Heaps

- Binary heap
- Binomial heap
- Fibonacci heap

Heap/Priority queue

- Find min/Delete; Insert;
- Decrease key (change value of the key)
- Merge two heaps ...

Binomial heaps:

- **Performance**: All of the following operations work in $O(\log n)$ time on a binomial heap with $n$ elements:
  - Insert a new element to the heap
  - Find the element with minimum key
  - Delete the element with minimum key from the heap
  - Decrease key of a given element
  - Delete given element from the heap
  - Merge two given heaps to one heap
  - Finding the element with minimum key can also be done in $O(1)$ by using an additional pointer to the minimum.

- [http://www.cse.yorku.ca/~aaw/Jason/FibonacciHeapAnimation.html](http://www.cse.yorku.ca/~aaw/Jason/FibonacciHeapAnimation.html)
- [http://www.jucs.org/jucs_7_5/animation_for_teaching_purposes/Lauer_T.html](http://www.jucs.org/jucs_7_5/animation_for_teaching_purposes/Lauer_T.html)
Binomial heaps, Fibonacci heaps, and applications

http://www.cs.tau.ac.il/~dannyF/0609/cs09a.htm
Dan Feldman

• CLRS:
  • http://net.pku.edu.cn/~course/101/resource/Intro2Algorithm/boost/chap30.htm

Binomial trees

Binomial trees

19.1.1 Binomial trees

Figure 19.2: (a) The recursive definition of the binomial tree $B_i$. Triangles represent rooted subtrees.
(b) The binomial trees $B_i$ through $B_4$, node depths in $B_i$ are shown. (c) Another way of looking at the binomial tree $B_i$. 

26.03.2009
Lemma 20.1

• For the binomial tree \( B_k \),
• 1. there are \( 2^k \) nodes,
• 2. the height of the tree is \( k \),
• 3. there are exactly \( \text{choose}(i \text{ from } k) \) nodes at depth \( i \) for \( i = 0, 1, \ldots, k \), and
• 4. the root has degree \( k \), which is greater than that of any other node; moreover if the children of the root are numbered from left to right by \( k - 1, k - 2, \ldots, 0 \), child \( i \) is the root of a subtree \( B_i \).

Properties of binomial trees

1) \( |B_k| = 2^k \)
2) \( \text{degree}(\text{root}(B_k)) = k \)
3) \( \text{depth}(B_k) = k \)

\( \Rightarrow \) The degree and depth of a binomial tree with at most \( n \) nodes is at most \( \log(n) \).

Define the rank of \( B_k \) to be \( k \).

Figure 20.4 The binomial tree \( B_k \) with nodes labeled in binary by a postorder walk.

Binomial heaps (def)

A collection of binomial trees at most one of every rank.

Items at the nodes, heap ordered.

Possible rep: Doubly link roots and children of every node. Parent pointers needed for delete.

Binomial heaps (operations)

Operations are defined via a basic operation, called linking, of binomial trees:

Produce a \( B_k \) from two \( B_{k-1} \), keep heap order.
Binomial heaps (ops cont.)

Basic operation is $\text{meld}(h1, h2)$:
Like addition of binary numbers.

$$
\begin{array}{cccc}
B_3 & B_4 & B_2 & B_1 \\
h1: & B_4 & B_3 & B_1 & B_0 \\
h2: & B_4 & B_3 & B_0 \\
\hline
& B_5 & B_4 & B_2 \\
\end{array}
$$

The execution of $\text{BINOMIAL-HEAP-UNION}$. (a)

Binomial heaps $H_1$ and $H_2$. (continued)

Delete min

Find min ($\leftarrow$)
Extract tree
Split tree, reverse
Merge/meld

Decrease key ($y=26 \Rightarrow y=7$)

Findmin($h$): obvious
insert($x, h$): meld a new heap with a single $B_0$
containing $x$, with $h$
delete($h$): Chop off the minimal root. Meld the subtrees with $h$. Update minimum pointer if needed.
delete($x, h$): Bubble up and continue like delete-min
decrease-key($x, h$, $\delta$): Bubble up, update min ptr if needed.

All operations take $O(\log n)$ time on the worst case, except
find-min($h$) that takes $O(1)$ time.
Amortized analysis

We are interested in the worst case running time of a sequence of operations.

Example: binary counter
single operation -- increment

00000
00001
00010
00011
00100
00101

Amortized analysis (Cont.)

On the worst case increment takes $O(k)$.

$k = \# \text{digits}$

What is the complexity of a sequence of increments (on the worst case) ?

Define a potential of the counter:

$\Phi (c) = ?$

Amortized(increment) = actual(increment) + $\Delta \Phi$

Amortized analysis (Cont.)

Amortized(increment$_1$) = actual(increment$_1$) + $\Phi_1$ - $\Phi_0$

Amortized(increment$_2$) = actual(increment$_2$) + $\Phi_2$ - $\Phi_1$

... +

Amortized(increment$_n$) = actual(increment$_n$) + $\Phi_n$ - $\Phi_{n-1}$

$\sum \text{Amortized(increment)} = \sum \text{actual(increment)} + \Phi_n - \Phi_0$

$\sum \text{Amortized(increment)} \geq \sum \text{actual(increment)}$

if $\Phi_n - \Phi_0 \geq 0$

Amortized analysis (Cont.)

Define a potential of the counter:

$\Phi (c) = \# \text{(ones)}$

Amortized(increment) = actual(increment) + $\Delta \Phi$

Amortized(increment) = 1 + $\#(1 \Rightarrow 0)$ + 1 - $\#(1 \Rightarrow 0)$ = $O(1)$

$\Rightarrow$ Sequence of $n$ increments takes $O(n)$ time

Binomial heaps - amortized ana.

$\Phi \text{(collection of heaps)} = \# \text{(trees)}$

Amortized cost of insert $O(1)$
Amortized cost of other operations still $O(\log n)$

Binomial heaps + lazy meld

Allow more than one tree of each rank.

Meld (h1,h2):

• Concatenate the lists of binomial trees.
• Update the minimum pointer to be the smaller of the minimums

$O(1)$ worst case and amortized.
Binomial heaps + lazy meld

As long as we do not do a delete-min our heaps are just doubly linked lists:

```
9
5
11
4
6
```

Delete-min : Chop off the minimum root, add its children to the list of trees.

**Successive linking:** Travers the forest keep linking trees of the same rank, maintain a pointer to the minimum root.

Possible implementation of delete-min is using an array indexed by rank to keep at most one binomial tree of each rank that we already traversed.

Once we encounter a second tree of some rank we link them and keep linking until we do not have two trees of the same rank. We record the resulting tree in the array

\[
\text{Amortized(delete-min)} = (\text{#links} + \text{max-rank}) - \text{#links} = O(\log(n))
\]

Fibonacci heaps (Fredman & Tarjan 84)

Want to do decrease-key(x,h,δ) faster than delete+insert. Ideally in \(O(1)\) time.

Why?

Dijkstra's shortest path algorithm

Let \(G = (V,E)\) be a weighted (weights are non-negative) undirected graph, let \(s \in V\). Want to find the distance (length of the shortest path), \(d(s,v)\) from \(s\) to every other vertex.

```
3
3 2
2
1
```

![Figure 23: A Fibonacci heap consisting of five min-heap-ordered trees and 14 nodes. The dashed line indicates the root list. The minimum node of the heap is the node containing the key 3. The three marked nodes are slackened. The potential of the particular Fibonacci heap is 0-2, 3 = 11. (a) A more complete representation showing pointers o (up arrows), endl (down arrows), and self and right node-up arrows. These details are omitted in the remaining figures in this chapter, since all the information shown here can be determined from what appears in part (a).](image)

Insert (left from root)
Finding the minimum node

- The minimum node of a Fibonacci heap \( H \) is given by the pointer \( \text{min}[H] \), so we can find the minimum node in \( O(1) \) actual time. Because the potential of \( H \) does not change, the amortized cost of this operation is equal to its \( O(1) \) actual cost.

Figure 21.3 The action of FIB-HEAP-EXTRACT-MIN.

Application #2: Prim’s algorithm for MST

Start with \( T \) a singleton vertex.

Grow a tree by repeating the following step:

Add the minimum cost edge connecting a vertex in \( T \) to a vertex out of \( T \).

Two calls of FIB-HEAP-DECREASE-KEY.

Application #2: Prim’s algorithm for MST

Maintain the vertices out of \( T \) but adjacent to \( T \) in a heap.

The key of a vertex \( v \) is the weight of the lightest edge \((v,w)\) where \( w \) is in the tree.

Iteration: Do a delete-min. Let \( v \) be the minimum vertex and \((v,w)\) the lightest edge as above. Add \((v,w)\) to \( T \). For each edge \((w,u)\) where \( u \in T \),

if \( \text{key}(u) = \infty \) insert \( u \) into the heap with \( \text{key}(u) = w(u) \)
if \( w(u) < \text{key}(u) \) decrease the key of \( u \) to be \( w(u) \).

With regular heaps \( O(m \log(n)) \).
With F-heaps \( O(n \log(n) + m) \).
Suggested implementation for decrease-key(x, h, δ):
If x with its new key is smaller than its parent, cut the subtree rooted at x and add it to the forest. Update the minimum pointer if necessary.

Decrease-key (cont.)

Does it work?
Obs1: Trees need not be binomial trees any more.. Do we need the trees to be binomial? Where have we used it?
In the analysis of delete-min we used the fact that at most log(n) new trees are added to the forest. This was obvious since trees were binomial and contained at most n nodes.

Fibonacci heaps (cont.)
We shall allow non-binomial trees, but will keep the degrees logarithmic in the number of nodes.
Rank of a tree = degree of the root.
Delete-min: do successive linking of trees of the same rank and update the minimum pointer as before.
Insert and meld also work as before.
**Fibonacci heaps (delete)**

\( \text{Delete}(x,h) : \text{Cut the subtree rooted at } x \text{ and then proceed with cascading cuts as for decrease key.} \)

Chop off \( x \) from being the root of its subtree and add the subtrees rooted by its children to the forest

If \( x \) is the minimum node do successive linking

---

**Fibonacci heaps (analysis)**

Want everything to be \( O(1) \) time except for delete and delete-min.

\( \implies \) cascading cuts should pay for themselves

\[ \Phi(\text{collection of heaps}) = \#(\text{trees}) + 2\#(\text{marked nodes}) \]

\[ \text{Actual(decrease-key)} = O(1) + \#(\text{cascading cuts}) \]

\[ \Delta \Phi(\text{decrease-key}) = O(1) - \#(\text{cascading cuts}) \]

\( \implies \) amortized(decrease-key) = \( O(1) \)

---

**Fibonacci heaps (analysis)**

The potential of a Fibonacci heap is given by

\[ \text{Potential} = t + 2m \] where \( t \) is the number of trees in the Fibonacci heap, and \( m \) is the number of marked nodes. A node is marked if at least one of its children was cut since this node was made a child of another node (all roots are unmarked).

---

**Fibonacci heaps (analysis)**

What about delete and delete-min?

Cascading cuts and successive linking will pay for themselves. The only question is what is the maximum degree of a node?

How many trees are being added into the forest when we chop off a root?
Lemma 1: Let x be any node in an F-heap. Arrange the children of x in the order they were linked to x, from earliest to latest. Then the i-th child of x has rank at least i-2.

Proof:
When the i-th node was linked it must have had at least i-1 children.
Since then it could have lost at most one.

Corollary 1: A node x of rank k in a F-heap has at least \( \phi^k \) descendants, where \( \phi = (1 + \sqrt{5})/2 \) is the golden ratio.

Proof:
Let \( s_k \) be the minimum number of descendants of a node of rank k in a F-heap.
By Lemma 1, \( s_k \geq s_0 + 2 \)
\( s_0 = 1, s_1 = 2 \)

Fibonacci numbers satisfy
\( F_{k+2} = \sum_{i=2}^{k} F_i + 2 \), for \( k \geq 2 \), and \( F_2 = 1 \)
so by induction \( s_k \geq F_{k+2} \)
It is well known that \( F_{k+2} \geq \phi^k \)

It follows that the maximum degree k in a F-heap with n nodes is such that \( \phi^k \leq n \)
so \( k \leq \log(n) / \log(\phi) = 1.4404 \log(n) \)

Summary of running times

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked List</th>
<th>Binary Tree</th>
<th>Min-Heap</th>
<th>Fibonacci Heap</th>
<th>Bounded Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>O(1)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>delete</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>decreaseKey</td>
<td>O(1)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>merge</td>
<td>O(1)</td>
<td>O(m log n/m)</td>
<td>O(m log n/m)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

*Amortized time

C code

- [http://www.cs.unc.edu/~bbb/foos/binheaps/heap.h](http://www.cs.unc.edu/~bbb/foos/binheaps/heap.h)
- [http://www.cs.unc.edu/~bbb/Binomial_heaps](http://www.cs.unc.edu/~bbb/Binomial_heaps)
More...

- http://www.cse.yorku.ca/~aaw/latex/FibonacciHeapAnimation.html