Binary and Binomial Heaps

These lecture slides are adapted from CLRS, Chapters 6, 19.

Priority Queues

Supports the following operations.
- Insert element \( x \).
- Return min element.
- Return and delete minimum element.
- Decrease key of element \( x \) to \( k \).

Applications.
- Dijkstra’s shortest path algorithm.
- Prim’s MST algorithm.
- Event-driven simulation.
- Huffman encoding.
- Heapsort.
- ...
**Binary Heap: Definition**

- **Binary heap.**
  - Almost complete binary tree.
    - filled on all levels, except last, where filled from left to right
  - Min-heap ordered.
    - every child greater than (or equal to) parent

**Binary Heaps: Array Implementation**

- Implementing binary heaps.
  - Use an array: no need for explicit parent or child pointers.
    - Parent(i) = ⌊i/2⌋
    - Left(i) = 2i
    - Right(i) = 2i + 1

**Binary Heap: Properties**

- **Properties.**
  - Min element is in root.
  - Heap with N elements has height = ⌊log₂ N⌋.

**Binary Heap: Insertion**

- Insert element x into heap.
  - Insert into next available slot.
  - Bubble up until it’s heap ordered.

  - Peter principle: nodes rise to level of incompetence
**Binary Heap: Insertion**

Insert element x into heap.
- Insert into next available slot.
- Bubble up until it's heap ordered.
  - Peter principle: nodes rise to level of incompetence

**Binary Heap: Insertion**

Insert element x into heap.
- Insert into next available slot.
- Bubble up until it's heap ordered.
  - Peter principle: nodes rise to level of incompetence

**Binary Heap: Insertion**

Insert element x into heap.
- Insert into next available slot.
- Bubble up until it's heap ordered.
  - Peter principle: nodes rise to level of incompetence
  - O(log N) operations.

**Binary Heap: Decrease Key**

Decrease key of element x to k.
- Bubble up until it's heap ordered.
  - O(log N) operations.
Delete minimum element from heap.
- Exchange root with rightmost leaf.
- Bubble root down until it’s heap ordered.
  - power struggle principle: better subordinate is promoted
Binary Heap: Delete Min

Delete minimum element from heap.
- Exchange root with rightmost leaf.
- Bubble root down until it’s heap ordered.
  - power struggle principle: better subordinate is promoted
- O(log N) operations.

Binary Heap: Heapsort

Heapsort.
- Insert N items into binary heap.
- Perform N delete-min operations.
- O(N log N) sort.
- No extra storage.

Binary Heap: Union

Union.
- Combine two binary heaps H₁ and H₂ into a single heap.
- No easy solution.
  - $\Omega(N)$ operations apparently required
- Can support fast union with fancier heaps.

Priority Queues

<table>
<thead>
<tr>
<th>Operation</th>
<th>Heaps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linked List</td>
</tr>
<tr>
<td>make-heap</td>
<td>1</td>
</tr>
<tr>
<td>insert</td>
<td>1</td>
</tr>
<tr>
<td>find-min</td>
<td>N</td>
</tr>
<tr>
<td>delete-min</td>
<td>N</td>
</tr>
<tr>
<td>union</td>
<td>1</td>
</tr>
<tr>
<td>decrease-key</td>
<td>1</td>
</tr>
<tr>
<td>delete</td>
<td>N</td>
</tr>
<tr>
<td>is-empty</td>
<td>1</td>
</tr>
</tbody>
</table>
Binomial Tree

• Recursive definition:
  \[ B_0 \]
  \[ B_{k-1} \]
  \[ B_k \]
  \[ B_{k-1} \]

Useful properties of order \( k \) binomial tree \( B_k \):

• Number of nodes = \( 2^k \).
• Height = \( k \).
• Degree of root = \( k \).
• Deleting root yields binomial trees \( B_{k-1}, \ldots, B_0 \).

Proof.

• By induction on \( k \).

Binomial Heap


• Sequence of binomial trees that satisfy binomial heap property.
  – each tree is min-heap ordered
  – 0 or 1 binomial tree of order \( k \)
Binomial Heap: Implementation

- Represent trees using left-child, right sibling pointers.
  - three links per node (parent, left, right)
- Roots of trees connected with singly linked list.
  - degrees of trees strictly decreasing from left to right

Binomial Heap: Properties

- Min key contained in root of $B_0$, $B_1$, ..., $B_k$.
- Contains binomial tree $B_i$ iff $b_i = 1$ where $b_n b_{n-1} b_{n-2} ... b_0$ is binary representation of $N$.
- At most $\lceil \log_2 N \rceil + 1$ binomial trees.
- Height $\leq \lceil \log_2 N \rceil$.

Binomial Heap: Union

- Create heap $H$ that is union of heaps $H'$ and $H''$.
  - "Mergeable heaps."
  - Easy if $H'$ and $H''$ are each order $k$ binomial trees.
    - connect roots of $H'$ and $H''$
    - choose smaller key to be root of $H$

Binomial Heap: Power-of-2 Heap

- Leftist power-of-2 heap

N = 19
# trees = 3
height = 4
binary = 10011

19 + 7 = 26

1 1 0 0 1 1

1 1 0 1 0
Binomial Heap: Union

+ 

+ 

+ 

+ 

12
18
**Binomial Heap: Union**

Create heap $H$ that is union of heaps $H'$ and $H''$.
- Analogous to binary addition.

**Running time. $O(\log N)$**
- Proportional to number of trees in root lists $\leq 2(\lceil \log_2 N \rceil + 1)$.

**Binomial Heap: Delete Min**

Delete node with minimum key in binomial heap $H$.
- Find root $x$ with min key in root list of $H$, and delete
  - $H' \leftarrow$ broken binomial trees
  - $H \leftarrow \text{Union}(H', H)$

**Running time. $O(\log N)$**
Binomial Heap: Delete Min

Delete node with minimum key in binomial heap H.
- Find root x with min key in root list of H, and delete
- \( H' \leftarrow \) broken binomial trees
- \( H \leftarrow \) Union\( (H', H) \)

Running time. \( O(\log N) \)

Binomial Heap: Insert

Insert a new node x into binomial heap H.
- \( H' \leftarrow \) MakeHeap\( (x) \)
- \( H \leftarrow \) Union\( (H', H) \)

Running time. \( O(\log N) \)

Binomial Heap: Decrease Key

Decrease key of node x in binomial heap H.
- Suppose x is in binomial tree \( B_k \).
- Bubble node x up the tree if x is too small.

Running time. \( O(\log N) \)

- Proportional to depth of node x \( \leq \lceil \log_2 N \rceil \).

Binomial Heap: Delete

Delete node x in binomial heap H.
- Decrease key of x to \(-\infty\).
- Delete min.

Running time. \( O(\log N) \)

Binomial Heap: Sequence of Inserts

Insert a new node x into binomial heap H.

- If \( N = \ldots .0 \), then only 1 steps.
- If \( N = \ldots .01 \), then only 2 steps.
- If \( N = \ldots .011 \), then only 3 steps.
- If \( N = \ldots .0111 \), then only 4 steps.

Inserting 1 item can take \( \Omega(\log N) \) time.

- If \( N = 11 \ldots 111 \), then \( \log_2 N \) steps.

But, inserting sequence of N items takes \( O(N) \) time!

- \( (N/2)(1) + (N/4)(2) + (N/8)(3) + \ldots \leq 2N \)
- Amortized analysis.
- Basis for getting most operations down to constant time.

\[
\sum_{n=1}^{N} \frac{n}{2^n} \leq \frac{2 - N}{2^{N-1}} \]

---

**Priority Queues**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Linked List</th>
<th>Binary</th>
<th>Binomial</th>
<th>Fibonacci*</th>
<th>Relaxed</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>insert</td>
<td>1</td>
<td>( \log 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>
<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>union</td>
<td>1</td>
<td>N</td>
<td>( \log N )</td>
<td>( \log N )</td>
<td>( \log N )</td>
</tr>
<tr>
<td>decrease-key</td>
<td>1</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>is-empty</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

Just did this