5 Binomial heaps

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

**Definition 5.1 (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 tree 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: $n = 13 = 1101_2 \rightarrow$ use $B_3, B_2, B_0$

**Definition 5.2 (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: $O(\log n)$

### 5.1 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:
<table>
<thead>
<tr>
<th>Rank</th>
<th>Worst case size</th>
<th>Size of binomial tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>B4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

- Marked node from previous deletion

Figure 4: Minimum size $B^e_4$ trees
6 F-heap

6.1 Properties

F-heaps have the following properties:

- maintain the minimum key in the heap all the time,
- relax the condition that we have at most one $B_k$ tree for any $k$, i.e., we can have any number of $B_k$ trees co-existing.
- trees are not true binomial trees.

For example,

![Diagram of F-heap](image)

Figure 5: Example of F-heap

Property: size of a tree, rooted at $x$, is in exponential in the degree of $x$.

In Binomial trees we have the property that $size(x) \geq 2^k$, here we will not have as strong a property, but we will have the following:

$$ size(x) \geq \phi^k $$

where $k$ is degree of $x$ and $\phi = \frac{1+\sqrt{5}}{2}$.

Note that since $\phi > 1$ this is sufficient to guarantee a $O(\log n)$ bound on the depth and the number of children of a node.

6.2 Decrease-Key operation

Mark strategy:

- when a node is cut off from its parent, tick one mark to its parent,
- when a node gets 2 marks, cut the edge to its parent and tick its parent as well,
- when a node becomes a root, erase all marks attached to that node,
- every time a node is cut, give $1$ to that node as a new root and $2$ to its parent.
Figure 6: Marking Strategy
The cost of Decrease-Key operation = cost of cutting link + $3. We can see that no extra dollars are needed when the parent is cut off recursively since when the cut off is done, that parent must have $4 in hand ($2 from each cut child and there must be 2 children have been cut), then we use those $4 dollars to pay for cutting link cost ($1), giving to its parent ($2), and for itself as a new root ($1).

Example: How does Decrease-Key work (see CLR also)?

![Diagram](image)

Figure 7: Example

The property we need when two trees are merged, is the degree of the roots of both trees should be the same.

Let $y_i$ be the oldest child and $y_k$ be the youngest child. Consider $y_i$, at the point $y_i$ was made a child of the root $x$, $x$ had degree at least $i - 1$ and so does $y_i$. Since $y_i$ has lost at most 1 child, now $y_i$ has at least degree $i - 2$.

We now study some properties about Fibonacci numbers, and see how they relate to the sizes of the trees with the decrease-key operation. Define
Figure 8: Example

\[
F_k = \begin{cases} 
0 & \text{if } k = 0 \\
1 & \text{if } k = 1 \\
F_{k-1} + F_{k-2} & \text{if } k \geq 2
\end{cases}
\]

These numbers are called Fibonacci numbers.

**Property 6.1** \( F_{k+2} = 1 + \sum_{i=1}^{k} F_i \)

**Proof:**
[By induction]

\[
k = 1 : \quad F_3 = 1 + F_1 = 1 + 1 = F_1 + F_2 \quad \text{(where } F_2 = F_1 + F_0 = 1 + 0)\]

Assume \( F_{k+2} = 1 + \sum_{i=1}^{k} F_i \) for all \( k \leq n \)

\[
k = n + 1 : \quad F_{n+3} = 1 + \sum_{i=1}^{n+1} F_i = 1 + \sum_{i=1}^{n} F_i + F_{n+1} = F_{n+2} + F_{n+1}
\]

\( \square \)

**Property 6.2** \( F_{k+2} \geq \phi^k \)

**Proof:**
[By induction]

\[
k = 0 : \quad F_2 = 1 \geq \phi^0 \]
\[
k = 1 : \quad F_3 = 2 \geq \phi = \frac{1 + \sqrt{5}}{2} = 1.618.. \]

Assume \( F_{k+2} \geq \phi^k \) for all \( k < n \)
\[ k = n : \quad F_{n+2} = F_{n+1} + F_n \]
\[ \geq \phi^{n-1} + \phi^{n-2} \]
\[ \geq \frac{1 + \phi}{\phi^2} \cdot \phi^n \]
\[ \geq \phi^n \]

\[ \square \]

**Theorem 6.1** \( x \) is a root of any subtree. \( size(x) \geq F_{k+2} \geq \phi^k \) where \( k \) is a degree of \( x \)

**Proof:**

[By induction]

\( k = 0: \)

\[ \begin{array}{c}
\times \\
\end{array} \]

\( size(x) = 1 \geq 1 \geq 1 \)

\( k = 1: \)

\[ \begin{array}{c}
\times \\
\end{array} \]

\( size(x) = 2 \geq 2 \geq \frac{1 + \sqrt{5}}{2} \)

Assume \( size(x) \geq F_{k+2} \geq \phi^k \) for any \( k \leq n \)

\[ \begin{array}{c}
x \\
/ \quad / \\
y_1 \quad y_2 \quad \ldots \\
\end{array} \]

\( k = n + 1: \)

\[ size(x) = 1 + size(y_1) + ... + size(y_k) + ... + size(y_k) \]
\[ \geq 1 + 1 + 1 + F_3 + ... + F_k \ (\text{from assumption}) \]
\[ \geq 1 + F_1 + F_2 + ... + F_k \]
\[ \geq F_{k+2} \ (\text{from property1}) \]
\[ \geq \phi^k \ (\text{from property2}) \]

\[ \log_\phi size(x) \geq k \]

So the number of children of node \( x \) is bounded by \( \log_\phi size(x) \). \[ \square \]