Heapsort and Priority Queues

CSE 680

Suggested reading: Chapter 6

1 Heaps

- **Complete binary tree:** a binary tree with exactly 2^h leaves, where h is the height of the tree.
- **Nearly complete binary tree:** a binary tree obtained from a complete binary tree by deleting one or more of its "rightmost" leaves.
- A **heap** is an array $A[1..n]$ or its subarray $A[1..m]$, $m < n$, viewed as a complete or nearly complete binary tree.
- heap-size$[A]$: number of elements in the heap.
- parent$(i) = \lfloor i/2 \rfloor$
- left$(i) = 2i$
- right$(i) = 2i + 1$
- **Max-heap:** $A[i] \leq A[\text{parent}(i)]$, for all $2 \leq i \leq$ heap-size$[A]$
- **Min-heap:** $A[i] \geq A[\text{parent}(i)]$, for all $2 \leq i \leq$ heap-size$[A]$

2 Max-Heapify

Problem: In a heap, suppose the subtrees rooted at left(i) and right(i) are max-heaps. We want to make the tree rooted at i a max-heap.

procedure MaxHeapify(A, i)

$l \leftarrow$ left(i)

$r \leftarrow$ right(i)

$\mathit{size} \leftarrow$ heap-size[A]

$\mathit{largest} \leftarrow i$

if ($l \leq \mathit{size}$) **and** ($A[l] > A[\mathit{largest}]$) **then** $\mathit{largest} \leftarrow l$

if ($r \leq \mathit{size}$) **and** ($A[r] > A[\mathit{largest}]$) **then** $\mathit{largest} \leftarrow r$

if ($\mathit{largest} \neq i$) **then**

exchange $A[i] \leftrightarrow A[\mathit{largest}]$

MaxHeapify($A, \mathit{largest}$)

Running time: $O(\log n)$.

3 Build a heap

Problem: Convert an array $A[1..n]$ of integers into a max-heap.

procedure Build-Max-Heap($A[1..n]$)

  heap-size$[A] \leftarrow n$

  for $i \leftarrow$ parent(n) **downto** 1 **do**

    MaxHeapify(A, i)

Running time: $O(n \log n)$.

4 Heapsort

Problem: Sort an array $A[1..n]$.

procedure Heapsort($A[1..n]$)

  Build-Max-Heap($A[1..n]$)

  for $i \leftarrow n$ **downto** 2 **do**

    exchange $A[1] \leftrightarrow A[i]$

    decrease heap-size[A] by 1

    MaxHeapify($A, 1$)

Running time: $O(n \log n)$.

5 Priority Queues

- A data structure for maintaining a set S of elements with the following operations:
  - Maximum(S): returns the element in S with the largest key.
  - Insert(S, x): inserts an element x into S.
  - Extract-Max(S): removes and returns the element in S with the largest key.
- The following function is useful in many applications of priority queue.
  - Increase-Key(S, i, k): increases i's key to k.
- Basic idea: use a max-heap.
- All the above operations can be done in $O(\log n)$ time, where n is the size of S.

procedure Increase-Key($A[1..n], i, k$)

/* increase $A[i]$ to k */

if ($k < A[i]$) **then error** "new key smaller than current key"

$A[i] \leftarrow k$

while ($i > 1$) **and** ($A[\text{parent}(i)] < A[i]$) **do**

exchange $A[i] \leftrightarrow A[\text{parent}(i)]$

$i \leftarrow \text{parent}(i)$