Fibonacci Heaps — Review

- 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 $\text{Decrease}$ or $\text{Delete}$ operations have been performed, the result is a binomial heap
- Amortized cost: $\hat{c} = O(D(n))$

Still to do: $\text{Decrease}$ and $\text{Delete}$
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 \textit{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 $\text{Decrease}(H, x, v)$

- update $\text{Min}[H]$
- if min-heap property is violated then
  
  repeat
  
  $y \leftarrow \text{parent}[x]$
  
  $(\ast)$ 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 \( c = \# \) of loop iterations

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

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

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 $\text{ExtractMin}$ is $O(D(n))$
- Without $\text{Decrease}$ and $\text{Delete}$, all trees are binomial trees, and $D(n) \leq \log_2 n$
- Even with $\text{Decrease}$ and $\text{Delete}$, 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 $t_1$: $\text{degree}[y_i] \geq i - 2$ \hspace{1cm} 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:

- **Create, Insert, FindMin, and Union** take time \( O(1) \)
- **Decrease** takes amortized time \( O(1) \)
- **ExtractMin** and **Delete** take amortized time \( O(\log n) \)
Disjoint Set Operations

We want to maintain a collection of disjoint sets
Each set is identified by one of its members, called the *representative* of the set

Operations:

- *MakeSet(x)* – create a the singleton set \{x\}
- *Union(x, y)* – form the union of sets whose representatives are \(x\) and \(y\) (original sets are lost)
- *Find(x)* – find the representative of the set containing \(x\)
A simple approach:

- A set is implemented as a doubly linked list of nodes
- the representative is the left-most node in the list
- each node in the list contains a pointer to the representative
- the representative contains the length of the list, and a pointer to the right-most node
- \textit{MakeSet} and \textit{Find} — trivial, \( O(1) \)
- \textit{Union}: concatenate lists Longest \( \parallel \) Shortest, and update pointers to representative in Shortest
Theorem

Any sequence of \( m \) operations, of which \( n \) are \textit{MakeSet}, takes time \( O(m + n \log n) \).

Proof. Want to show that total time spent updating representatives is \( O(n \log n) \).

Key observation: each time the representative pointer of a node is updated, the set in which it is contained at least doubles in size.

\[ \text{∴ if a node’s representative pointer is updated } k \text{ times, then } 2^k \leq n \implies k \leq \log_2 n \]

QED