1. Priority queues

2018年9月11日 14:46

Abstract data type

An object (i.e. set, graph, sequence)

Operations can perform on this object

Data structure implements the ADT, specifies

1) how to represent the object in memory,
2) how to implement the operations as algorithms

Example of ADT: Priority Queues

Object set(/multiset) S of elements with keys (as integers)

Operations

max( ) returns the elements of S with the largest key

$extract_$max( ) returns the element of S with the largest key and removes it from S

$insert(x)$ add x (x has a key) to S

Examples of implement

DS	Insert WCRT	Extract-max WCRT
Unsorted Linkedlist	1	n
Sorted linkedlist	n	1
Heap	$\log n$	$\log n$

Max Heap (DS)

Shape(where to store) complete binary tree of size n

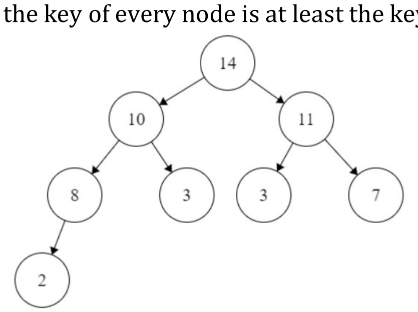
Complete binary tree in which every level is full except possibly the last one where all nodes are fully to the left. Root -> lv0, level := #edges on the path to the root

Consider a complete binary tree with all full h-1 levels plus one more node

There are 2^h nodes in such tree $1 + 2 + 2^2 + \cdots + 2^{h-1} + 1 = (2^h - 1) + 1$

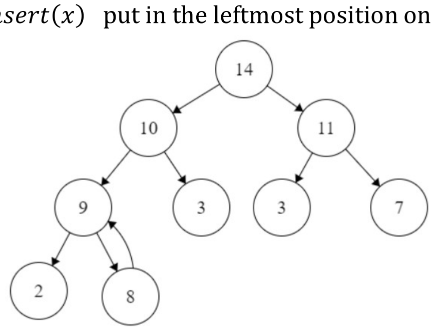
Properties(how to store)

the key of every node is at least the keys of its children. Example



max( ) returns the root, max( ) $\in O(1)$

$insert(x)$ put in the leftmost position on the last level, swap with its parent till it's larger than both of its children



Note that the binary tree is only a visualization, in an actual computer, data are stored more like an array

For example, the tree above will be in the form of [14,10,11,9,3,3,7,2,8]

Notice that pick any node, take its index being i

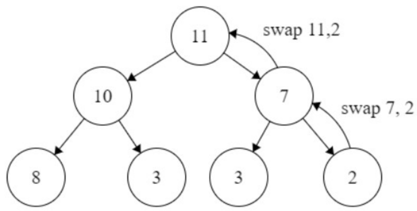
$left(i) = 2i$ if $2i \leq n$ or its left child is empty

$right(i) = 2i + 1$

$parent(i) = floor\left(\frac{i}{2}\right)$

Hence, the swapping will have at most $\lg n$ times $insert(x) \in O(\log n)$

$extract\_\max(\,)$ removes the root, move the leftmost leaf to the root position. Swap with its larger child till it's larger than its children.



Similarly, $extract\_\max(\,) \in \mathrm{O}(\log n)$

Max_Heapify(A, i)

\# Precondition: the left and right subtrees, L, R are max-heap

\# Postcondition: tree rooted at $A[i]$ will be a max-heap

```
1.  l = left(i) // the left node
2.  r = right(i) // the right node
3.  if l ≤ A.heap-size and A[r] > A[largest]
4.     largest = l
5.  else largest = i
6.  if r ≤ A.heap-size and A[r] > A[largest]
7.     largest = r
8.  if largest ≠ i
9.     swap(A[i], A[largest])
10.    max_heapify(A, largest)
```

For line 1-9, the alg takes constant steps.

Hence, for the recurrence $T(n) = \mathrm{T}\left(\frac{2\mathrm{n}}{3}\right) + \mathrm{c}$ since the size of a subtree of a compelete binary tree is $\frac{2n}{3}$ (when the last level is exactly half full)

By Master's thrm, $T(n) \in \mathrm{O}(\lg \mathrm{n})$

Or, since each time, the swap happens between a parent and a child, $A[largest]$ can at most be swapped to a leaf.

Resulting $T(n) \in O(h)$

Build_Max_Heap(A)

```
n = length(A)
for i = range(floor(n/2), 0, -1)
    max_heapify(A, i)
```

Lemma there are at most $\frac{n}{2^h}$ nodes, whose rooted subtrees are of heigh h

Since $2^n \leq n \leq 2^{n+1} - 1$

There's only one node whose rooted subtrees are of height h (The root)

Then there are two nodes whose rooted subtrees ar of height $h - 1 = \frac{n}{2^{h-1}} = 2\left(\frac{n}{2^h}\right)$

...

Claim $T(n) \in O(n)$

Proof

$$T(n) \leq \sum_{h=0}^{floor(\lg n)} \frac{n}{2^h} O(h)$$

by the lemma there are at most $\frac{n}{2^h}$ nodes on each level and each node will take $O(h)$ time to heapify

$$= cn \sum_{h=0}^{floor(\lg n)} \frac{h}{2^h}$$

c is the constant in $O(h)$

$$\leq cn \sum_{h=0}^{\infty} \frac{h}{2^h} = cn \sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h$$

dx dx

By Maclaurin's series $\sum_0^\infty h\left(\frac{1}{x}\right)^h = \frac{d}{dx}\sum_1^\infty \left(\frac{1}{x}\right)^h = \frac{d}{dx}\left((1-x)^{-1}\right) = \frac{x}{(1-x)^2}$

$= \dfrac{0.5}{0.25} = 2cn \in O(n)$

Heapsort(A)

1. $build\_max\_heap(A)$
2. $for\ i\ in\ range(n, 2, -1)$:
3. $\quad swap(A[1], A[i])$ //the largest number got moved to the last element
4. $\quad n--$ //remove the last element (the largest element), shorten the heap
5. $\quad max\_heapify(A, 1)$ //maintain property

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