CS 151
Heaps & Heap sort
Announcements

Prelab 7 is due now.

Lab 7 due on Sunday night.

CS department winter term organizational meeting:
   Tuesday Nov 6, 4:30pm in King 237
A **priority queue** stores a collection of comparable elements such that the highest priority element is “at the front”. Operations are

- `insert()` aka `offer` — add element to the queue
- `removeMin()` aka `poll` — remove the highest priority element
- `findMin()` aka `peek` — return but do not remove the highest priority element
- `size()`
- `isEmpty()`
- `traverse()` / `iterate()`

We implement the priority queue using a **binary heap**:

- a complete binary tree (a binary tree where each level is completely filled except for possibly the last, which is filled from left-to-right.)
- with the **heap order property**:
  - for all nodes \( n \), the value/key of \( n \) is \( \leq \) value/key of its children

The nodes of a heap are stored in an array in level-order, starting at index 1.

We showed that `size()`, `isEmpty()`, `findMin()` is \( O(1) \); `insert`, `remove` \( O(\log n) \)
percolateDown( index ) -- move the elt at index index down to the correct spot
  • loop
    • if index is a leaf, return (it’s smaller than its non-existent children)
    • else, let minChild = index of smaller of index’s children (or, child)
      • if data[minChild] < data[index] then we’re out of heap order
        • swap the two entries
        • set index = minChild
      • else, return (we’re done.)

RT: if percolate down to the bottom, get O(log n). Note that percolate( index ) is actually O( height of the tree rooted at index ).
Consider trying to construct a heap out of an unordered array.

- `heapify( array )` - make array into a heap (enforce the heap order prop)

How could you do this?

**Idea #1:**
- construct a new empty temporary array, view it as a heap
- for each element in given array, add it to our new heap
- at the end, new heap contains all of the array, but with heap order prop

Problem is... takes $O(n \log n)$ time. (Each of the $n$ inserts takes $O(\log n)$.)

Want a way to heapify in $O(n)$ time.
Consider trying to construct a heap out of an unordered array.
- `heapify(array)` – make array into a heap (enforce the heap order prop)

How could you do this?

idea #2:
- view given array as a complete binary tree without the heap order prop
- start at the leaves, move up the tree level-by-level
  - call `percolateDown(i)` for each node in the tree
  - i.e. enforce the heap order prop node-by-node, going up the tree

Ex. array 92, 47, 21, 20, 12, 45, 63, 61, 17, 55, 37, 25, 64, 83, 73

Since `percolateDown(i)` takes $O(\text{height}(i))$, the RT of `heapify` is:
$$\sum_{\text{nodes } n \in T} O(\text{height}(n)) \in O(n) \quad \text{(we’ll prove this by induction later)}$$

Bottom Line: If you need to construct a new heap out of n elements, use `percolateDown` on every node, starting from the leaves!
Heapsort

Note that heapify gives us a new sorting algorithm, called heapsort:

- heapsort(array):
  - call heapify(array)
  - removeMin n times to repeatedly get the next min element
- heapify is $O(n)$, $n$ removeMins are $O(n \log n)$, so heapsort is $O(n \log n)$
A complete binary tree $T$ is either

- an empty tree, or
- a root node $r$ with left and right complete binary trees $L$ and $R$ such that if $L$ is not perfect (it’s “missing” some leaves) then $h(R)=h(L)-1$ (o.w. $h(R)=h(L)$)

For any complete binary tree $T$, let $P(T)$ be the boolean property that $\sum_{v \in T} h(v) \in O(n(T))$, where $h(v)$ is the height of node $v$.

(Remember, you can think of $P(T)$ as a boolean function that sums up the heights of all the nodes in $T$ then checks that it is at most $\text{numNodes}(T)$-ish.)

We will show that $P(T)$ is true for all complete binary trees $T$ “by structural induction” on $T$.

Step 0: Play around and convince yourself $P(T)$ is true for some specific $T$’s.
A complete binary tree $T$ is either
- an empty tree, or
- a root node $r$ with left and right complete binary trees $L$ and $R$ such that if $L$ is not perfect (it’s “missing” some leaves) then $h(R)=h(L)-1$ (o.w. $h(R)=h(L)$).

For any complete binary tree $T$, let $P(T)$ be the boolean property that $\sum_{\text{nodes } v \in T} h(v) \in O(n(T))$, where $h(v)$ is the height of node $v$.

Step 1: Show that $P(T)$ is true when $T$ is your base case.

**base case:** $T$ is an empty tree

We want to show (WTS) that $P(T)$ is true when $T$ is an empty tree, i.e. that $\sum_{\text{nodes } v \in T} h(v) \in O(n(T))$ holds when $T$ is an empty tree.

When $T$ is empty, there are no nodes in $T$ so $n(T)=0$ and the sum is empty:
$$\sum_{\text{nodes } v \in T} h(v) = 0 \in O(n(T)) \quad \text{because there are no nodes to sum over}$$
$$\because n(T) = 0.$$
A complete binary tree $T$ is either
• an empty tree, or
• a root node $r$ with left and right complete binary trees $L$ and $R$ such that if
  $L$ is not perfect (it’s “missing” some leaves) then $h(R)=h(L)-1$ (o.w. $h(R)=h(L)$)

For any complete binary tree $T$, let $P(T)$ be the boolean property that
$$\sum_{\text{nodes } v \in T} h(v) \in O(n(T)),$$
where $h(v)$ is the height of node $v$.

Step 2: Show that $P(T)$ is true when $T$ is your recursive case.

**recursive case:** $T$ is a root node $r$ with $L$ and $R$ complete binary trees. Suppose hypothetically, that $P(L)$ and $P(R)$ are true, i.e. that
$$\sum_{\text{nodes } v \in L} h(v) \in O(n(L)) \quad \text{and} \quad \sum_{\text{nodes } v \in R} h(v) \in O(n(R))$$

This is our **induction hypothesis**

We want to use these two hypothetical assumptions and the definition of $T$ to show $P(T)$ is true, i.e. that $$\sum_{\text{nodes } v \in T} h(v) \in O(n(T))$$ is always true.
Structural Induction Example

- Algebraically connect \( \sum_{v \in T} h(v) \) to \( \sum_{v \in L} h(v) \), \( \sum_{v \in R} h(v) \)

- Algebraically connect \( n(T) \) to \( n(L), n(R) \): \( n(T) = 1 + n(L) + n(R) \)

Use these facts along with our IH \( \sum_{v \in L} h(v) \in O(n(L)), \sum_{v \in R} h(v) \in O(n(R)) \)

and the definition of \( T \) to show \( P(T) \) is true, ie. that \( \sum_{v \in T} h(v) \in O(n(T)) \) is true.

\[
\sum_{v \in T} h(v) = h(r) + \sum_{v \in L} h(v) + \sum_{v \in R} h(v)
\]

\( \text{b/c T’s nodes are r, L, & R} \)

\[
= h(r) + O(n(L)) + O(n(R))
\]

\( \text{by the induction hypothesis} \)

\[
= h(r) + O(n(T))
\]

\( \text{b/c } n(L) + n(R) = n(T) - 1 \)

\[
\in O(n(T))
\]

\( \text{b/c } h(r) \leq n(T) \)

Thus \( P(T) \) is true for all \( T \) by structural induction on \( T \).