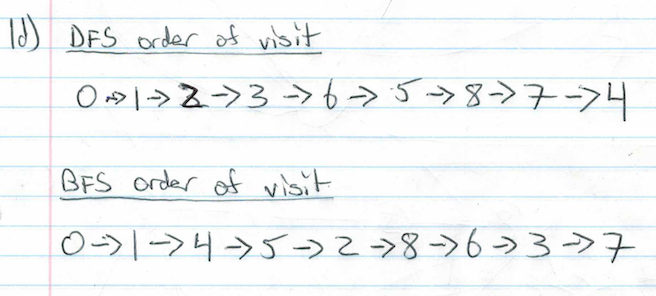
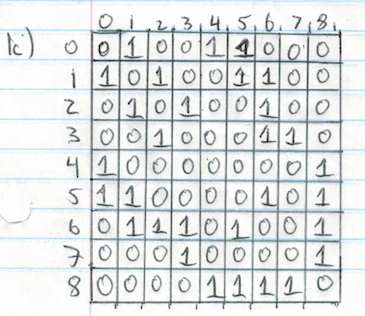
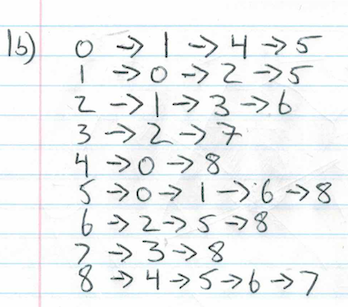
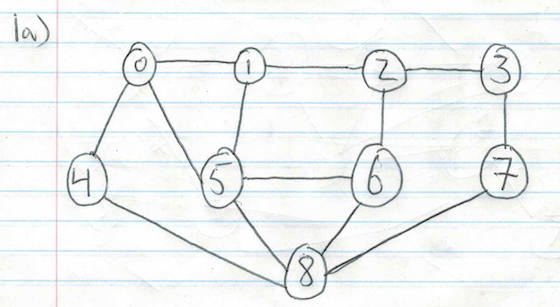
Ryan Woodward

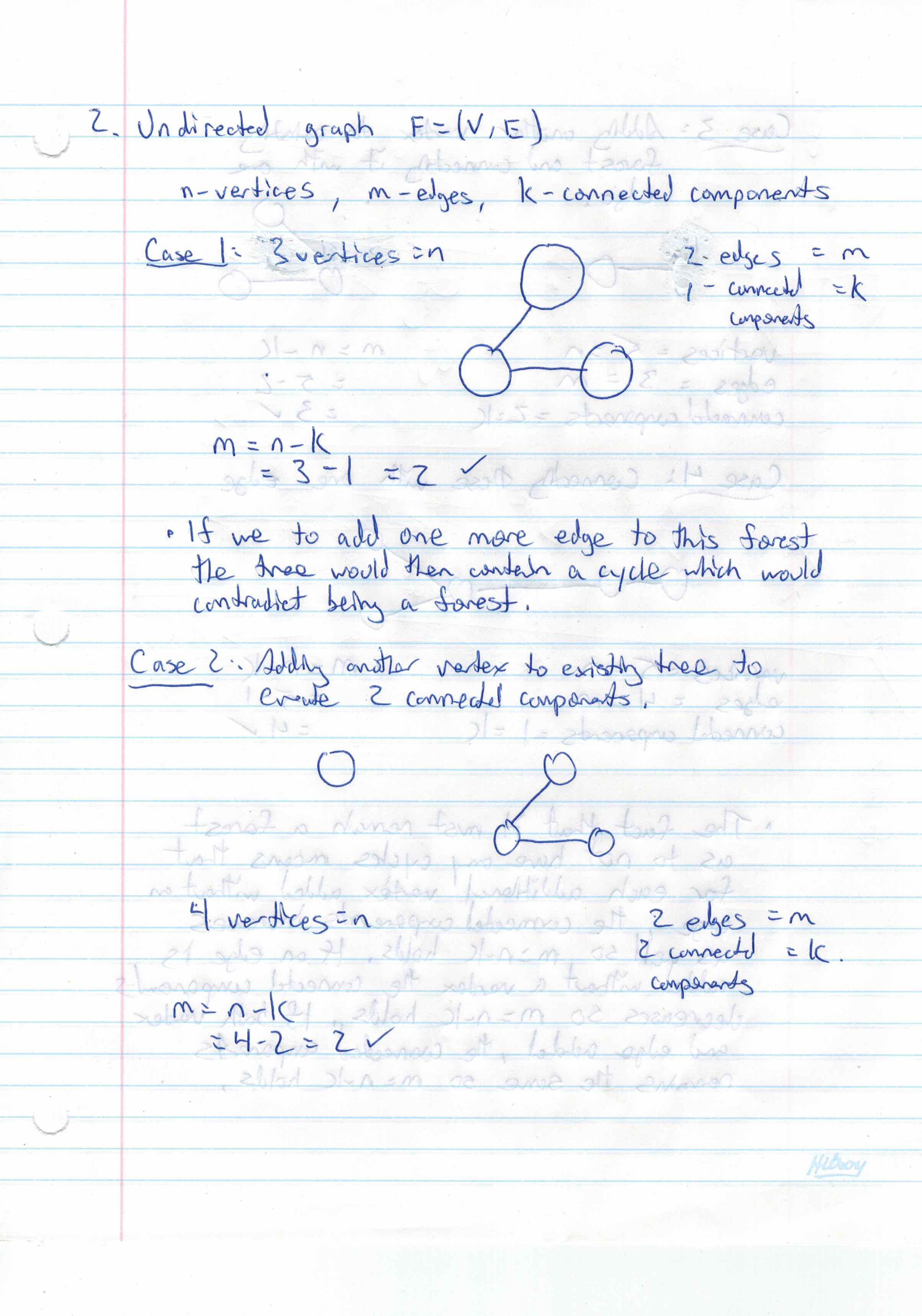
V00857268

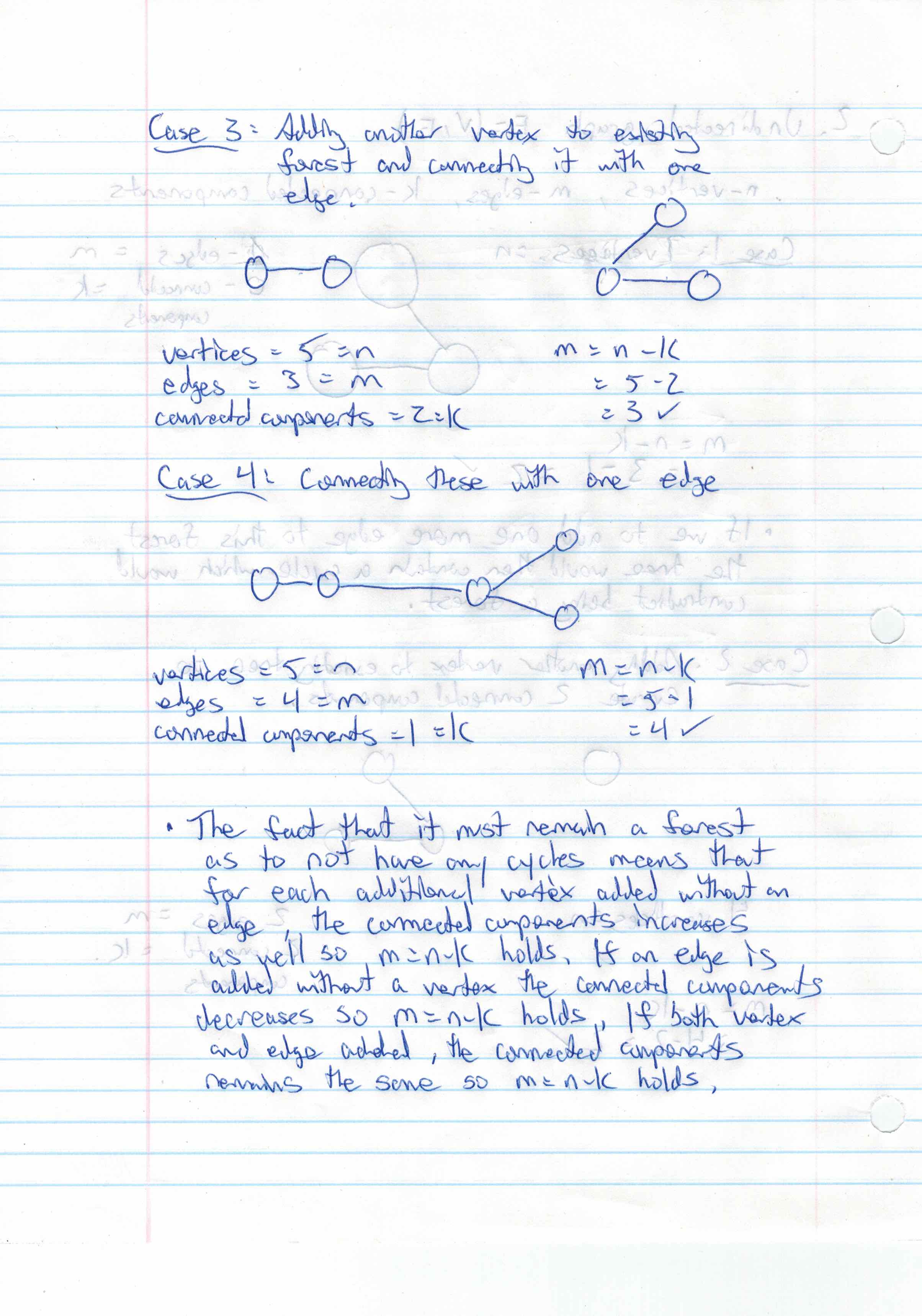
Assignment 4: Theoretical

CSC 225

Nov 24, 2016







**3a)**

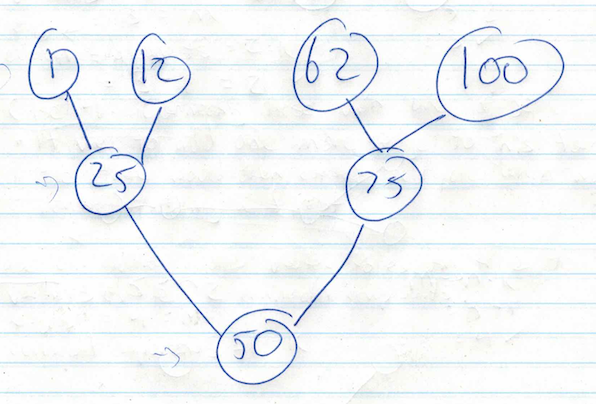
**Algorithm**: Unique Topological Sort

* + Input
    - A digraph G
  + Output
    - A list L of the ordered vertices if one exists that is unique or indication that it isn’t unique

1. L <- new empty List
2. S <- new empty Stack
3. for all u E G.vertices() do
   1. incounter(u) <- indeg(u)
   2. if incounter(u) = 0 then
      1. S.push(u)
4. i <- 1
5. while !S.isEmpty() do
   1. u < - S.pop()
      1. if S.isEmpty() then
         1. return “G has no unique topological order”
   2. add u to List as position i
   3. i < i + 1
   4. for all e E G.outIncidentEdges(u)
      1. w <- opposite(u,e)
      2. incounter(w) <- incounter(w) – 1
      3. if incounter(w) = 0 then
         1. S.push(w)
6. If i<= n then
   1. Return “G has a directed cycle”
7. return v1, v2, …., vn

**3bi)**

This is false. A post order traversal on a tree always follows the structure that it will attempt to go as left as possible on the tree until it can’t anymore. Then it will attempt to go down the right path, if it can’t do that it will go to the node.

Now if the children are directing towards the parents this is essentially like flipping the tree upside down.

If we were to being at node (1) and traverse down we would go to node (25) and then to node(50). Once there we would see that we can’t go right so we would add (50) to our traversal list. Then we would recurse to node (25) and see that we can’t go right so we would add that to our traversal list. We then would return to node (1) and see that we can’t go right anymore so we would add that to our traversal list. We would then input the next node with no incoming edges which is (12). Once again we see that we’ve already reached (25) and (50) so we put (12) on the traversal list. Then we move to (62), we move down to (75) and see that (50) is already on the traversal list. So we put (75) on the list. However this is wrong because (75) is earlier in the traversal list than lower tiered nodes then incoming (62) and (100) so it is not considered topological ordering.

Traversal List: (50) -> (25) -> (1) -> (12) -> (75).

**3bii)**

This is true. If topological ordering holds true then there should be no back edges in depth first search. In order for topological ordering to hold properly, if a back edge were to occur there is a possibility that some of the vertices would be out of order due to back edge traversal, at this point the topological ordering would then be contradictory.

**4.**

In order to find the largest group of people within a social network that are friends with other, each person would have to be friends with each other person of the group.

If this were a graph problem, let each person be a vertex, and each connection be and edge. In order to find the largest group of people we need to find the largest complete subgraph that exists within the original graph.

The input would be the original graph, and the output would be a set of complete subgraphs. Then the largest complete subgraph would represent the largest possible group of friends. By solving this graph problem we find the largest complete subgraph that has the most adjacent edges, meaning that we find the group of friends where each person is friends with each other person.