Aman Patel

December 12, 2019

CSCI-B 403

Assignment 5

1. Given n nodes of a forest and their edges, describe and prove an algorithm (i.e., show correctness) that finds the number of trees in the forest. Trees can be directed or undirected.
   1. Each node of the forest will be given a Boolean attribute representing if the node has been visited throughout the traversals.
   2. The first step for this algorithm is performing a Depth-first search on an arbitrary node. This will set the value of all nodes within the first tree to true. A global variable is set to 1 to account for this tree.
      1. Depth-first search involves traversing the full path from one node to another, then backtracking to visit nodes that branched off the original path or the starting node.
   3. Once this first DFS is complete, a DFS needs to be performed on all other nodes.
   4. If the value of the root is true, do not continue with DFS and do not increment the global count. Else, continue with DFS and increment the global count.
   5. If a visited node is encountered during DFS, the starting node’s tree was counted twice, and the global count should be decremented.
   6. This process is continued until there are no more unvisited nodes. The global count is then returned.
   7. This approach works because a DFS will visit every node in one tree and no nodes from other trees. This isolates complete trees from unassigned nodes.
   8. A DFS needs to be performed on every node to ensure that each edge is accounted for. This is especially important in directed trees, where the arbitrary node selected may not have an outgoing edge.
2. Design, in the most efficient way you can, an algorithm which, given the inputs s and t, outputs the path with the largest probability of passing "I love B403" from s to t.
   1. The probabilities are independent, meaning they will be multiplied when calculating the final probability.
   2. To get the highest probability when multiplying numbers below 1, the number of multiplications should be minimized and the values that are being multiplied should be maximized.
   3. One way to solve this problem would be to find all paths from *s* to *t* and find the path with the largest probability. However, this approach is far too inefficient, especially for graphs with many nodes.
   4. Another approach would be a combination of a Greedy approach and the shortest path approach.
      1. For the Greedy portion of this approach, the edge with the greatest probability is selected, with no regard to the length of the path. If the chosen path leads to a dead-end, the path is backtracked and the next highest probability branch is selected.
      2. The shortest path between *s* and *t* is also be found, and its overall probability is calculated.
      3. The two probability values are compared and the path with greater probability is returned.
3. Given a directed graph G, give an algorithm that tests if G is acyclic. Analyze running time.
   1. Each node of graph G will be given a Boolean attribute representing if the node has been visited during the current traversal.
   2. A Depth-first search will be performed starting from an arbitrary node of the graph.
   3. The starting node will have an initial value of true, while all other nodes will have a value of false. Throughout the DFS, the previously visited node will be maintained. To begin, the previously visited node will be null.
   4. As soon as a new node is reached, its value is checked. If true, the traversal is broken, and the function returns false. Otherwise, is set to true.
   5. If at any point in the traversal a node does not have any outgoing edges, the value of it and the previously visited node are set to false. The current node is saved in a temporary variable, the current node is set to the previously visited node and the previous node is set to the temporary node. This prevents the program from traversing the same path repeatedly.
   6. This process is repeated for any remaining outgoing edges of the current node.
   7. If the starting node is reached with a value of false and a non-null previously visited node, the starting node is checked for any incoming nodes.
   8. If there are none present, return true. Else, set the node on the opposing side of the incoming edge as the starting node and reset the traversal. Maintain the original starting node and prevent the next traversal from following the original path.
   9. If all incoming nodes have been traversed, return true.
   10. Time complexity: O ()
       1. In the worst-case scenario, the graph is acyclic, and every node and edge is visited once.
       2. The best-case scenario is a cyclic edge connecting the starting node to itself.
4. GCD (Greatest Common Divisor) of two integers is the largest number that divides both of them (without remainder). Given two integers, give an algorithm that finds their GCD efficiently.
   1. The algorithm begins with the initialization of two variables: and .
   2. Set equal to the larger of the two given numbers and set equal to the other.
   3. Create a while loop with the condition:
   4. Within the loop, create a temporary variable and set it equal to . Then set equal to and set equal to the temporary variable.
   5. Outside of the while loop, return .