1 Introduction

A Fibonacci heap is a specific data structure that supports these operations:

- \textsc{Insert}(H, k)
- \textsc{Minimum}(H)
- \textsc{ExtractMinimum}(H)
- \textsc{Union}(H_1, H_2)
- \textsc{DecreaseKey}(H, x, k)
- \textsc{Delete}(H, x)

The primary advantages of a Fibonacci heap are the \textsc{Union} and \textsc{DecreaseKey} operations, which each take $\Theta(1)$ amortized time.

2 Binary heaps?

We could implement these operations using a binary heap (which we called a “heap” earlier this semester).

<table>
<thead>
<tr>
<th>operation</th>
<th>binary heap worst-case</th>
<th>Fibonacci heap amortized</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{Insert}(H, k)</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>\textsc{Minimum}(H)</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>\textsc{ExtractMinimum}(H)</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(\lg n)$</td>
</tr>
<tr>
<td>\textsc{Union}(H_1, H_2)</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>\textsc{DecreaseKey}(H, x, k)</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>\textsc{Delete}(H, x)</td>
<td>$\Theta(\lg n)$</td>
<td>$\Theta(\lg n)$</td>
</tr>
</tbody>
</table>

3 Fibonacci heap organization

A Fibonacci heap is a collection of min-heap ordered trees.

Each node $x$ in each tree has these attributes:

- $x$.key
- $x$.parent
• .child (a pointer to any of the children)
• .left (a pointer to the left sibling)
• .right (a pointer to the right sibling)
• .degree
• .mark

The heap itself keeps this attribute:

• H.min – a pointer to the root of the tree containing the smallest key
• H.n – the number of keys in the heap

4 Fibonacci heap example

5 Potential function

We’ll analyze the Fibonacci heap data structure using the potential method for amortized analysis.

\[ \Phi(H) = t(H) + 2m(H) \]

• t(H) – number of trees in H

\(^1\)Has x lost a child since the last time it was made the child of another node?
- $m(H)$ – number of marked nodes in $H$

In an application with multiple heaps that may be merged, use the total potential:

$$\Phi(H_{1..n}) = \sum_{i=1}^{n} \Phi(H_i)$$

Is this a valid potential function?

### 6 Fibonacci heap: Simple operations

**INSERT($H, k$):**
- Create a new 1-node tree, and insert it as a sibling of $H$.min.
- Actual run time: $O(1)$
- Amortized run time: $\hat{c}_i = c + \Phi(H') - \Phi(H) = 1 + 1 = O(1)$

**UNION($H_1, H_2$):**
- Join the two linked lists of trees and select the new minimum.
- Actual run time: $O(1)$
- Amortized run time: $\hat{c}_i = c_i + \Phi(H') - (\Phi(H_1) + \Phi(H_2)) = 1 + 0 = O(1)$

### 7 Fibonacci heap: ExtractMin

**ExtractMin:**
- Remove $H$.min from the list of trees.
- Promote each of the children of $H$.min to be top-level trees.
- “Consolidate” the heap, ensuring that no two trees have the same degree.
  - Use a direct address table, keyed on the degree of the root nodes.
  - Scan through the list of trees.
  - If we find two trees with the same degree $d$, **link** them, making one tree a child of the other, to create a combined tree with degree $d + 1$. 
8 Consolidate

Let $D(n)$ denote the maximum degree of any node in a Fibonacci heap with $n$ elements. (We’ll show later that $D(n)$ is $O(\log n)$.)

Amortized run time of $\text{EXTRACTMIN}$:

$$\hat{c}_i = D(n) + t(H) + \overbrace{D(n) + 1 + 2m(H)}^{\text{actual}} - t(H) - 2m(H)$$

$$= O(D(n)) = O(\log n)$$

9 ExtractMin analysis

Let $D(n)$ denote the maximum degree of any node in a Fibonacci heap with $n$ elements. (We’ll show later that $D(n)$ is $O(\log n)$.)

Amortized run time of $\text{EXTRACTMIN}$:

$$\hat{c}_i = D(n) + t(H) + \overbrace{D(n) + 1 + 2m(H)}^{\text{actual}} - t(H) - 2m(H)$$

$$= O(D(n)) = O(\log n)$$

10 Fibonacci heap: DecreaseKey

$\text{DECREASEKEY}(x, k)$:

- If the new key is greater than the parent’s key, update $x\text{.key}$ and return.
- Otherwise, cut $x$ from its parent, and add it as a new tree. Update $H\text{.min}$ if needed.
- Use the mark attributes to promote any node that has lost two children since its last link to be a new tree. Search upward from former the parent of $x$ toward the root for marked nodes. ("cascading cut")

Idea: When a node loses its second child, promote it to the root level, to be folded into other trees on the next $\text{EXTRACTMIN}$.

Reminder: $x\text{.mark}$: Has $x$ lost a child since the last time it was made the child of another node?

11 Fibonacci heap: DecreaseKey analysis

Let $c$ denote the number of calls to $\text{CASCADINGCUT}$. Then $c - 1$ trees were created by the cascading cuts.

- Actual cost: $c$
- Change in potential:
- $t(H)$ increases by $c$.
- $m(H)$ decreases by at least $c - 2$.
- $\Phi(H') - \Phi(H) = c - 2(c - 2) = 4 - c$

- Amortized cost: $\hat{c}_i = c + 4 - c = O(1)$

12 Fibonacci heap: Delete

**DELETE**($x$):
- **DECREASE**$\text{KEY}$($x, -\infty$)
- **EXTRACT**$\text{MIN}$($H$)

Amortized Analysis: $O(1) + O(\log n) = O(\log n)$

13 Bounding the maximal degree

We still need to show that, in a Fibonacci heap with $n$ nodes, the maximum degree of any node is $O(\log n)$.

**Intuition**: The only way to get a large degree is to have many descendants.

**Lemma**: Consider a node $x$ with degree $k$. Let $y_1, \ldots, y_k$ denote the children of $x$ in the order in which they were added. Then the degree of $y_i \geq i - 2$.

**Proof**: When $y_i$ was linked to $x$, $x$ already had $y_1, \ldots, y_{i-1}$ as children, so at the time, $x.$degree $\geq i - 1$. The consolidate process only links nodes with equal degree, so we also have $y_i.$degree $\geq i - 1$. Since then, $y_i$ has lost at most one child, so now $y_i.$degree $\geq i - 2$.

14 Fibonacci numbers

Recall the Fibonacci sequence: $F_0 = 0, F_1 = 1, F_k = F_{k-1} + F_{k-2}$.

Let $\phi = \frac{1 + \sqrt{5}}{2}$.

**Lemma**: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$ (Prove by induction on $k$.)

**Lemma**: $F_{k+2} \geq \phi^k$ (Prove by induction on $k$.)
15 How small can a subtree in a Fibonacci heap be?

**Lemma:** Let $x$ be a node in a Fibonacci heap with degree $k$. Then

$$\text{size}(x) \geq F_{k+2}$$

**Proof:** Let $s_i$ denote the smallest possible size for a node of degree $i$. Induction on $k$. Base cases, for $k = 0$ and $k = 1$, are trivial. Assume the result for $1, \ldots, k - 1$ to show for $k$.

$$s_k \geq 2 + \sum_{i=2}^{k} s_{i, \text{degree}}$$

$$\geq 2 + \sum_{i=2}^{k} s_{i-2}$$

$$\geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i$$

$$= F_{k+2}$$

16 Maximum degree

Finally, if $k$ is the maximum degree in the heap, we have,

$$n \geq \text{size}(x) \geq F_{k+2} \geq \phi^k$$

which implies

$$\log_\phi n \geq k \quad \Rightarrow \quad k = O(\log n).$$