Fibonacci Heaps

**History.** [Fredman and Tarjan, 1986]
- Ingenious data structure and analysis.
- Original motivation: improve Dijkstra’s shortest path algorithm
- Repeat: extract-min
  - for all neighbors
  - if new value lower then decrease key
- Complexity reduced from $O(E \log V)$ to $O(E + V \log V)$.  

**Theorem.** Starting from empty Fibonacci heap, any sequence of $a_1$ insert, $a_2$ delete-min, and $a_3$ decrease-key operations takes $O(a_1 + a_2 \log n + a_3)$ time.

**Priority Queues Performance Cost Summary**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked List</th>
<th>Binomial Heap</th>
<th>Binomial Heap</th>
<th>Fibonacci Heap</th>
<th>Reduced Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>make-heap</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>is-empty</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>insert</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>delete-min</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>decrease-key</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>delete</td>
<td>1</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>union</td>
<td>1</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>find-min</td>
<td>1</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
</tbody>
</table>

$n = \text{number of elements in priority queue}$  

Our pictures vs. the detailed implementation

Fibonacci Heaps: Structure

- **Fibonacci heap.**
  - Set of heap-ordered trees.
  - Maintain pointer to minimum element.
  - Set of marked nodes.

Fibonacci Heaps: Structure

- **Fibonacci heap.**
  - Set of heap-ordered trees.
  - Maintain pointer to minimum element.
  - Set of marked nodes.
Fibonacci Heaps: Structure

- Fibonacci heap.
  - Set of heap-ordered trees.
  - Maintain pointer to minimum element.
  - Set of marked nodes.

Part I: intuition: insert, extract-min, decrease-key

First we go through some ideas for the operations and from there for the potential function.

Cost will turn out to depend on rank and we will make sure via extra operations that rank is bounded.

Fibonacci Heaps: Insert

Insert:
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

Insert 21

Delete Min
Fibonacci Heaps: Delete Min

Delete min.
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have the same rank.

Consolidate trees so that no two roots have the same rank.

But now we have to find the new min element and this may be expensive. Plan to pay for this step using potential function.

Potential proportional to length of root list.

Decrease Key

Case 2a. (Heap order violated)
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn’t yet lost a child), mark it.
  - Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).
Case 2a. [heap order violated]

- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
- If parent p of x is unmarked (hasn’t yet lost a child), mark it; otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).

Fibonacci Heaps: Decrease Key

Notation:
- $n$ = number of nodes in heap.
- $\text{rank}(x)$ = number of children of node x.
- $\text{rank}(H)$ = max rank of any node in heap H.
- $\text{trees}(H)$ = number of trees in heap H.
- $\text{marks}(H)$ = number of marked nodes in heap H.

Potential Function

$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

## Insert
Fibonacci Heaps: Insert

- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

- Insert 21

Fibonacci Heaps: Insert Analysis

Actual cost: $O(1)$

Change in potential: +1

Amortized cost: $O(1)$

Delete Min

Linking Operation

Linking operation. Make larger root be a child of smaller root.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.
Fibonacci Heaps: Delete Min

Delete min.
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

link 23 into 17
Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.
Fibonacci Heaps: Delete Min

Delete min.
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

Fibonacci Heaps: Delete Min Analysis

Delete min.

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]

Actual cost.
- O(rank(H)) to meld min's children into root list.
- O(rank(H)) + O(trees(H)) to update min.
- O(rank(H)) + O(trees(H)) to consolidate trees.

Change in potential.
- O(rank(H)) - trees(H)
  - trees(H') = rank(H) + 1 since no two trees have same rank.
  - \( \Delta \Phi(H) = \text{rank}(H) + 1 - \text{trees}(H) \).

Amortized cost.
- O(rank(H))

Decrease Key

Intuition for decreasing the key of node x.
- If heap-order is not violated, just decrease the key of x.
- Otherwise, cut tree rooted at x and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).
**Case 1.** [heap order not violated]
- Decrease key of \(x\).
- Change heap min pointer (if necessary).

\[
\begin{array}{c}
24 \\
29 \\
17 \\
30 \\
23 \\
7 \\
88 \\
26 \\
21 \\
52 \\
39 \\
18 \\
41 \\
38 \\
72 \\
\end{array}
\]

**Case 2a.** [heap order violated]
- Decrease key of \(x\).
- Cut tree rooted at \(x\), meld into root list, and unmark.
- If parent \(p\) of \(x\) is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut \(p\), meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

\[
\begin{array}{c}
24 \\
15 \\
17 \\
30 \\
23 \\
7 \\
88 \\
26 \\
21 \\
52 \\
39 \\
18 \\
41 \\
38 \\
72 \\
\end{array}
\]

**Mark parent**
Case 2b. [heap order violated]
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
  - Otherwise, cut p, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).
**Case 2b.** [heap order violated]
- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn’t yet lost a child), mark it.
  - Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).

---

**Analysis**

**Fibonacci Heaps: Decrease Key**

- Decrease key of x.
- Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn’t yet lost a child), mark it.
  - Otherwise, cut p, meld into root list, and unmark (and do so recursively for all ancestors that lose a second child).

**Fibonacci Heaps: Decrease Key Analysis**

- **Decrease key.**
  - Actual cost: O(c)
    - O(1) time for changing the key.
    - O(1) time for each of c cuts, plus melding into root list.
  - Change in potential: O(1) - c
    - trees(H') = trees(H) + c.
    - marks(H') = marks(H) - c + 2.
    - ΔΦ ≤ c + 2 ⋅ (-c + 2) = 4 - c.
  - Amortized cost: O(1)

---

**Fibonacci Heaps: Bounding the Rank**

**Lemma.** Fix a point in time. Let x be a node, and let y₁, ..., yₖ denote its children in the order in which they were linked to x. Then:

- rank(yᵢ) ≥ \( \begin{cases} 
0 & \text{if } i = 1 \\
 i-2 & \text{if } i > 1 
\end{cases} \)

**Pf.**
- When yᵢ was linked into x, x had at least i-1 children y₁, ..., yᵢ-1.
- Since only trees of equal rank are linked, at that time rank(yᵢ) = rank(x) ≥ i - 1.
- Since then, yᵢ has lost at most one child.
- Thus, right now rank(yᵢ) ≥ i - 2.  
  - or yᵢ would have been cut.

---

**Fibonacci Heaps: Bounding the Rank**

**Lemma.** Fix a point in time. Let x be a node, and let y₁, ..., yₖ denote its children in the order in which they were linked to x. Then:

- rank(yᵢ) ≥ \( \begin{cases} 
0 & \text{if } i = 1 \\
 i-2 & \text{if } i > 1 
\end{cases} \)

**Def.** Let \( F_k \) be smallest possible tree of rank k satisfying property.
Fibonacci Heaps: Bounding the Rank

**Lemma.** Fix a point in time. Let x be a node, and let \( y_1, \ldots, y_k \) denote its children in the order in which they were linked to x. Then:

\[
\text{rank}(y_i) \geq 0 \quad \text{if} \quad i = 1 \\
\text{rank}(y_i) \geq \text{rank}(y_{i-2}) \quad \text{if} \quad i \geq 1
\]

**Def.** Let \( F_k \) be smallest possible tree of rank k satisfying property.

**Fibonacci Numbers**

**Def.** The Fibonacci sequence is: 1, 2, 3, 5, 8, 13, 21, ...

\[
F_n = \begin{cases} 
1 & \text{if } n = 0 \\
2 & \text{if } n = 1 \\
F_{n-1} + F_{n-2} & \text{if } n \geq 2
\end{cases}
\]

**Lemma.** \( F_\phi = \phi^k \), where \( \phi = (1 + \sqrt{5}) / 2 \approx 1.618 \).

**Pf.** (by induction on k)

- **Base cases:** \( F_0 = 1 \), \( F_1 = 2 \).
- **Inductive hypotheses:** \( F_k = \phi^k \) and \( F_{k+1} = \phi^{k+1} \).

\[
F_{k+2} = F_k + F_{k+1} = \phi^k + \phi^{k+1} = \phi^k (1 + \phi) = \phi^{k+2}
\]

**Union**

**Union.** Combine two Fibonacci heaps.

**Representation.** Root lists are circular, doubly linked lists.
Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.

Actual cost. \( O(1) \)

Change in potential. 0

Amortized cost. \( O(1) \)

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]

Fibonacci Heaps: Delete

Delete node \( x \).
- decrease key of \( x \) to \(-\infty\).
- delete-min element in heap.

Amortized cost. \( O(\text{rank}(H)) \)
- \( O(1) \) amortized for decrease-key.
- \( O(\text{rank}(H)) \) amortized for delete-min.

\[ \Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H) \]

Priority Queues Performance Cost Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked List</th>
<th>Binary Heap</th>
<th>Binomial Heap</th>
<th>Fibonacci Heap</th>
<th>Relaxed Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>make-heap</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>is-empty</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>insert</td>
<td>1</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>delete-nodes</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>decrease-key</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>delete</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>union</td>
<td>1</td>
<td>( \log n )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>find-min</td>
<td>1</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
</tr>
</tbody>
</table>

\( n \) is number of elements in priority queue

amortized