CS2700 : Quiz 2 Practice Questions

Quiz 2 All

October 7, 2024

Information

- This document comprises practice questions for CS2700 Quiz 2. All topics covered in class till Oct 4th can be tested in Quiz-2.

Trees (General / Binary / BST / AVL trees)

1. True/False. A binary tree of height h can have greater than $2^{(h+1)}$ - 1 nodes.

Solution: False, as the maximum number of nodes in a binary tree of height h is given by the number of nodes in the complete binary tree of height h, which is $2^0 + 2^1 + \ldots 2^h = 2^{h+1} - 1$.

2. A binary tree can be uniquely reconstructed from which of these traversals (select all that apply):
   A. Inorder and Preorder traversals
   B. Inorder and Postorder traversals
   C. Preorder and Postorder traversals
   D. Inorder and Levelorder traversals

Solution: All but third choice. With inorder traversal, knowing the root can help us recurse into reconstructing the left and right subtrees of the root. The preorder, postorder or levelorder can tell us what the root is. But preorder and postorder alone cannot identify the tree uniquely (e.g., 1 root and 2 left child of 1, and 1 root and 2 right child of 1 – these two trees have the same pre- and post-order traversals). See also here: `https://www.cmi.ac.in/~madhavan/courses/programming08/old-notes/lecture14-13oct2008.txt`.

3. Different insertion orders of the same set S of elements can result in different binary search trees. The inorder traversal sequence of these different BSTs corresponding to S can be different – True/False.

Solution: False

4. The numbers $1, 2, \ldots, n$ are inserted in a binary search tree in some order. In the resulting tree, the right subtree of the root contains p nodes. The first number to be inserted in the tree must be ———-.

Solution: n-p, since first number inserted should have only p numbers greater than it in the list.

5. Prove that the number of leaves is one more than the number of nodes with 2 children in a binary tree.

Solution: Let n_i be the number of nodes with i children in a binary tree (e.g., n_0 is the number of leaves). Then, the number of nodes in the tree is $n_0 + n_1 + n_2$ and edges in the tree is $n_1 + 2n_2$. Since the number of edges in any tree is one less than the number of nodes, we've $n_1 + 2n_2 = n_0 + n_1 + n_2 - 1$, which implies $n_0 = n_2 + 1$.

6. Assume that a tree is represented by the leftMostChild, rightSibling representation:

```
struct TNode {
    TNode* leftMostChild;
    TNode* rightSibling;
    ElemType data;
};
```

The height of a tree is defined as the length of the longest path from the root node to a leaf node. Complete the following C++ function to compute the height of the tree. Pay special attention to the relationship between the left-most child and right siblings.

```
int computeHeight(TNode* node) {
    if (node == NULL)
        return 0;

    int childHeight = ________(node->______);

    int siblingHeight = ________(node->______);

    return max(childHeight + 1, siblingHeight);
}
```

Solution: 1) `computeHeight(node->leftMostChild)`
2) `computeHeight(node->rightSibling)`

7. Write a recursive C++ function to check if a given binary tree is a BST, and another to check if a given BST is an AVL tree. Use the following Node struct and function signature.

```
struct TreeNode
{
  int data;
  TreeNode *left;
  TreeNode *right;
};
bool isBST(TreeNode *nodep); //called as isBST(root)
bool isAVL(TreeNode *nodep); //called as isAVL(root)
```

8. Write recursive functions that take only a pointer to the root of a binary tree, T and compute:
   1. The number of nodes in T.
   2. The number of leaves in T.
   3. The number of full nodes (i.e., nodes with two children) in T.
   4. The height of T.

   (a) What is the running time of your routines?

   (b) What will the running time if you use write these functions in a non-recursive (iterative) fashion?

   (c) If T is a general tree instead of a binary tree T, how will your codes and running time analyses change?

Solution:

```
int countNodes( Node *t )
{
  if( t == NULL )
     return 0;
  return 1 + countNodes( t->left ) + countNodes( t->right );
}
int countLeaves( Node *t )
{
   if( t == NULL )
      return 0;
   else if( t->left == NULL && t->right == NULL )
      return 1;
   return countLeaves( t->left ) + countLeaves( t->right );
}

int countFullNodes( Node *t )
{
  if( t == NULL )
     return 0;
  else if(t->left==NULL || t->right==NULL)
      return countFullNodes( t->left ) + countFullNodes( t->right )
  else
      return 1 + countFullNodes( t->left ) + countFullNodes( t->right );

}

int height( Node *t )
{
  if( t == NULL )
     return -1;
  return 1 + max(height( t->left ) , height( t->right ));
}


// All routines from 1-4 take linear time O(n).
//No change in running time
```

```
        // These functions use the type Node, which can contain a list of all it
        children instead of just right and left child, and iterate through the list.
```

Stacks and Queues

9. In a stack, what is the time complexity for finding the minimum element if we implement the stack with a singly linked list and don't use any auxiliary space?
   A. $O(1)$
   B. $O(\log n)$
   C. $O(n)$
   D. $O(n \log n)$

Solution: Ans: C

If we don't use any auxiliary space (such as an additional stack to store the minimum values), finding the minimum element would require traversing the entire stack, which takes $O(n)$.

10. Which of the following applications does not typically use a stack?
    A. Reversing a string
    B. Finding the shortest path in a graph
    C. Evaluating postfix expressions
    D. Validating balanced parentheses

Solution: Ans: B

Finding the shortest path in a graph is typically done using breadth-first search (BFS), which uses a queue, not a stack.

11. Which of the following statements is false about circular array implementation of queues?
    A. The dequeue operation always removes the element at the front of the queue.
    B. Circular arrays are used to efficiently utilize memory underlying queues.
    C. A queue can be implemented using a circular array but not using a linked list data structure.
    D. The front and rear indices of the circular array wrap around when they reach the end of the array.

Solution: Ans: C

Queue can be implemented using both arrays, circular arrays and linked lists.

12. Devise an algorithm for checking balanced parentheses in an expression using a **stack ADT**. The list of parentheses includes `(, [, {, }, ], )`. Write your function using a stack to ensure that the parentheses are properly balanced.

Solution:

```
bool isMatchingPair(char char1, char char2) {
    if (char1 == '(' && char2 == ')') return true;
    if (char1 == '[' && char2 == ']') return true;
    if (char1 == '{' && char2 == '}') return true;
    return false;
}

bool areParenthesesBalanced(string expr) {
    stack<char> s;  // Create a stack to hold parentheses

    for (int i = 0; i < expr.length(); i++) {
        char c = expr[i];

        if (c == '(' || c == '[' || c == '{') {
            ______;  // Fill in the blank to push
        } else if (c == ')' || c == ']' || c == '}') {
            if (______ || !isMatchingPair(______, c)) {  // Fill in the blank for top
                return false;
            }
            ______;  // Fill in the blank to pop
        }
    }

    return s.empty();
}
```

1) `s.push(c)`

2) `s.empty()`

3) `s.top()`

4) `s.pop()`

13. A given string is very long. Assume that the given string is of length *N*. Devise an algorithm using a **stack** and a **queue** to determine whether this long string is a palindrome. Complete the following C++ function by filling in the blanks.

```
bool isPalindrome(string str) {
    stack<char> S;
    queue<char> Q;
    int length = str.length();

    for (int i = 0; i < length; i++) {
        Q.______;
        S.______;
    }

    bool flag = true;

    for (int i = 0; i < length / 2; i++) {
        if (______) {
            flag = false;
            break;
        }
        Q.pop();
        S.pop()
    }

    return flag;
}
```

Solution: 1) `Q.push(str[i])`

2) `S.push(str[i])`

3) `Q.front() != S.top()`

Final Notes

14. Please also review the conceptual as well as programming questions in Tutorial/Prep documents shared so far with the class, as well as the slides/codes shared with the class. Topics covered in class till Oct 4th are included for Quiz-2.