Mergeable Heaps
Mergeable Heaps

Operations:

• \( H \leftarrow Create() \)

• \( \text{Insert}(H, x) \) – 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

• \( \text{Decrease}(H, x, \nu) \) – decrease value of node \( x \) to \( \nu \)

• \( \text{Delete}(H, x) \) – delete node \( x \)
<table>
<thead>
<tr>
<th>procedure</th>
<th>binary heap</th>
<th>2-3 trees</th>
<th>binom heap</th>
<th>fib 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>$O(\log n)\ast$</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>$O(1)\ast$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)\ast$</td>
</tr>
</tbody>
</table>

* = amortized cost
Binomial Trees

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

$B_0 = \text{single node}$

$B_k :$

$B_k = B_{k-1} \quad (k = 1, 2, 3, \ldots)$

$B_0 = \text{single node}$
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

$H = \text{a set of binomial trees}$

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:
  \[
  (B_{k_1}, B_{k_2}, \ldots, B_{k_t})
  \]
  \[
  0 \leq k_1 < k_2 < \cdots < k_t \leq \log_2 n, \quad t \leq \log_2 n + 1
  \]

- \textit{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)

\[ x \geq y \]
Use a simple “merge sort like” procedure:

<table>
<thead>
<tr>
<th>Result:</th>
<th>Inputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{k_1}, \ldots, B_{k_t}$</td>
<td>$B_{\ell_1}, B_{\ell_2}, \ldots$</td>
</tr>
<tr>
<td></td>
<td>$B_{m_1}, B_{m_2}, \ldots$</td>
</tr>
</tbody>
</table>

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
  merge (if $\ell_1 = k_t$) or append (o/w) $B_{\ell_1}$ to result
- else
  merge (if $m_1 = k_t$) or append (o/w) $B_{m_1}$ 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

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

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

Maximum degree

- \( D(n) := \text{an upper bound on the degree (\# of children) of any node in an } n\text{-node Fibonacci heap} \)
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):

- \( x \leftarrow \text{Min}[H] \)
- Update \( \text{Min}[H] \) 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 \textit{Decrease} or \textit{Delete} operations have been performed, the result is a binomial heap
Details of the merge step:

- \( n := \# \text{ of nodes in } H \)
- \( t := \# \text{ of trees in } H = t(H) \)
- \( d := \# \text{ of children of } x \leq D(n) \)
- We need to consolidate \( d + t - 1 \) trees: \( T_1, T_2, \ldots, T_{d+t-1} \)
- Let \( d_i := \text{ the degree of } T_i \text{'s root} \)
- Initialize an array \( A[0..D(n)] \) of trees (each initialized to “⊥”)
- Let “\textit{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
    $(\ast)$ $T_i \leftarrow \text{Merge}(T_i, A[k])$
    $A[k] \leftarrow \bot$
    $k \leftarrow k + 1$
  $A[k] \leftarrow T_i$

Invariants:

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

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

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: $\hat{c} := c + \Delta \Phi \leq 2D(n) + 1$
\[ \hat{c} \leq 2D(n) + 1 \]

If these are the only operations performed, then

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

Next up: Decrease and Delete

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

- Create/destroy a single-node tree
- Merge node \( x \) into node \( y \):
  - \( x \) and \( y \) are roots of trees, with \( \text{degree}[x] = \text{degree}[y] \)
  - we make \( x \) a new child of \( y \)
- Cut node \( x \):
  - \( x \) has a parent \( y \)
  - we detach \( x \) from \( y \), making \( x \) the root of its own tree

These are the only structure-modifying operations we will use
Marking nodes:

• We will place “marks” on certain nodes
• When a node is created, it is unmarked
• Whenever a node is cut, any mark on it is removed
  ◦ the logic of ExtractMin needs to be modified to deal with this
  ◦ does not increase the amortized cost of any operation discussed so far
• Roots will never have marks
Operation $Decrease(H, x, v)$

- update $Min[H]$
- if min-heap property is violated then
  
  repeat
  
  $y \leftarrow parent[x]$
  
  $(\ast)$ cut $x$
  
  $x \leftarrow y$

  until $x$ is unmarked

  if $x$ is not a root then

  $(\ast\ast)$ mark $x$
Node “lifecycle”:  
- initially, node is a root  
- Gain/lose several children  
- Merge into another node  
- Lose (at most) one child  
- Cut, becoming a root again
Amortized cost of *Decrease*

Let $c = \# \text{ of loop iterations}$

Recall $\Phi(H) = t(H) + 2m(H)$, where $t(H) = \# \text{ of trees in } H$, and $m(H) = \# \text{ of marked nodes in } H$

$t(H)$ increases by $c$

$m(H)$ decreases by at least $c - 2$:

• each execution of $(\ast)$, except possibly the first, removes a mark $\Rightarrow \Phi$ decreases by at least $c - 1$

• one mark may be added at line $(\ast \ast)$ $\Rightarrow \Phi$ may increase by 1

$\therefore \hat{c} = c + \Delta \Phi \leq c + (c - 2(c - 2)) = 4$
Implementation of $Delete(H, x)$

- $Decrease(H, x, -\infty), ExtractMin(H)$
- $\hat{c} = O(D(n))$

Bounding $D(n)$

- Recall that $D(n)$ is an upper bound on the degree of any node in an $n$-node Fibonacci heap, and that the amortized cost of $ExtractMin$ is $O(D(n))$
- Without $Decrease$ and $Delete$, all trees are binomial trees, and $D(n) \leq \log_2 n$
- Even with $Decrease$ and $Delete$, we can still show that $D(n) = O(\log n)$
Lemma 1

Let $x$ be a node in a Fibonacci heap, with $\text{degree}[x] = k$. Suppose $y_1, \ldots, y_k$ are the children of $x$, listed in the order in which they were last merged into $x$. Then $\text{degree}[y_i] \geq i - 2$ for $i = 2 \ldots k$.

Proof. Let $t_1 := \text{current time}$,

$t_0 := \text{time when } y_i \text{ was last merged into } x$

At time $t_0$: nodes $y_1, \ldots, y_{i-1}$ are children of $x$

At time $t_0$: $\text{degree}[y_i] = \text{degree}[x] \geq i - 1$

Between time $t_0$ and $t_1$:

$y_i$ is not cut $\Rightarrow$ $y_i$ looses at most one child

$\therefore$ at time $t_1$: $\text{degree}[y_i] \geq i - 2$  QED
Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, $F_{k+2} = F_k + F_{k+1}$

**Facts:**

- $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$
- $F_{k+2} \geq \phi^k$, where $\phi = (1 + \sqrt{5})/2 \approx 1.618$ is a root of $x^2 = x + 1$
Lemma 2

Let $x$ be any node in a Fibonacci heap, let $k = \text{degree}[x]$, and let $n = \# \text{ of nodes in the tree rooted at } x$. Then $n \geq F_{k+2}$.

**Proof.** Induction on $n$. $n = 1$: $k = 0$, $F_2 = 1$

$n > 1$: Let $y_1, \ldots, y_k$ be the children of $x$, as in Lemma 1, let $d_i := \text{the degree of } y_i$, and let $n_i := \text{the size of the sub-tree rooted at } y_i$

\[
    n = 1 + \sum_{i=1}^{k} n_i \geq 2 + \sum_{i=2}^{k} n_i \geq 2 + \sum_{i=2}^{k} F_{d_i+2} \quad \text{(induction)}
\]

\[
    \geq 2 + \sum_{i=2}^{k} F_i \quad \text{(Lemma 1, } F_i \text{ increasing)}
\]

\[
    = 1 + \sum_{i=0}^{k} F_i = F_{k+2} \quad \text{QED}
\]
Corollary

\[ n \geq \phi^{D(n)} \]

Thus, \( D(n) \leq \log_\phi(n) \)

Putting it all together — for a Fibonacci heap:

- \textit{Create, Insert, FindMin, and Union} take time \( O(1) \)
- \textit{Decrease} takes amortized time \( O(1) \)
- \textit{ExtractMin} and \textit{Delete} take amortized time \( O(\log n) \)