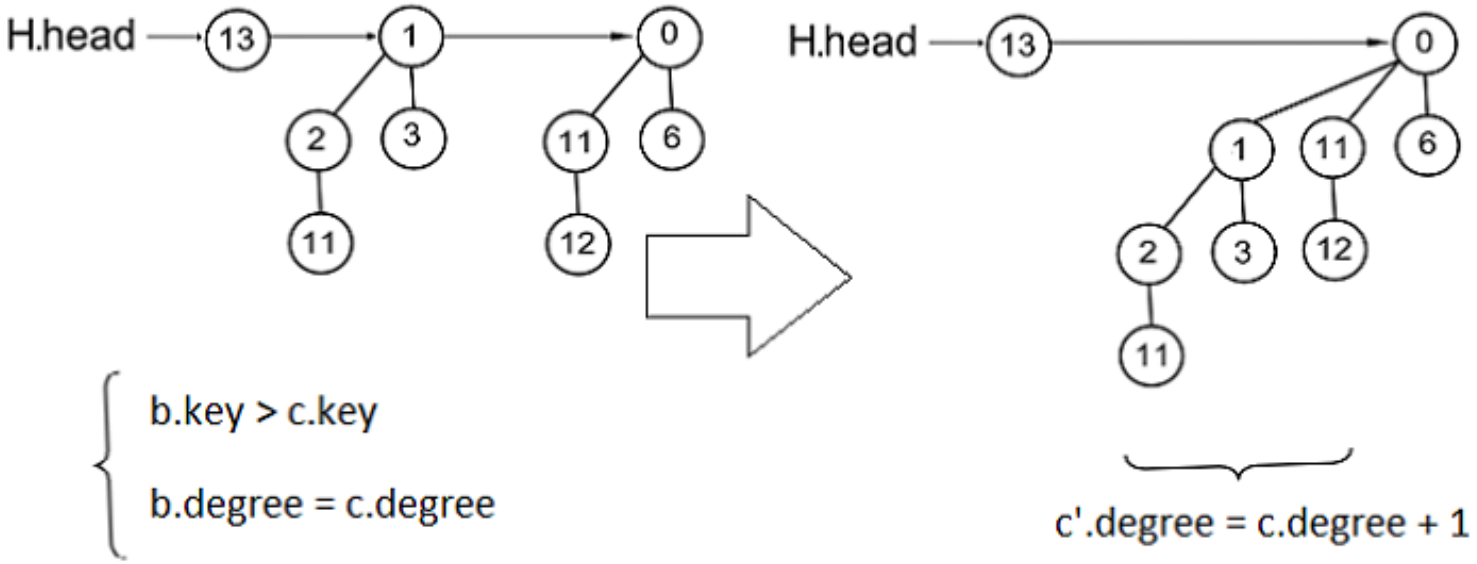


При использовании указателя на биномиальное дерево, которое содержит минимальный элемент, время для этой операции может быть сведено к $O(1)$. Указатель должен обновляться при выполнении любой операции, кроме getMinimum. Это может быть сделано за $O(\log n)$, не ухудшая время работы других операций.

merge

Эта операция, соединяющая две биномиальные кучи в одну, используется в качестве подпрограммы большинством остальных операций.

Вот в чем состоит ее суть: пусть есть две биномиальные кучи с H и H'. Размеры деревьев в кучах соответствуют двоичным числам m и n, то есть при наличии дерева соответствующего порядка в этом разряде числа стоит единица, иначе ноль. При сложении столбиком в двоичной системе происходят переносы, которые соответствуют слияниям двух биномиальных деревьев B_{k-1} в дерево B_k. Надо только посмотреть, в каком из сливаемых деревьев корень меньше, и считать его верхним (пример работы для одного случая приведен на рисунке справа; в другом случае подвешиваем наоборот).



Работа этой процедуры начинается с соединения корневых списков куч в единый список, в котором корневые вершины идут в порядке неубывания их степеней.

В получившемся списке могут встречаться пары соседних вершин одинаковой степени. Поэтому мы начинаем соединять деревья равной степени и делаем это до тех пор, пока деревьев одинаковой степени не останется. Этот процесс соответствует сложению двоичных чисел столбиком, и время его работы пропорционально числу корневых вершин, то есть операция выполняется за $\Omega(\log n)$.

```
BinomialHeap merge(H1 : BinomialHeap, H2 : BinomialHeap):
   if H1 == null
       return H2
   if H2 == null
       return H1
   H.head = null                       // H — результат слияния
   curH = H.head                       // слияние корневых списков
   curH1 = H1.head
   curH2 = H2.head
   while curH1 != null and curH2 != null
       if curH1.degree < curH2.degree
           curH.sibling = curH1
           curH = curH1
           curH1 = curH1.sibling
      else
           curH.sibling = curH2
           curH = curH2
           curH2 = curH2.sibling
   if curH1 == null
       while curH2 != null
           curH.sibling = curH2
           curH2 = curH2.sibling
   else
       while curH1 != null
           curH.sibling = curH1
           curH1 = curH1.sibling
   curH = H.head                       // объединение деревьев одной степени
   while curH.sibling != null
       if curH.degree == curH.sibling.degree
           p[curH] = curH.sibling
           tmp = curH.sibling
           curH.sibling = curH.sibling.child
           curH = tmp
           continue
       curH = curH.sibling
   return H
```

insert

Чтобы добавить новый элемент в биномиальную кучу нужно создать биномиальную кучу H' с единственным узлом, содержащим этот элемент, за время $O(1)$ и объединить ее с биномиальной кучей H за $O(\log n)$, так как в данном случае куча H' содержит лишь одно дерево.

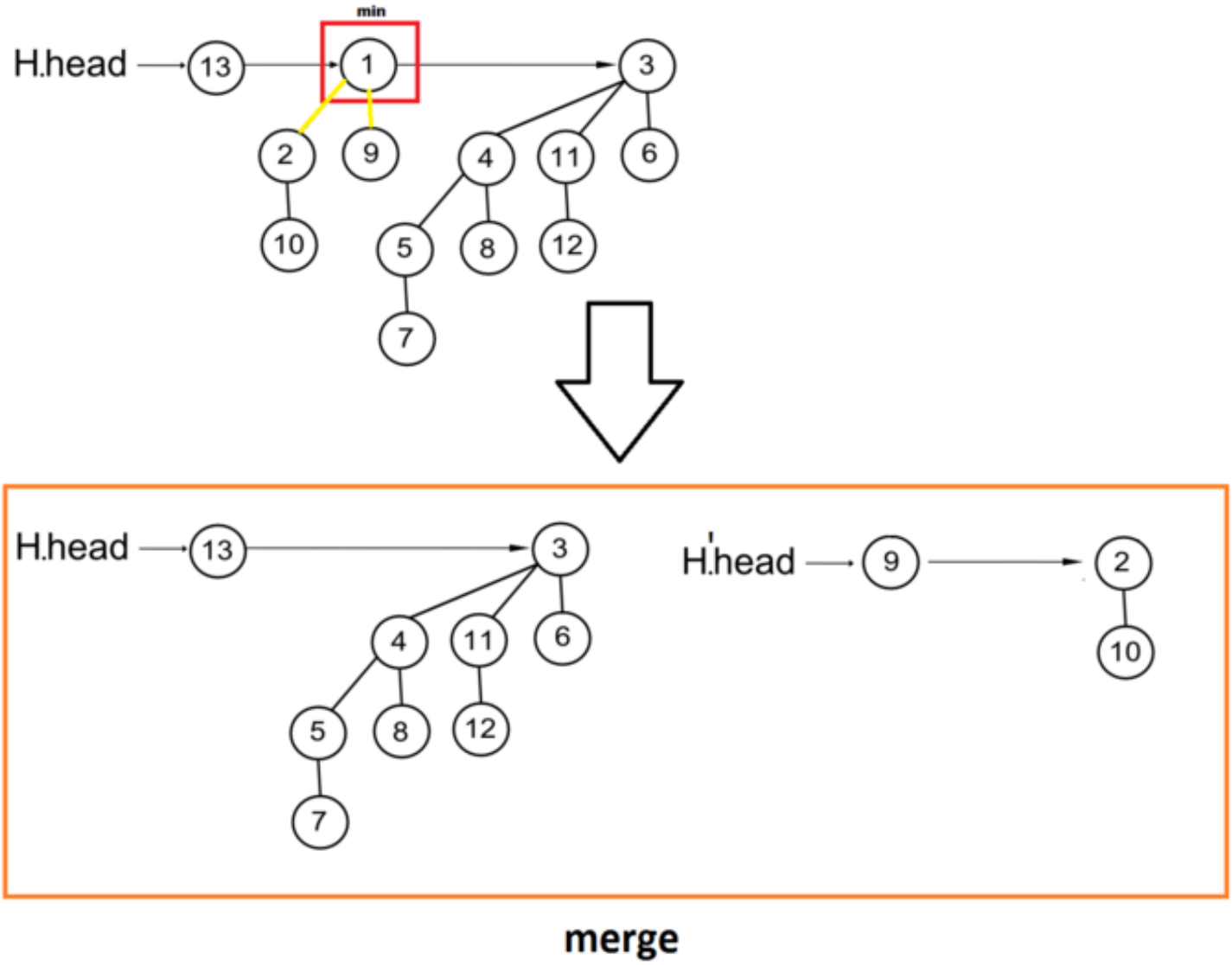
extractMin

Приведенная ниже процедура извлекает узел с минимальным ключом из биномиальной кучи и возвращает указатель на извлеченный узел.

Рассмотрим пошагово алгоритм:

- Найдем биномиальное дерево с минимальным корневым значением. Предположим, что это дерево B_k . Время работы этого шага алгоритма $\Theta(\log n)$.
- Удаляем дерево B_k из кучи H. Иными словами, удаляем его корень из списка корней кучи. Это можно сделать за время $O(1)$.
- Пусть H' — куча детей найденного корня. При этом мы для каждого из ребенка устанавливаем указатель на предка равным $null$. После этого сливаем кучу H' с H за $\Omega(\log n)$.

Процедура выполняется за время $\Theta(\log n)$, поскольку всего в списке $\Theta(\log n)$ корней биномиальных деревьев. И всего у найденного дерева k порядка (с минимальным значением ключа) ровно k детей, то сложность перебора этих детей будет тоже $\Theta(\log n)$. А процесс слияния выполняется за $\Omega(\log n)$. Таким образом, операция выполняется $\Theta(\log n)$.



merge

```
Node extractMin(H : BinomialHeap): //поиск корня x с минимальным значением ключа в списке корней H:
    min =
    x = null
    xBefore = null
    curx = H.head
```

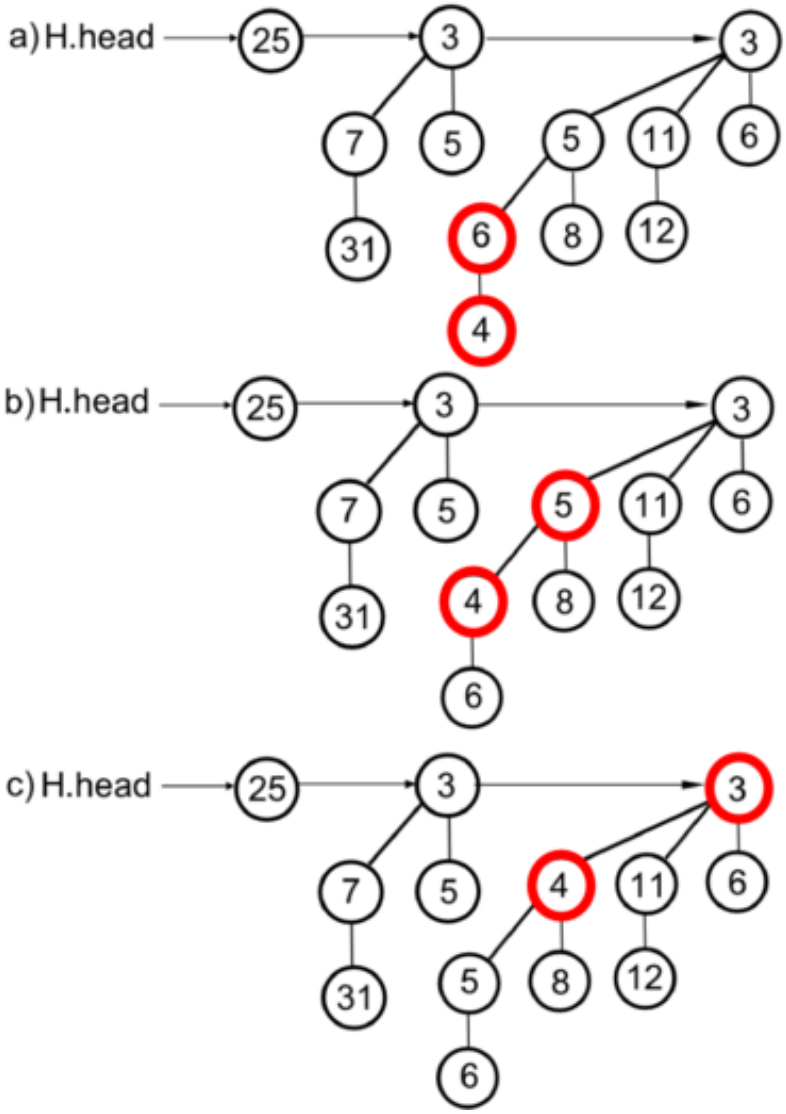
```
    curxBefore = null
    while curx != null
        if curx.key < min       // релаксируем текущий минимум
            min = curx.key
            x = curx
            xBefore = curxBefore
        curxBefore = curx
        curx = curx.sibling
   if xBefore == null           //удаление найденного корня x из списка корней деревьев кучи
        H.head = x.sibling
   else
        xBefore.sibling = x.sibling
   H' = null                    //построение кучи детей вершины x, при этом изменяем предка соответствующ
его ребенка на null:
   curx = x.child
   H'.head = x.child
   while curx != null
        p[curx] = null          // меняем указатель на родителя узла curx
        curx = curx.sibling     // переход к следующему ребенку
   H = merge(H, H')             // слияние нашего дерева с деревом H'
   return x
```

decreaseKey

Следующая процедура уменьшает ключ элемента x биномиальной кучи, присваивая ему новое значение. Вершина, ключ которой был уменьшен, «всплывает» как в обычной куче. Процедура выполняется за время $\Theta(\log n)$, поскольку глубина вершины x в худшем случае есть $\Theta(\log n)$ (свойства биномиального дерева), а при выполнении каждого шага алгоритма мы поднимаемся вверх.

```
function decreaseKey(H : BinomialHeap, x : Node, k : int):
    if k > key[x]                       // проверка на то, что текущий ключ x не меньше передаваемого ключ
а k
        return
    key[x] = k
    y = x
    z = p[y]
    while z != null and key[y] < key[z] // поднимаем  x с новым ключом k, пока это значение меньше значен
ия в родительской вершине
        swap(key[y], key[z])
        y = z
        z = p[y]
```

Пример работы процедуры проиллюстрирован на рисунке (y — уменьшаемый элемент, z — его предок).



delete

Удаление ключа сводится к операциям decreaseKey и extractMin: сначала нужно уменьшить ключ до минимально возможного значения, а затем извлечь вершину с минимальным ключом. В процессе выполнения процедуры этот узел всплывает вверх, откуда и удаляется. Процедура выполняется за время $\Theta(\log n)$, поскольку каждая из операций, которые используется в реализации, работают за $\Theta(\log n)$.

```
function delete(H : BinomialHeap, x : Node):
  decreaseKey(H, x, ) // уменьшение ключа до минимально возможного значения
  extractMin(H)           // удаление "всплывшего" элемента
```

Персистентность

Биноминальную кучу можно сделать персистентной при реализации на односвязных списках[1]. Для этого будем хранить список корней в порядке возрастания ранга, а детей будем хранить по убыванию ранга. Каждый родитель будет знать ребенка с большим рангом, который является головой списка детей, но ребенок не будет знать родителя. Односвязанные списки хороши с точки зрения

функционального программирования, так как голова списка не будет достижима из потомков. Тогда при добавлениии новой версии в голову или удалении объявляя другую вершину новой головой мы не будем терять старые версии, которые останутся на месте, так как фактически односвязный список с операциями на голове это персистентный стек, который является полностью персистентной функциональной структурой. При этом каждая версия будет поддерживать возможность изменения, что является полным уровнем персистентности. Также поддерживается операция merge для всех версий биномиальных куч, что позволяет получать новую версию путём сливания старых. Это добавляет конфлюэнтный уровень персистентности.

См. также

- Двоичная куча
- Фибоначчиева куча
- Левосторонняя куча
- Куча Бродала-Окасаки

Примечания

1. Github — реализация на Haskell (https://github.com/kgeorgiy/okasaki/tree/master/Okasaki/Chapter3)

Источники информации

- Википедия — Биномиальная куча (http://ru.wikipedia.org/wiki/Биномиальная_куча)
- Wikipedia — Binomial heap (http://en.wikipedia.org/wiki/Binomial_heap)
- INTUIT.ru — Биномиальные кучи (http://www.intuit.ru/department/algorithms/dscm/7/)
- Лекция А.С. Станкевича по приоритетным очередям (http://www.lektorium.tv/lecture/?id=14234)
- Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн Алгоритмы: построение и анализ — 2-е изд. — М.: «Вильямс», 2007. — с. 538—558. — ISBN 5-8489-0857-4
