Fibonacci Heaps

Lecture slides adapted from:
- Chapter 20 of *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein.
- Chapter 9 of *The Design and Analysis of Algorithms* by Dexter Kozen.

**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-min</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>log n</td>
<td>log n</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>n</td>
<td>log n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>find-min</td>
<td>n</td>
<td>1</td>
<td>log n</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( n \) = number of elements in priority queue  
\( \dagger \) amortized

**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.

**History.** [Fredman and Tarjan, 1986]
- Ingenious data structure and analysis.
- Original motivation: improve Dijkstra’s shortest path algorithm from \( O(\mathcal{E} \log V) \) to \( O(\mathcal{E} + V \log V) \).

**Basic idea.**
- Similar to binomial heaps, but less rigid structure.
- Binomial heap: *eagerly* consolidate trees after each insert.

- Fibonacci heap: *lazily* defer consolidation until next delete-min.

**Hopeless challenge.** \( O(1) \) insert, delete-min and decrease-key. Why?
**Fibonacci Heaps: Structure**

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

```
17  24  23  7
30  26  46
18  52  41  44
19
```

**Fibonacci Heaps: Notation**

**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 \( H \).
Fibonacci Heaps: Potential Function

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

potential of heap \( H \)

Insert

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

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

Fibonacci Heaps: Insert

insert 21

Heap \( H \)
Fibonacci Heaps: Insert Analysis

Actual cost.  $O(1)$

Change in potential.  $+1$

Amortized cost.  $O(1)$

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

potential of heap $H$

---

Delete Min

---

Linking Operation

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

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

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

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

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

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

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

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

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

Amortized cost. \( O(\text{rank}(H)) \)
Fibonacci Heaps: Delete Min Analysis

Q. Is amortized cost of $O(\text{rank}(H))$ good?

A. Yes, if only insert and delete-min operations.
   - In this case, all trees are binomial trees.
   - This implies $\text{rank}(H) \leq \lg n$.

A. Yes, we’ll implement decrease-key so that $\text{rank}(H) = O(\log n)$.

Fibonacci Heaps: 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).
Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]
- Decrease key of $x$.
- Change heap min pointer (if necessary).

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 2. [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

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 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).
**Fibonacci Heaps: Decrease Key**

**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).

![Diagram](image1)

*decrease-key of $x$ from 35 to 5*

**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).

![Diagram](image2)

*decrease-key of $x$ from 35 to 5*
Fibonacci Heaps: Decrease Key Analysis

Decrease-key.

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

potential function

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. \( O(1) \cdot c \)

- \( \text{trees}(H) = \text{trees}(H) + c \).
- \( \text{marks}(H) \leq \text{marks}(H) - c + 2 \).
- \( \Delta \Phi \leq c + 2 \cdot (c + 2) = 4 \cdot c \).

Amortized cost. \( O(1) \)

Analysis Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Amortized Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Delete-min.</td>
<td>( O(\text{rank}(H)) )</td>
</tr>
<tr>
<td>Decrease-key.</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

† amortized

Key lemma. \( \text{rank}(H) = O(\log n) \).

number of nodes is exponential in rank

Analysis

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:

\[
\text{rank}(y_i) = \begin{cases} 
0 & \text{if } i = 1 \\
2i - 2 & \text{if } i \geq 1
\end{cases}
\]

Pf.

- When \( y_i \) was linked into \( x \), \( x \) had at least \( i - 1 \) children \( y_1, \ldots, y_{i-1} \).
- Since only trees of equal rank are linked, at that time \( \text{rank}(y_i) = \text{rank}(x) = i - 1 \).
- Since then, \( y_i \) has lost at most one child.
- Thus, right now \( \text{rank}(y_i) \geq i - 2 \). 

or \( y_i \) would have been cut
Fibonacci Heaps: Bounding the Rank

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

\[
\text{rank}(y_i) = \begin{cases} 
0 & \text{if } i = 1 \\
|y| - 2 & \text{if } i \geq 1
\end{cases}
\]

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

**Fibonacci Numbers**

**Fibonacci Numbers**

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

\[
\text{rank}(y_i) = \begin{cases} 
0 & \text{if } i = 1 \\
|y| - 2 & \text{if } i \geq 1
\end{cases}
\]

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

**Fibonacci fact.** \( F_k \geq \phi^k \), where \( \phi = (1 + \sqrt{5}) / 2 \approx 1.618 \).

**Corollary.** \( \text{rank}(H) \leq \log_{\phi} n \). golden ratio
**Fibonacci Numbers: Exponential Growth**

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

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

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

**Pf.** [by induction on \(k\)]

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

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

---

**Fibonacci Numbers and Nature**

**Union.** Combine two Fibonacci heaps.

**Representation.** Root lists are circular, doubly linked lists.

---

**Fibonacci Heaps: Union**

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

**Union.** Combine two Fibonacci heaps.

**Representation.** Root lists are circular, doubly linked lists.

![Diagram of Fibonacci Heaps Union]

**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*.

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