Design & Analysis of Algorithms

Final Exam

Date: Monday, Dec. 14, 2020

Name: _____Chris Grimes_____

- You have 150 minutes for this exam.
- It consists of 5 problems worth 50 points each, plus a problem 6 for extra 30 points credit.
- The extra credit will be recorded separately, so make sure you have answered all questions from first 5 problems before moving on to problem 6.
- You need to collect 200 points out of 280 to have an A.

Instructions

Work as many problems as possible. All problems have the same value, but subparts of a problem may have different values (depending on their difficulties, importance, etc). Provide a short preliminary explanation of how an algorithm works before running an algorithm or presenting a formal algorithm description, and use examples or diagrams if they are needed to make your presentation clear. Please be concise and give well-organized explanations. Long, rambling, or poorly organized explanations, which are difficult to follow, will receive less credit.

Problem 1 (50)	Problem 2 (50)	Problem 3 (50)	Problem 4 (50)	Problem 5 (50)	Extra Credit (30)	Total (200/280)

Problem 1 [Fundamental Design Techniques] (50 points)

(a) (15 points) Describe the greedy method design paradigm. Does it work in all situations? Explain.

The greedy method paradigm is a method by which one will choose the locally optimal choice in the hopes that this choice will eventually lead to the globally optimal solution. This does not work in all situations as some problems can't be solved by making a series of small choices like matrix multiplication.

(b) (20 points) Characterize the running-time of the following divide-and-conquer algorithms described by the given recurrence relations

i. $T(n) = 3T(n/3) + n^2$

$\log_b a=1$ $f(n)= n^2$ so, T(n) is big theta of n^2

ii. $T(n) = 4T(n/2) + n$

$\log_b a= 2$ $f(n)= n$ so, T(n) is big theta of n^2

iii. $T(n) = 2T(n/2) + n \log^2 n$

$\log_b a= 1$ $f(n)= n \log^2 n$ so, T(n) is big theta of $n \log^2 n$

iv. $T(n) = 8T(n/2) + n \log n$

$\log_b a=3$ $f(n)= n \log n$ so, T(n) is big theta of n^3

(c) (15 points) Find the optimal way to parenthesize the following chain of seven matrices to be multiplied, where A_0 is a 2 × 3 matrix, A_1 is a 3 × 2 matrix, A_2 is a 2 × 5 matrix, A_3 is a 5 × 1 matrix, A_4 is a 1 × 4 matrix, A_5 is a 4 × 2 matrix, and A_6 is a 2 × 1 matrix. Most of the $N[i][j]$ values (left) and k values (right) have been filled in already. Complete both tables and use them to give the optimal parenthesization to multiply matrices $A = A_0 \cdot A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5 \cdot A_6$.

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	12	32	22	30	34	34
1		0	30	16	28	30	28
2			0	10	18	22	22
3				0	20		15
4					0	8	10
5						0	8
6							0

	0	**1**	**2**	**3**	**4**	**5**	**6**
0		0	1	0	3	3	3
1			1	1	3	3	1
2				2	3	3	3
3					3		3
4						4	5
5							5
6							

A3(A4*A5)=0+8+5*1*2=18 (A3*A4)A5=0+20+5*4*2=60

So, (3,5) in the left should be 18, and (3,5) in the right should be 3.

Leaving the optimal solution as, [(((A0*A1)*A2)*A3)*(A4*A5)]

Problem 2 [Graph Traversals] (50 points)

(a) (20 points) Explain how either depth first search (DFS) or breadth first search (BFS) can be used to solve the following problems.
i. Find a vertex *t* known to be close to a starting vertex *s*.

Run dfs specialized algorithm pathDFS on t and s and when the path is found it will be returned as the contents of the stack used in said algorithm.

ii. Determine if a vertex *t* is reachable from a vertex *v*.

Run dfs specialized algorithm pathDFS on t and v and if there is a path it will be returned as the contents of the stack used in said algorithm.

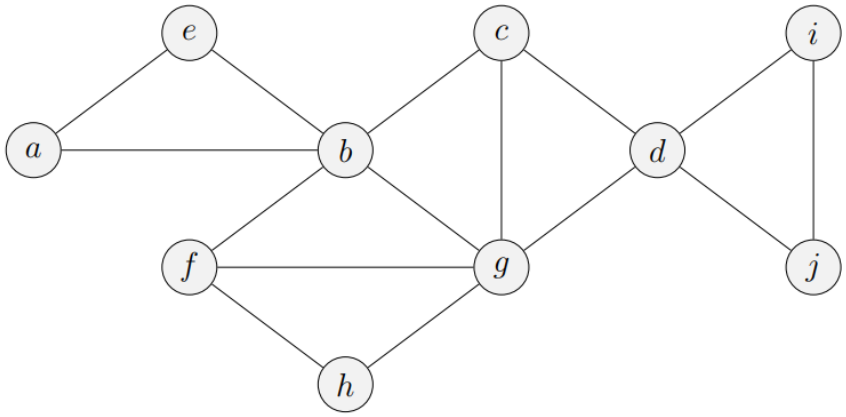
iii. Determine if a graph is acyclic.

Run dfs specialized algorithm cycleDFS and if no back edges are encountered the given graph is acyclic.

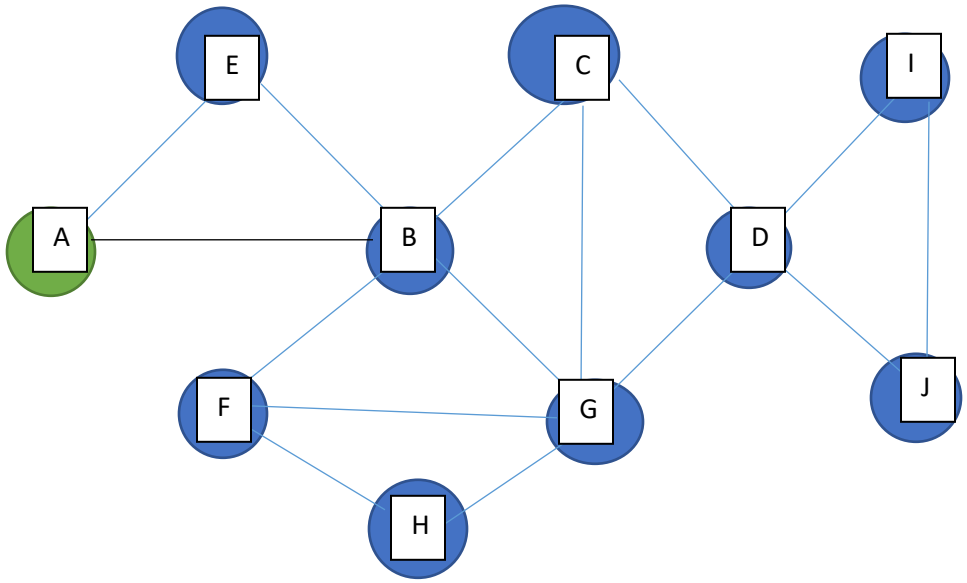
iv. Compute a spanning forest.

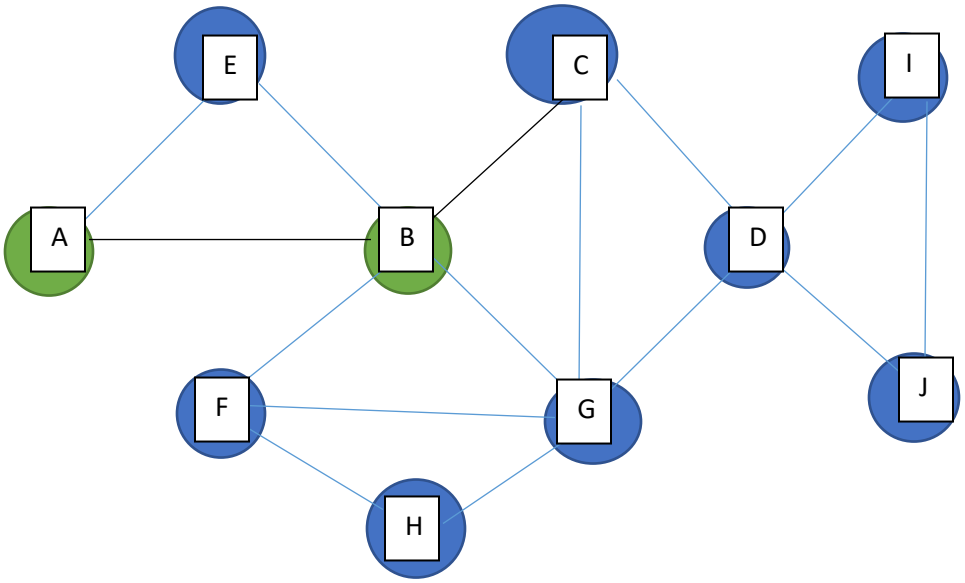
Run dfs on said graph and the resulting graph without back edges will result in a spanning forest.

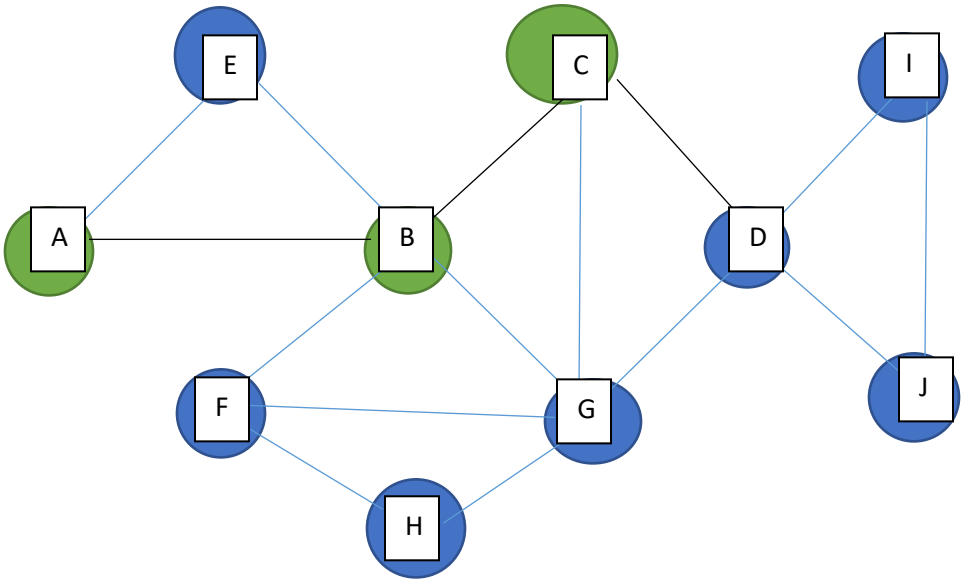
(b) (30 points) Run a depth first search (DFS) on the graph below. Start at vertex *a*. Label every edge on the graph as a *discovery* edge or *back* edge. Whenever faced with a decision of which vertex to pick from a set of vertices, pick the vertex whose label occurs earliest in the alphabet.

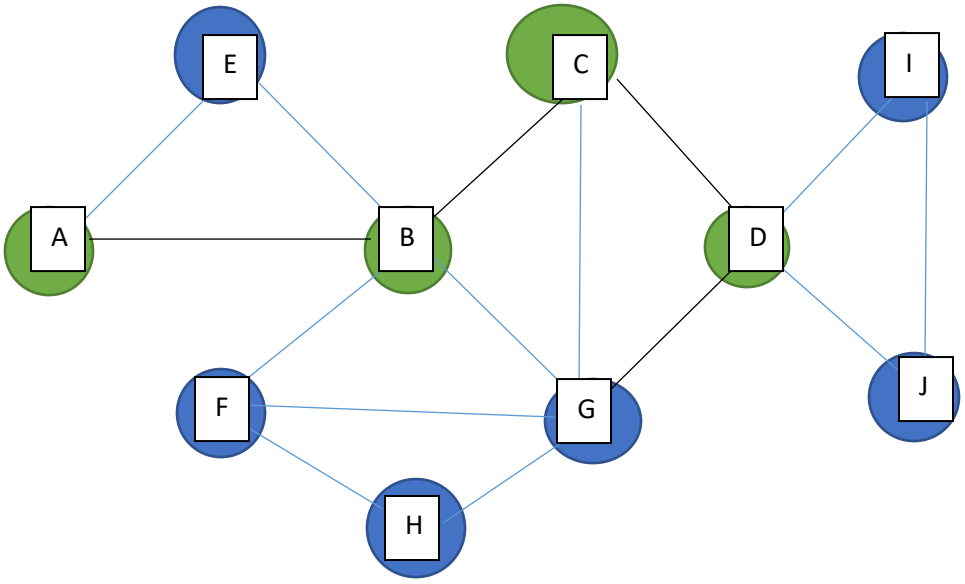


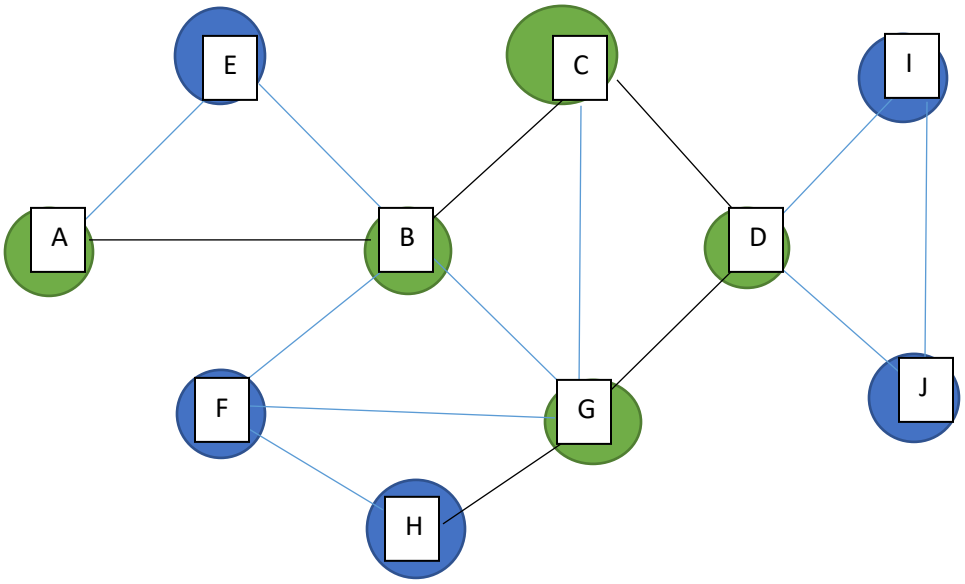
Blue circle= unexplored vertex blue line= unexplored edge green circle= visited vertex
black line= discovery edge red line= back edge

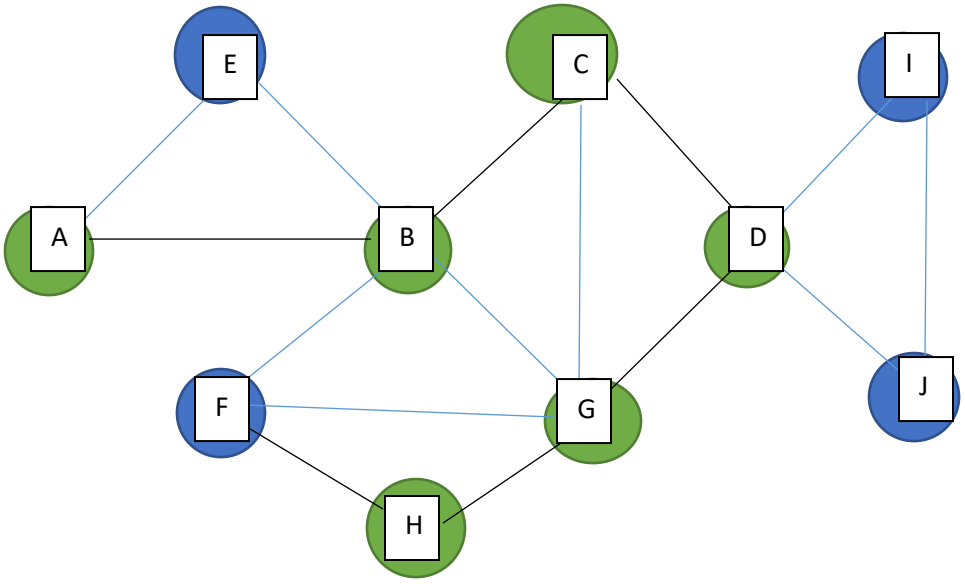


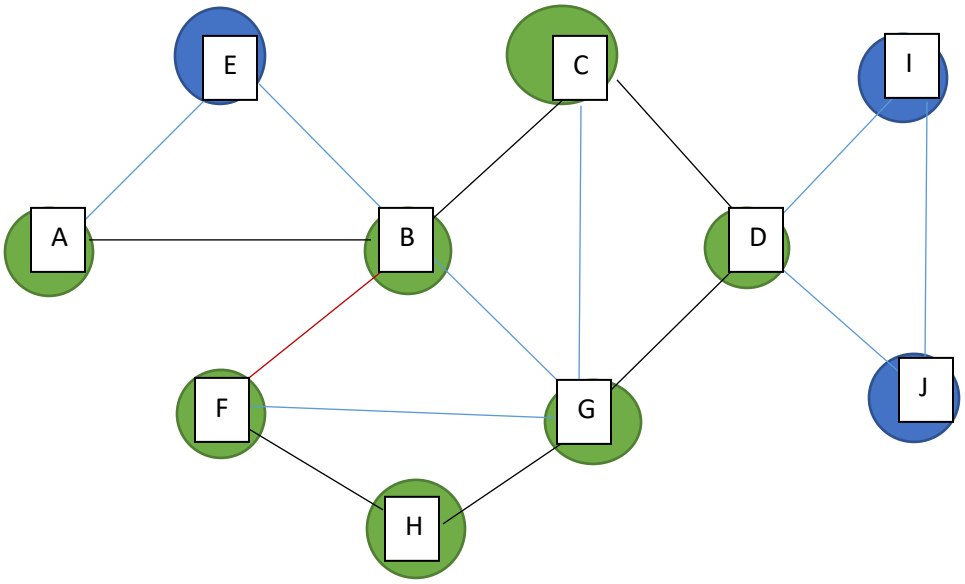

E
C
I
A
B
D
F
G
J
H

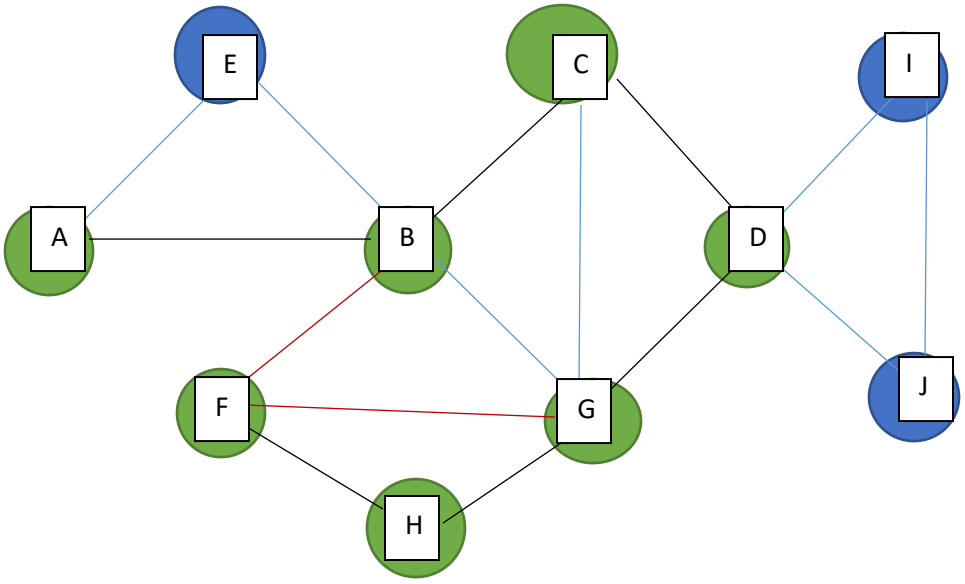

E
C
I
A
B
D
F
G
J
H

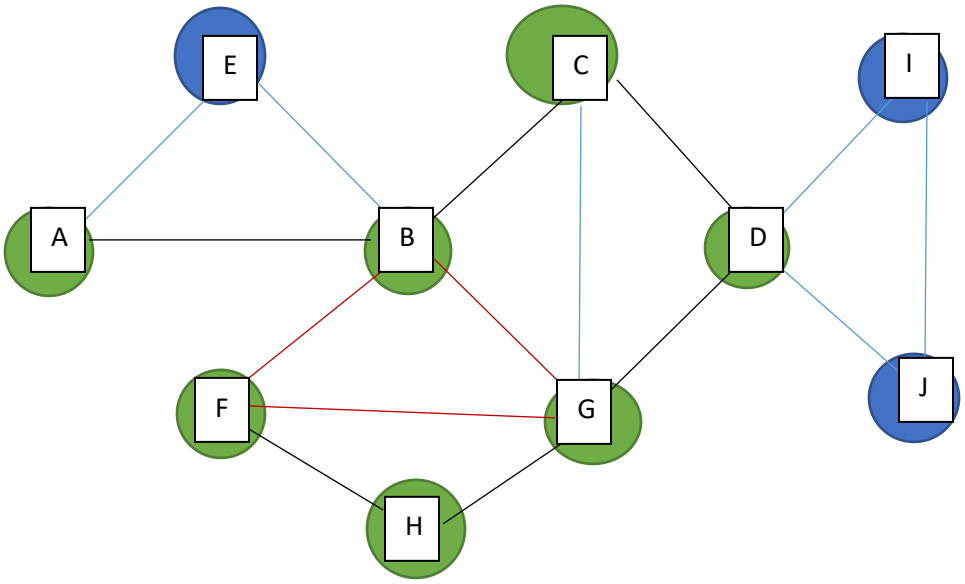

E
C
I
A
B
D
F
G
J
H

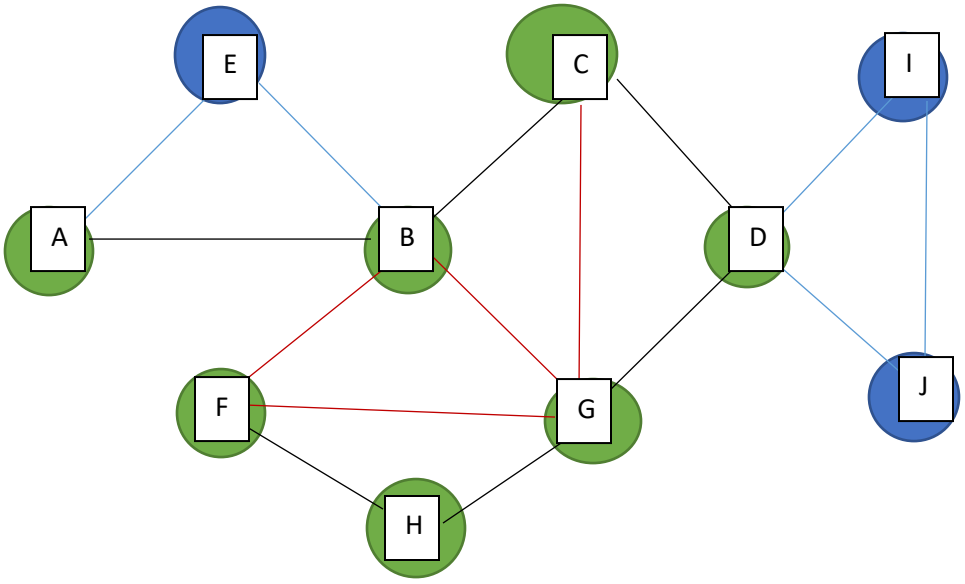

E
C
I
A
B
D
F
G
J
H

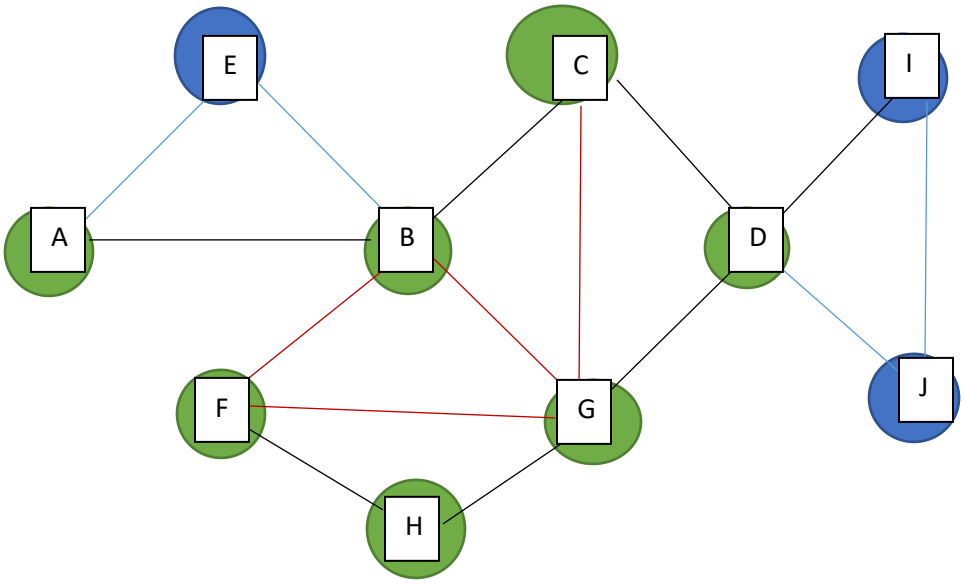

E
C
I
A
B
D
F
G
J
H

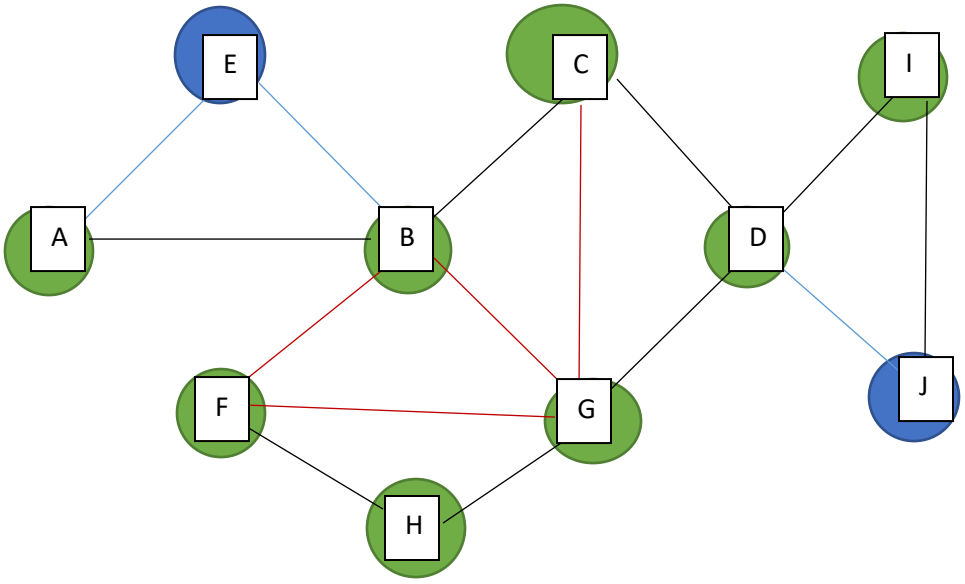

E
C
I
A
B
D
F
G
J
H

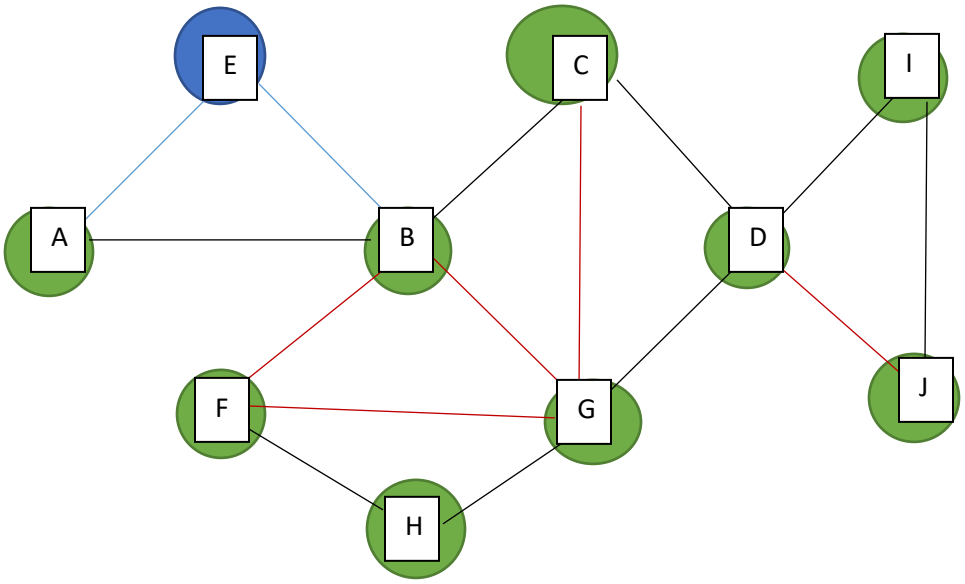

E
C
I
A
B
D
F
G
J
H

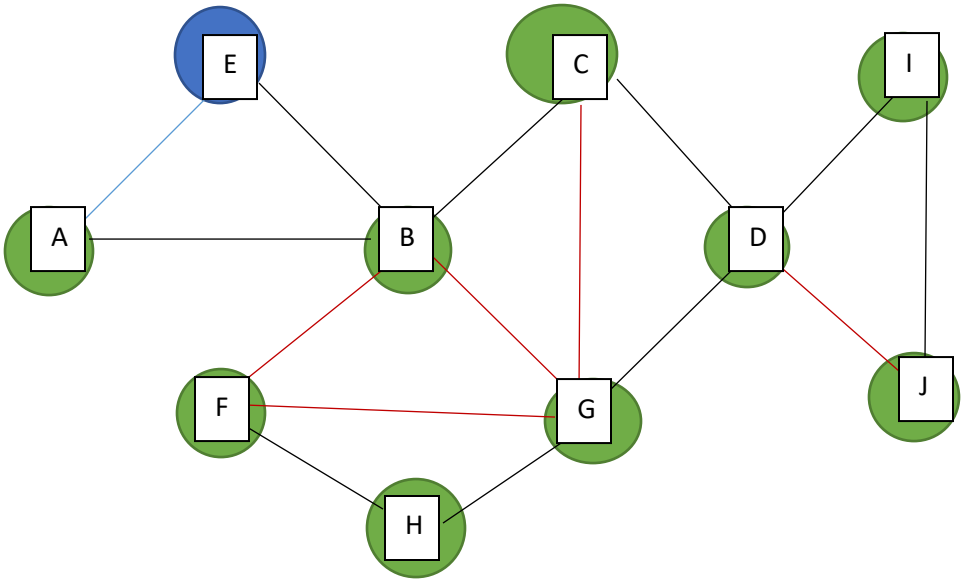

E
C
I
A
B
D
F
G
J
H

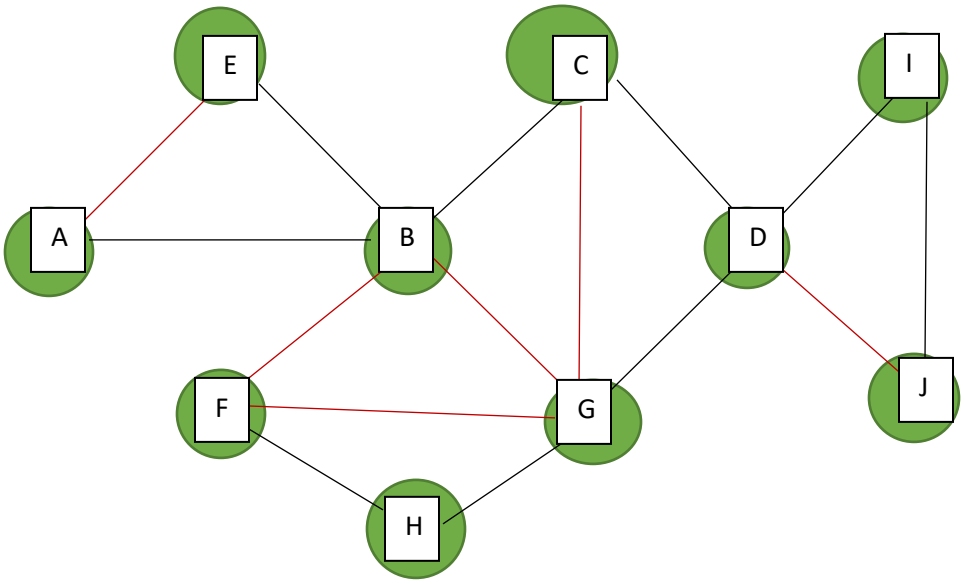

E
C
I
A
B
D
F
G
J
H


E
C
I
A
B
D
F
G
J
H


E
C
I
A
B
D
F
G
J
H


E
C
I
A
B
D
F
G
J
H


E
C
I
A
B
D
F
G
J
H


E
C
I
A
B
D
F
G
J
H

Problem 3 [MST] (50 points)

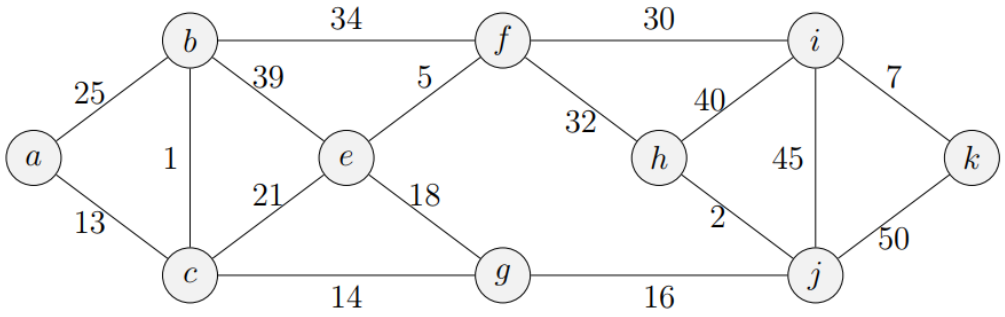
(a) (5 points) Define a minimum spanning tree (MST).

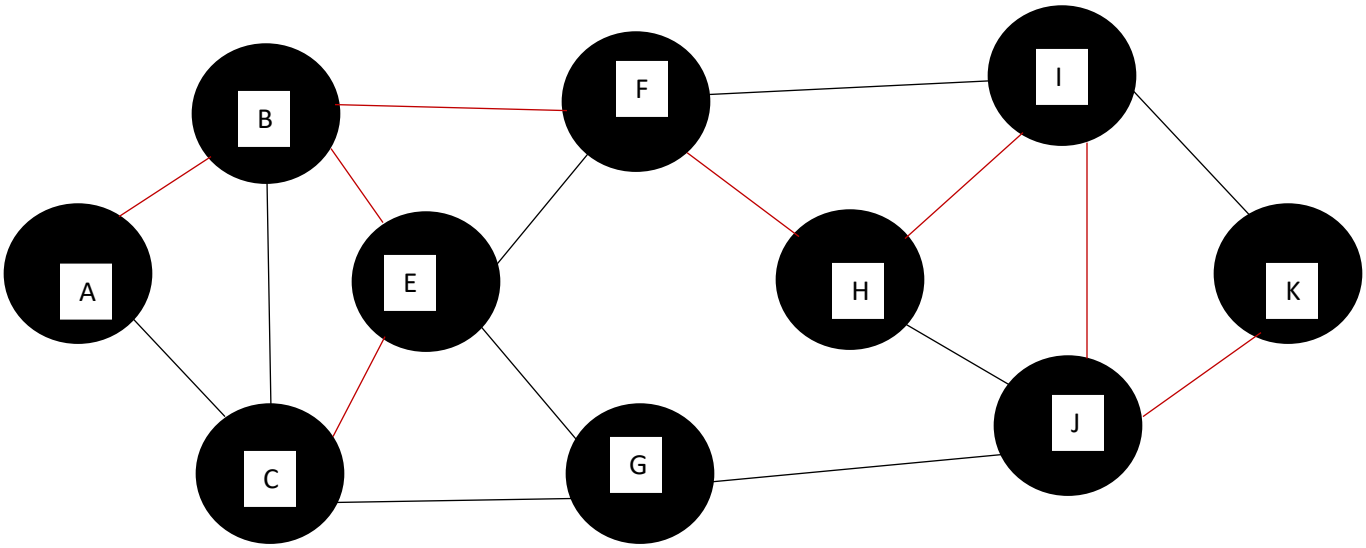
A minimum spanning tree is a tree of a weighted graph that is connected using the minimum total edge weights. That is it's connected using the edges that total to the minimum whilst still connecting the whole graph.

(b) (5 points) What is the worst-case complexity of Kruskal's algorithm to find a MST?

The worst case running time for Kruskal's algorithm is O(m log(n)).

(c) (40 points) Construct the MST of the following graph using Kruskal's algorithm. Give a list of edges in the order in which they are considered, and indicate if that edge is used in the MST.





Edge (b,c) used as it connects two clouds

Edge(e,f) used as it connects two clouds

Edge(a,c) used as it connects two clouds

Edge(g,j) used as it connects two clouds

Edge(h,j) used as it connects two clouds

Edge(i,k) used as it connects two clouds

Edge(c,g) used as it connects two clouds

Edge(e,g) used as it connects two clouds

Edge(c,e) not used as these two clouds are already connected

Edge(a,b) not used as these two clouds are already connected

Edge(f,i) used as it connects two clouds

Edge(f,h) not used as these two clouds are already connected

Edge(b,f) not used as these two clouds are already connected

Edge(b,e) not used as these two clouds are already connected

Edge(h,i) not used as these two clouds are already connected

Edge(i,j) not used as these two clouds are already connected

Edge(j,k) not used as these two clouds are already connected

Problem 4 [Single Destination Shortest Path] (50 points)

The *single-destination shortest path problem* for a directed graph seeks the shortest path from every vertex to a specified vertex *v*. Give an efficient algorithm (running in $O((n + m) \log n)$ time) to solve the single-destination shortest paths problem on a connected digraph with positive edge weights. Analyze the running time of your algorithm.

```
Algorithm sdsp(G,s):
Input: a graph G
output: a connected tree
a<- new heap based priority queue
for each (vertex in G)//runs n times
        if(vertex!=s)
                vertex.distance=-1
        else
                vertex.distance=0
while(!a.empty())//runs n times
        z<-a.removeMin()
        for each(incident edge of z)//runs m times
                z.relax()
                i<-g.opposite(z, incident edge of z)
                j<-getDistance(z)+ weight(incident edge of z)
                if(j< getDistance(i))
                        setDistance(i,j)
```

The worst case running time for the above algorithm is O(m log(n)) as it must run for each edge, but each edge only checks some of the verticies.

Problem 5 [Graph Traversal Variants] (50 points)

Amazon has a network of *k* distribution centers which are connected via *m* roads to *n* residential houses. Suppose the network is represented by a graph $G = (V, E)$ that contains $k + n$ vertices and *m* edges, where each vertex can be a distribution center or a house. Due to an increase in demand, the shipping trucks need to return back to the distribution center frequently, so they can't go too far away. Design an efficient algorithm which will compute for each distribution center the set of houses it can reach using no more than 3 roads. Analyze its running time.

```
Algorithm amazonShipping(a vertex A, distance from source):
    input: a vertex A and an int I which is the distance from the distribution center
    // I should begin at 0 the first time this method is called
    output: a set of vertices within 3 edges of vertex A
    set rtnval=null
    for each(edge in the graph that contains vertex A)//runs O(n+m)
        edge.label()= -1
    ++I
    if(I <4)
        for each(incident edge of A)// runs a max of O(n+m)
            if(edge.label()==-1)
                edge.label()=I
                rtnval.add(other vertex connected to edge)
                amazonShipping(other vertex connected to edge, I)
    return rtnval
```

Do to the two for loops in the above algorithm, the span4video algorithm will have a worst case running time of O(n+m).

Problem 6 [Challenging problem - Burgers would be great right about now] (30 points) Suppose that you want to get from vertex s to vertex t in an unweighted graph $G = (V, E)$, but you would like to stop by vertex u if it is possible to do so without increasing the length of your path by more than a factor of α. Describe an efficient algorithm that would determine an optimal s to t path given your preference for stopping at u along the way if doing so is not prohibitively costly. (It should either return the shortest path from s to t, or the shortest path from s to t containing u, depending on the situation.) Analyze the running time of your algorithm. If it helps, imagine that there are burgers at u.