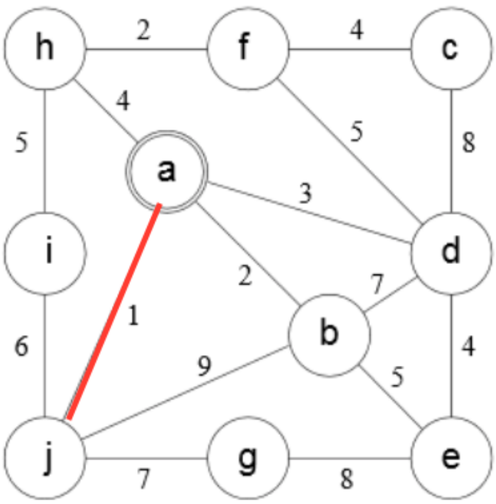
Class: CS-325
Term: Fall 2017
Author: Jon-Eric Cook
Date: November 5, 2017
Homework: #5

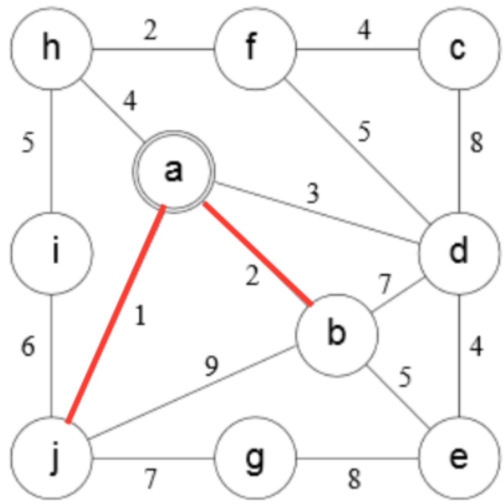
Problem 1: (3 points)

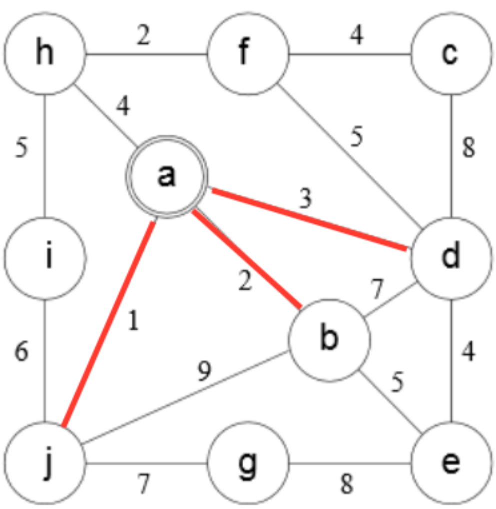
Demonstrate Prim's algorithm on the graph below by showing the steps in subsequent graphs as shown in Figure 23.5 on page 635 of the text. What is the weight of the minimum spanning tree? Start at vertex a.

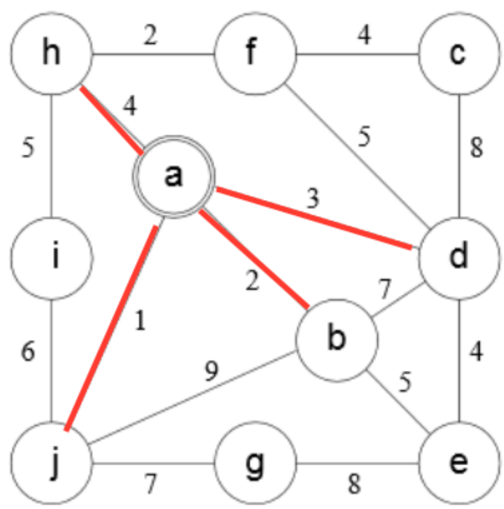
Answer:

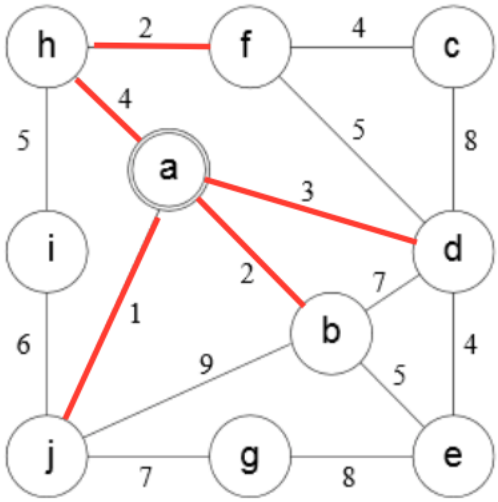
See screen shots below for steps. The weight of the minimum spanning tree is 32.

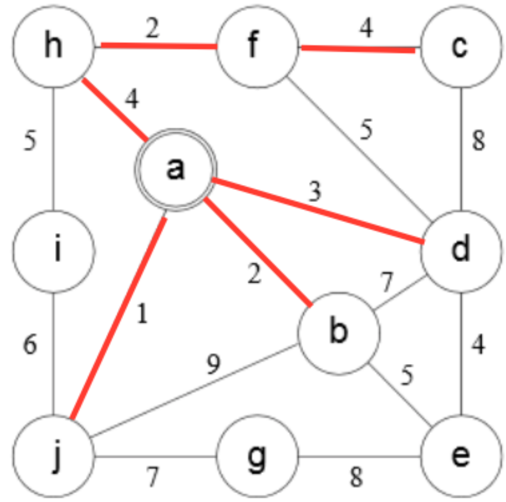


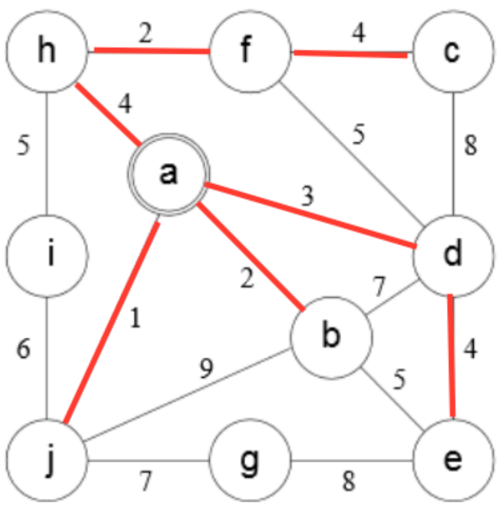


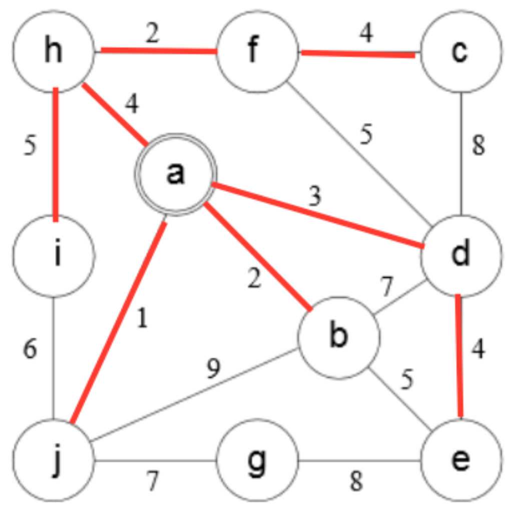


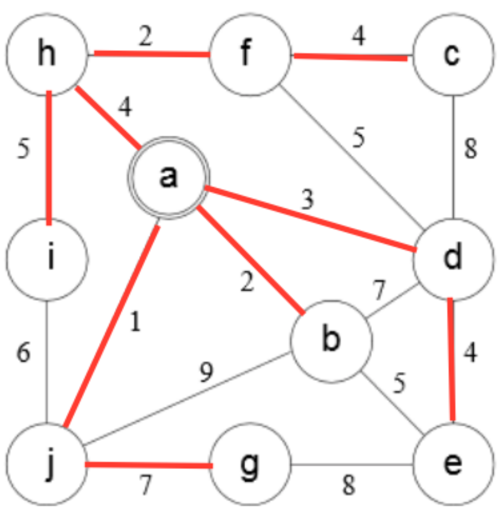



h
2
f
4
c
4
5
a
5
3
8
i
d
2
7
1
b
6
9
5
4
j
g
e
7
8


h
2
f
4
c
4
5
a
5
3
8
i
d
2
7
1
b
6
9
5
4
j
g
e
7
8


h
2
f
4
c
4
5
a
5
3
8
i
d
2
7
1
b
6
9
5
4
j
g
e
7
8


h
2
f
4
c
4
5
a
5
3
8
i
d
2
7
1
b
6
9
5
4
j
g
e
7
8


h
2
f
4
c
4
5
a
5
3
8
i
d
2
7
1
b
6
9
5
4
j
g
e
7
8

Problem 2: (6 points)

Consider an undirected graph G=(V,E) with nonnegative edge weights w(u,v)>=0. Suppose that you have competed 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.

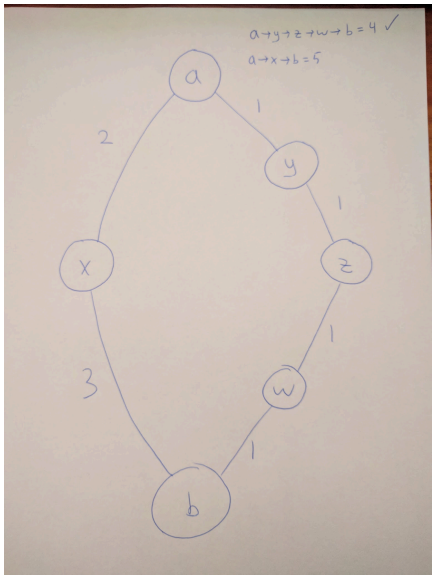
a) Does the minimum spanning tree change? Give an example it changes or prove it cannot change.

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

Answer:

a) No, the minimum spanning tree does not change. Seeing how Kruskal's algorithm was proven to be correct in a class lecture, lets use it. When it runs, it will build the MST by taking a certain path through the undirected graph. This path consists of the lowest weights. Due to the fact that the edge weights are distinct, the algorithm will follow the same path each time it is run on a given tree. By simply adding 1 to each weight, it does not change the path the algorithm would take. Therefore, the MST will not change.

b) Yes, the shortest path may change. This could happen because the length of a path is determined by the number of edges and the value of each edge. Suppose in the tree G, there is a path between points a and b that has 4 edges and results in a length of 4. Also suppose that another possible path is one that consists of two edges and results in a length of 5. Clearly the path with the length of 4 is the shortest. By increasing each edge by 1, the shortest path would now have a length of 8 and the previously rejected path would have a length of 7. As a result, there would be a new shortest path.



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.

b) What is the running time of your algorithm?

Answer:

a) An efficient algorithm that would solve this problem would be a modified BFS. The modification would come by ignoring edges with a weight value less than W.

b) The running time would be O(V+E).

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.

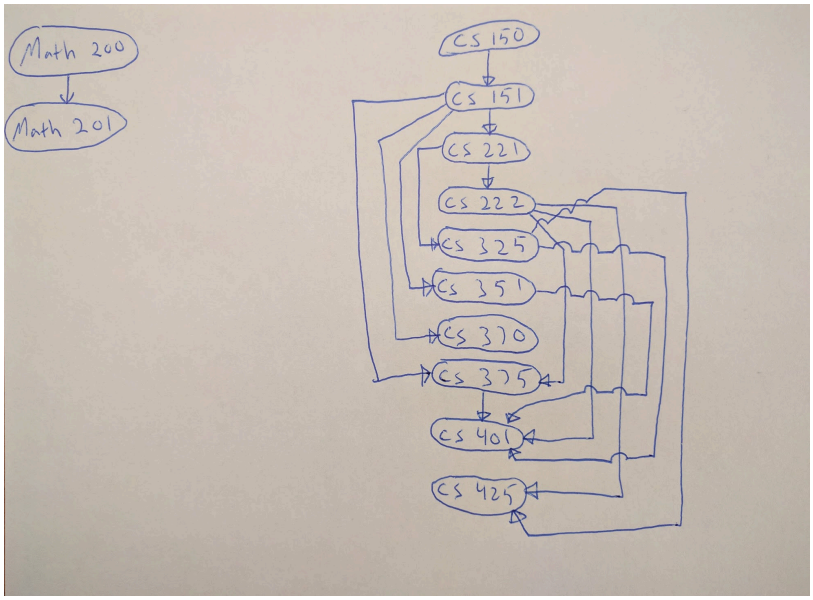
b) Give a topological sort of the graph.

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.

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

Answer:

a) See picture below.



b) A possible topological sort of the graph could be as follows:

It's important to note that the order below makes sure that a course's prerequisite precedes it.

Math 200(21,24)

Math 201(22,23)

CS 150(1,20)

CS 151(2,19)

CS 221(3,18)

CS 222(4,17)

CS 325(9,10)

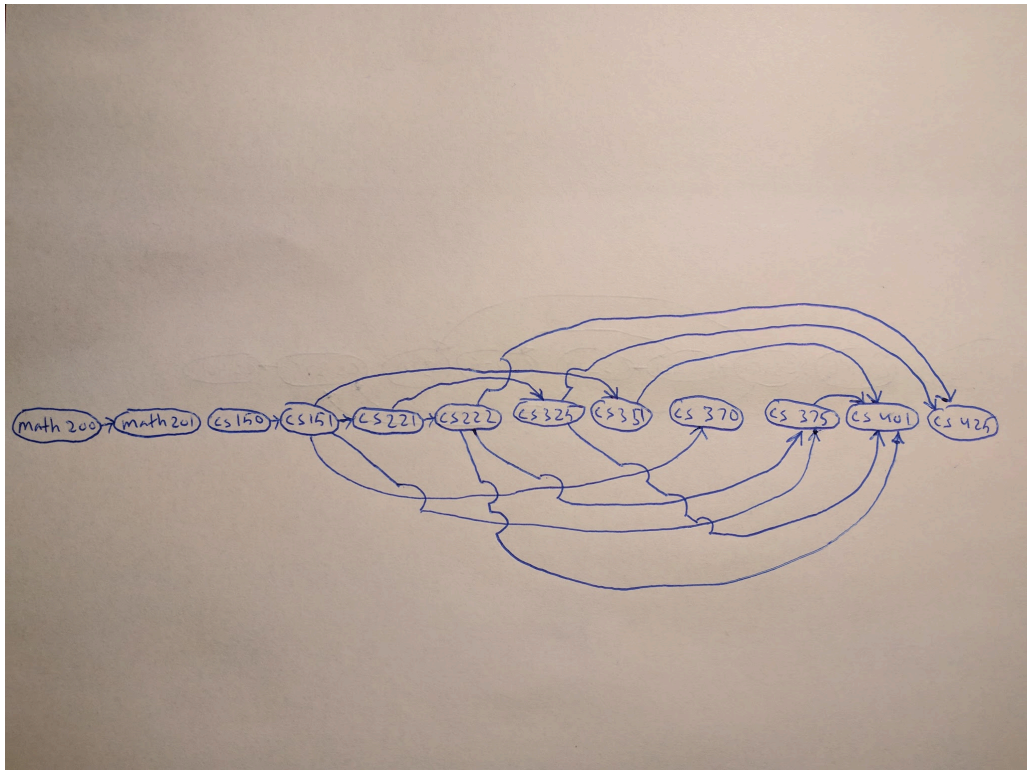
CS 351(11,12)

CS 370(13,14)

CS 375(15,16

CS 401(7,8)

CS 425(5,6)



c) Below is a layout of classes that would result in requiring the fewest number of terms.

Term 1

Math 200

CS 150

Term 2

Math 201

CS 151

Term 3

CS 221

CS 351

CS 370

Term 4

CS 222

CS 325

Term 5

CS 375

CS 425

Term 6

CS 401

d) The length of the longest path in the DAG is 5. I found this by looking through the graph and counting the length of each path through the graph. The longest path represents the longest continuous list of prerequisites. Below is a list of classes that make up the longest path:

CS 150

CS 151

CS 221

CS 222

CS 375

CS 401

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.

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

(c) Implement: Babyfaces vs Heels.

Input: Input is read in from a file specified in the command line at run time. The file contains the number of wrestlers, n, followed by their names, the number of rivalries r and rivalries listed in pairs. *Note: The file only contains one list of rivalries*

Output: Results are outputted to the terminal.

-Yes, if possible followed by a list of the Babyface wrestlers and a list of the Heels.

-No, if impossible.

Sample Input file:

5

Ace

Duke

Jax

Biggs

Stone

6

Ace Duke

Ace Biggs

Jax Duke

Stone Biggs

Stone Duke

Biggs Jax

Sample Output:

Yes

Babyfaces: Ace Jax Stone

Heels: Biggs Duke

Answer:

a)

Designation can work

```
g = newGraph(inputfile)
BFS(g,first vertex)
        setup vertices
        queue = {first vertex}
        while (queue is not empty)
                node = queue.pop()
                for each neighbor in g.getNeighbors()
                        if neighbor was not visited
                                mark as visited
                        if neighbor's name is even
                                put on team babyface
                        else
                                put on team heel
                        if node and neighbor are on the same team
                                designation cant work
If designation cant work
        print not possible
Else
        print teams
```

b)

The running time of my algorithm would be O(n+r). This is due to the BFS.

c)

Code submitted to TEACH.