CS352 (Winter 2018) - Homework 5

Marc Tibbs (tibbsm@oregonstate.edu)

Due Date: February 18, 2018

Problem 1: *(3 points)*

Demonstrate Prim's algorithm on the graph below by showing the steps in subsequent graphs as shown in Figures 23.5 on page 635 of the text. What is the weight of the minimum spanning tree? Start at vertex a.

The weight of the minimum spanning tree is 32. Please see Appendix A for the graphs showing the steps Prim's algorithm would take.

Problem 2: *(6 points)*

Consider an undirected graph G=(V,E) with nonnegative edge weights w(u,v)≥0. Suppose that you have computed a minimum spanning tree G, and that you have also computed shortest paths to all vertices from vertex s∈V. Now suppose each edge weight is increased by 1: the new weights w'(u,v) = w(u,v) + 1.

(a) Does the minimum spanning tree change? Give an example it changes or prove it cannot change.

The minimum spanning tree would not change. Since the all weights are increased by the same amount, Prim's algorithm would still traverse the tree in the same order and the same minimum spanning tree would be created with the new weights. Consider a graph with vertices A, B, C, and D. The edge weight for AB = 1, AC = 2, CD = 3, DB = 4. Starting from A the minimum spanning tree would be created by (1) Adding AB to the tree (2) adding AC (3) adding CD. If you added one to each edge weight the minimum spanning tree would be created in the exact same manner since AB < AC < CD < DB still holds true.

(b) Do the shortest paths change? Give an example where they change or prove they cannot change.

The shortest path might change after a change in the edge weights. Consider a graph with vertices A, B, C, and D. The edge weight for AB = 1 and the edge weights for AC, CD, and DB = 0. The shortest path for this graph then is A, C, D, B. However, if all weights were increased by one so that AB = 2, and AC, CD, and DB = 1, then the shortest path for the graph would be A, B and not A, C, D, B.

Problem 3: *(4 points)*

In the bottleneck-path problem, you are given a graph G with edge weights, two vertices s and t and a particular weight W; your goal is to find a path from s to t in which every edge has at least weight W.

(a) Describe an efficient algorithm to solve this problem.

There are two approaches for this problem. If you are simply looking for any path between the vertices s and t, then you could use a modified version of a BFS algorithm that ignored all edges with weights less than W. However, if you are looking for the shortest path between the two vertices, you would have to use a modified version of Dijkstra's algorithm that ignores all edges with weights less than W. Since the question does not specify shortest path, I will only show the modified pseudocode for the BFS algorithm.

```
BFS(G,s) {
    initialize vertices;
    Q = {s};
    while(!Q.empty()) {
        u = DEQUEUE(Q);
        for each v ∈ G.Adj[u] {
            if (v.color == WHITE && distance between u and v ≥ W) {
                v.color = GREY;
                v.d = u.d+1;
                v.p = u;
                ENQUEUE(Q, v)
        }
        u.color = BLACK;
    }
}
```

(b) What is the running time of your algorithm?

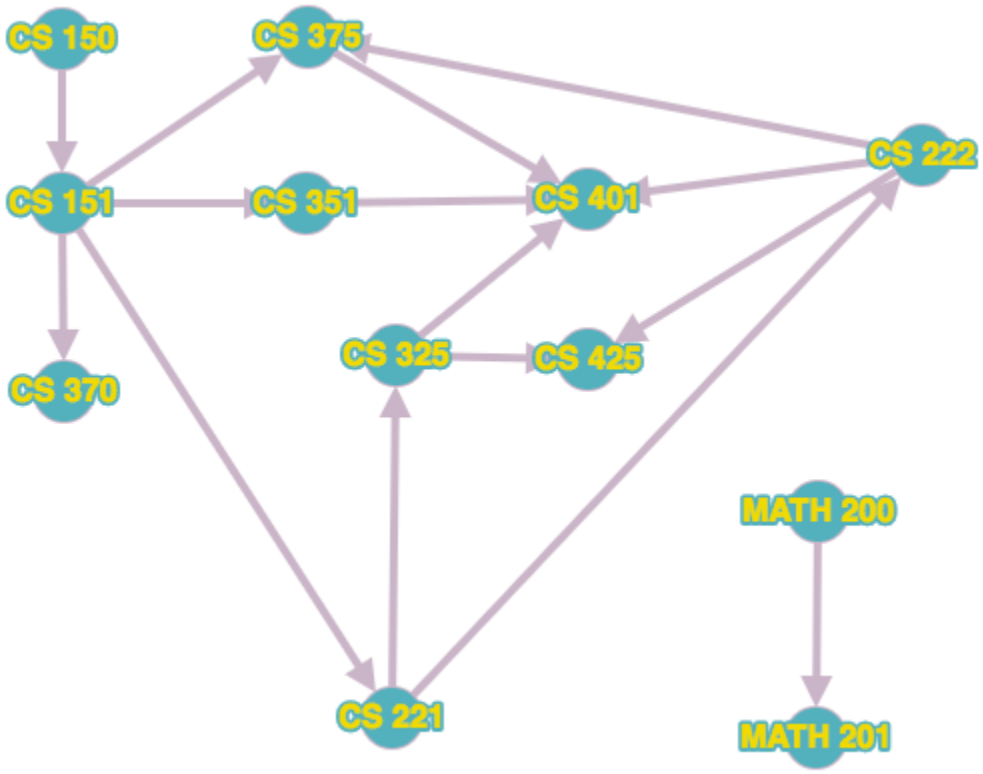
The running time of the algorithm to find a path between s and t would be $\mathcal{O}(E + V)$ using a modified version of a BFS algorithm. If you are interested in finding the shortest path in between the to vertices, then you would need to use Dijkstra's algorithm so the runtime would be $\mathcal{O}(E\ lg\ V)$.

Problem 4: *(5 points)*

Below is a list of courses and prerequisites for a factious CS degree.

Course	Prerequisite
CS 150	None
CS 151	CS 150
CS 221	CS 151
CS 222	CS 221
CS 325	CS 221
CS 351	CS 151
CS 370	CS 151
CS 375	CS 151, CS 222
CS 401	CS 375, CS 351, CS 325, CS 222
CS 425	CS 325, CS 222
MATH 200	None
MATH 201	MATH 200

(a) Draw a directed acyclic graph (DAG) that represents the precedence among the courses.



(b) Give a topological sort of the graph.

MATH 200, MATH 201, CS 150, CS 151, CS 351, CS 221, CS 375, CS 325, CS 222, CS 425, CS 401

(c) If you are allowed to take multiple courses at one time as long as there is no prerequisite conflict, find an order in which all the classes can be taken in the fewest number of terms.

Term 1: CS 150, MATH 200
Term 2: CS 151, MATH 201
Term 3: CS 370, CS 221, CS351
Term 4: CS 325, CS 222
Term 5: CS 375, CS 425
Term 6: CS 401

(d) Determine the length of the longest path in the DAG. How did you find it? What does this represent?

The longest path in the DAG is CS150, CS151, CS221, CS222, CS375, CS401 with a weight of 5. To find the longest path you need to keep track of the prerequisite class for each class. Then starting with the class with the smallest number of prerequisite classes you pop it off and record the distance it took to get there. Before you pop it off you would give the max(parent distance + 1, child distance). In this way you would get the longest paths to each vertex where the longest path represents the minimum number of steps required to list all vertices in the graph.

Problem 5: *(12 points)*

Suppose there are two types of professional wrestlers: "Babyfaces" ("good guys") and "Heels" ("bad guys"). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n wrestlers and we have a list of r pairs of rivalries.

(a) Give pseudocode for an efficient algorithm that determines whether it is possible to designate some of the wrestlers as Babyfaces and the remainder as Heels such that each rivalry is between a Babyface and a Heel. If it is possible to perform such a designation, your algorithm should produce it.

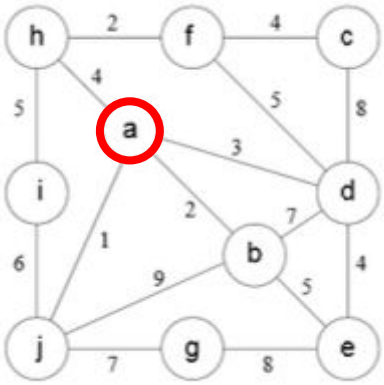
```
BFS(G,s) {
initialize vertices;
Q = {s};
s.type = 'BABYFACE';
s.d = 0;

for each r ∈ rivals{
    r.d = s.d + 1;
    r.type = 'HEEL';
    QUEUE(r);
}

while(!Q.empty()) {
    u = DEQUEUE(Q);

    for each v ∈ rivals[u] {
        if (v.color == WHITE) {
            if (v.d > u.d + 1) {
                v.d = u.d + 1                }
            if (v.d is EVEN) }
                v.type = 'BABYFACE'                }
            else }
```

```
                v.type = 'HEEL'
            if (v.type == u.type) }
                "Matchup not possible"
            }
        }
        u.color = BLACK;
    }
```

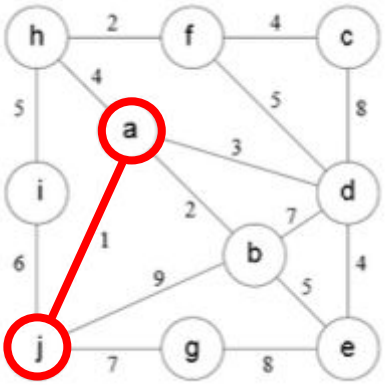
(b) What is the running time of your algorithm?
The running time for the algorithm would be $\mathcal{O}$(V+E) because it uses BFS.

(c) Implement: Babyfaces vs Heels.
A copy of my files including a README file that explains how to compile and run my code has been submitted in a ZIP file to TEACH.

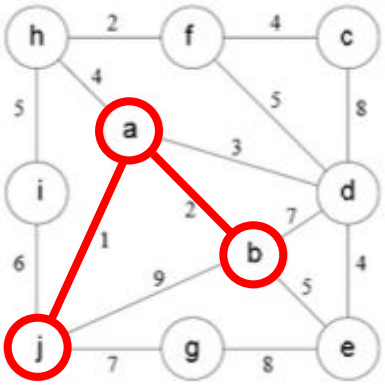
Appendix A: Problem 1 - Minimum Spanning Tree



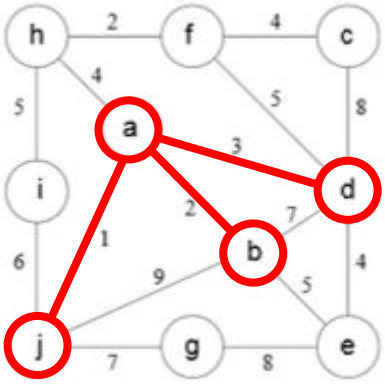
(a)



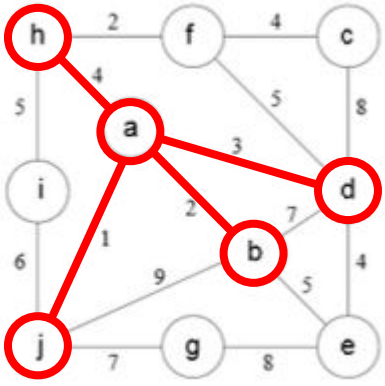
(b)



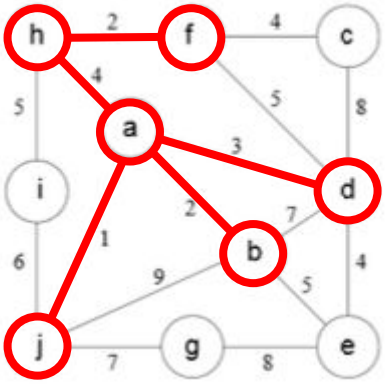
(c)



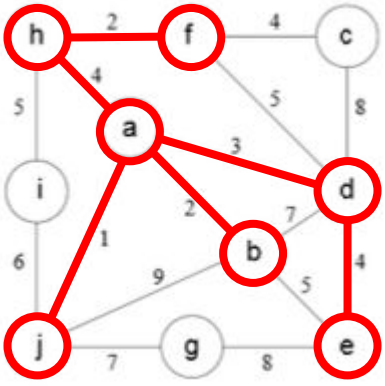
(d)



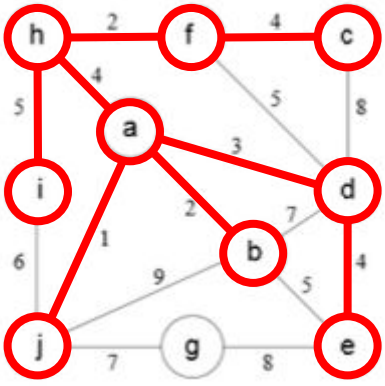
(e)



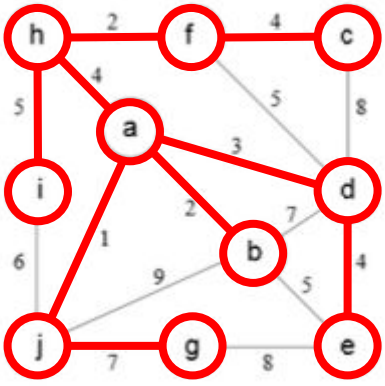
(f)



(g)



(h)



(i)