Data structures lab – week 7

Welcome back!

Wake-up quiz – order statistics

- Finding the *i*th smallest key in a standard red-black tree takes:

  a) O(1)

  b) O(lg n)

  c) O(n)

  d) O(n lg n)

Wake-up quiz – order statistics

- Finding the *i*th smallest key in a standard red-black tree takes:

  a) O(1)

  b) O(lg n)

  c) O(n) ← c is correct!

  d) O(n lg n)

  - The tree is “standard” – not augmented with size of the sub trees.
    - Sorry, I might have tricked you but this IS a wake-up quiz :-)

Week 6 recap

- Augmenting data structures
- Order statistics
- Assignment 3, part 2
  - Hope you're all near completion

Week 6 class evaluation

- Comments (slightly edited):
  - ”Is there a possibility to have the class meet in Deschutes 100? At least once.”
    - Probably not
      - Only 16 machines or so
      - Room used by other students
      - Go to HELP sessions.
    - We talked about it at the beginning of the term. “How a lab lecture works”
  - “Too much review from last week.”
    - Will do a slightly faster recap

Week 6 class evaluation

- Comments (slightly edited):
  - “i know it's probably frustrating to have such a weak turn out, but keep in mind that the people that were there today most likely were the same people that were there last week and will be there next week. ;)”
    - Yes, I agree, I'm sorry if anyone was offended.
    - I did express disappointment on the blog also.

Outline for today

- A final note on order statistics
- Heaps
- Priority queues
- Assignment 4, part 1
- Hint number 10

Order statistics tree

- Parent pointers are needed for the book's (CLRS') solution to OS-rank
- LLRB insert does not maintain parent pointers.
- You suggested to use top-down approach
- I showed the pseudocode for bottom-up last week.
- I'll show you equivalent top-down approach

Top-down OS-rank

- Topdown-OSRank(T,x)

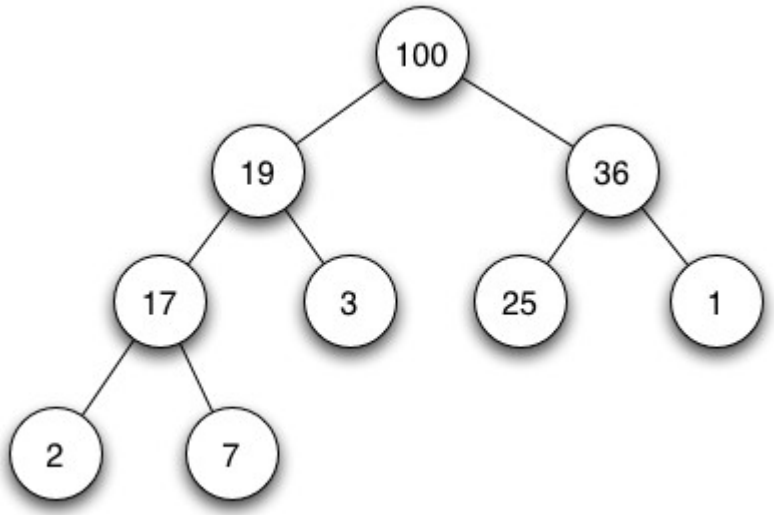
```
r = 0
y = T.root
while (y.key != x.key)
        if (x.key > y.key)
                r = r + y.left.size + 1
                y = y.right
        else
                y = y.left;
r = r + y.left.size+1;
return r;
```

(Binary) Heaps

- “Nearly complete binary tree”
- Max-Heap-property
  - For every node *x*, other than the root
    - *x*.parent.key >= *x*.key
- Min-Heap-property
  - For every node *x*, other than the root
    - *x*.parent.key <= *x*.key

Max-heap

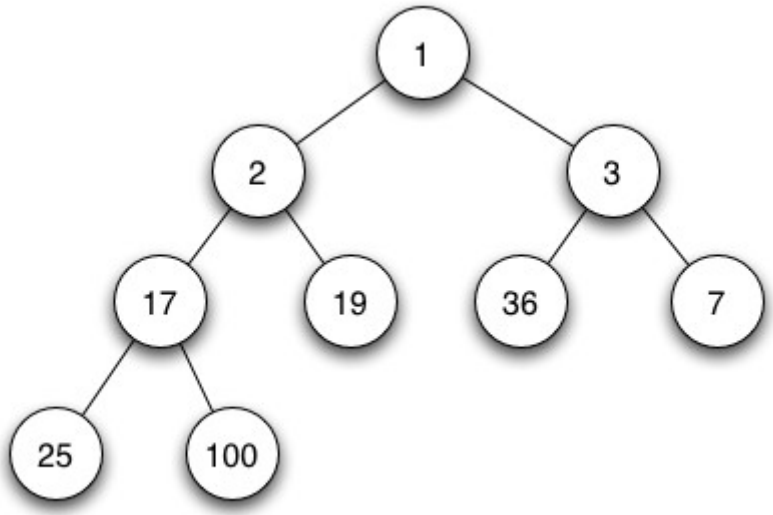
- This is a max-heap



- Notice that it is NOT a binary search tree

Min-heap

- This is a min-heap



- Notice that this, also, is NOT a binary search tree

Heaps

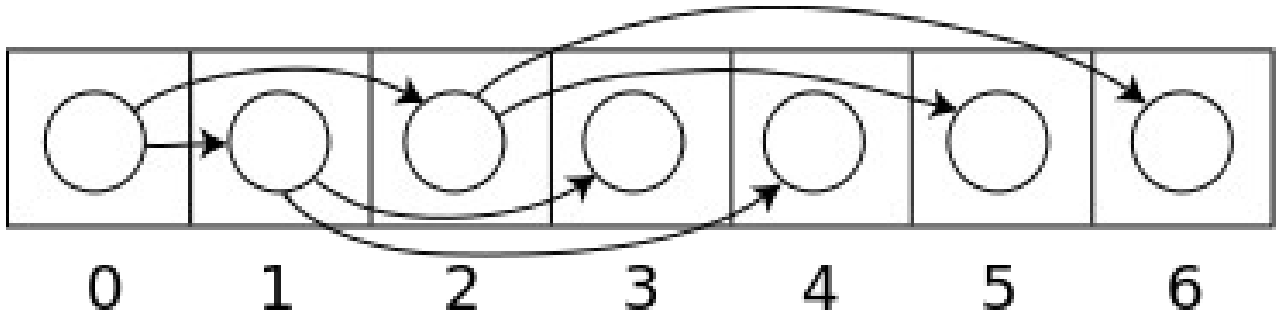
- Often, we implement heaps with an array
- For an array *A*, we have:
  - A.length
    - Capacity of the heap/array
  - A.heapsize
    - Number of elements currently in the heap
  - A[0] is the root of the tree
    - Beware that the book uses 1-indexing instead of 0-indexing like me.

Heaps

- Max-Heap-property
  - For every node with index *i*, other than the root
    - A[parent(*i*)] >= A[i]
- Min-Heap-property
  - For every node with index *i*, other than the root
    - A[parent(*i*)] <= A[i]
- Be careful, *i* != *key*

Array heap

- Array representation



- Why is this cool?

Array heap

- No pointers are needed!
- Parent(i)

  return (i-1)/2 (round down)
- Left(i)

  return $2i + 1$
- Right(i)

  return $2i + 2$

Wake-up quiz – Heaps

- Consider the array:

  A = (23, 17, 14, 6, 13, 10, 1, 5, 7, 12)

- Is A a max-heap?

Wake-up quiz – Heaps

- Consider the array:

  A = (23, 17, 14, 6, 13, 10, 1, 5, 7, 12)

- Is A a max-heap?
  - Nope, 7 is a right child of 6
  - Drawing it is straightforward.

Maintaining heap property

- Max-Heapify(A, i)
  - Make a heap rooted at index *i*
  - A[*i*] “floats down” the tree
- Procedure
  - Determine A[*i*], A[Left(*i*)], A[Right(*i*)]
  - Swap A[*i*] with the largest number
  - Recursively call Max-Heapify on the subtree rooted at the largest number
  - Stops when A[*i*] is the largest number

Building a heap

- Build-Max-Heap(A)
  - Call Max-Heapify on every element down to the first element.
  - Start at middle of array
    - Because everything after middle will automatically be heapified
- Surprise: Runs in linear time O(n) !

Heaps – applications

- What can they be used for?
  - Heap-sort
    - In-place sorting
      - With array representation
  - Priority queues

Heapsort

- Heapsort(A)

  Build-max-heap(A)

  For *i* = A.length downto 2

    Exchange A[*i*] with A[*1*]

    A.heapsize = A.heapsize-1

    Max-Heapify(A,1)

- Running time Θ(n lg n)

Heap data structure

```
// C++ heap
struct Heap {
  int heapSize;
  int length;
  int * heap;
  Heap(int h[], int l, int hs) {
    heap = h;
    length = l;
    heapSize = hs;
  }
};
```

Heap data structure

```
int parent(int i) {
  return (i-1)/2;
}

int left(int i) {
  return 2*i+1;
}

int right(int i) {
  return 2*i+2;
}
```

Heapsort implementation

```
// C++ heapsort
void heapSort(Heap& A) {
   buildMaxHeap(A);
   for (int i = A.length-1; i >= 1; i--) {
      int temp = A.heap[i];
      A.heap[i] = A.heap[0];
      A.heap[0] = temp;
      A.heapSize--;
      maxHeapify(A,0);
   }
}
```

Priority queues

- A queue with prioritized elements
  - In a way, a sorted queue
- Applications
  - Scheduling
  - Shortest path algorithms
  - Sorting

Priority queues

- Important operations
- Insert(S,x)
  - Inserts x into the set S
- Extract-min(S) / Extract-max(S)
  - Extracts min/max element of S
- Min(S) / Max(S)
  - Shows min/max element of S

Priority queues

- Implementation methods
  - Sorting algorithm
  - Binary search tree
  - Heap
- A heap can also be used for sorting
  - Heap sort
- A binary search tree can also be used for sorting
  - Inorder-tree walk

Priority queues

- Using a heap, all priority queue operations run in O(lg n)
- Looking at the max/min only takes O(1)
  - Compared to binary search tree O(lg n)

Assignment 4

- Implement a Binary Min-Heap
  - Use any language
- Use it as a Min-priority-queue
- Accept unspecified number of elements
  - Grow as necessary
- Do a number of Extract-min
- Find info on max-heap in CLRS, chapter 6
  - Figure out min-heap by yourself.

Assignment 4

- Don't pre-count number of elements
- You can either:
  - Maintain heap-property while inserting
  - Build the heap after all elements are inserted
  - In both cases, grow as necessary
    - Because you do not know how many nodes to insert

Assignment 4

- I will supply some testcases that will each have a “hidden” message.
- Along with your program, I would like you to find and submit the hidden messages from these testcases.

Hint number 10

- Hint number 10 is:
  - Read the textbook (7 votes)
- Runners up:
  - Use Google (6 votes)
  - Ask questions (5 votes)
  - Take breaks when you get stuck (4 votes)

Thank you

Questions?