Heaps and Sorting
Our First Data Structure

S.V.N. (vishy) Vishwanathan

University of California, Santa Cruz
vishy@ucsc.edu

February 11, 2016
Outline

1. Heaps
2. Heapsort
3. Priority Queue
Heaps

(a)

(b)
Max Heaps

Each node satisfies a max heap property:

\[ A[\text{PARENT}(i)] \geq A[i] \]
Encoding a Heap in an Array

- Key Observation: A heap is a nearly complete binary tree

\[
PARENT(i) = \lfloor i/2 \rfloor \\
LEFT(i) = 2i \\
RIGHT(i) = 2i + 1
\]

- Also need to distinguish between \( A.length \) and \( A.heap-size \)

\[0 \leq A.heap-size \leq A.length\]

- Think: in this representation, where do the leaf nodes reside in the array?
Encoding a Heap in an Array

- Key Observation: A heap is a nearly complete binary tree

  \[
  \text{PARENT}(i) = \left\lfloor \frac{i}{2} \right\rfloor \\
  \text{LEFT}(i) = 2i \\
  \text{RIGHT}(i) = 2i + 1
  \]

- Also need to distinguish between \textit{A.length} and \textit{A.heap-size}

  \[\ 0 \leq \text{A.heap-size} \leq \text{A.length}\]

- Think: in this representation, where do the leaf nodes reside in the array?
Max Heapify

- Suppose the binary trees rooted at $\text{LEFT}(i)$ and $\text{RIGHT}(i)$ are max heaps
- We want to merge the two heaps and add $A[i]$ as their root
Max Heapify

(a)

(b)

(c)
Max Heapify

MAX-HEAPIFY(A, i)

1   \( l = \text{LEFT}(i) \)
2   \( r = \text{RIGHT}(i) \)
3   \textbf{if} \ l \leq A.\text{heap-size} \text{ and } A[l] > A[i] \\
4      largest = l
5   \textbf{else} largest = i
6   \textbf{if} \ r \leq A.\text{heap-size} \text{ and } A[r] > A[largest] \\
7      largest = r
8   \textbf{if} \ largest \neq i \\
9      exchange A[i] with A[largest]
10  \text{MAX-HEAPIFY}(A, largest)
Runtime

\[ T(n) \leq T(2n/3) + \Theta(1) \]

Therefore \[ T(n) = O(\log n) \]
Building a Heap

- Remember: Leaves are at position $A[\lfloor n/2 \rfloor + 1 .. n]$
- Simply go through the first half of the array and run \texttt{MAX-HEAPIFY} on each element

\textbf{BUILD-MAX-HEAP}(A)

1. $A.\text{heap-size} = A.\text{length}$
2. \textbf{for} $i = \lfloor A.\text{length}/2 \rfloor$ \textbf{downto} 1
3. \texttt{MAX-HEAPIFY}(A, i)
Building a Heap

![Diagram of heap building process](image-url)
Loop Invariant

At the start of each iteration of the for loop of lines 2-3, each node $i + 1, i + 1, \ldots, n$ is the root of a max-heap.
Initialization

- At the beginning $i = \lceil n/2 \rceil$
- All nodes $\lceil n/2 \rceil + 1, \lceil n/2 \rceil + 2, \ldots, n$ are leaf nodes
- Therefore each node is trivially the root of a one-node max-heap
Maintenance

- All children of node $i$ are numbered higher than $i$
- By loop invariant, both left and right sub-trees of $i$ are max-heaps
- `MAX-HEAPIFY` will make a valid max-heap, with element $i$ as its root
- Does not touch nodes which are not subtrees of $i$
Termination

- Each node $1, 2, \ldots, n$ is the root of a max-heap
- In particular, node 1 is the root of a max-heap
Runtime: Loose bound

- At most \( \frac{n}{2} \) calls to MAX-HEAPIFY
- Each call takes \( O(\log n) \) time
- So total time is \( O(n \log n) \)
Heaps

Runtime: Tight bound

- Height is bounded by $\lceil \log n \rceil$
- Maximum number of nodes at any height $h$ is at most $\lceil n/2^{h+1} \rceil$

\[
\sum_{h=1}^{\lceil \log n \rceil} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O \left( n \sum_{h=1}^{\lceil \log n \rceil} \frac{h}{2^{h+1}} \right) \leq 2 = O(n)
\]
Outline

1 Heaps

2 Heapsort

3 Priority Queue
Heapsort

Basic Idea
Pseudocode

HEAPSORT($A$)
1. BUILD-MAX-HEAP($A$)
2. for $i = A.length$ downto 2
4. $A.heap-size = A.heap-size - 1$
5. MAX-HEAPIFY($A, 1$)
Time complexity

- **BUILD-MAX-HEAP** takes $O(n)$ time
- There are $O(n)$ calls to **MAX-HEAPIFY**
- Each call takes at most $O(\log n)$ time
- Total time is therefore $O(n \log n)$
Outline

1. Heaps
2. Heapsort
3. Priority Queue
Suppose there is only one doctor at the hospital ER and a steady stream of patients coming in.

- Alice has a cold
- Bob has chopped off a finger when working on his tablesaw
- Cathy has cut herself with a kitchen knife
- Dyan fell when biking and broke his collar bone
- Esther came in with a fever, but now she has developed serious concussions and her heart rate is falling rapidly
- . . .

**Question:** Whom should the doctor attend to?
Priority Queue

- Items arrive in a steady stream
- Each item has a priority
- Priorities of items can change (usually they increase)
- You want to pick the item with the top priority
- Delete the top priority item from the queue

How to implement a data structure that performs these operations?
Supported Operations

- \textsc{insert}(S, x, k)
- \textsc{maximum}(S)
- \textsc{extract-max}(S)
- \textsc{increase-key}(S, x, k)
Let’s Try an Array
Let’s Try a Linked List
Max-Priority Heap

- \textsc{Insert}(S, x, k) in \(O(\log n)\)
- \textsc{Maximum}(S) in \(O(1)\)
- \textsc{Extract-Max}(S) in \(O(\log n)\)
- \textsc{Increase-Key}(S, x, k) in \(O(\log n)\)
Maximum

1. \textbf{return} \ A[1]

\textsc{heap-maximum}(A)
Extract Maximum

HEAP-EXTRACT-MAXIMUM(A)
1 if A.heap-size < 1
2    error “heap underflow”
3   max = A[1]
4   A.heap-size = A.heap-size − 1
5   MAX-HEAPIFY(A, 1)
6 return max
HEAP-INCREASE-KEY($A$, $i$, $key$)

1. if $key < A[i]$
2.   error “new key is smaller than current key”
3. $A[i] = key$
4. while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
5.   exchange $A[i]$ with $A[\text{PARENT}(i)]$
6. $i = \text{PARENT}(i)$
Illustration

(a) 16
   /   
  14   10
 /   /
8 7 9 3
 /  \
2 4 1

(b) 16
   /   
  14   10
 /   /
8 7 9 3
 /  \
2 15 1

(c) 16
   /   
  14   10
 /   /
15 7 9 3
 /  \
2 8 1

(d) 16
   /   
  14   10
 /   /
15 7 9 3
 /  \
2 8 1
Insert

HEAP-INSERT\((A, key)\)

1. \(A.\text{heap-size} = A.\text{heap-size} + 1\)
2. \(A[A.\text{heap-size}] = -\infty\)
3. HEAP-INCREASE-KEY\((A, A.\text{heap-size}, key)\)
Questions?