Due: Thu May 9th at the start of class

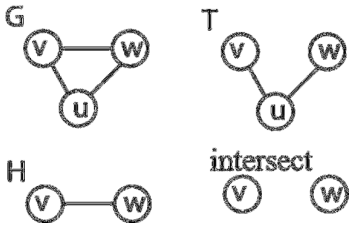
Homework #5

CMSC351 - Spring 2013

PRINT Name _____:

- *Grades depend on neatness and clarity.*
- *Write your answers with enough detail about your approach and concepts used, so that the grader will be able to understand it easily. You should ALWAYS prove the correctness of your algorithms either directly or by referring to a proof in the book.*
- *Write your answers in the spaces provided. If needed, attach other pages.*
- *The grades would be out of 16. Four problems would be selected and everyone's grade would be based only on those problems. You will also get 4 bonus points for trying to solve all problems.*

1. [Prob 7.2, pg. 248] Let $G = (V, E)$ be a connected, undirected graph, and let T be a DFS tree of G rooted at v.
   a. Let H be an arbitrary induced subgraph of G. Show that the intersection of H and T is not necessarily a spanning tree of H.



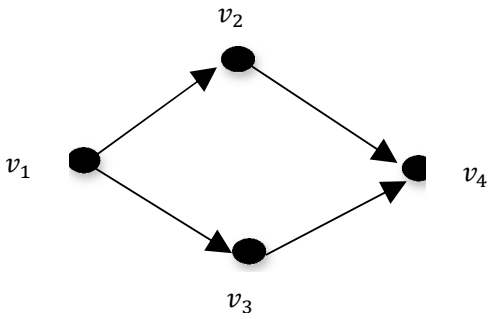
   b. Let R be a subtree of T, and let S be the subgraph of G induced by the vertices in R. Prove that R could be a DFS tree of S.

Let *u*, be the vertex in *R* that is closest to root *v* in *T*, and take that as the root of *R*. Any edge in *S* not in *R*, is an edge in *G* not in *T* and therefore a back-edge (connects a vertex to an ancestor in *T*). By construction, this is back-edge in *R* as well. Therefore subtree *R* rooted at *u*, is a DFS of *S*.

2. Let G be a directed acyclic graph. Does G have a unique topological ordering? If so, prove it or else give an example of a directed acyclic graph having at least two topological orderings.

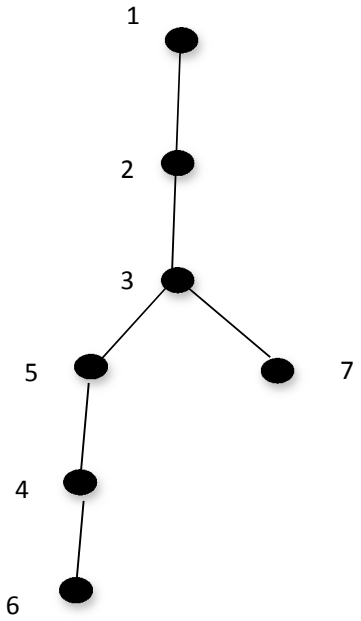
Consider the following directed acyclic graph. It has 2 topological orderings:

- v_1, v_2, v_3, v_4
- v_1, v_3, v_2, v_4

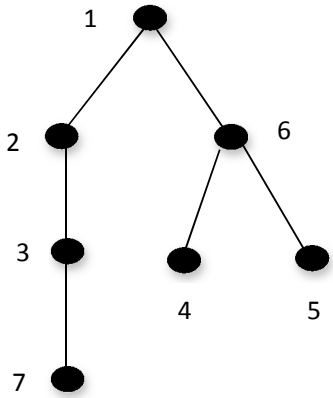


3. Consider the undirected graph below which is represented by its adjacency matrix.
    a. Run the DFS algorithm starting from vertex 1, and draw the DFS tree.
    b. Run the BFS algorithm starting from vertex 1, and draw the BFS tree.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

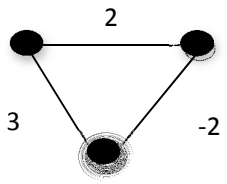


DFS Tree



BFS Tree

4. Give an example that the Dijkstra algorithm is not working properly with negative edge weights. The graph should not contain a negative cycle (a negative cycle is a cycle which the sum of the weights of its edges is negative). Briefly explain why Dijkstra does not work in your example.



Starting from the leftmost vertex, the distance of the rightmost vertex in Dijkstra would be 2. However, the length of the shortest path to the rightmost vertex is actually 1.

5. Consider the weighted graph below which is represented by its adjacency matrix.
    a. Run the Dijkstra algorithm starting from vertex 1. Write the vertices in the order which they are marked.
    b. Run the Prim's algorithm starting from vertex 1. Again write the vertices in the order which they are marked.

$$\begin{bmatrix} 0 & 5 & 0 & 0 & 8 & 0 & 6 \\ 5 & 0 & 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 3 \\ 0 & 7 & 0 & 0 & 2 & 0 & 0 \\ 8 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 4 \\ 6 & 0 & 3 & 0 & 0 & 4 & 0 \end{bmatrix}$$

Prim: 1, 2, 7, 3, 6, 4, 5

Dijkstra: 1, 2, 7, 5, 3, 4, 6

6. Recall that given a graph G, the Bellman-Ford algorithm runs for $n-1$ iterations, where n is the number of vertices in G. Prove that G contains a negative cycle, if and only if at least one edge gets relaxed if we run the algorithm for one extra iteration.

```
procedure BellmanFord(list vertices, list edges, vertex source)
   // Step 1: initialize graph
   for each vertex v in vertices:
       if v is source then v.distance := 0
       else v.distance := infinity

   // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge uv in edges: // Try relaxing the edge from u to v
           u := uv.source
           v := uv.destination
           if u.distance + uv.weight < v.distance: // uv gets relaxed
               v.distance := u.distance + uv.weight
```

You can prove the problem by showing that this lemma holds (you can use induction):

Lemma. After *i* repetitions of *for* cycle:

- If "u.distance" is not infinity, it is equal to the length of some path from *s* to *u*;
- If there is a path from *s* to *u* with at most *i* edges, then "u.distance" is at most the length of the shortest path from *s* to *u* with at most *i* edges.

See http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm#Proof_of_correctness for more details.

7. Suppose that $G = (V, E)$ is a connected graph with weights on the edges. Show that the following algorithm gives a minimum cost spanning tree of G:
    - Arrange the edges in E in decreasing order of weights
    - Traverse the edges according this order
    - For each edge, keep the edge if its deletion disconnects the graph. Otherwise delete the edge.

    Hint: The proof of correctness is similar to the one given in Section 7.6 of the book.

    With small perturbations, we can assume that all edge costs are distinct. The following lemma tells us which edges can be guaranteed to not be in a MCST.

    Lemma: Let C be a cycle in G and let e be the most expensive edge in C. Then e does not belong to any MCST of G.
    Proof: Suppose there is a MCST say T such that $e \in T$. Let the edge $e = \{v, w\}$. Then we have that $T - e$ divides G into two components, say S and V-S. If we follow the cycle C, then get a path P from v to w. Hence this path contains an edge $e' \in C$ such that e' has one endpoint in S and the other in V-S. Consider $T' = (T \setminus e) \cup e'$. It is easy to see that T' is indeed a spanning tree of G, and it has less cost than that of T since e was the most expensive edge in C. This is a contradiction since T was a MCST of G.

    Now, the proof follows by induction since the only edges that we delete are those whose deletion does not disconnect the graphs, i..e, those edges which belong to a cycle. By the Lemma, these edges do not belong to **any** MCST and hence it is safe to delete them.

8. A Hamiltonian path in graph G=(V,E) is a simple path that includes every vertex in V. Design an algorithm that runs in O(n+m) time, to determine if a Hamiltonian path exists in a given directed acyclic graph.

Perform a topological sort of the DAG, then check if successive vertices in the sort are connected in the graph. If so, the topological sort gives a Hamiltonian path. On the other hand, if there is a Hamiltonian path, then the path gives a topological sort of the DAG.

Remark: Problem 2 states that a given directed acyclic graph may have many topological orderings. Argue why is it enough to check any single topological ordering for paths between successive vertices.

9. [Prob. 7.61, pg. 256]. Let G=(V,E) be a connected, undirected graph, and let *T* be a minimum cost spanning tree of G. Suppose the cost of one edge *e* in G is changed. Discuss the conditions in which *T* is no longer a MCST of G. Design an efficient algorithm to either find a new MCST or to determine that *T* is still an MCST. (*e* may or may not belong to *T*).

Suppose *e*=(u,v) is in T, then *T* may no longer be an MCST if the cost of *e* becomes larger than the cost of an edge not in T that is in any path between u and v. To find an MCST *e* is removed such that now *T* is disconnected into trees T_1 and T_2 (you can show that these are MCSTs for the corresponding induced subgraphs), then the minimum cost edge joining a vertex in T_1 to a vertex in T_2 is added to make the MCST.

Suppose *e=(u,v)* is *not* in T, then *T* may not be an MCST if the cost of *e* becomes smaller than the largest cost in the path between *u* and *v* in *T.* In this case, this edge is removed and new edge is added to *T* as before.