Fibonacci Heaps

COMP 160 – Algorithms – Spring 2012 - Tufts University

Original Slides from Kevin Wayne, Princeton
http://www.cs.princeton.edu/courses/archive/spring07/cos423/lectures/fibonacci-heap.ppt
Slightly adapted by Roni Khardon 4/2012

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>Binary 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>$n$</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>$n$</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>1</td>
<td>1</td>
</tr>
<tr>
<td>$n$ = number of elements in priority queue</td>
<td>amortized</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $O(E + V \log V)$.

Fibonacci Heaps: Structure

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

Our pictures vs. the detailed implementation

Heap $H_{min}$

(a) $23 \rightarrow 7$
(b) $23 \rightarrow 7$

Heap H

Heaps $H$
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

Insert

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

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 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.

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 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: 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**

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

---

**Insert**
Fibonacci Heaps: 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 Analysis

- Actual cost: $O(1)$
- Change in potential: +1
- Amortized cost: $O(1)$

Delete Min

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

Linking Operation

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

Fibonacci Heaps: 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 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 17 into 7

link 24 into 7

link 41 into 18
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.
\[ \Phi(H) = trees(H) + 2 \cdot \text{marks}(H) \]

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

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

Amortized cost: \( O(\text{rank}(H)) \)

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).
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 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
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).

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. \( O(1) \cdot c \)
- \( \text{trees}(H') = \text{trees}(H) + c. \)
- \( \text{marks}(H') = \text{marks}(H) \cdot c + 2. \)
- \( \Delta \Phi = c + 2 - (c + 2) = 0. \)

Amortized cost. \( O(1) \)

Analysis Summary

- Insert. \( O(1) \)
- Delete-min. \( O(\text{rank}(H)) \)
- Decrease-key. \( O(1) \)

Key lemma. \( \text{rank}(H) = O(\log n) \)
(number of nodes is exponential in rank)

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:

\[ \text{rank}(y_i) = \begin{cases} 0 & \text{if } i = 0 \\ c & \text{if } i > 0 \end{cases} \]

Proof.
- 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 \( \text{rank}(y_i) = \text{rank}(x) \cdot i - 1. \)
- Since then, \( y_i \) has lost at most one child.
- Thus, right now \( \text{rank}(y_i) = i - 2. \)

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:

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

Def. Let $F_k$ be smallest possible tree of rank $k$ satisfying property.

Fibonacci fact. $F_k \geq \phi^k$, where $\phi = \frac{1 + \sqrt{5}}{2} \approx 1.618.$

Corollary. $\text{rank}(H) \leq \log_\phi n.$

Fibonacci Heaps: 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.

Fibonacci Heaps: Delete

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

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

Fibonacci Heaps: Union

Actual cost. \(O(1)\)

Change in potential. 0

Amortized cost. \(O(1)\)

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

Potential function

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>Relaxation 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>log n</td>
<td>log n</td>
<td>log n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>decrease-key</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>\text{rank}(H)</td>
<td>1</td>
</tr>
<tr>
<td>delete</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>union</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>find-min</td>
<td>0</td>
<td>log n</td>
<td>log n</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

n = number of elements in priority queue

\(\text{amortized}\)