6 Priority Queues

6.1 Model and simple implementations

Reading: MAW 6.1 and 6.2

- A data structure $H$ that supports $\text{Insert}(H, x)$ and $\text{DeleteMin}(H)$.
- Implementations:

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Insert</th>
<th>DeleteMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted array</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>BST</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>AVL</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

Question: Can we use an array for data storage to achieve $O(\log n)$ worst-case time for both operations?

6.2 Heap: An implementation of priority queues

Reading: MAW 6.3

- What is a heap?
  - A left-complete binary tree;
  - Each node has a number (key); and
  - Heap-order property (for min heaps): Parent $\leq$ children.

Example: The left tree in Figure 6.5 on page 215.

- How is a heap implemented?
  Array: top-down, level-by-level, and left-right.

For example, the heap in our previous example can be represented by array $H$: 13, 21, 16, 24, 31, 19, 68, 65, 26, 32. In general, for $H[i]$, its left child is $H[2i]$, its right child is $H[2i+1]$, and its parent is $H[\lfloor i/2 \rfloor]$.

- Height of a heap with $n$ nodes:
  Height $h = \lfloor \log n \rfloor + 1 = O(\log n)$.

For example,

<table>
<thead>
<tr>
<th>$n$</th>
<th>$h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 = $\lfloor \log 1 \rfloor + 1$</td>
</tr>
<tr>
<td>2</td>
<td>2 = $\lfloor \log 2 \rfloor + 1$</td>
</tr>
<tr>
<td>3</td>
<td>2 = $\lfloor \log 3 \rfloor + 1$</td>
</tr>
<tr>
<td>4</td>
<td>3 = $\lfloor \log 4 \rfloor + 1$</td>
</tr>
</tbody>
</table>

- Rebuilding a heap:
  Assumptions:
  - The left subtree of $H[i]$ is a heap;
  - The right subtree of $H[i]$ is a heap; but
  - The heap-order property is not satisfied at $H[i]$, i.e., $H[i] > H[2i]$ or $H[i] > H[2i+1]$.

Question: How can we rebuild the tree rooted at $H[i]$ into a heap?

Example: 10, 5, 6, 7, 8 $\rightarrow$ 5, 7, 6, 10, 8.
RebuildHeap(H, i)
   if H[i] is not a leaf
      if H[i] > H[2i] or H[i] > H[2i+1]
         if H[2i] < H[2i+1]
            j = 2i
         else j = 2i+1
         swap H[i] and H[j]
      RebuildHeap(H, j)

Worst-case time: $O(\log n)$.

• Insert $(H, x)$:
  Example: Insert 14 to 13, 21, 16, 24, 31, 19, 68, 65, 26, 32.

  Insert $(H, x)$
  i = size + 1
  H[i] = x
  parent = i / 2
  while parent > 0 and H[i] < H[parent]
      swap H[i] and H[parent]
      i = parent
      parent = i / 2
  size ++

Worst-case time: $O(\log n)$.

• DeleteMin $(H)$:
  By the heap property of a min heap, the minimum is at the root of the heap, which is $H[1]$.
  Example: Delete the minimum from 13, 14, 16, 24, 21, 19, 68, 65, 26, 32, 31.

  DeleteMin $(H)$
  min = H[1]
  H[1] = H[size]
  size --
  RebuildHeap(H, 1)
  return min

Worst-case time: $O(\log n)$.

• Initialization:
  Example: Input array 150, 80, 40, 30, 10, 70, 110, 100, 20, 90, 60, 50, 120, 140, 130.

  Initialization $(H)$
  for i = size / 2 to 1
     RebuildHeap(H, i)

Worst-case time: $O(n)$. (See page 223 for proof)

Note: Another method for initialization is to insert the keys one by one into an initially empty heap. What is the time complexity?
6.3 Applications

Reading: MAW 6.4

• Operating system: Scheduling jobs on processors to run based on priorities.
  When a new job comes, if the processors are all busy, the job is inserted into a waiting queue based on its priority.
  When a processor becomes idle, if the waiting queue is not empty, the job with the highest priority is removed from
  the queue to start execution on the processor.

• The selection problem: Finding the kth largest number among n.
  – Algorithm 1: Sort and then select.
    Time complexity: \( O(n^2) \) or \( O(n \log n) \).
  – Algorithm 2: Sort \( A[1..k] \) into decreasing order. We call the sorted list \( S \). \( S[k] \) is therefore the smallest of the k
    numbers.) For \( i = k + 1..n \), if \( A[i] > S[k] \), \( S[k] \) is removed and \( A[i] \) is inserted into the correct position in \( S \). At
    the end, return \( S[k] \) as the kth largest.
    Time complexity: \( O(k \log k + (n - k)k) = O(nk) \).
  – Algorithm 3: Initialize into a max heap in \( O(n) \) time and then perform \( \text{DeleteMax} \) \( k \) times.
    Time complexity: \( O(n + k \log n) \).
  – Algorithm 4: Similar idea to Algorithm 2, but maintain \( S \) as a min heap instead of a sorted list. \( A[1..k] \) is first
    initialized into a heap \( S \). For \( i = k + 1..n \), if \( A[i] > S[1] \), do \( \text{DeleteMin}(H) \) and \( \text{Insert}(S,A[i]) \). At the end, return
    \( S[1] \).
    Time complexity: \( O(k + (n - k) \log k) = O(n \log k) \).