Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dmytro Chibisov
Sandeep Sadanandan

Winter Semester 2007/08
Solution Sheet 5
November 22, 2007

Fundamental Algorithms

Problem 1 (10 Points)

What is an undirected tree? Give three definitions of tree and show they are equivalent.

Solution

Three definitions:

1. A tree is an undirected graph that is connected and acyclic.
2. Any connected, undirected graph with n nodes and $n-1$ edges is a tree.
3. An undirected graph with a unique path between any pair of nodes is a tree.

1=2 Since the graph is connected, there must be atleast one way to go from one node to another which means there must be atleast $n-1$ edges. And if there is a cycle in the graph, that means there is more than one way from one node to another which would say there are more than $n-1$ edges. so acyclic connected graph is a connected graph with $n-1$ edges. The other way around is also similar.
1=3 If there is a unique path between any pair of nodes, that means there are no cycles. And the graph is connected.
All the three definitions are equivalent.

Problem 2 (10 Points)

A Binary Tree is a rooted tree in which every node has at most two children. The root node is said to be in level one. The children of the noded at level n are in level $n+1$.

Calculate

1. The maximum number of nodes in level h of a binary tree
2. The maximum number of nodes in a binary tree of h levels.
3. How many nodes does a *complete*[1] binary tree with n leaves have?

[1] A binary tree is *complete* if all of its vertices have either zero or two children and all the leaves are at levels l and $l-1$

Solution

1. The maximum number of nodes in level h of a binary tree

   A level will have maximum number of nodes iff the level above it has maximum nodes and all the nodes have maximum (2) children. So the maximum nodes which can occur in level h is 2 times the maximum of level $h-1$.

   The maximum of level 1 is 1 which is 2^0

   The maximum of level 2 is 2 which is 2^1

   The maximum of level 3 is 4 which is 2^2

   Hence, the maximum of level h will be 2^{h-1}.

2. The maximum number of nodes in a binary tree of h levels.

   A tree of h levels will have maximum nodes iff all the h levels have maximum nodes in each of them. So the total number of nodes will be $1 + 2 + \ldots + 2^{h-1}$ which is equal to $2^h - 1$.

3. How many nodes does a *complete* binary tree with n leaves have?

   If the tree has maximum number of nodes, it's last level has to have maximum number of leaves. If we assume that the last level is h, then from the first part, we know that the maximum number of leaves at level h is 2^{h-1}. So in this case 2^{h-1} equals n.

   And from the part two, we know that the maximum number of nodes in a tree of levels h is $2^h - 1$.

   We can see that $2^h - 1 = 2(2^{h-1}) - 1 = 2 \cdot n - 1$. So the solution is $2 \cdot n - 1$.

   A different and more mathematical approach is given below. If the complete binary tree is of height h, the number of leaves will be between 2^{h-2} and 2^{h-1}. Hence the height could be calculated to be $\lceil \lg n \rceil + 1$. The maximum number of nodes a tree of height h is $2^h - 1$.

   So in this case, it is $2^{\lceil \lg n \rceil + 1} - 1 = 2^{\lceil \lg n \rceil} \cdot 2 - 1$

Problem 3 (10 Points)

Perform the `heapify()` operation on an array containing the keys $\{0, .., 9\}$ where we assume that the keys are stored in the tree in the following order (level-wise, left-to-right): $6; 3, 4; 1, 8, 5, 0; 7, 9, 2$ ((levels separated by semicolon))

Solution

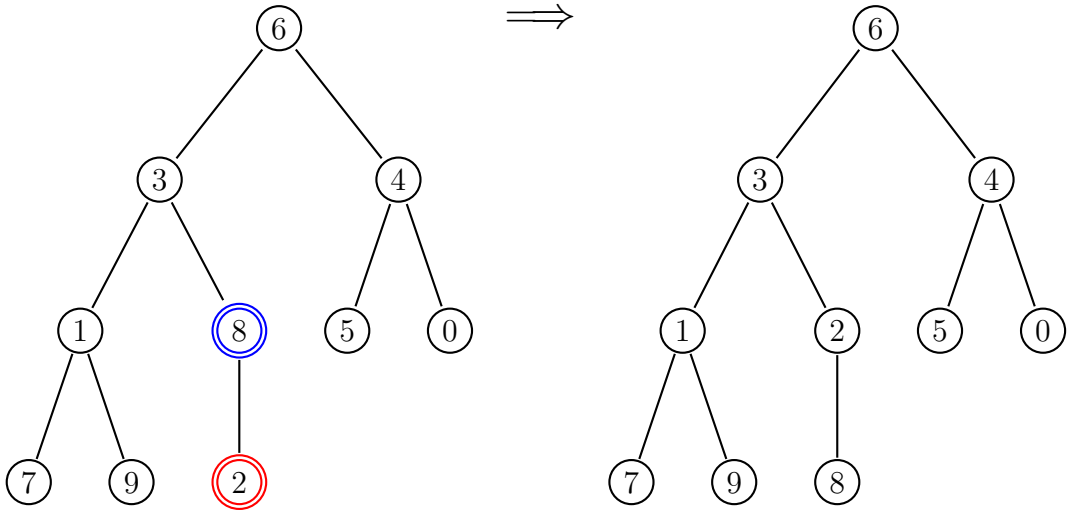
The `heapify` algorithm is given below at **??**.

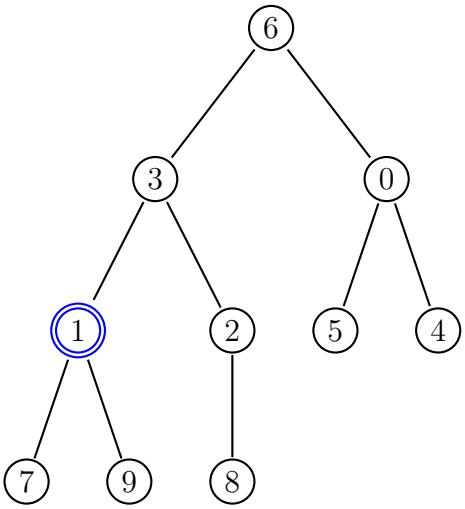
Algorithm *heapify*(Array A, size)
($*$ The algorithm for creating the heap $*$)
1. **for** $i \leftarrow \lfloor \frac{size}{2} \rfloor$ downto 1
2. $siftdown(A, i, size)$;
3. **return**

The algorithm for `siftdown` is given at **??**.

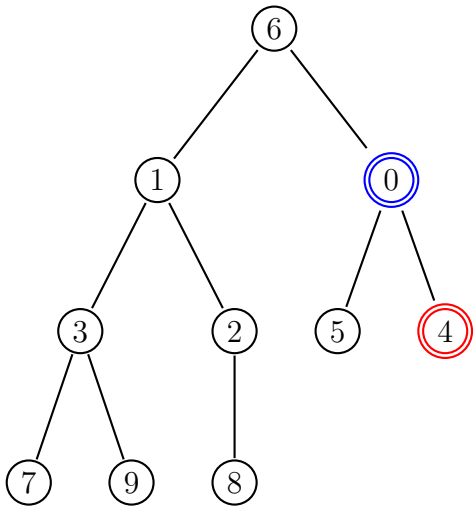
Algorithm *siftdown*(Array A, root, size)
(∗ The algorithm for sifting down ∗)
1. $child \leftarrow 2 \times root + 1$
2.
(∗ See whether there is a child ∗)
3. **if** $child > size$
4. **return**
5.
(∗ See whether there is a right child and if yes then see whether it is smaller ∗)
6. **if** $child < size$ and $A[child + 1] < A[child]$
7. $child \leftarrow child + 1$
8.
(∗ If the parent is larger, then do the swap and continue to reheap ∗)
9. **if** $A[root] > A[child]$
10. $swap(A, root, child)$
11. $siftdown(A, child, size)$;
12. **return**

The output of the algorithm can be seen in the figure. The blue nodes are the node which is being re-heaped. The red nodes are the children which needs to be swapped with his dad!

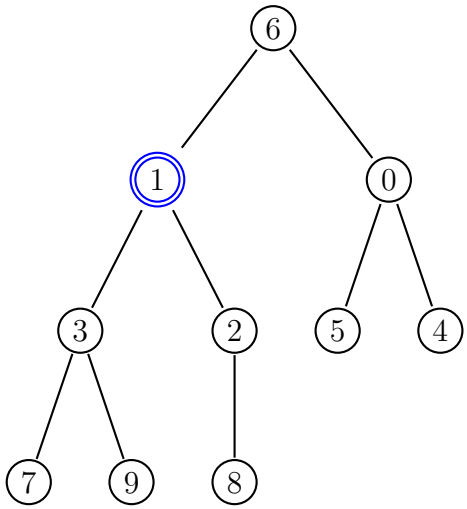




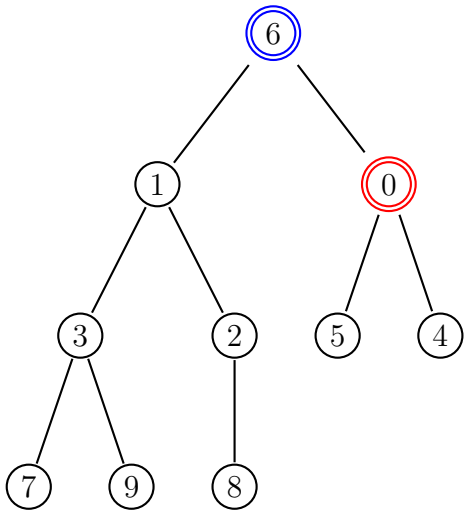
Nothing to change



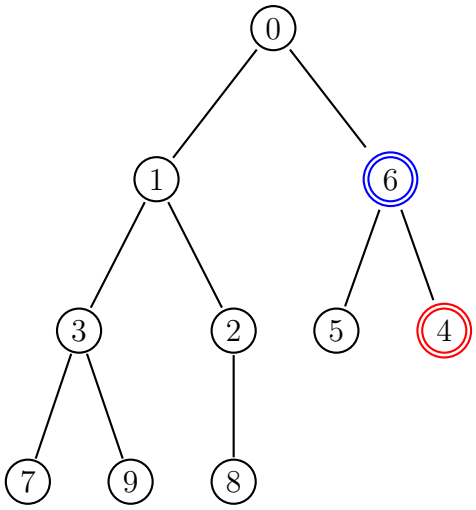
Nothing to change



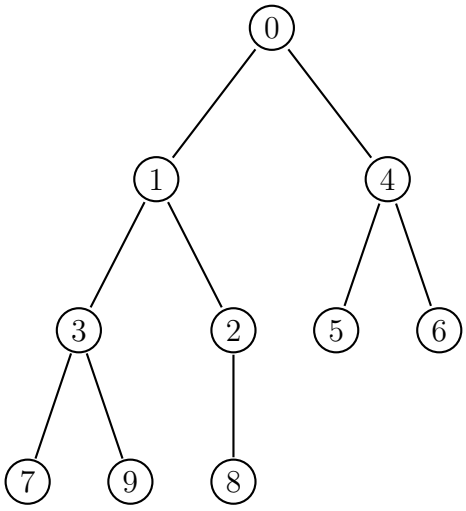
$\Longrightarrow$ Nothing to change



$\Longrightarrow$



$\Longrightarrow$



Problem 4 (10 Points)

Analyze the complexity of the `heapify()` function, applied to a complete binary tree containing n nodes. The result should be given as exactly as possible in Landau notation.

Solution

At each step of the `siftdown` algorithm, a node is compared to its children and one of them is chosen as the next root. So, after every `siftdown` call, we go to the next level which means, one level of the tree is dropped at each step of this process. Heap is a binary tree with a maximum of $\lg n$ levels. Thus, the worst case time complexity of `siftdown` is $O(\lg n)$.

Each call to `siftdown` takes $O(\lg n)$ time and there are $O(n)$ such calls. Thus the running time is $O(n \lg n)$.