CS161 – Fall 2016 — Homework2

Jenny Zeng, SID 52082740, zhaohuaz@uci.edu

R-4.17, R-5.11, C-6.1, A-6.8

R-4.17

For each of the following statements about wavl trees, determine whether it is true or false. If you think it is true, provide a justification. If you think it is false, give a counterexample.

(a) A subtree of a wavl tree is itself a wavl tree.

(b) The sibling of an external node is either external or it has rank 1.

Answer:
WAVL tree's properties are:

1. External node $\rightarrow$ rank = 0
2. If a node with 2 external node as child $\rightarrow$ rank = 1
3. Rank(parent) $\in$ {Rank(child+1), Rank(child +2)}

(a) True. A subtree of a wavl tree is itself a wavl tree.
justification:
Suppose a node n in a subtree is not following WAVL's properties, then this subtree is not a WAVL tree. Because this node n is also in the original tree, then the original tree is also not a WAVL tree.

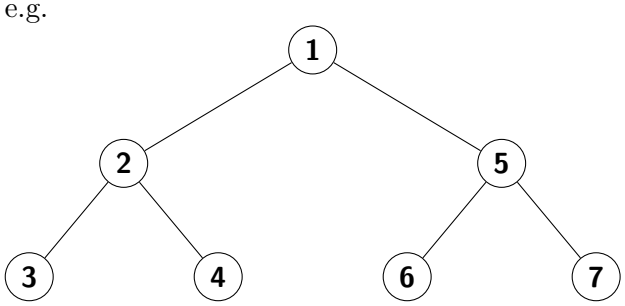
(b) True. according to WAVL's property:Rank(parent) $\in$ {Rank(child+1), Rank(child +2)}, the rank difference between two children is ≤ 1. Because an external node ranks 0, then it's sibling can rank 0 (an external node) or rank 1.

R-5.11

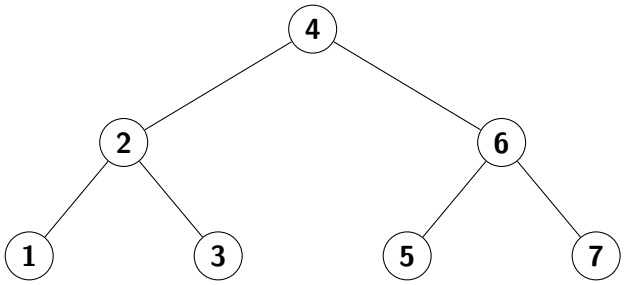
Is there a heap T storing seven distinct elements such that a preorder traversal of T yields the elements of T in sorted order? How about an inorder traversal? How about a postorder traversal?

Answer:

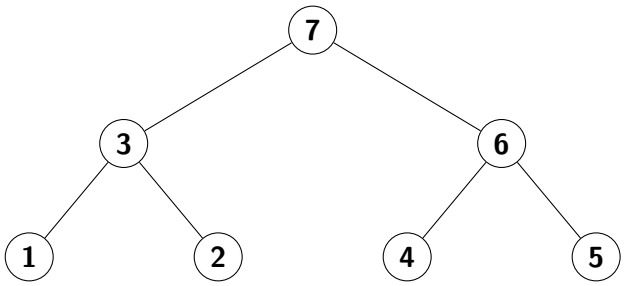
(a) preorder:
Exist. in preorder, $root < left < right$. This can be done in a min-heap.
e.g.



(b) inorder:
Not exist. In inorder, $left < root < right$.
In the example below, to yield elements of T in sorted order, the left child would be smaller than its parent. (1 is 2's child). This violate the rule that the parent should be smaller than its children in a heap.



(c) postorder:
Exist. In postorder, $left < right < root$. This can be done in a max-heap.
In the example below, the parent is greater than both of its children. It follows the property of a max-heap.



C-6.1

Give the pseudocode description for performing insertion, searching, and removal from a hash table that uses linear probing to resolve collisions where we use a special marker to represent deleted elements.

Answer:

In the algorithms below, A is the array, N is the length of A, h is the hash function, and if a cell is marked, it is not empty (NULL).

```
function SEARCH (key k)
    i ← h(k)
    while A[i] ≠NULL do
        if A[i] is not marked and A[i].key = k then
            return A[i]
        i ← (i + 1)modN
    return NULL

function REMOVE (key k)
    i ← h(k)
    while A[i] ≠ NULL do
        if A[i] is not marked and A[i].key = k then
            temp ← A[i]
            A[i] ← NULL
            mark A[i]
            return temp
        i ← (i + 1)modN
    return NULL

function INSERT (key k, value v)
    i ← h(k)
    m = NULL
    while A[i] ≠NULL do
        if A[i] is not marked and A[i].key = k then
            A[i] ← (k, v) // replace the old (k, v′)
            return
        else if A[i] is marked and m = NULL then
            m ← i
        i ← (i + 1)modN
    if m ≠ NULL then
        A[m] ← (k, v)
    else
        A[i] ← (k, v)
    return
```

A-6.8

Imagine that you work for an insurance company that is insuring people against identity theft. You have just learned about a major security breach at a prominent bank used by many of your customers. Through back channels, you have obtained the list of Social Security numbers of the bank customers whose banking records were stolen, and, of course, you know the Social Security numbers for your own customers. Describe an efficient scheme for identifying which of your customers were victims in this security breach. What is the running time of your method in terms of n, the number of customers of your insurance company, and m, the number of bank customers who were victims in this security breach?

Answer:

1. First, we use security number as the key. We can put all the customers into a hash table and we can assume that the hash function works reasonably well. This takes time $O(m)$
2. Then, for every social security number, search in hash table and append the customer found into our result list. This takes time $O(n)$.
3. At the end we get the result list and know who are victims in the security breach. Total running time would be$O(m + n)$.

this procedure can be represented in the pseudocode description below.

```
function VICTIMSCHECK([int] m, [int] n)
    for i ← 0 to m.size() − 1 do
        insert m[i] into a hash table //O(1)
    for j ← 0 to n.size() − 1 do
        search n[i] in hash table //O(1)
        append victim found to a result list //O(1)
    return result list
```