Mergeable Heaps

Operations:

- \( H \leftarrow \text{Create}() \)
- \( \text{Insert}(H, x) \) – insert node \( x \)
- \( x \leftarrow \text{FindMin}(H) \) – return node with minimum value
- \( x \leftarrow \text{ExtractMin}(H) \) – delete node with minimum value
- \( H \leftarrow \text{Union}(H_1, H_2) \) – destructive union
- \( \text{Decrease}(H, x, v) \) – decrease value of node \( x \) to \( v \)
- \( \text{Delete}(H, x) \) – delete node \( x \)
<table>
<thead>
<tr>
<th>procedure</th>
<th>binary heap</th>
<th>2-3 trees</th>
<th>binomial heap</th>
<th>fibonacci heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insert</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>FindMin</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(\log n)*</td>
</tr>
<tr>
<td>Union</td>
<td>O(n)</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Decrease</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(1)*</td>
</tr>
<tr>
<td>Delete</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(\log n)</td>
<td>O(\log n)*</td>
</tr>
</tbody>
</table>

* = amortized cost
Binomial Trees

\( B_k \ (k = 0, 1, 2, \ldots) \)

\( B_0 = \) single node

\[ B_k = \begin{array}{c}
B_{k-1} \\
B_{k-1}
\end{array} \]
Properties of $B_k$

- $2^k$ nodes
- height = $k$
- at depth $i$, there are $\binom{k}{i}$ nodes
  (Pascal’s triangle)
- root has $k$ children, which are roots of $B_{k-1}, \ldots, B_0$

- all nodes beside the root have $< k$ children

Corollary: in an $n$-node binomial tree, every node has degree $\leq \log_2 n$
Binomial Heaps

$H = a set$ of binomial trees

Each node stores an item

Binomial Heap Properties:

- each tree in $H$ satisfies the usual min-heap property
- for each $k \geq 0$, $B_k$ occurs in $H$ at most once

Implication: $|H| \leq \log_2 n + 1$

Proof. Let $|H| = t$

$n \geq 2^0 + 2^1 + \cdots + 2^{t-1} = 2^t - 1$

$\Rightarrow 2^t \leq n + 1 \Rightarrow t \leq \log_2(n + 1) \leq \log_2 n + 1$
Some implementation details:

- each node has
  - a value field
  - a pointer to its list of children
  - and a count of the # of children
  - a pointer to its parent

- the heap itself is a list of binomial trees in order of increasing size:
  \[(B_{k_1}, B_{k_2}, \ldots, B_{k_t})\]
  \[0 \leq k_1 < k_2 < \ldots < k_t \leq \log_2 n, \quad t \leq \log_2 n + 1\]

- \(\text{Min}[H] :=\) pointer to node with minimum value (a root of one of the trees)
Mergeable Heap Operations

Create():

FindMin(H): return Min[H]

H ← Union(H₁, H₂):

Low-level merge step — time = O(1)
Use a simple “merge sort like” procedure:

**Result:**  
$B_{k_1}, \ldots, B_{k_t}$

**Inputs:**  
$B_{\ell_1}, B_{\ell_2}, \ldots$  
$B_{m_1}, B_{m_2}, \ldots$

**Invariants:**  
$k_1 < \cdots < k_t \leq \ell_1 < \ell_2 < \cdots$  
$k_t \leq m_1 < m_2 < \cdots$

**Logic:**

if $\ell_1 = m_1$ then
    append merge of $B_{\ell_1}$ and $B_{m_1}$ to result
else if $\ell_1 < m_1$ then
    append/merge $B_{\ell_1}$ to result
else
    append/merge $B_{m_1}$ to result
**Insert**($H, x$): make a heap $H_1$ out of $x$, 
$H \leftarrow \text{Union}(H, H_1)$

**ExtractMin**($H$):

- $x \leftarrow \text{Min}[H]$

- Let $H_1$ be the heap obtained by removing the tree rooted at $x$ from $H$
- Let $H_2$ be the heap consisting of the trees rooted at $x$’s children (in reverse order)
- $H \leftarrow \text{Union}(H_1, H_2)$, return $x$
Decrease($H, x, v$):

- Usual “bubble up” procedure (no structural changes)

Delete($H, x$):

- $Decrease(H, x, -\infty), ExtractMin(H)$
Fibonacci Heaps

- A list $H$ of min-ordered trees
- Each node $x$ has:
  - a value field
  - a pointer to a list of children
  - a child count
  - a parent pointer
  - a boolean field $mark[x]$ (initially false)
- $Min[H] :=$ pointer to node with minimum value (a root of one of the trees)
Potential Function

\[ t(H) := \# \text{ of trees} \]
\[ m(H) := \# \text{ of marked nodes} \]
\[ \Phi(H) := t(H) + 2m(H) \]

Actually, we maintain a collection of heaps, and the “global” \( \Phi = \text{sum of the individual } \Phi \text{'s} \)

Maximum degree

- \( D(n) := \text{an upper bound on the degree (}\# \text{ of children}\text{) of any node in an } n\text{-node Fibonacci heap} \)
If no *Decrease* or *Delete* operations are performed:

- all trees are binomial trees (although some trees may have the same size, and the trees are not sorted by size)
- \( D(n) \leq \log_2 n \)
- all nodes are unmarked
Create(): $c = 1, \Delta \Phi = 0 \Rightarrow \hat{c} = 1$

Insert($H, x$): just append a new 1-item tree, and update $Min[H]$

$$c = 1, \Delta \Phi = 1 \Rightarrow \hat{c} = 2$$

$FindMin(H)$: return $Min[H]$

$$c = 1, \Delta \Phi = 0 \Rightarrow \hat{c} = 1$$

$H \leftarrow \text{Union}(H_1, H_2)$: just concatenate the two lists of trees, and calculate $Min[H]$

$$c = 1, \Delta \Phi = 0 \Rightarrow \hat{c} = 1$$
**ExtractMin**(\(H\)):

- \(x \leftarrow Min[H]\)
- Update \(Min[H]\) by examining \(x\)’s children, and the roots of all the other trees in \(H\)
- Merge the trees rooted at the children of \(x\) with the other trees in \(H\)
  - consolidate trees so that no two have roots with the same degree
  - if no \(Decrease\) or \(Delete\) operations have been performed, the result is a binomial heap
Details of the merge step:

- \( n := \# \) of nodes in \( H \)
- \( t := \# \) of trees in \( H = t(H) \)
- \( d := \# \) of children of \( x \leq D(n) \)
- We need to consolidate \( d + t - 1 \) trees:
  \[ T_1, T_2, \ldots, T_{d+t-1} \]

- Let \( d_i := \) the degree of \( T_i \)'s root
- Initialize an array \( A[0..D(n)] \) of trees (each initialized to “⊥”)
- Let “Merge” be the low-level merge operation that we used to merge two binomial trees
Logic:

for $i \leftarrow 1$ to $d + t - 1$ do
  $k \leftarrow d_i$
  while $A[k] \neq \bot$ do
    $(*)$ $T_i \leftarrow \text{Merge}(T_i, A[k])$
    $A[k] \leftarrow \bot$
    $k \leftarrow k + 1$
  $A[k] \leftarrow T_i$

Invariants:

- at any time, $A[k] = \bot$ or is a tree whose root has degree $k$
- at the line marked “$(*)$”, the degree of $T_i$’s root increases by 1
Actual cost:

- The consolidate routine works like a binary counter, and takes time $O(d + t - 1)$
- All other steps also take time $O(d + t - 1)$
- $\therefore$ we may set $c := D(n) + t(H) - 1$

Change in potential:

- $\Phi_0 = t(H) + 2m(H)$
- $\Phi_1 \leq (D(n) + 1) + 2m(H)$, since after consolidation, at most $D(n) + 1$ trees remain
- $\Delta \Phi := \Phi_1 - \Phi_0 \leq D(n) + 1 - t(H)$

Amortized cost: $\hat{c} := c + \Delta \Phi \leq 2D(n)$
\[ \hat{c} \leq 2D(n) \]

If these are the only operations performed, then

- \( D(n) \leq \log_2 n \)
- amortized cost of \( ExtractMin \) is \( O(\log n) \)

Next time: \( Decrease \) and \( Delete \)

- Binomial tree structure will be destroyed
- We’ll finally make use of “marks”
- We’ll need to derive an upper bound
  \[ D(n) = O(\log n) \]