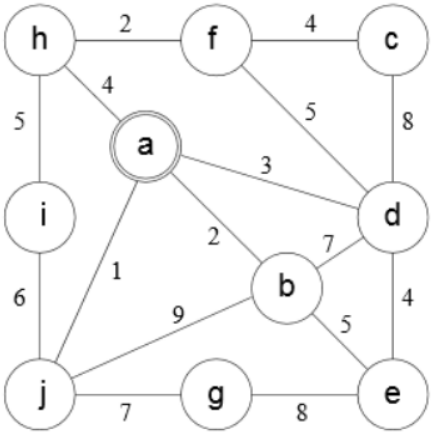
Homework 5

CS 325

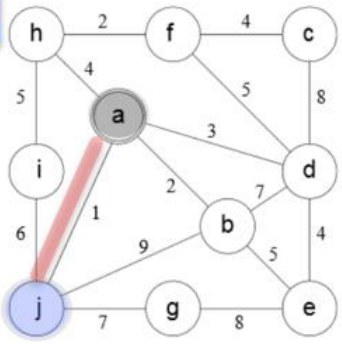
Daniel Kim

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.

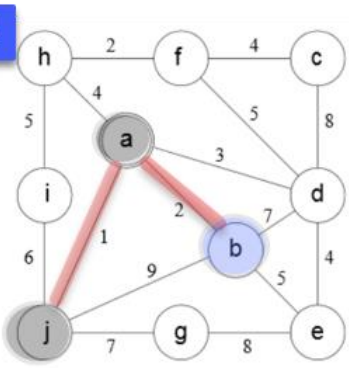


Prim's algorithm utilizes the greedy method to find a minimum spanning tree. The blue vertices are the current visited or explored vertex at that step. The gray vertices are already visited or explored vertices. Diagram is below.

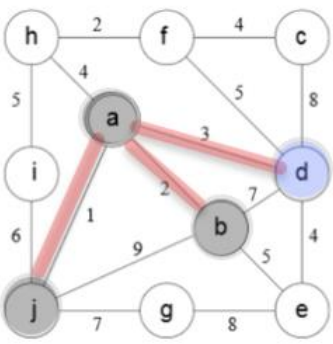
1


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

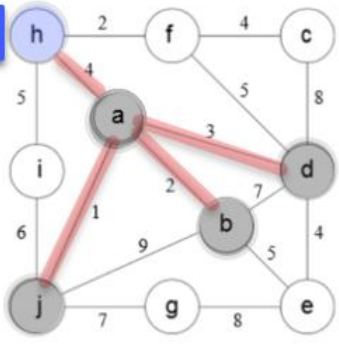
2


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

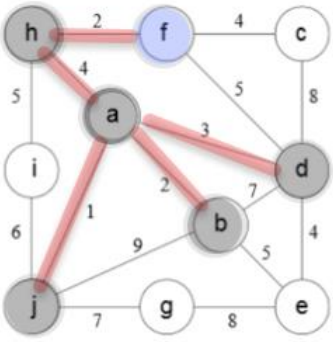
3


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

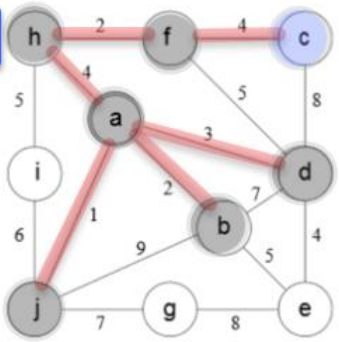
4


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

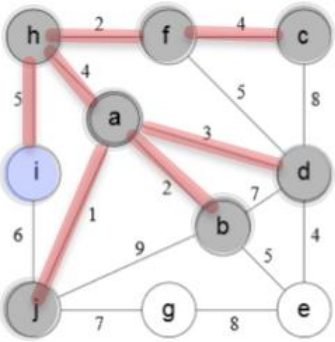
5


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

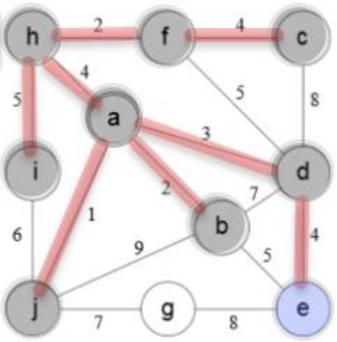
6


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

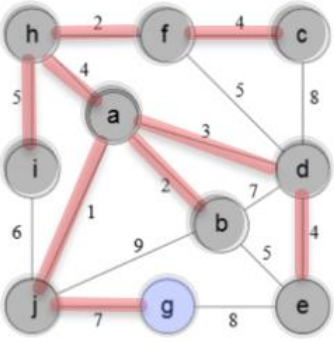
7


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

8


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

9


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

1. a-j, MinSpanSet = {a,**j**}, Weight of edge = 1

2. a-b, MinSpanSet = {a,**b**,j}, Weight of edge = 2

3. a-d, MinSpanSet = {a,b,j,**d**}, Weight of edge = 3

4. a-h. MinSpanSet ={a,b,d,**h**,j}, Weight of edge = 4

5. h-f, MinSpanSet = {a,b,d,h,**f**,j} Weight of edge = 2

6. f-c, MinSpanSet = {a,b,d,h,f,j,**c**}, Weight of edge = 4

7. h-i, MinSpanSet ={a,b,d,h,f,j,c,**i**}, Weight of edge = 5

8. d-e, MinSpanSet ={a,b,d,h,f,j,c,i,**e**}, Weight of edge = 4

9. j-g, MinSpanSet ={a,b,d,h,f,j,c,i,e,**g**}, Weight of edge = 7

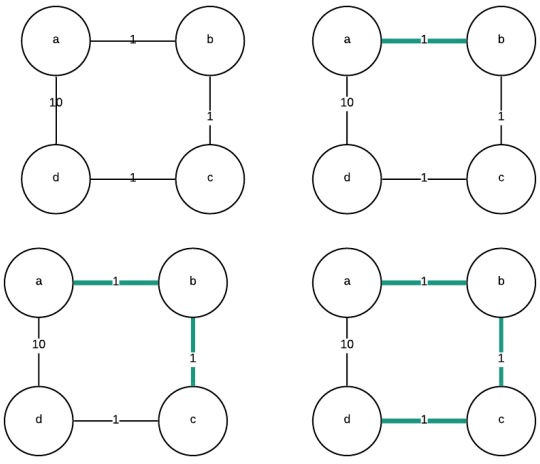
Total Weight = 32

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.

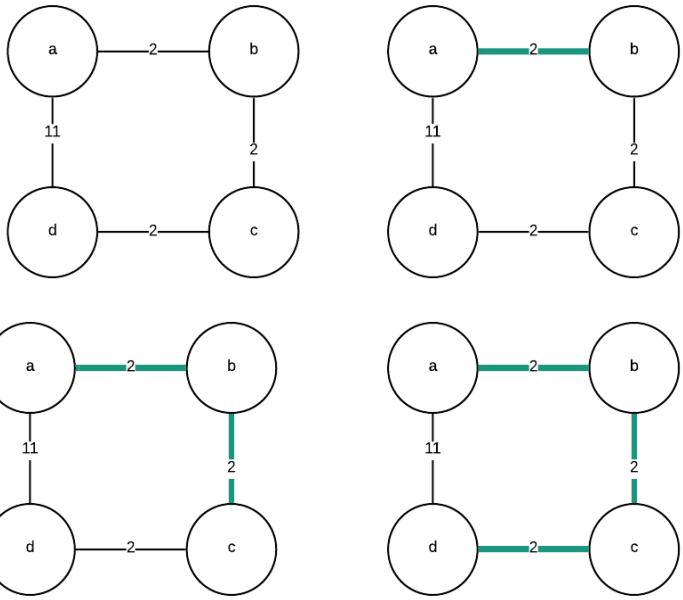
(b) Do the shortest paths change? Give an example where they change or prove they cannot change.

a) No, the minimum spanning tree does not change when all edge weights are increased by 1. If undirected graph G has n vertices, then a spanning tree will have at least n-1 edges. We can see this with a graph with vertexes a,b,c,d in the diagram.

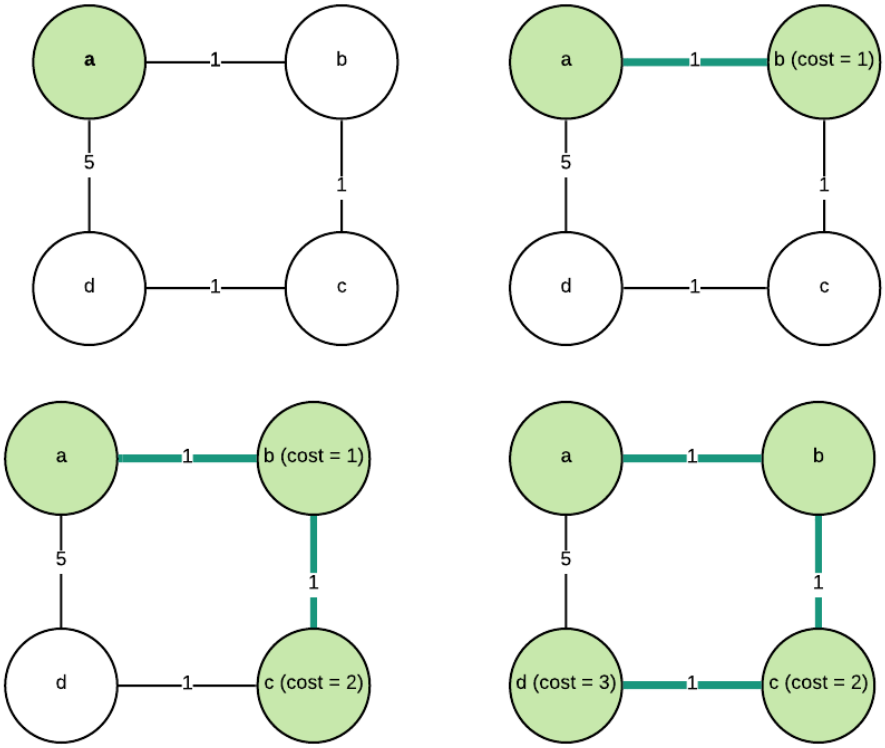


Using Kruskal's algorithm, we can see that the minimum spanning tree includes vertex a,b,c with a cost of 3.

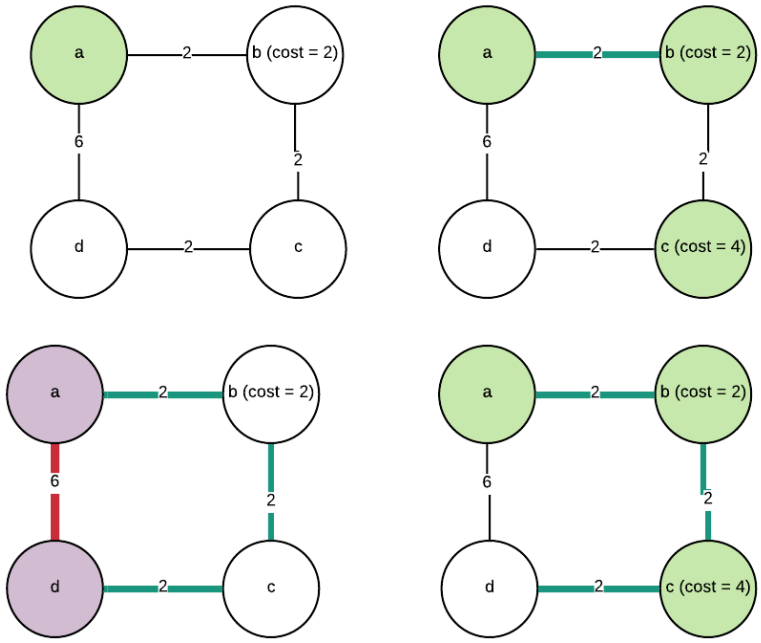
Increasing the edge weights by 1, we can see that the minimum spanning tree using Kruskal's algorithm is still the same. This is shown by the second diagram with updated weights.



b) Using Dijkstra's algorithm, we can see that the shortest path does change. For our first diagram, w(a,b) = 1, w(b,c) = 2 < w(a,d) =5, w(c,d) =3 < w(a,d) = 5. So a->b->c->d with a cost of 3 is our shortest path. After adjusted the weights of the edges, we can see that the shortest path changes. The edges a->b->c->d would produce a cost of 6.



However, the path a->d is already cost of 6. So, we use a->d because a->b->c->d doesn't produce a lower cost. The shortest paths are different.



a->b->c->d would cost 6
which is not less than a->d = 6.
Thus, we use a -> d = 6

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 youor algorithm.

a) Breadth first search algorithm can be used to traverse the graph with tree data structure. We would use BFS and ignore any edges of weight less than W.
b) The running time for the algorithm will be O(V+E) time.

4. *(5 points)* Below is a list of courses and prerequisites for a factious CS degree.

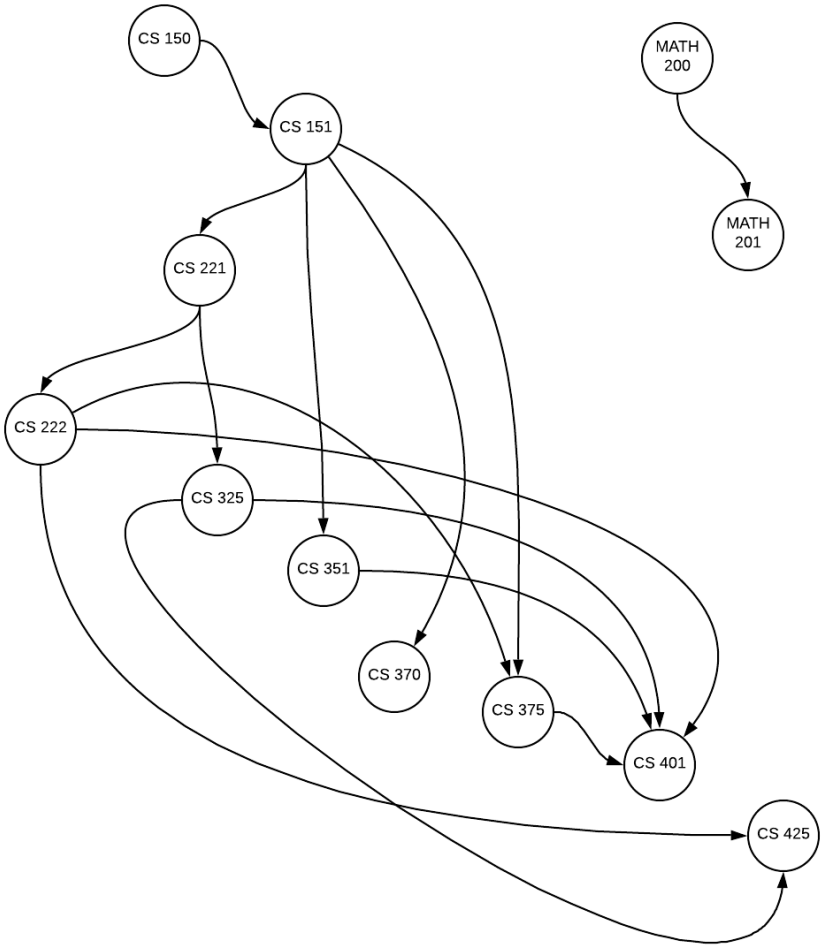
Course	Prerequisite
CS 150	None
CS 151	CS 150
CS 221	CS 151
CS 222	CS 221
CS 325	CS 221
CS 351	CS 151
CS 370	CS 151
CS 375	CS 151, CS 222
CS 401	CS 375, CS 351, CS 325, CS 222
CS 425	CS 325, CS 222
MATH 200	None
MATH 201	MATH 200

(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.

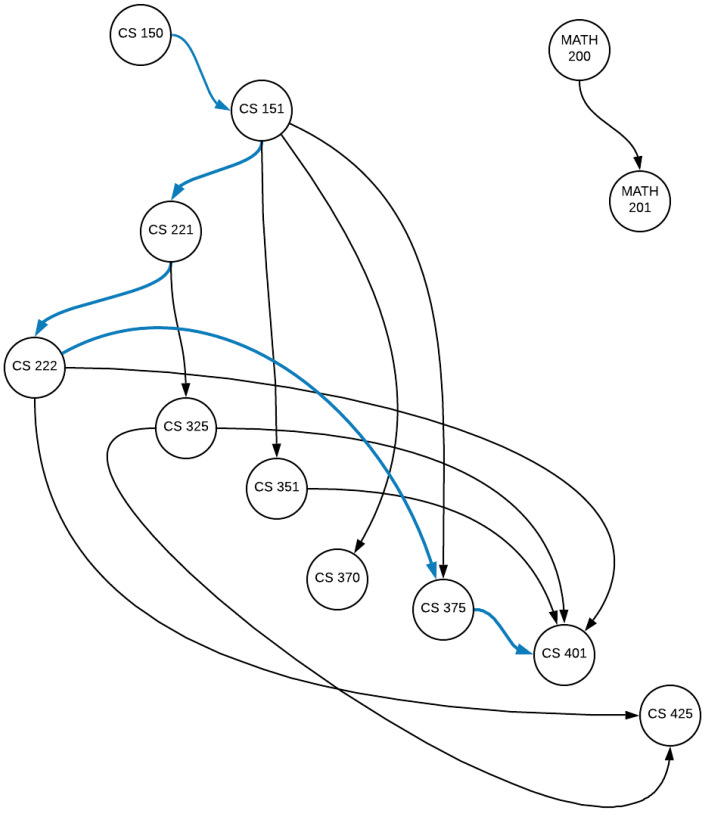
(d) Determine the length of the longest path in the DAG. How did you find it? What does this represent?



a)

b) Topological sort: MATH 200, MATH 201, CS 150, CS 151, CS 370, CS 351, CS 221, CS 325, CS 222, CS 375, CS 401, CS 425

c) In term 1, take CS 150 and MATH 200 because there are no prerequisites required. In term 2, take CS 151 and MATH 201 because the only prerequisites are the already taken CS 150 and MATH 200. In term 3: CS 221, CS 351 and CS 370. In term 4: CS 222 and CS 325. Term 5, we take CS 375, CS 401 and CS 425. This schedule would result in a total of 5 semesters needed to complete all courses.



d)

The length of the longest path in the DAG is of length 5. The path is as follows: **CS 150 --> CS 151 --> CS 221 --> CS 222 --> CS 375 --> CS 401**. The longest path in the DAG shows us that CS 401 requires the most prerequisite courses. The 5 courses prior, **CS 150 --> CS 151 --> CS 221 --> CS 222 --> CS 375,** represent the direct or indirect prerequisites for CS 401.

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

Submit a copy of your files including a README file that explains how to compile and run your code in a ZIP file to TEACH.

5. a) Let the graph G(V, E) represent the rivalries with each vertex representing a wrestler and each edge representing a rivalry.

Modifying the coloring attributes of the breadth-first-search algorithm, we can "color" the vertices of the graph of rivalries with two labels (colors), "baby faces" and "heels". Our graph, G, will have n vertices and r edges. Thus, $|V| = n$ and $|E| = r$

Algorithm:

1. For each vertex perform BFS operations as needed to visit all vertices.
2. If vertex distance is even, the vertex will have the attribute “good guys” and if the vertex distance of a wrestler is odd, the vertex will have the attribute “bad guys”. (Utilizing graph coloring, odd distance “bad guys” are assigned one color. Even distance “good guys” are assigned a differing color.)
3. For each edge, verify that coloring is valid. A valid edge between two vertices is one that doesn’t connect the same team of wrestlers and connects one wrestler from "babyfaces" to one wrestler from "heels" which means a rivalry between the “good guys” and ”bad guys” is established.

b)For BFS, the algorithm will take O(n + r) time. It will take O(n) time to designate each wrestler as a "babyface" or "heel". It will take O(r) time to validate edges. The solution will have take O(n + r). Therefore, the overall running time of the algorithm is O(n + r).

c) Uploaded on TEACH! Thank you!