COMP2119A Introduction to Data Structures and Algorithms
Homework 2 **Due Date:** 7pm, Oct 30, 2018

Rules: discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual pages of another student). Please make your answers precise and concise.

Course Outcomes

- **[O1]. Mathematics foundation**
- **[O2]. Basic data structures**
- **[O3]. Problem solving**
- **[O4]. Implementation**

1. **[O1]** (20 pts)

   (a) (5 pts) If we have 60000 items to store in a hash table **H-Open** using open addressing with linear probing and we desire a load factor of 0.75, how big should the hash table be?

   Solution: Table size $= \frac{n}{\alpha} = 60000/0.75 = 80000$

   (b) (5 pts) If we have 60000 items to store in a hash table **H-Chaining** using chaining and we desire the expected number of comparisons in a successful search to be 2.5, how big should the hash table be?

   Solution:

   Average number of comparisons in successful search $= 1 + \alpha/2 = 2.5$

   So the table size $= \frac{n}{\alpha} = 60000/3 = 20000$

   (c) (10 pts) Analyse and compare the memory usage for **H-Open** and **H-Chaining**. Suppose item's size and pointer's size are both 8 bytes.

   Solution:

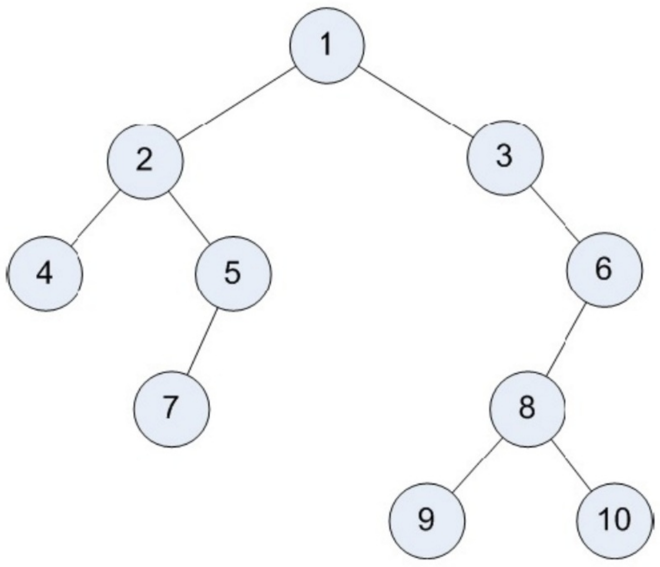
   - **H-Open**: 80000 * 8 bytes = 0.64MB
   - **H-Chaining**: 20000 * 8 bytes + 60000 * (2 * 8 bytes) = 1.12MB
     20000 * 8, one for each "bucket" and 2 * 8 (key and next) per node.

   Here we assume 1MB = 10^6 bytes.

2. **[O2]** (20 pts) Given the following pre-order traversal sequence: $1, 2, 4, 5, 7, 3, 6, 8, 9, 10$ and in-order traversal sequence: $4, 2, 7, 5, 1, 3, 9, 8, 10, 6$ of a binary tree, draw the binary tree.

   Suppose that all nodes in a binary tree have distinct values, given its pre-order traversal sequence and in-order traversal sequence, does there exist a different binary tree with the same pre-order traversal sequence and in-order traversal sequence? Justify your answer.

   Solution:



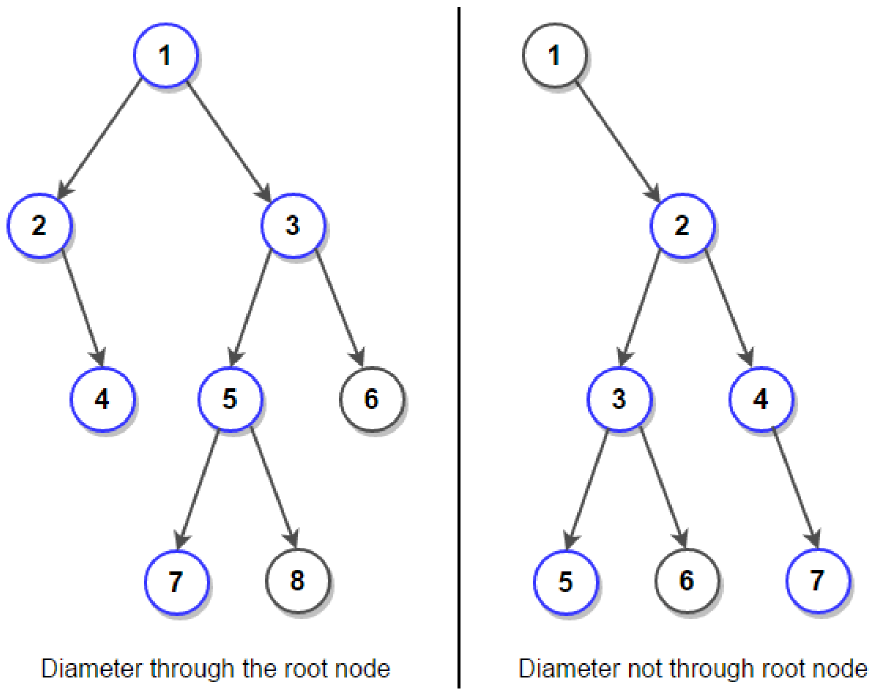
If all values in the binary tree are different, then we can uniquely recover the tree from its pre-order traversal sequence and in-order traversal sequence by recursively identifying the root and the traversal sequences of its sub-trees.

Note that if there are elements of the same value in the tree, then in some cases it is impossible to recover the tree uniquely (i.e., all elements are of the same value).

3. **[O3, O4]** (20 pts) Given a binary tree, write an efficient algorithm to compute the diameter of it. The diameter of a binary tree is equal to the number of nodes on the longest path between any two leaves of it.

   For example, below figures shows two binary trees having diameter 6 and 5 respectively(nodes highlighted in blue color.)

   What is the time complexity of your algorithm? How much extra space does your program need? Prove your claim.



Diameter through the root node Diameter not through root node

Solution: We can solve this problem in linear time by doing *post-order* traversal. We can get height of left/right subtree for every node by staring from the bottom of the tree and return the height of subtree to its parent node.

The time complexity of the solution is $O(n)$ and need $O(h)$ extra space for the call stack where h is the height of the tree.

Below is a sample solution written in *C++*. We pass *diameter* by reference to getDiameter() function (instead of returning it) and update its value within the function itself using left and right subtree height.

```
// Data structure to store a Binary Tree node
struct Node
{
  int data;
  Node *left, *right;
};


// Function to find diameter of the binary tree. Note that the function
// returns the height of the subtree rooted at given node and diameter
// is updated within the function as it is passed by reference
int getDiameter(Node* root, int &diameter) {
  // base case: tree is empty
  if (root == nullptr)
  return 0;

  // get heights of left and right subtrees
  int left_height = getDiameter(root->left, diameter);
  int right_height = getDiameter(root->right, diameter);

  // calculate diameter "through" the current node
  int current_max_diameter = left_height + right_height + 1;

  // update Maximum Diameter
  diameter = max(diameter, current_max_diameter);

  return max(left_height, right_height) + 1;
}

int getDiameter(Node* root)
{
  int diameter = 0;
  getDiameter(root, diameter);

  return diameter;
}
```

4. **[O4]** (20 pts)

   Given a binary search tree, write a function *kthSmallest*(*TreeNode*∗, *int*) in $C/C{+}{+}$ style to find the k-th smallest element in it.

   Note: You may assume k is always valid, $1 \leq k \leq$ number of BST's total elements.

```
/**
* Definition for a binary tree node.
**/

struct TreeNode {
  int val;
  TreeNode *left;
  TreeNode *right;
  TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int kthSmallest(TreeNode* root, int k) {
// you code starts here





}
```

Solution: Inorder traversal can solve this problem in time $O(h + k)$.

```
    int kthSmallest(TreeNode* root, int k) {
      TreeNode * p=root;
      Stack stack;

      // go to far left of the tree
      while(p){
```

```
    stack.push(p);
    p = p->left;
  }

  //index of the node to be visited
  int cnt = 1;

  while(!stack.empty() && cnt <= k){
    // assign p with top of stack
    p = stack.top();
    stack.pop();
    ++cnt;
    TreeNode * q = p->right;
    while(q){
      stack.push(q);
      q = q->left;
    }
  }

  return p->val;
}
```

5. **[O1]** (20 pts) Suppose a given tree T has n_1 nodes that have 1 child, n_2 nodes that have 2 children, . . . , n_m nodes that have m children and no node has more than m children, how many nodes have NO child are there in T?

   Solution:

   Let n_0 be the number of nodes in T that have no child, then the total number of nodes in T is $n_0 + n_1 + \ldots + n_m$. Since T is a tree, the total number of edges in T is $n_0 + n_1 + \ldots + n_m - 1$.

   At the same time, the total number of edges in T is $n_1 + 2n_2 + 3n_3 + \ldots + mn_m$, which implies that $n_0 = 1 + n_2 + 2n_3 + 3n_4 + \ldots + (m-1)n_m$.