ICCS200: Assignment 6

Hasdin Ghogar
Collaborators: 1408
The date

1: Facts About Graphs

(a) Since each edge connects two nodes, which means for each edge there is are two nodes where each node has a degree of 1, which sums up to 2 degree. Since an edge always has two nodes the number of degrees of nodes for each edge is always 2. Therefore The sum of the vertex degrees is exactly 2*(number of edges).

(b) By proving contraposition, if G contains no cycles, then G has a vertex with degree less than 2.G is a forest if it has no cycle if it has no cycle. If N a component of G is trivial then G has a vertex of degree 0. If N is nontrivial then N is a nontrivial tree. Which implies that N has at least two end-vertices of degree 1. Therefore if G has no cycle then G has at least one vertex with degree less than 2. Then if this is true therefore if G has degree at least 2,then G contains a cycle.

2: Lets Grow a Tree

(2) Since the runtime for it's recursion is n and each time we divide it into left and right subtree therefore it's depth is logn since we do it logn time. Worst case we may be left with an extra node left which is the leaf of the tree which cannot be divided anymore. Therefore it's depth logn + 1.

```java
import java.util.Arrays;

public class MakeTree {
        public static BinaryTreeNode buildBST(int[] keys){
            Arrays.sort(keys);                  //Takes nlogn =O(nlogn).
            int medium = keys.length / 2;
            return new BinaryTreeNode(tree(keys, low: 0, high: medium - 1), keys[medium], tree(keys, low: medium + 1, high: keys.length - 1));
            //2T(n/2)   (From helper)

            //Therefore it's run time is 2*logn + nlogn = O(nlogn)
        }
        public static BinaryTreeNode tree(int[] keys, int low, int high) {
            if(low==high){
                return new BinaryTreeNode(keys[(low+high)/2]);
            }
            else{
                int mid=(low+high)/2;
                if(mid-1<low){
                    return new BinaryTreeNode( left_: null,keys[mid],tree(keys, low: mid+1,high));
                }
                if(mid+1>high){
                    return new BinaryTreeNode(tree(keys,low, high: mid-1),keys[mid], right_: null);
                }

                return new BinaryTreeNode(tree(keys,low, high: mid-1),keys[mid],tree(keys, low: mid+1,high));
            }
        }
}
```