2017-07-28

Fibonacci Heap

刘予希

0 目录

1 基本思想与操作

斐波拉契堆数据结构有两种操作。第一种，它支持一系列操作，这些操作构成了所谓的“可合并堆”。第二种，斐波那契堆的一些操作可以在均摊时间内完成。

MAKE-HEAP : O(1) 创建与返回一个新的不含任何元素的堆

INSERT(H, x) : O(1) 将一个已被填入关键字的元素x插入堆H中

MINIMUM(H) : O(1) 返回堆中最小值

DELETE-MINIMUM(H): O(log n) （均摊）从堆中删除最小关键字的元素

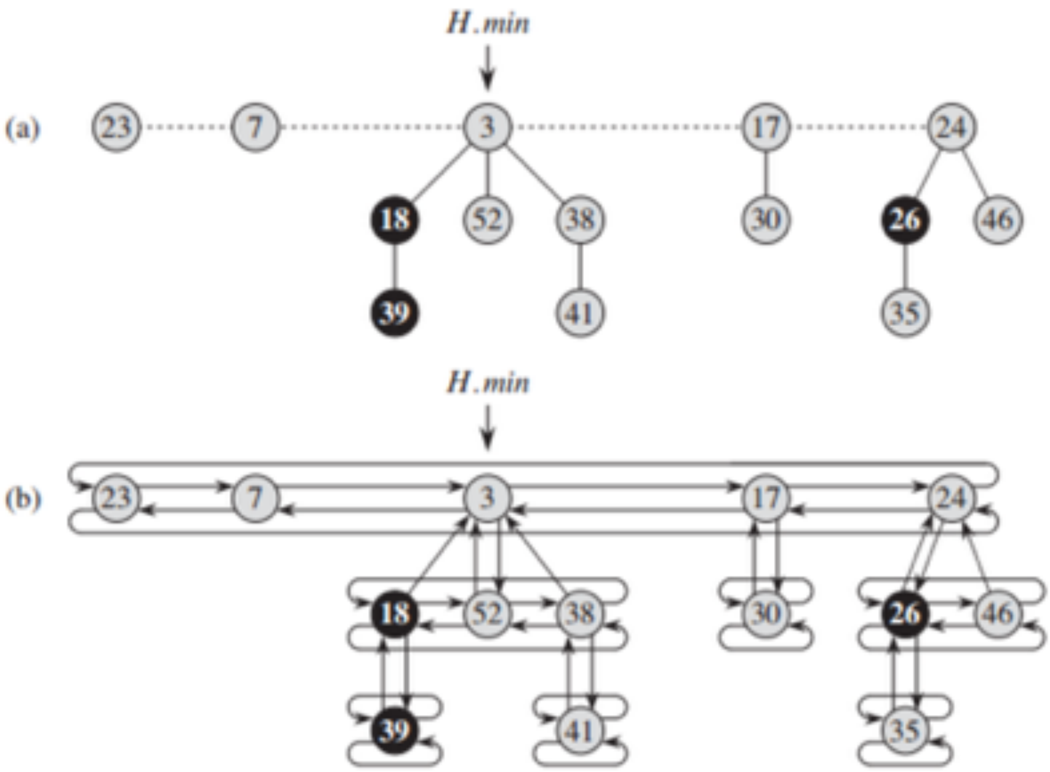
MERGE(H1, H2) : O(1) 合并两个斐波拉契堆

DECREASE-KEY(H, x, k) : O(1)（均摊）将堆中x元素的关键字修改成一个更小的k

DELETE(x) : O(log n) 删除堆中某个元素

2 斐波拉契堆结构

斐波拉契堆是由一组最小堆有序树组成，每棵树遵循最小堆性质，并且每棵树都是有根而无序的。每棵有序树的根通过双向链表连接，形成堆的根表，每个结点与他的兄弟们也通过双向链表连接。斐波拉契堆的结构如下：



（图中黑色结点，表示mark为true，即被标记了）

斐波拉契堆的每一个结点包含：

key（关键值）

degree（度数，孩子数）

left（左兄弟）

right（右兄弟）

parent（父亲）

child（某一个儿子）

mark（指示该结点自从上次称为另一个结点的孩子后，是否失去过孩子。新产生的结点是未标记的，并且当该结点成为另一个结点的孩子时便成为未标记的）

斐波拉契堆包含：

root （根链表中关键字最小的结点，即这个堆最小关键字结点）

size (堆的大小)

我们通过势方法来分析斐波拉契堆操作的性能。我们定义其势函数为：

potential = t + 2m

其中t为根链表中树的数目，m为被标记的结点个数。

我们定义D(n)为在一个n个结点的斐波拉契堆中任何结点的最大度数的上界。在第4节我们会证明它的大小 (D(n) = O(log n))，在DELETE操作和DECREASE-KEY操作中，D(n)会被用于证明时间复杂度。

3 可合并堆操作

3.1 创建一个新的斐波拉契堆

分配一个斐波拉契堆对象H，其中H.root = nullptr, H.size = 0，即H初始为空。

可见由于m = t = 0, 则potential = 0，均摊代价等于实际代价，O(1)

3.2 插入一个结点

把新加入的包含了关键值的堆结点插入到根链表中，如果原本根链表为空，则创建一个根链表，否则则通过简单的双向链表插入方法插入到根链表中。假设新的堆为_H，则 _t = t + 1, _m = m, _potential = potential + 1, 则均摊代价等于实际代价，O(1)

3.3 寻找最小结点

直接通过root得到，由于势并没有发生变化，则均摊代价等于实际代价，O(1)

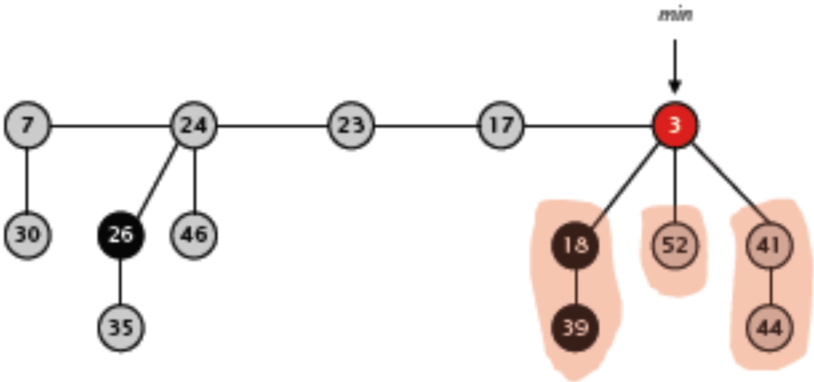
3.4 两个斐波拉契堆的合并

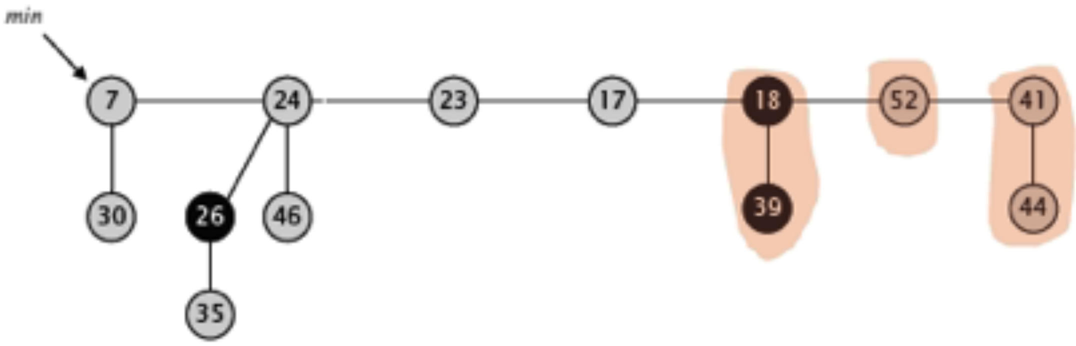
将两个斐波拉契堆的根链表通过双向链表的方法简单的合并在一起，并判断两个的root的关键值大小，获得合并的堆新的root。

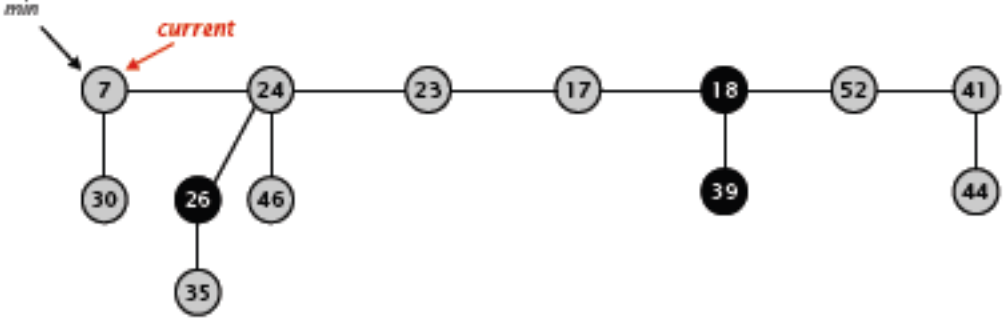
因为 t = t1 + t2 , m = m1 + m2, 则potential = potential1 + potential2，所以势并没有发生变化，则均摊代价等于实际代价，O(1)

3.5 删除最小结点

把root从根链表中删除，并把它的所有儿子都加到根链表中，并且更新root，注意这个root可能变为nullptr。



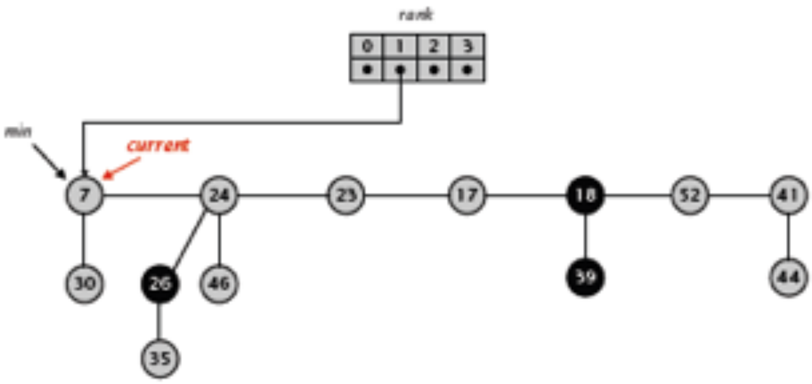




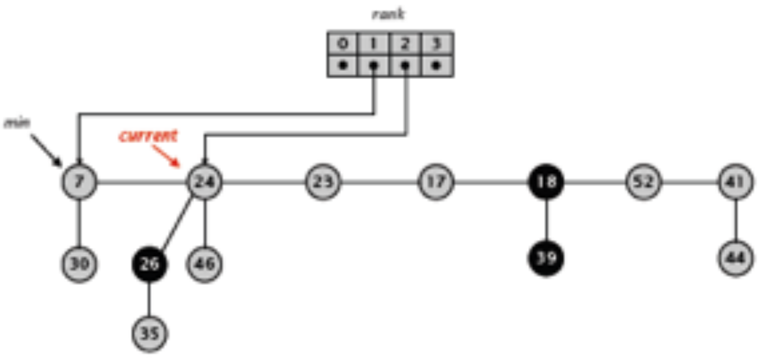
然后我们调用一个辅助过程consolidate()

其作用为把根链表中所有相同度数的结点合并起来，即保证该辅助过程结束后根链表中每个结点度数尽不想同。此处我们会开一个对应的度数数组A[D(n)]，记录对应度数所指向的结点。以下是一个例子：

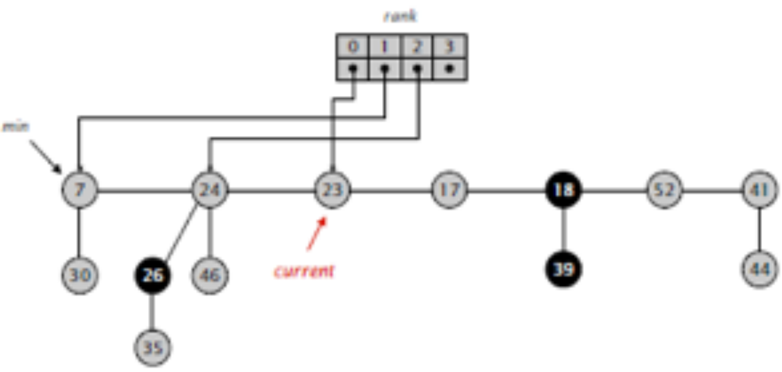
观察到7有1个孩子结点，即度为1，先保存起来，由于是初始的，肯定没有和7度相同的



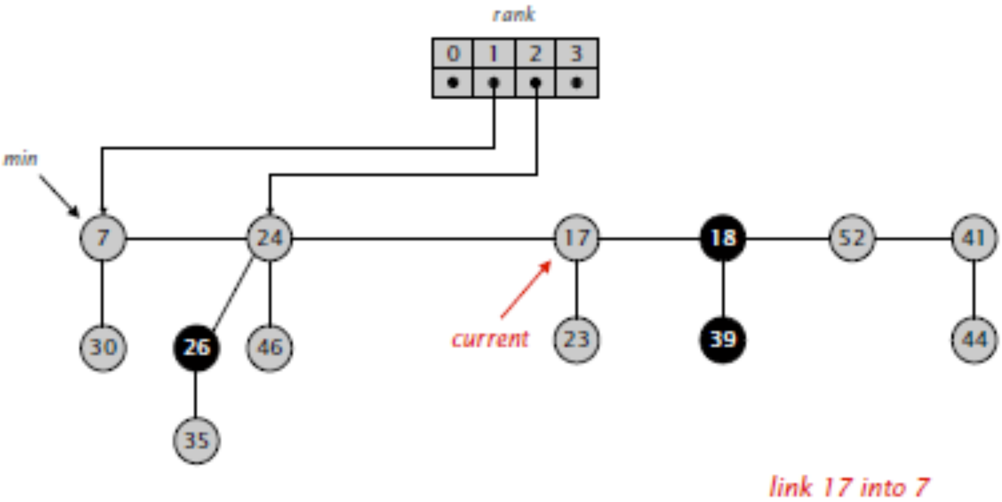
接着下一个根结点24，度为2，继续。



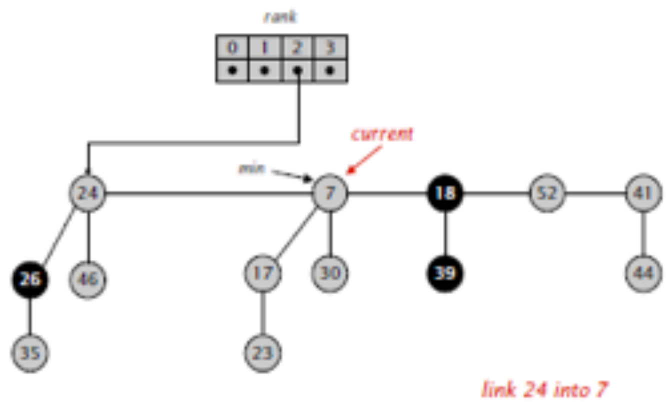
23, 度为0，继续

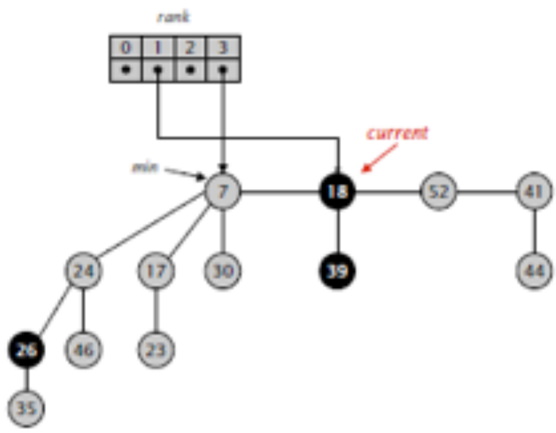


17, 度为0。 由于已经有度为1的根结点了（7），所以需要合并这两个结点，根据最小堆的性质，我们直接另关键值较大的根结点成为较小的结点的儿子即可。

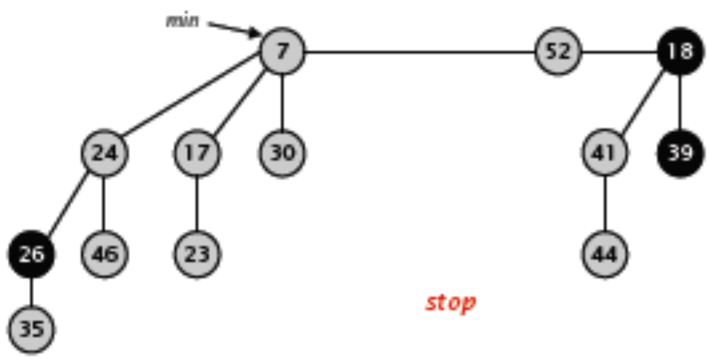


此时17的度为2，仍然重复，继续合并，直到没有度一样的根结点





最后结果如下图：



首先对于删结点，加儿子（最多为D(n)）的操作，以及申请A[D(n)]数组的操作均是O(D(n))的。

在相同度数的结点合并操作中，根链表的大小为D(n) + t - 1，而循环操作的总次数最多为根链表中的根的数目，所以抽取最小结点的实际代价为O(D(n) + t)。

抽取最小结点前potential = t + 2m，抽取并合并后最多有D(n) + 1个根结点留下，且在该过程中没有结点被标记（可能有结点被删除标记），所以该操作后势至多为D(n) + 1 + 2m。

则摊还代价至多为

$$O(D(n) + t) + (D(n) + 1 + 2m) - (t + 2m) = O(D(n)) + O(t) - t = O(D(n)) = O(\log n)$$

4 关键字减值和删除一个结点

4.1 关键字减值

首先先要保证关键字不比x当前的关键字大。然后如果修改之后，x的关键字依然不比父亲的关键字小，那么斐波拉契堆的结构依然得到保持，并不需要做修改；又或者x是根链表的一员，没有父亲，那也并不需要做修改，因为并没有违反最小堆序。

如果违反了最小堆序，那么我们需要进行很多操作。

首先切断x与它的父亲y，并将x加入到根链表中。

此处我们会使用先前定义的mark标记来得到需要的时间界，该标记记录了每个结点的一小段历史，假定下列步骤已经发生在结点x上：

1 在某个时刻，x是根

2 然后x被链接到了另一个结点y（成为y的孩子结点）

3 然后x的两个孩子被切断操作移除

一旦失掉第二个孩子，就切断x与其父亲结点的链接，使其成为根链表的一个新的成员。如果发生了第一步和第二步且x的一个孩子被切掉了，那么x.mark会被标记成TRUE。这也迎合了我们在第2节中对mark的定义。

我们的工作还没完成，因为上述操作可能会导致x与它的父亲结点y断开，那么同样的x可能是y断开的第二个孩子结点，y可能会成为新的x。

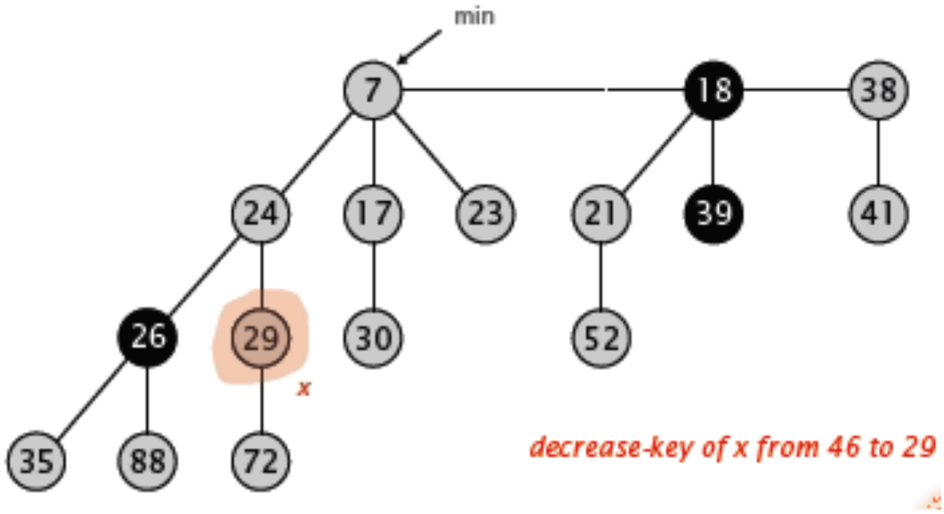
所以我们调用级联切断，cascading-cut，通过递归的形式，逐层往上，直至当前结点的mark标记还不是TRUE（也就是说未被删除过孩子)，或者说我们已经递归到了某个根，就结束（递归边界条件）。

当所有过程做完了，我们要更新root，那么唯一改变过的关键字可能成为答案，只要判断一下即可。

举个例子：

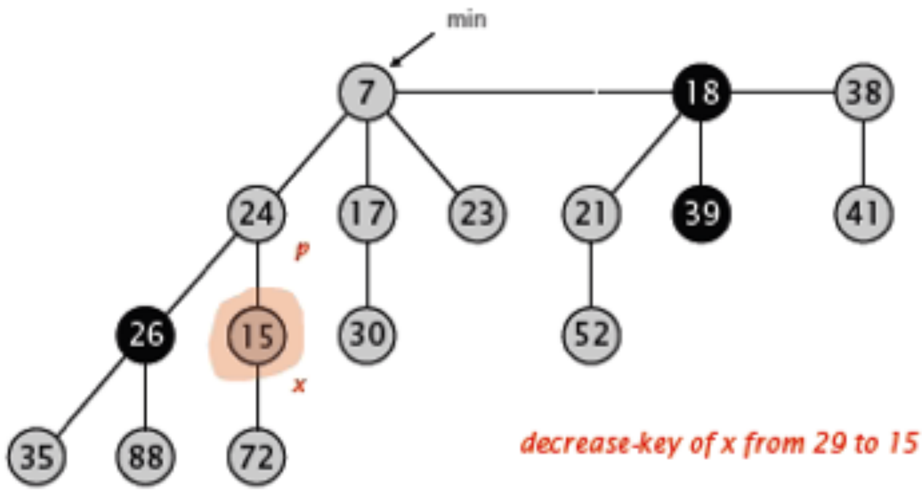
1、不违反最小堆性质

把46减小为29，不违反最小堆性质，不改变堆结构



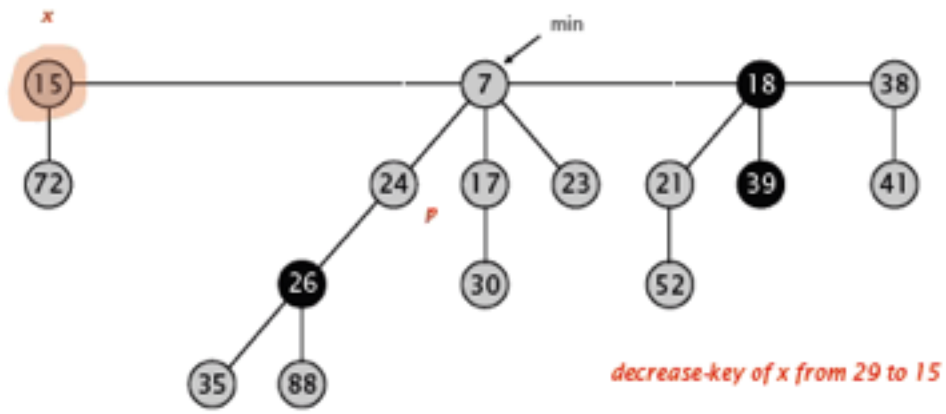
2、违反最小堆性质，合并到根链表中，并且unmark 它

把29减小为15，违反了堆性质

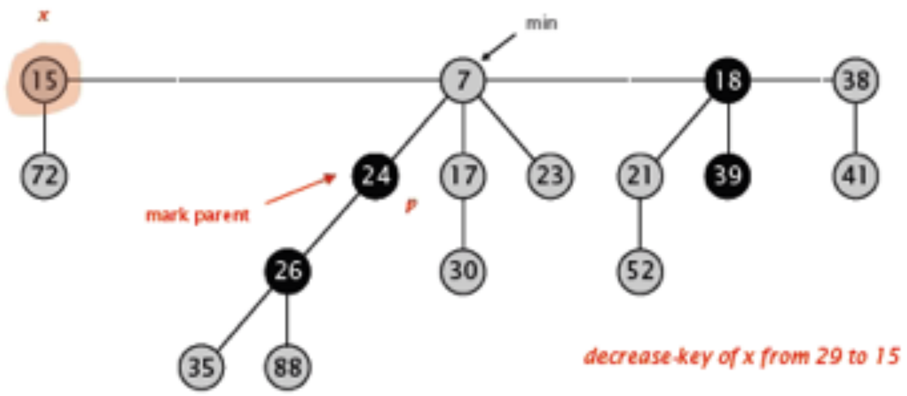


把15合并到根链表中

2017-07-28

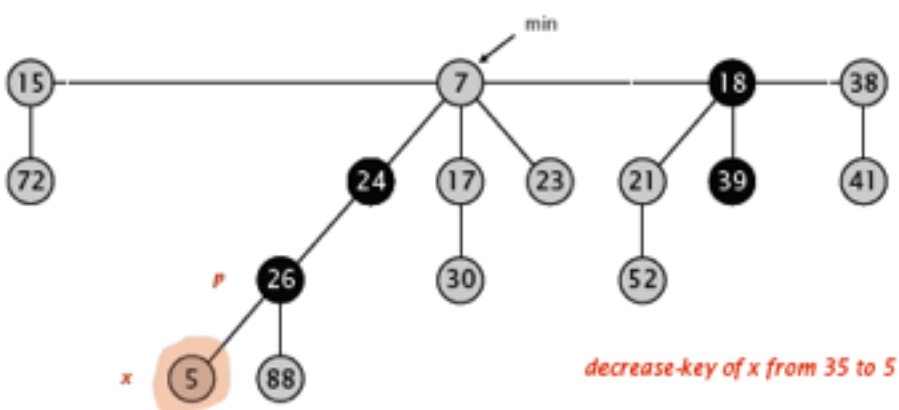


如果父节点没有mark(没有失去孩子), 设置它为mark



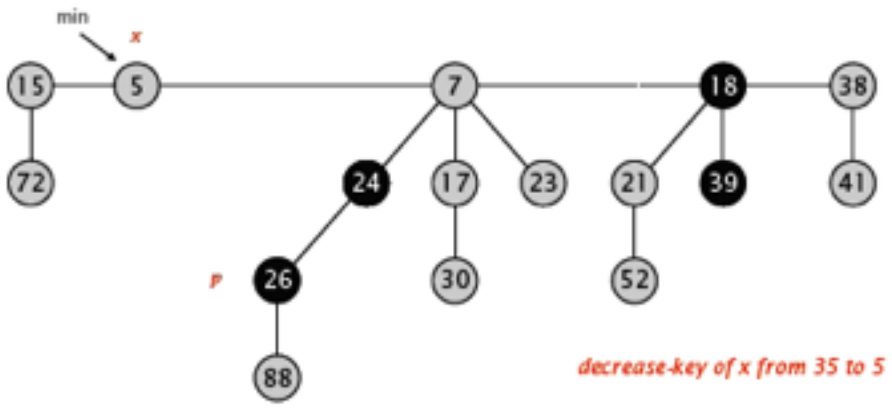
如果父节点已经是mark，则把父节点合并到根链表中，并设置为unmark。

把节点35减小到5

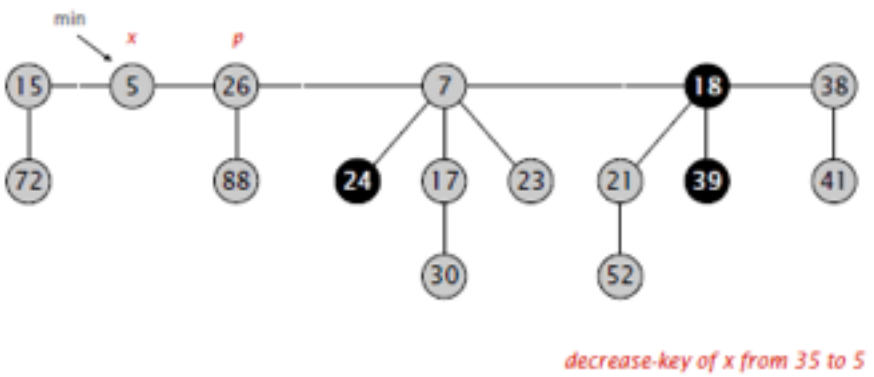


-

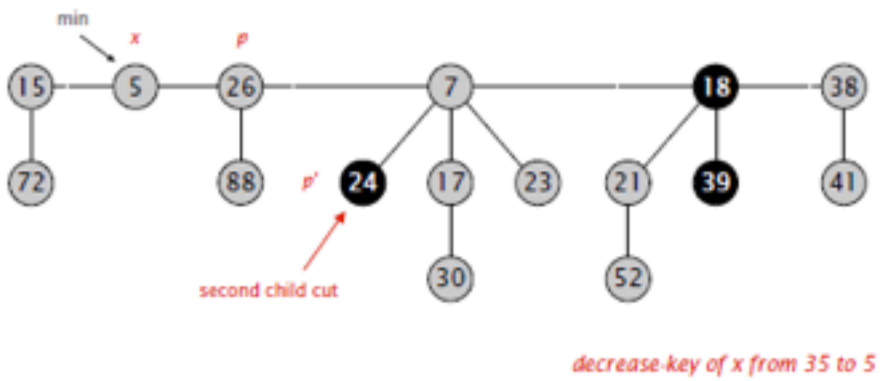
由于违反了，把5合并到根



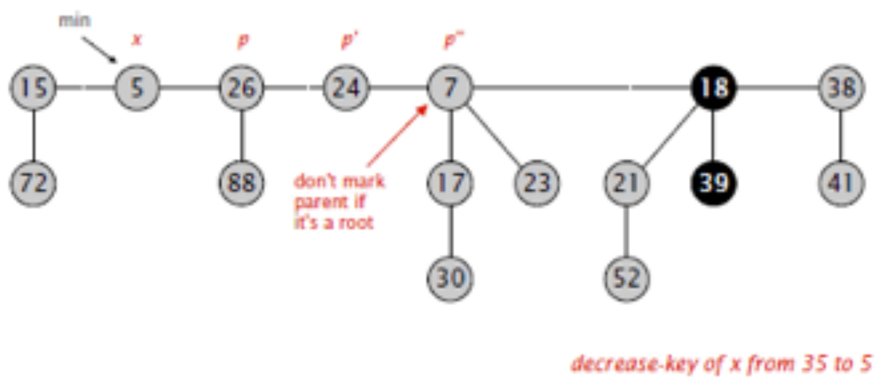
由于26已经mark，把26这个子树合并到根



同理24合并到根



由于7已经是根节点了，停止，全部结束



在级联切断的操作中，如果递归了c - 1次，那么实际代价是O(c)的。

接下来计算势的变化，设关键值减小操作执行前的斐波拉契堆为H，那么在接下来每次级联切断操作中（c - 1次），均将一个被mark的结点加入到根链表中，成为一棵新树，并把mark标记清除，此后斐波拉契堆包含t + c棵树（原来的t棵，c - 1次递归的新树，以及x这棵新树），而且最多有m - c + 2个标记结点 （c - 1次清空标记，最后一次调用级联操作可能有标记了一个点），那么势的变化至多为：

$$(t + c) + 2\ (m - c + 2) - t - 2m = 4 - c$$

那么关键字减值的摊还代价是：

$$O(c) + 4 - c = O(1)$$

至此，我们会明白为什么定义势函数的时候，要包含一个2倍于标记节点数目的项。当一个结点被一个级联切断操作切断时，它的标记为清空，势减小2。一个单位的势支付切断和标记的清除，一个单位的势不长了因为结点变成新的根而增加的势。

4.2 删除一个节点

删除一个节点的操作比较简单，只需要把该结点的关键值调整为负无穷（足够小的数），然后调用删除最小结点即可。时间复杂度为关键字减值的O(1)摊还时间加上删除最小结点的O(log n)摊还时间。

5 最大度数的界

特别地，要证明D(n) <= log φ (n) ，φ为底数。

以下，我们定义size(x)，为x结点的子树大小，x可为任意节点。

引理5.1: 假定x为斐波拉契堆中任意节点，并假定x.degree = k， 设y1, y2, ….., yk为x的孩子，并以它们链入顺序为准，那么y1.degree >= 0, yi >= i - 2 (i = 2, 3, ….. ,k)。

证明: 显然y1.degree >= 0， 那么对于 i >= 2 ，当yi链入x时，y1, y2, …, yi-1已经是x的孩子，则x.degree >= i - 1。因为结点yi 只有在 x.degree = yi.degree 的时候才会链入x（合并操作consolidate中），此时一定有 yi >= i - 1, 从这之后 yi至多失去一个孩子（失去两个则会被cascading-cut切断），所以 yi >= i - 2

我们回顾一下斐波拉契堆的定义：

F(0) = 0 F(1) = 1 F(k) = F(k - 1) + F(k - 2) (k = 2, 3, …..)

我们给出另一种表示F(k)的方法：

引理5.2: 对于所有的整数k >= 0, F(k + 2) = 1 + ∑F(k), 其中∑上下界分别为k， 0。

证明: 运用数学归纳法易得

引理5.3: 对于所有的整数k >= 0, 斐波拉契数满足 F(k + 2) >= φ^k 。

证明: 对k进行归纳，此处省略看k = 0, k = 1。对于k >= 2， 假定对于 i= 0, 1，… , k - 1, 有F(i + 2) > φ^i。因为φ是 x^2 = x + 1 的正根， 因此有：

F(k + 2) = F(k + 1) + F(k) >= φ^(k - 1) + φ^(k - 2) = φ^(k - 2) * (φ + 1)

= φ^(k - 2) * φ^2 = φ^k

综上所述，得证。

引理5.4: 设x是斐波拉契堆中的任意结点，并设k = x.degree，则有 size(x) >=

F(k + 2) >= φ^k。

证明: 设sk是斐波拉契堆中度数为k的任意结点的最小可能size。平凡地，s0 = 1，s1 = 2, sk最大为size(x), 且因为往一个结点上添加孩子不能减小结点的size，sk随着k单调增。在任意斐波拉契堆中，考虑某个结点z，有z.degree = k 和 size(z) = sk。因为 sk <= size(x), 所以可以通过计算sk的下界来得到 size(x)的一个下界。如同引理5.1中对x孩子的定义，并按照它们链入顺序。为了求sk的界，我们把z本身和z的第一个孩子y1(size(y1) >= 1)各算一个，则有：

size(x) >= sk

>= 2 + s(y2.degree) + s(y3.degree) + … + s(yk.degree)

>= 2 + s(0) + s(1) + … + s(k - 2)

依据引理5.1以及sk的单调性：

现在对k进行归纳证明，对于所有的非负整数k，有sk >= F(k + 2), 归纳基础k = 0 和k = 1先让城里。 假定k >= 2且对于i = 0, 1, … , k - 1 , 均有si >= F(i + 2)，则有：

sk >= 2 + s(0) + s(1) + … + s(k - 2)

>= 2 + F(2) + F(3) + … + F(k)

= F(k + 2)

>= φ^k

于是就证明了size(x) >= sk >= F(k + 2) >= φ^k

推论5.5: 一个n个结点的斐波拉契堆中任意结点的最大度数D(n)为O(log n)

6 正确性测试

6.1 insert_deleteMin_checker.cpp

6.2 heap_sort_checker.cpp

6.3 merge_checker.cpp

6.4 shortest_path_check.cpp

7 参考资料

1 《算法导论》

2 http://www.cnblogs.com/junyuhuang/p/4463758.html