LECTURES 20-21
Priority Queues and Binary Heaps

Adam Smith

S. Raskhodnikova and A. Smith. Based on slides by C. Leiserson and E. Demaine.
Trees

- Rooted Tree: collection of nodes and edges
  - Edges go down from root (from parents to children)
  - No two paths to the same node
  - Sometimes, children have “names” (e.g. left, right, middle, etc)
Where are trees useful?

- Some important applications
  - Parse trees (compilers)
  - Heaps
  - Search trees
  - String search
  - Game trees (e.g. chess program)

- E.g. parse tree for $3 \times (4+2) \times (2^6) \times (8/3)$
Priority Queue ADT

• Dynamic set of pairs (key, data), called elements

• Supports operations:
  - MakeNewPQ()
  - Insert(S,x) where S is a PQ and x is a (key,data) pair
  - Extract-Max(S) removes and returns the element with the highest key value (or one of them if several have the same value)

• Example: managing jobs on a processor, want to execute job in queue with the highest priority

• Can also get a “min” version, that supports Extract-Min instead of Extract-Max

• Sometimes support additional operations like Delete, Increase-Key, Decrease-Key, etc.
Heaps: Tree Structure

- Data Structure that implements Priority Queue
  - Conceptually: binary tree
  - In memory: (often) stored in an array
- Recall: complete binary tree
  - all leaves at the same level, and
  - every internal node has exactly 2 children
- Heaps are nearly complete binary trees
  - Every level except possibly the bottom one is full
  - Bottom layer is filled left to right
Height

• Heap Height =
  - length of longest simple path from root to some leaf
  - e.g. a one-node heap has height 0,
    a two- or three-node heap has height 1, ...

• Exercises:
  - What is are max. and min. number of elements in a heap of height $h$?
    - ans: min = $2^h$, max = $2^{h+1} - 1$
  - What is height as a function of number of nodes $n$?
    - ans: floor( log(n) )
Array representation

- Instead of dynamically allocated tree, heaps often represented in a fixed-size array
  - smaller constants, works well in hardware.
- Idea: store elements of tree layer by layer, top to bottom, left to right
- Navigate tree by calculating positions of neighboring nodes:
  - Left(i) := 2i
  - Right(i) := 2i+1
  - Parent(i) := floor(i/2)
- Example: [20 15 8 10 7 5 6]
Review Questions

• Is the following a valid max-heap?
  ➢ 20 10 4 9 6 3 2 8 7 5 12 1
  ➢ (If not, repair it to make it valid)

• Is an array sorted in decreasing order a valid max-heap?
Local Repair: Heapify “Down”

- Two important “local repair operations”: 
  - Heapify “Down”, Heapify “Up”
- Suppose we start from a valid heap and decrease the key of node i, so
  - it is still smaller than parent
  - the two subtrees rooted at i remain valid
- How do we rearrange nodes to make the heap valid again?
  - Idea: let new, smaller key “sink” to correct position
    - Find index with largest key among \{i, \text{Left}(i), \text{Right}(i)\}
    - If i \neq \text{largest}, EXCHANGE i with largest and recurse on largest
Local Repair: Heapify “Down”

- Pseudocode in book

- Running time: $O(\text{height}) = O(\log n)$
  - Tight in worst case

- Correctness: by induction on height
Exercise: what does Max-Heapify do on the following input?

- 2.5 10 4 9 6 3 2 8 7 5 0 1

This gives us our first PQ operation:

**Extract-Max(A)**

1. tmp := A[1]
3. heap-size := heap-size-1
4. MAX-HEAPIFY-DOWN(A, 1)
5. return tmp
Local Repair: Heapify “Up”

- Two important “local repair operations”
  - Heapify “Down”
  - Heapify “Up”
- Suppose we start from a valid heap and increase the key of node $i$, so
  - it is still larger than both children, but
  - might be larger than parent
- How do we rearrange nodes to make the heap valid again?
  - Idea: let new, larger key “float” to right position
    - If $A[i] > A[\text{Parent}(i)]$, EXCHANGE $i$ with parent and recurse on parent
  - Pseudocode in CLRS Chap 6.2: “MAX-HEAPIFY”
Local Repair: Heapify “Up”

- Pseudocode in book (see “Increase-key”)

- Running time: $O(\log n)$
  - tight in worst case

- Correctness: by induction on height
MAX-HEAPIFY(-UP)

- Exercise: what does Max-Heapify do on the following input?
  - 20 10 4 9 6 3 2 8 7 21 0 1
- This gives us our second PQ operation:
- Insert(A,x)
  1. A[heap-size+1] := x
  2. heap-size := heap-size+1
  3. MAX-HEAPIFY-UP(A,heap-size)
Other PQ operations

- **Max(S):** return max element without extracting it

- **Increase-Key(S,i,new-key)**
- **Decrease-Key(S,i,new-key)**
  - Both of these require knowing position of desired element
  - Can be taken care by “augmenting” using a dictionary data structure, such as hash table

- **Delete(S,i)**
  - Delete the element in position i
Heap Sort

- Heaps give an easy $O(n \log n)$ sorting algorithm:
  - For $i=2$ to $n$
    - Insert($A, A[i]$)
  - For $i=n$ downto 3
    - $A[i] := \text{Extract-Max}(A)$

- There is a faster way ($O(n)$ time) to build a heap from an unsorted array.
Search

• Can we search for an element in a heap in sublinear (that is, o(n)) time, in the worst case?

• How would you prove that such a search algorithm does not exist?