| | |
|---|---|
| CMSC 651 Advanced Algorithms | Lecture 5 |
| Lecturer: Samir Khuller | Tu. Feb. 10, 1998 |

Original notes by King-Ip Lin.

# 1 Binomial heaps

These are heaps that also provide the power to merge two heaps into one.

**Definition 1.1 (Binomial tree)** *A* binomial tree of height $k$ *(denoted as $B_k$) is defined recursively as follows:*

1. *$B_0$ is a single node*
2. *$B_{i+1}$ is formed by joining two $B_i$ heaps, making one's root the child of the other*

Basic properties of $B_k$

- Number of nodes : $2^k$
- Height : $k$
- Number of children of root (degree of root) : $k$

In order to store $n$ nodes in binomial tress when $n \neq 2^k$, write $n$ in binary notation; for every bit that is "1" in the representation, create a corresponding $B_k$ tree, treating the rightmost bit as 0.

Example : $n = 13 = 1101_2 \longrightarrow$ use $B_3, B_2, B_0$

**Definition 1.2 (Binomial heap)** *A* binomial heap *is a (set of) binomial tree(s) where each node has a key and the heap-order property is preserved. We also have the requirement that for any given $i$ there is at most one $B_i$.*

Algorithms for the binomial heap :

**Find minimum** Given $n$, there will be $\log n$ binomial trees and each tree's minimum will be its root. Thus only need to find the minimum among the roots.

$Time = \log n$

**Insertion** Invariant to be maintained : only 1 $B_i$ tree for each $i$

Step 1: create a new $B_0$ tree for the new element

Step 2: $i \leftarrow 0$

Step 3: **while** there are still two $B_i$ trees do
join the two $B_i$ trees to form a $B_{i+1}$ tree
$i \leftarrow i + 1$

Clearly this takes at most $O(\log n)$ time.

**Deletion**

**Delete min** Key observation: Removing the root from a $B_i$ tree will form $i$ binomial trees, from $B_0$ to $B_{i-1}$

Step 1: Find the minimum root (assume its $B_k$)

Step 2: Break $B_k$, forming $k$ smaller trees

Step 3: **while** there are still at least two $B_i$ trees do
join the two $B_i$ trees to form a $B_{i+1}$ tree
$i \leftarrow i+1$

Note that at each stage there will be at most three $B_i$ trees, thus for each $i$ only one join is required.

**Delete**

Step 1: Find the element

Step 2: Change the element key to $-\infty$

Step 3: Push the key up the tree to maintain the heap-order property

Step 4: Call Delete min

Steps 2 and 3 are grouped and called DECREASE_KEY($x$)
Time for insertion and deletion : $O(\log n)$

## 1.1 Fibonacci Heaps (F-Heaps)

Amortized running time :

- Insert, Findmin, Union, Decrease-key : $O(1)$
- Delete-min, Delete : $O(\log n)$

Added features (compared to the binomial heaps)

- Individual trees are not necessary binomial (denote trees by $B_i'$)
- Always maintain a pointer to the smallest root
- permit many copies of $B_i'$

Algorithms for F-heaps:

**Insert**

Step 1: Create a new $B_0'$

Step 2: Compare with the current minimum and update pointer if necessary

Step 3: Store \$1 at the new root

(Notice that the \$ is only for accounting purposes, and the implementation of the data structure does not need to keep track of the \$'s.)

**Delete-min**

Step 1: Remove the minimum, breaking that tree into smaller tree again

Step 2: Find the new minimum, merge trees in the process, resulting in at most one $B_i'$ tree for each $i$

**Decrease-key**

Step 1: Decrease the key

Step 2: **if** heap-order is violated
break the link between the node and its parent (note: resulting tree may not be a true binomial tree)

Step 3: Compare the new root with the current minimum and update pointer if necessary

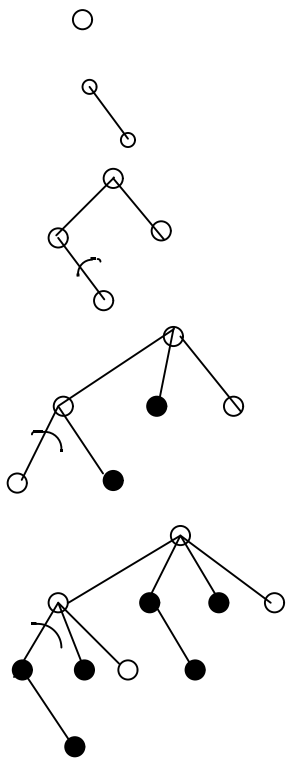
Step 4: Put \$1 to the new root

Step 5: Pay \$1 for the cut

Problem with the above algorithm: Can result in trees where the root has a very large number of children.
Solution:

- whenever a node is being cut
  - mark the parent of the cut node in the original tree
  - put \$2 on that node
- when a second child of that node is lost (by that time that node will have \$4), recursively cut that node from its parent, use the \$4 to pay for it:
  - \$1 for the cut
  - \$1 for new root
  - \$2 to its original parent
- repeat the recursion upward if necessary

Thus each cut requires only \$4.
Thus decrease-key takes amortized time $O(1)$
Define rank of the tree = Number of children of the root of the tree.
Consider the minimum number of nodes of a tree with a given rank:



| | Rank | Worst case size | Size of binomial tree |
|---|---|---|---|
| B0 | 0 | 1 | 1 |
| B1 | 1 | 2 | 2 |
| B2 | 2 | 3 | 4 |
| B3 | 3 | 5 | 8 |
| B4 | 4 | 8 | 16 |

Marked node from previous deletion

Figure 1: Minimum size $B_i'$ trees