ICCS200: Assignment 6

Vikrom Narula
vikrom.nar@gmail.com

25/06/2019

People in 1408 & 1409(not including Parm)

Exercise 1:

Proposition A:

We want to proof $\sum_{v \varepsilon V} deg_G(v) = 2|E|$

Base Case:

$$P(0) := 2n = 2 * 0 = 0 \equiv \Sigma deg(v) = 0$$

Inductive Step:

We assume that k is true from $0 \leq k \leq n$

$$P(n+1) := 2(n+1) = \sum_{v \varepsilon V} deg(v)$$

If we remove one edge we will get subgraph which can assume

$G' := \sum_{v \varepsilon V'} deg_G(v) = 2n$ We know this is true by IH.

Then we know that graph G is equal to it's subgraph plus one edge we know that each edge connects two nodes we can say

$$2n + 2 = 2(n+1) = \sum_{v \varepsilon V'} deg(v) \equiv 2(n+1) = \sum_{v \varepsilon V} deg(v)$$

Which proof for P(n+1) Hence Proved.

We can also say that, we know that each edge in the graph contribute plus two to the summation of the degree

because each edge connects two nodes and when we get the magnitude of the edges we will have

$$e_1 + e_2 + e_3 + ... + e_n \equiv +2_1 + 2_2 + 2_3 + ... + +2_n \equiv 2 * n$$

Which means the sum of all degree of all nodes is equal to the magnitude of edges times 2 Hence Proved.

Proposition B:

Proof by contradiction

If we pick one node to walk to all the edges we know there's always at least an unused edge due to each node having at least two degrees for the graph to not have a cycle we have to have an infinite amount of node or it will visit an already visited node.

Exercise 2:

$$O(n\ log(n))$$

```java
import java.util.Arrays;

public class MakeTree { // Code Runtime O(n log (n))
  public static BinaryTreeNode buildBST(int[] keys) {
    Arrays.sort(keys); // O( n * log n )
    int medium = (keys.length-1) /2; // O(1)
    return new BinaryTreeNode( // O(n * log (n))
        treeMaker(keys, 0, medium - 1), // O(n/2 * log (n/2))
        keys[medium],
        treeMaker(keys, medium + 1, keys.length - 1) // O(n/2 * log (n/2))
    );
  } // n = keys.length

  public static BinaryTreeNode treeMaker(int[] keys, int low, int high) { // T(m) = 2T(m/2) + O(m) => O(m * log (m))
    if (low == high) return new BinaryTreeNode(keys[(low + high) / 2]); // O(1)
    else {
      int medium = (low + high) / 2; // O(1)
      if (low > medium-1) return new BinaryTreeNode(
          null,
          keys[medium],
          treeMaker(keys, medium + 1, high)); // T(m/2)

      if (medium + 1 > high ) return new BinaryTreeNode(
          treeMaker(keys, low, medium - 1), // T(m/2)
          keys[medium],
          null);

      return new BinaryTreeNode(
          treeMaker(keys, low, medium - 1), // T(m/2)
          keys[medium],
          treeMaker(keys, medium + 1, high)); // T(m/2)
    }
  } // m = keys.length
}
```