CSE 100 Advanced Data Structures Fall 2018

Homework 2

Instructors: *Leo Porter (Sec. A)* & *Debashis Sahoo (Sec. B)*

Due on: Tuesday, 10/30 *(24 points)*

Name: ShihHan Chan PID: A15677346 *Date:* 10 / 22 / 2018

Instructions

1. Answer each problem in the boxes provided. Any writing outside of the boxes *will NOT be graded*. Do not turn in responses recorded on separate sheets.
2. Handwritten or typed responses are accepted. In either case, make sure all answers are in the appropriate boxes.
3. All responses *must* be neat and legible. Illegible answers will result in zero points.
4. Make sure to *scan in portrait mode* and to *select the corresponding pages* on Gradescope for each question.
5. You may use code from any of the class resources, including Stepik. You may not use code from other sources.

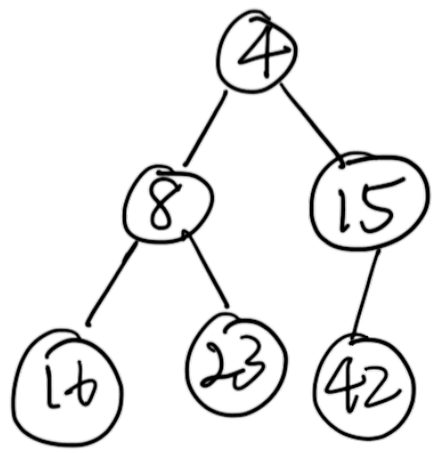
---

1. (6 points **- Correctness**) *Tree Reconstruction* : You ran depth first search and breadth first search on an unordered binary tree and got the following outputs:

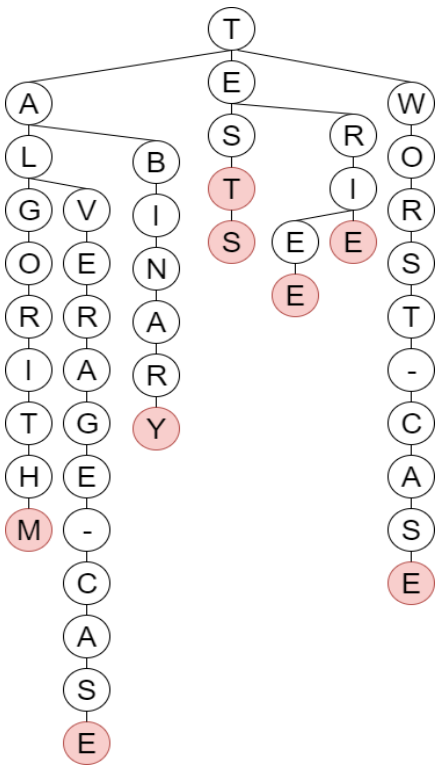
   BFS: 4 8 15 16 23 42

   DFS: 4 8 16 23 15 42

   What is a possible tree that could have given you this output? (Note: this tree is unordered, so for instance the node 4 will be the root, and its children in both directions will be bigger than it. Assume that a node is visited once it's taken out of the stack/queue. You may also assume that when either of the search methods are deciding which node to visit among nodes of the same depth, that they visit the left-most node first.)



2. (6 points - **Correctness**) *Ternary Trie Insert* : Refer to the ternary trie shown below to answer both items.

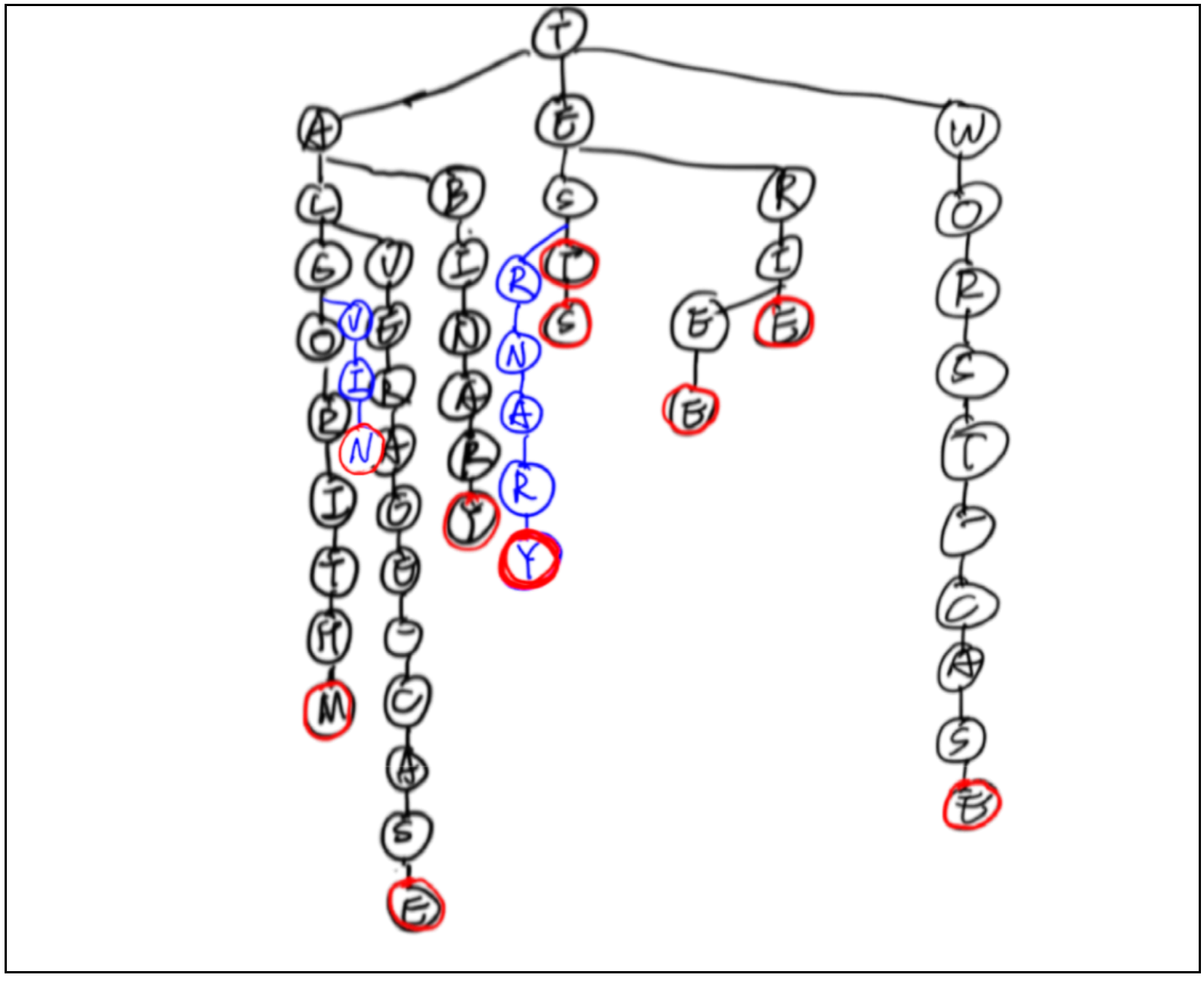


(a) List all the words stored in this ternary trie, arranged in *lexicographical order* in their corresponding box below. We *may* have included more boxes than there are words in the trie. If you think that's the case, leave the unneeded boxes *completely* blank.

(Note: If the words were *A, B* and *C,* the right answer would have *A* in square #1, *B* in square #2 and *C* in square #3, and boxes #4-12 completely blank.)

1. ALGORITHM	2. AVERAGE-CASE	3. BINARY
4. TEST	5. TESTS	6. TREE
7. TRIE	8. WORST-CASE	9.
10.	11.	12.

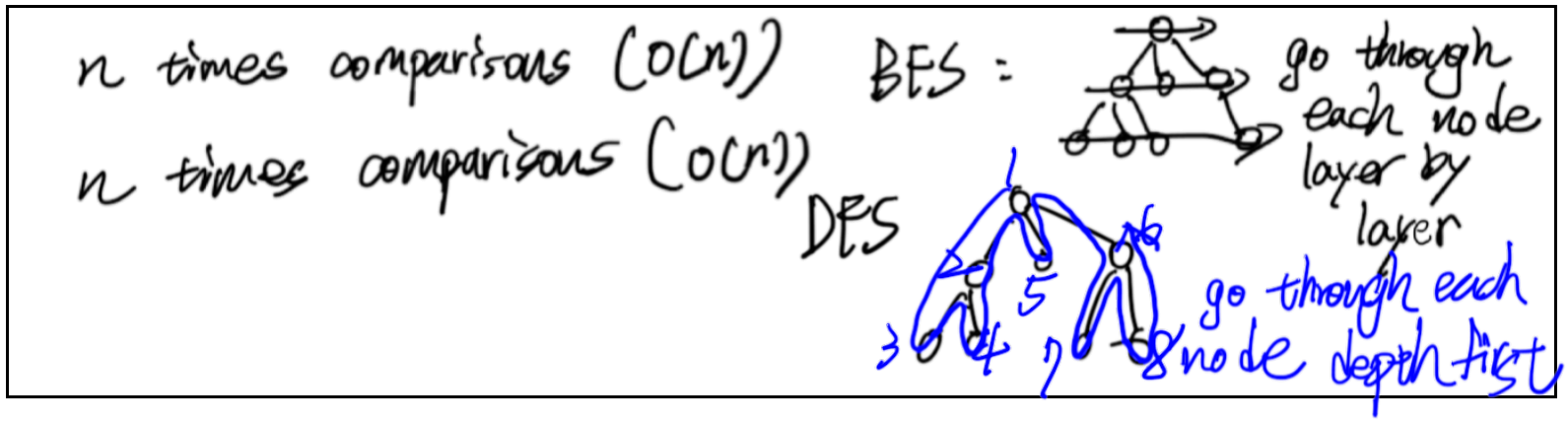
(b) Say we insert the words **ALVIN** and **TERNARY** into the trie shown above. Draw the resulting tree.



3. (6 points **- Correctness**) *Tree Search* :

(a) Suppose you have an unordered tree with n nodes. And suppose you're searching for a node with the value k. This should take the longest when k isn't in the tree. Briefly justify your answers.

- *(BFS)* If k isn't in the tree, how long does it take breadth first search to tell you that?
- *(DFS)* If k isn't in the tree, how long does it take depth first search to tell you that?



(b) *Iterative deepening DFS* (IDDFS) is a tree search strategy "[...] in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found."[1] It's often used in game tree exploration, e.g. solving chess, as it allows for more efficient exploration of the space of possibilities, especially at earlier levels of the tree. Fill in the blanks below to implement IDDFS.

```
class TreeNode<T> {
public:
  TreeNode * left, right;
  T value;
  int height; // root node stores tree height; a single node tree has height 0
};

bool find(const TreeNode<T> * root, const T& value) {
  if(root == nullptr) return false;
  for (int max_level = 0; [max_level<=root->height]; max_level++)
      if (iddfs(root, value, max_level)) return true;
  return false;
}
bool iddfs(const TreeNode<T> * root, const T& value, int max_level) {
  if([(!root)||(root->value!=value&&max_level<=0)]) return false;
  if([root->value==value]) return true;
  return [iddfs(root->left,value,max_level-1)||iddfs(root->right,value,max_level-1)];
}
```

[1] Source: *Iterative deepening depth-first search*. Wikipedia.

4. (6 points **- Completeness**) *Ternary Trie Delete* : Using pseudocode, write the *delete* method for a ternary trie. Please prune the tree so that it does NOT contain any unnecessary nodes after the deletion. You are allowed to make reasonable assumptions about the trie class you're working with. Make sure to *state any assumptions* you've made that aren't trivial and self-evident.

```
//assumptions: 1. suppose each node represents only one existing string at most (we don't need to count how many duplicate string at the same node)
2. call delete_node(root,string_you_want_to_delete,0); to delete the string you want to delete
struct Node{
    char key;
    bool leaf;
    struct Node *left;
    struct Node *down;
    struct Node *right;};

//return true if the function deletes the node successfully
bool delete_node(Node *cur, string s, int level){
    //the string doesn't exist in the tree
    if (!cur)return false;
    //string exists in TST
    if (level+1>=(int)s.length()){
    // free leaf node if there is a string
    if (cur->leaf) {
    cur->leaf = false;
    return !(cur->left ||cur->down ||cur->right);}
    else return false;}
    //search string in TST
    if(s[level] < cur->key)return delete_node(cur->left, s, level);
    if(s[level] > cur->key)return delete_node(cur->right, s, level);
    if (s[level]==cur->key){
    //recursively delete the node if cur node is free and it's not a leaf
    if(delete_node(cur->down, s, level+1)){
    delete(cur->down);
    return !cur->leaf && !(cur->left ||cur->down ||cur->right); }
    }
    return 0;}
```