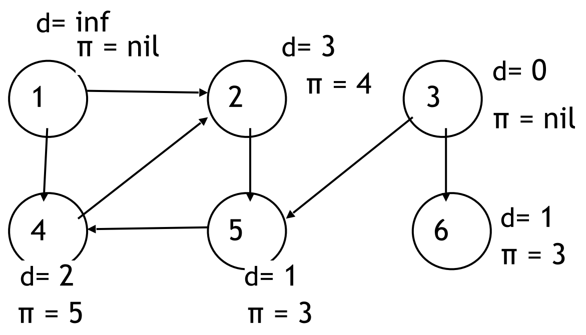
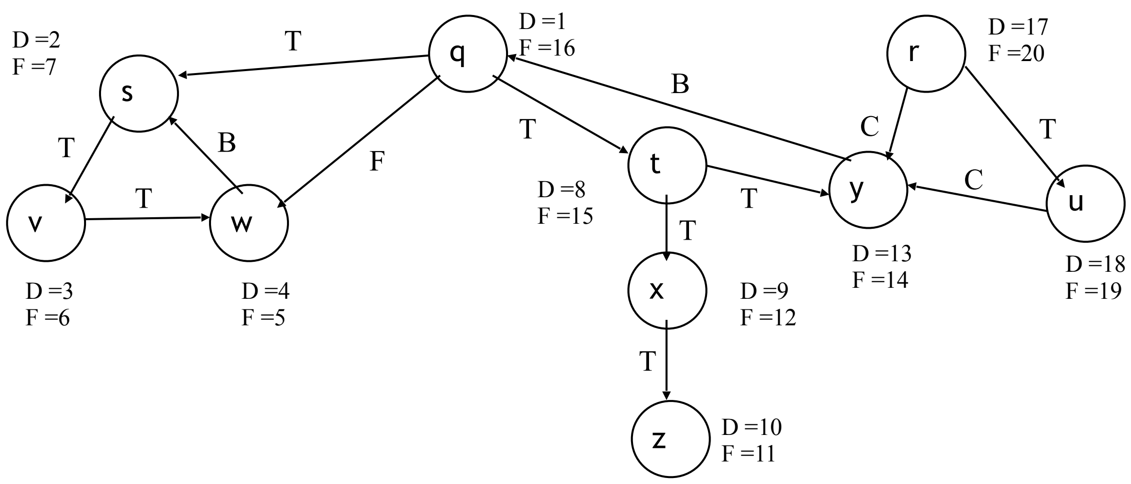
Instructions:

1. This is an individual assignment. You should do your own work. Any evidence of copying will result in a zero grade and additional penalties/actions.
2. Late submissions **will not** be accepted unless prior permission has been granted or there is a valid and verifiable excuse.
3. Think carefully; formulate your answers, and then write them out concisely using English, logic, mathematics and pseudocode (<u>no programming language syntax</u>).
4. Type your final answers in this Word document.
5. Don't turn in handwritten answers with scribbling, cross-outs, erasures, etc. If an answer is unreadable, it will earn zero points. **Neatly and cleanly handwritten submissions are acceptable**.

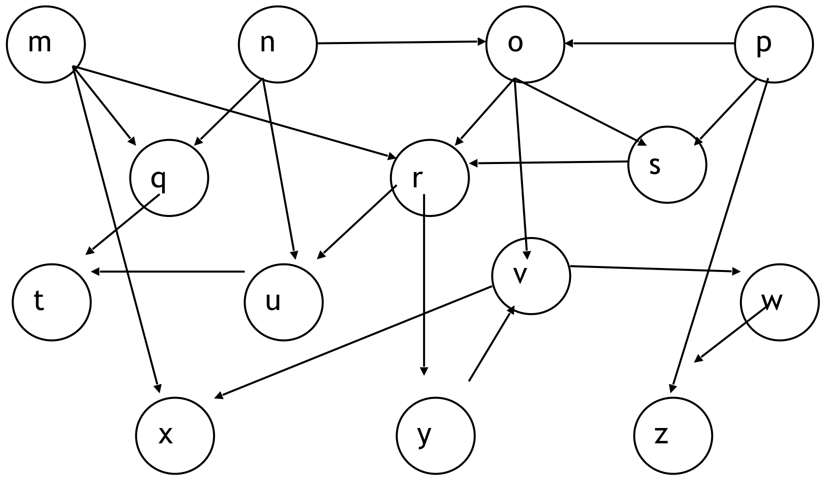
1. (15 points) Show d and π values that result from running Breadth First Search on the directed graph below using vertex 3 as the start node.



2. (10 points) Show how Depth First Search works on the graph below by marking on the graph the discovery and finishing times (d and f) for each vertex and the classification of each edge. Assume that the for loops in DFS and DFS-VISIT consider vertices alphabetically.



3. (15 points) List the vertices of the graph below in Topological Order, as produced by the Topological Sort algorithm. Assume that the for loops in DFS and DFS-VISIT consider vertices alphabetically.



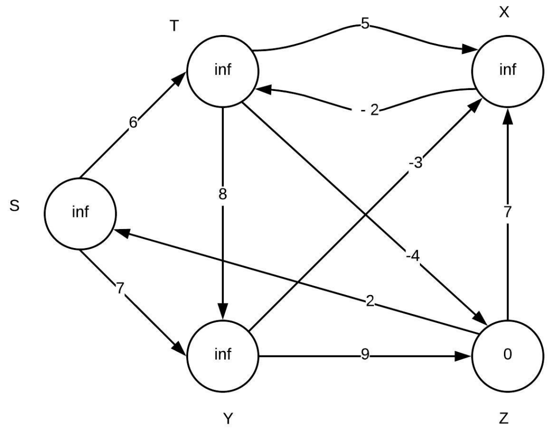
Linked list order: P->N->O->S->M->R->Y->V->X->W->Z->U->Q->T

4. (15 points) Do Problem 24.1-1 (p. 654) (you do not have to do the last part, i.e., running the algorithm again after changing an edge weight).

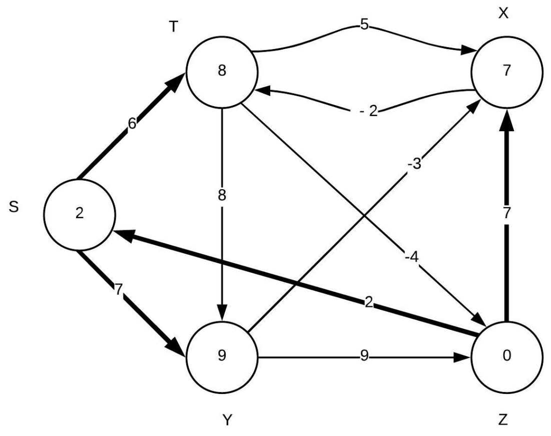
D values are maintained within the vertex; PI values are maintained using bold arrows.

relaxing in order: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

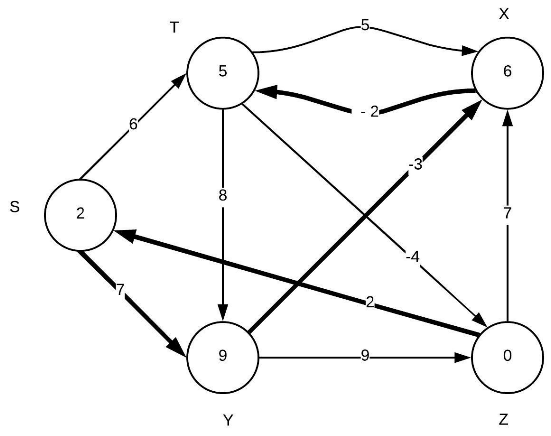
after init



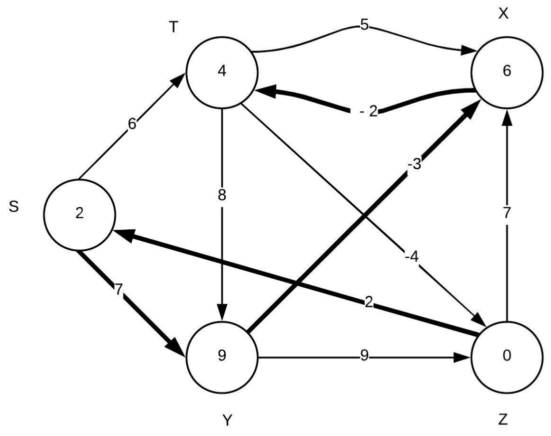
after pass 1



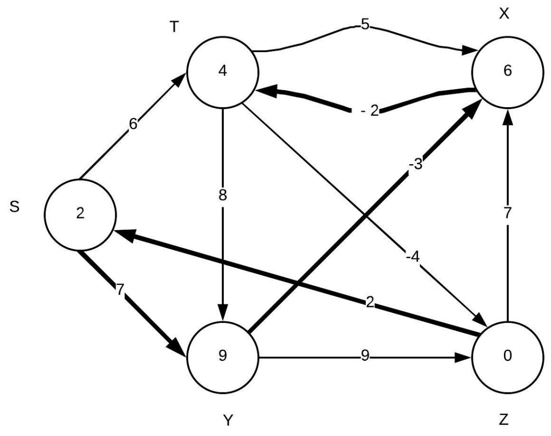
after pass 2



after pass 3

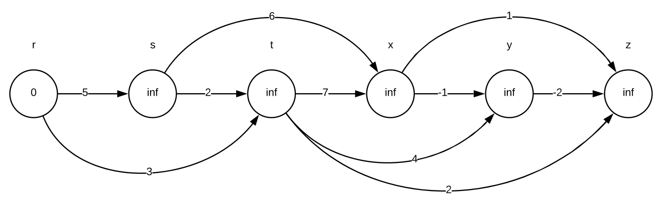


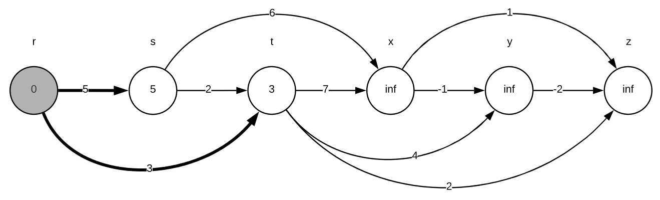
after pass 4

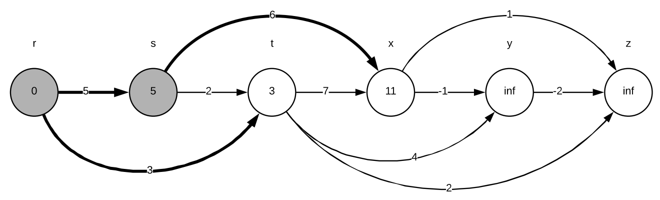


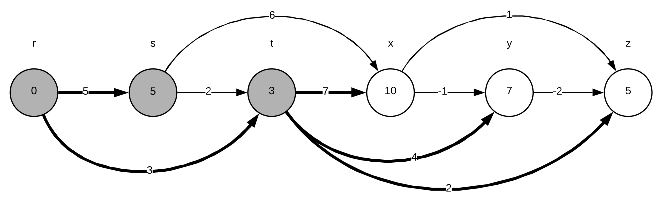
5. (15 points) Do Problem 24.2-1 (p. 657). Show the results similar to Fig. 24.5.

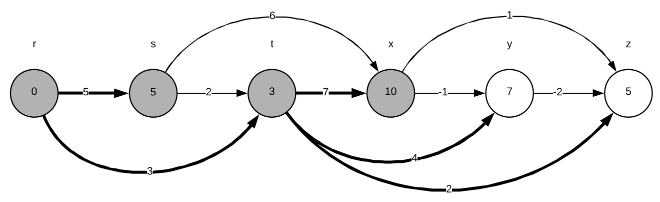
D values are maintained within the vertex; PI values are maintained using bold arrows.

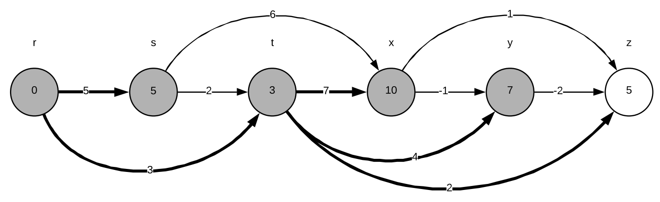


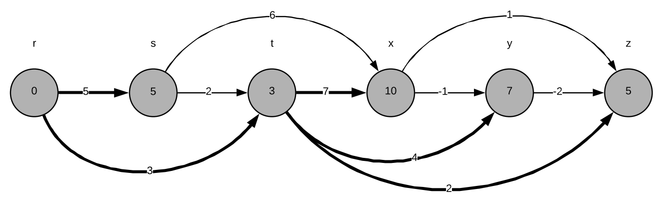






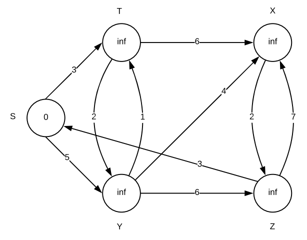


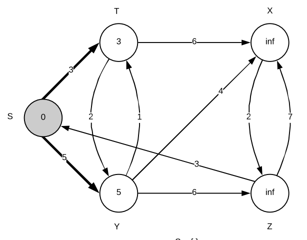




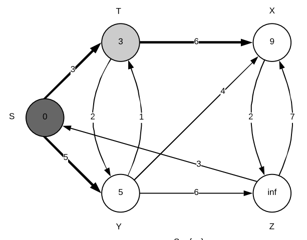
6. (20 points) Do Problem 24.3-1 (p. 662).

D values are maintained within the vertex; PI values are maintained using bold arrows.

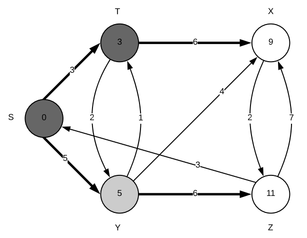




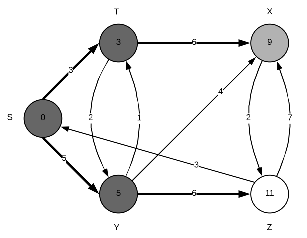
S = { }



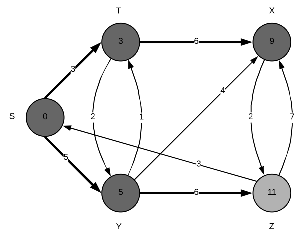
S = { s }



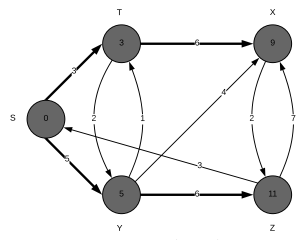
S = { s, t }



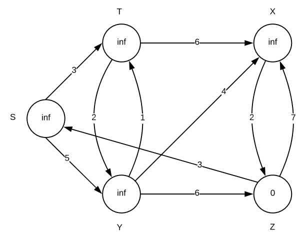
S = { s, t, y }

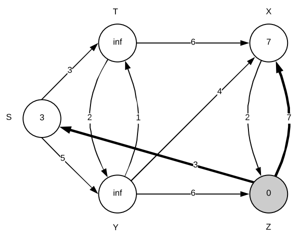


S = { s, t, y, x }

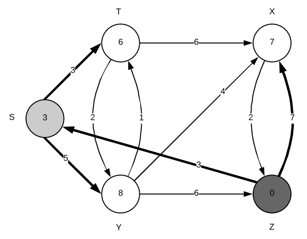


S = { s, t, y, x, z}

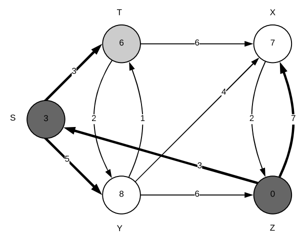




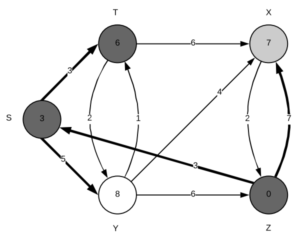
S = { }



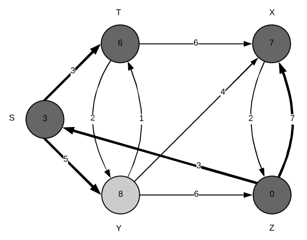
S = { z }



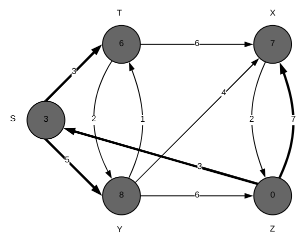
S = { z, s }



S = { z, s, t }



S = { z, s, t, x }



S = { z, s, t, x, y }

(7) (10 points) Suppose that a graph G has a Minimum Spanning Tree (MST) computed. How quickly can we update the MST if we add a new vertex and incident edges to G. Propose and outline a strategy and present an algorithm (you can reuse graph algorithms covered in class as building blocks as part of your solution) and evaluate its asymptotic complexity.

Assuming our existing MST is a valid MST, we will create an algorithm which computes a new MST. There are two basic approaches: (1) compute a new MST from scratch or (2) use the existing MST and the newly added vertex as parameters to compute an updated MST. Approach (1) is trivial (use either Prim's or Kruskal's without modification to achieve this) and approach (2) is more efficient. We will argue a solution to approach (2).

Our algorithm will take in two parameters, the existing MST and the new vertex, v. We will compute the new MST by adding all of v's edges to the graph. This will purposefully cause multiple cycles. We will then use DFS to detect the cycle and break its heaviest link. This will not only retain all the vertices in the MST (including v) but also account for the case when adding v will cause another vertex in the MST to be accessible using a path with a lesser weight.

A cycle can be detected in DFS using colored vertex. A white vertex indicates that a vertex that has not yet been visited. A gray vertex indicates we have visited it but it has not finished. A black vertex indicates that it and all its decedents have been visited and finished. While in DFS-Visit, if we encounter a gray vertex then we have traversed a back edge and therefore we have detected our cycle.

Algorithm:

1. Add all of v's edges to the MST.
2. Use a modified version DFS which detects all cycles.
3. For each cycle found, remove the edge with the heaviest weight.
4. If a cycle is found, remove the edge with the heaviest weight in the cycle.
5. Return the new MST.

The runtime complexity of this approach is dominated by DFS. DFS is $\theta(|V| + |E|)$. In addition to calling DFS once, we are also iterating over all the cycles we detect. However, it is obvious that the number of cycles in a graph is always less than the sum of that graph's edges and vertices. Therefore the runtime cost of our algorithm is $\theta(|V| + |E|)$. This is faster than Kruskal or Prim's (which are both $O(|E| lg |V|)$) therefore it is an ideal solution.