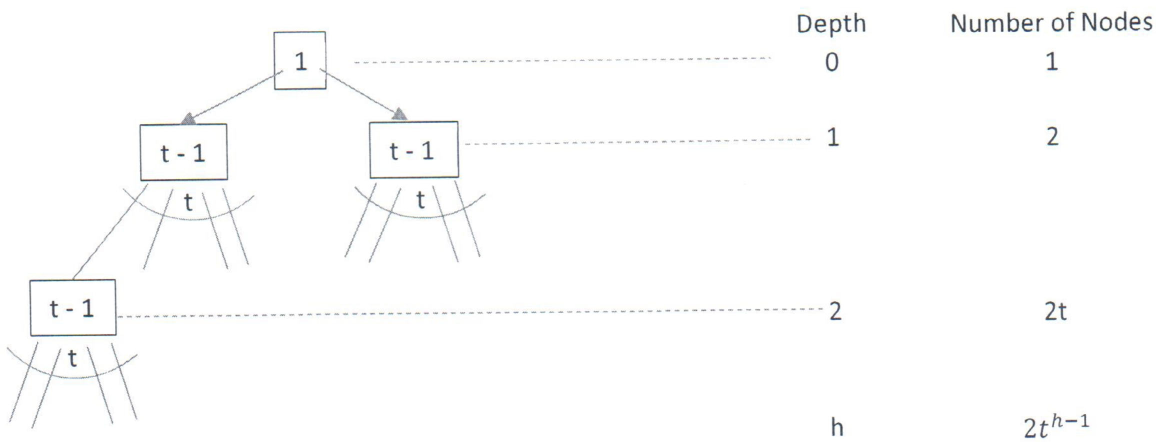
CSCE 500 Midterm Exam

1. For any n-key of height h and with the minimum node degree of t ≥ 2, prove that h is no larger than $\log_t \frac{n+1}{2}$. (Hint: consider the number of keys stored in each tree level.)

Show that $h \leq \log_t \frac{n+1}{2}$.



Let n = the total number of keys.

n ≥ number of root keys + (keys per nodes) * (total number of nodes)

$$n \geq 1 + (t-1) * \sum_{i=1}^{h} 2t^{i-1}$$

$$n \geq 1 + 2(t-1)\left[\frac{1-t^h}{1-t}\right]$$

$$n \geq 1 + 2(t-1)\left[\frac{1-t^h}{1-t}\right]$$

$$n \geq 1 + 2(t^h - 1)$$

$$n \geq 1 + 2t^h - 2$$

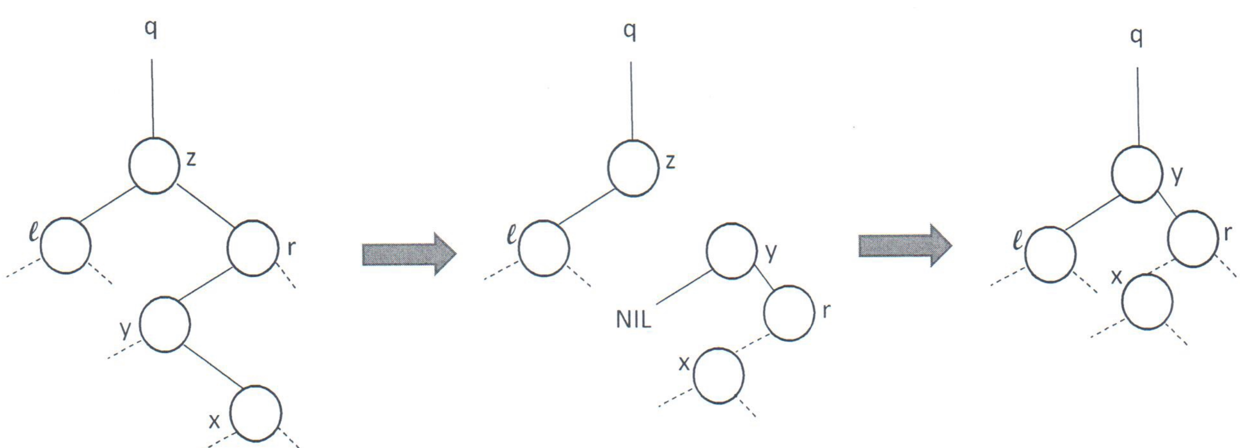
$$n \geq 2t^h - 1$$

* Adding 1, dividing by 2 and taking the log base t yields the following result.

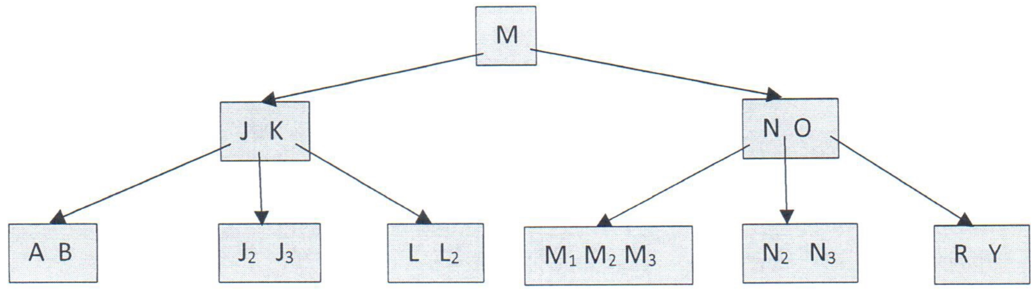
$$h \leq \log_t \frac{n+1}{2}.$$

2. The utilization efficiency of a hash table depends heavily on its hashing function(s) employed. Explain briefly (1) how <u>perfect hashing</u> works, and (2) <u>Cuckoo hashing</u> works under two has functions h_1 and h_2.
   (1) Perfect hashing uses primary and secondary hash tables. The primary hash table consists of an array of pointers. The pointers in the primary table point to the secondary table whose individual sizes are determined by the square of the number of collisions in each entry in the primary tables. The total storage size is O(n). The set of keys is static. Search takes O(1).
   (2) Cuckoo hashing uses two equally-sized tables, T_1 and T_2, with two unique hash functions h_1 and h_2. Initially, keys are hashed into table T_1 using hash function h_1. However, if a subsequent key is hashed into the same location as previously hashed, it will be removed , replaced with the subsequent key and hashed into table T_2 using hash function h_2. If there is a collision is table T2, the current key in the location will be removed and replaced with the new key. The subsequent key will be hashed into table T_1 using hash function h_1. It is possible that a key may not be able to be inserted into either table if the new edge (defined by the new key) contains at most one cycle.
3. The binary search tree (T) facilitates key search, and it involves several operations to maintain the tree property when a node (z) is deleted, as shown in the following pseudo code, TREE-DELETE(T, z), where TRANSPLANT(T, u, v) replaces the subtree rooted at u with one rooted at v. Fill in those three missing statements in the pseudo code below and sketch an example binary search tree to illustrate such a deletion case.

```
TREE-DELETE(T, z)
if z.left == NIL
    TRANSPLANT(T, z, z.right)    // z has no left child
elseif z.right == NIL
    TRANSPLANT(T, z, z.left)     // z has just one left child
else //z has two children
    y = TREE-MINIMUM(z.right)  //y is z's successor
    if y.p ≠ z
    // y lies within z's right subtree but is not the root of this subtree
    TRANSPLANT(T, y, y.right)
    y.right = z.right
    y.right.p = y
```

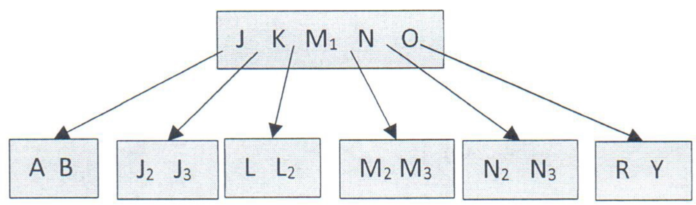


4. Given the initial B-tree with the minimum node degree of t = 3 below, show the results (a) after deleting two keys in order: M then R and (b) followed by inserting the key of L_1, with $L < L_1 < L_2$.

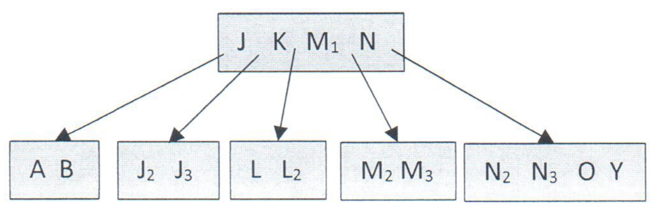


(a)

Deleting M…



Deleting R…



(b) Inserting L_1…