CS240 - Module 2

Graham Cooper

May 14th, 2015

ADTS

Stack

Array

Array: $[e_1, e_1, ...e_{size-1}]$
Pop(): If n > 0, n = n-1, return A[n].
Push(x): n=n+1 and now A[n-1] = x

Linked List

$$heap \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow \emptyset$$

$$heap \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow \emptyset$$

Push(x): head = new(x, head)
Pop(): Assume n > 0
ret = head− >item head = $head \rightarrow next$ return ret;

Queue

Array

array: A = $[... \overset{front}{e_1}, e_2, e_3 ... \overset{last}{e_n} ...]$
Enqueue(x):
last = last + 1 mod size
A[last] = x
n = n +1
Dequeue....?

Heaps

- binary tree
- heap-order property

- structural property

Max heap: all of the nodes are less than the nodes above.(The nodes get smaller as they go down)
Min Heap: All of the nodes are greater than the nodes above. (The nodes get larger as they go down)

Assuming the level of a heap is h, the number of nodes is $1 + 2 + ... + 2^{h-2} = 2^{h-1} - 1$

$$h > 2^{h-1}$$

$$h + 1 > 2^{h-1}$$

$$\log n + 1 > h - 1$$

$$h < \log(n + 1) + 1$$

$$h \in O(\log n)$$

	0	1	2	3	4	5	6	7	8	9
A	100	50	20	30	40	15	19	1	8	2

$$A[0] = 100$$

$$A[0] \rightarrow left = 50 = A[1]$$

$$A[0] \rightarrow right = 20 = A[2]$$

$$A[1] = 50$$

$$A[1] \rightarrow left = 30 = A[3]$$

$$A[1] \rightarrow right = 40 = A[4]$$

$$A[3] \rightarrow left = A[2 \times 3 + 1] = A[7] = 1$$

$$A[3] \rightarrow right = A[2 \times 3 + 2] = A[8] = 8$$

$$A[i] \rightarrow left = A[2 \times i + 1]$$

$$A[i] \rightarrow right = A[2 \times i + 2]$$

Insertion into Heaps

From our table above, we can see that if we insert 70, it goes into the 10th element in the array. We want to "Bubble" this value up since now 70 is greater than its parent, 40. So we swap upwards until we find hte correct position for the new inserted value. See the table below.

	0	1	2	3	4	5	6	7	8	9	10
A	100	50	20	30	40	15	19	1	8	2	<u>70</u>

	0	1	2	3	4	5	6	7	8	9	10
A	100	50	20	30	<u>70</u>	15	19	1	8	2	<u>40</u>

	0	1	2	3	4	5	6	7	8	9	10
A	100	<u>70</u>	20	30	<u>50</u>	15	19	1	8	2	40

insert 70
swap(70 and 40)
swap(70 and 50)

Each swap takes constant time, and in the worst case will be log(n)

Delete Max

	0	1	2	3	4	5	6	7	8	9	10
A	<u>100</u>	70	20	30	50	15	19	1	8	2	40

	0	1	2	3	4	5	6	7	8	9	10
A	<u>40</u>	70	20	30	50	15	19	1	8	2	_

	0	1	2	3	4	5	6	7	8	9	10
A	<u>70</u>	<u>40</u>	20	30	50	15	19	1	8	2	_

	0	1	2	3	4	5	6	7	8	9	10
A	70	<u>50</u>	20	30	<u>40</u>	15	19	1	8	2	_

delete max
swap(40, 70)
swap(50, 50)
Yay we are happy now!
The swap takes constant time and the bubble down also takes log(n) time

Height of a heap is $\Theta(log(n))$
Proof:

$$n = 2^0 + 2^1 + 2^2 + ... + 2^{h-1} + k$$

$$1 \leq k \leq 2^h$$

$$2^0 + 2^1 + ... + 2^{h-1} + 1 \leq n \leq 2^0 + 2^1 + ... + 2^{h-1} + 2^h$$

$$2^h \leq n \leq 2^{h+1} - 1 < 2^{h+1}$$

$$h \leq log(n) < h + 1$$

$$h = floor(logn)$$

getMax: $\Theta(1)$
extractMax: O(logn)
insert: O(logn)
delete: O(logn)

Sorting

Turn A into a heap, then we can find max in $\Theta(1)$ time and fix the heap takes O(log(n)) time.

Building a Heap

Array: [2,4,6,8,1,3,5,7]
Just using the first element in the array is a heap, because only one element holds up all of the properties of a heap. We are going to insert the rest of the elements into the heap and continue to hold the ordering property.

	0	1	2	3	4	5	6	7
A	2	4	6	8	1	3	5	7

	0	1	2	3	4	5	6	7
A	<u>4</u>	<u>2</u>	6	8	1	3	5	7

	0	1	2	3	4	5	6	7
A	<u>6</u>	2	<u>6</u>	8	1	3	5	7

...

	0	1	2	3	4	5	6	7
A	8	7	5	6	1	3	4	2

Upper Bound:
Cost of bubble-up is O(depth of a node)
Depth of each node is O(logn)
Number of nodes to process is n so therefore runtime is O(nlogn)

Lower Bound:
The worst-case:
Initial ordering is in increasing order.
We will need to bubble up to the root node every single time.

For any complete tree of size n, roof(n/2) nodes are leaves (prove by induction left to exercise).
Leaves have depth $\geq$ h-1 = floor(logn) - 1 in the worst case number of comparisons for bubble up is at least roof(n/2)(log(n) - 1) $\in \Omega(nlogn)$

Let's view the array as arleady a heap out of order then "fix" the heap.

	0	1	2	3	4	5	6	7
A	2	4	6	8	1	3	5	7

We can see that the bottom row of 1,3,5, and 8. We know that all of these nodes with no child nodes are proper heaps, and the 8 which has a child of 7 is also a proper max-heap. Examine 8: do nothing
Examine 6: do nothing
Examine 4: bubble 4 down (swap 4 with 8)
Examine 2: swap 2 with 8, 7, then 4

	0	1	2	3	4	5	6	7
A	8	7	6	4	1	3	5	2

Runtime Analysis

number of nodes	height of node
$= 2^0$	h
$= 2^1$	$\leq h-1$
$= 2^{h-1}$	≤ 1
$\leq 2^h$	0

A bubble down call on a node of height k takes O(k) time.
(at most k swaps to get to the bottom)
Lemma: There are no more than 2^{h-1} nodes of height i.

Total number of swaps is at most

$$\sum_{i=0}^{h} i \cdot 2^{h-i}$$

$$= 2^h \sum_{i=0}^{h} \frac{i}{2^i} \leq n \sum_{i=0}^{h} \frac{i}{2^i}$$

$$< n \sum_{i=0}^{\infty} \frac{i}{2^i}$$

$$S = 1/2 + 2/4 + 3/8 + 4/16 + ...$$

$$2S = 1 + 1/2 + 3/4 + 4/8 + ...$$

$$2S - S = 1 + 1/2 + 1/4 + 1/8 + ...$$

$$s = 2$$

$\therefore$ the number of swaps is at most 2n