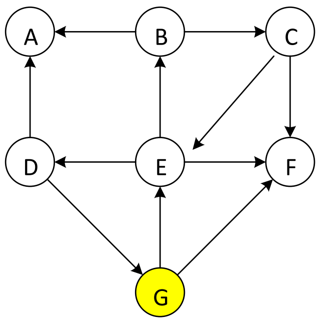
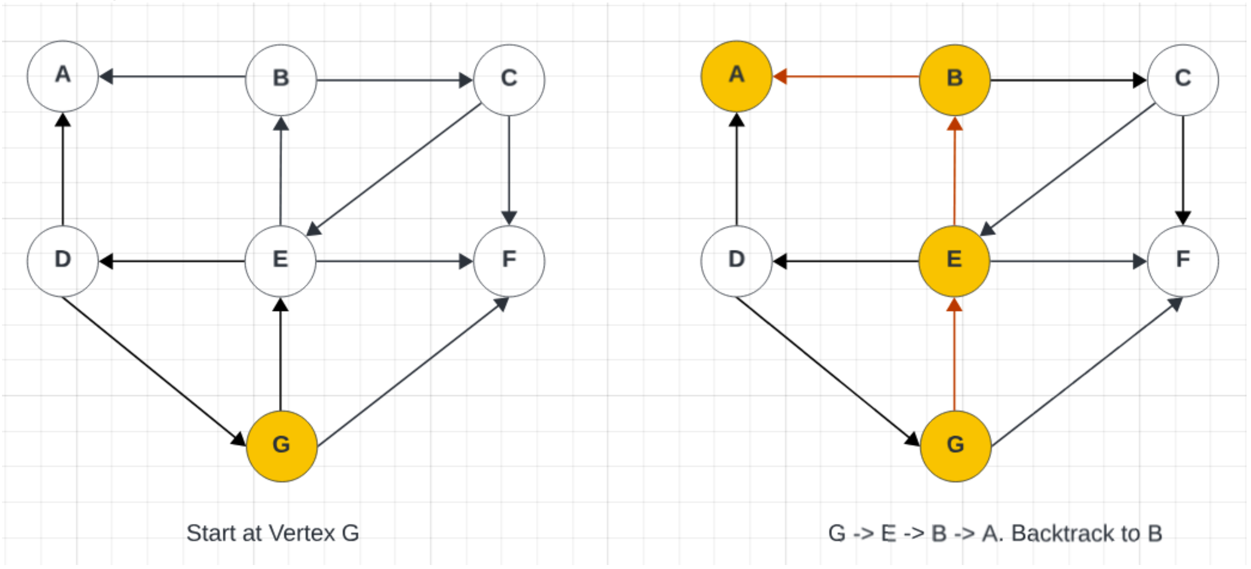
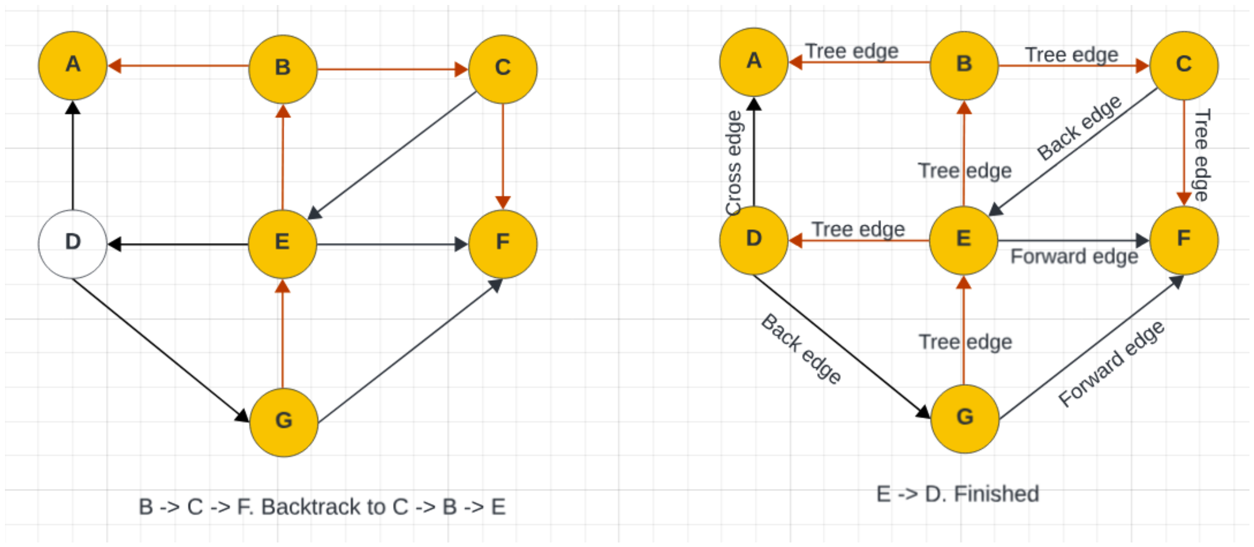
Assignment 6

Problem 1 (10 points). Run DFS on the following graph beginning at node G and show the sequence of nodes generated by the search. When you have two or more choices as the next node to visit, choose them in the alphabetical order.



After completing the DFS, classify each edge as a *tree edge*, a *forward edge*, a *back edge*, or a *cross edge*.

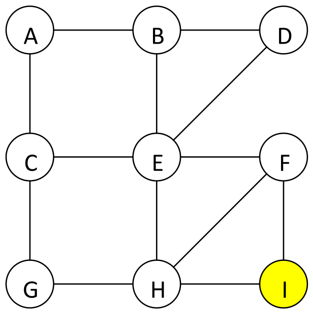




The sequence of nodes generated by the search:

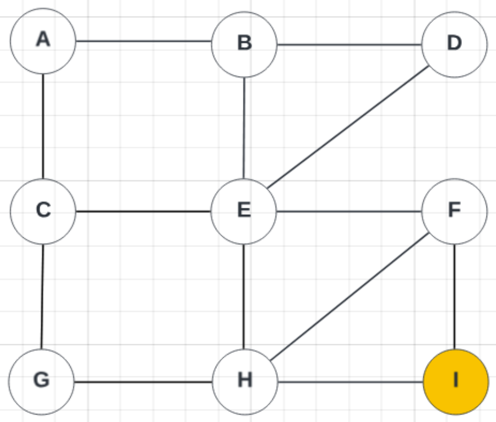
G → E → B → A → C → F → D

Problem 2 (10 points). Run BFS on the following graph beginning at node I and show the sequence of nodes generated by the search. When you have two or more choices as the next node to visit, choose them in the alphabetical order.

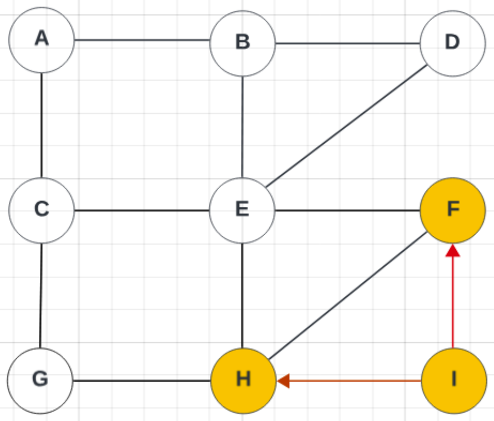


Sequence of nodes generated by the search:

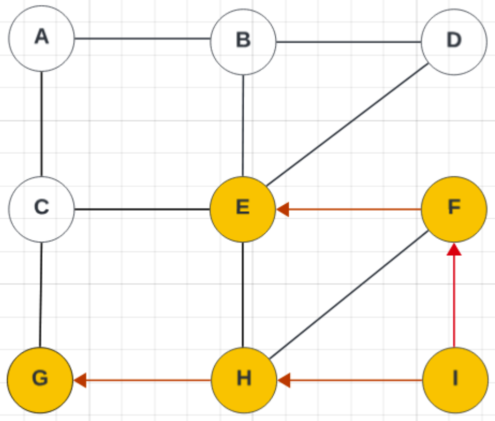
{I, F, H, E, G, B, C, D, A}



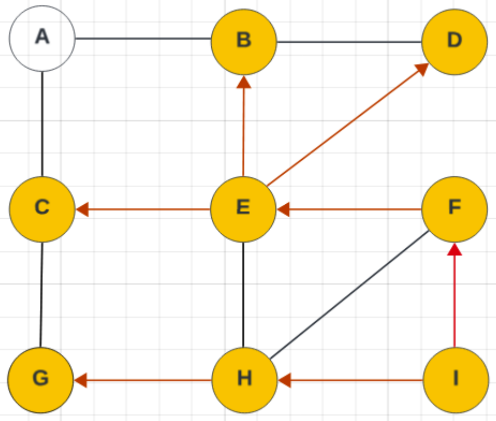
Start at Vertex I



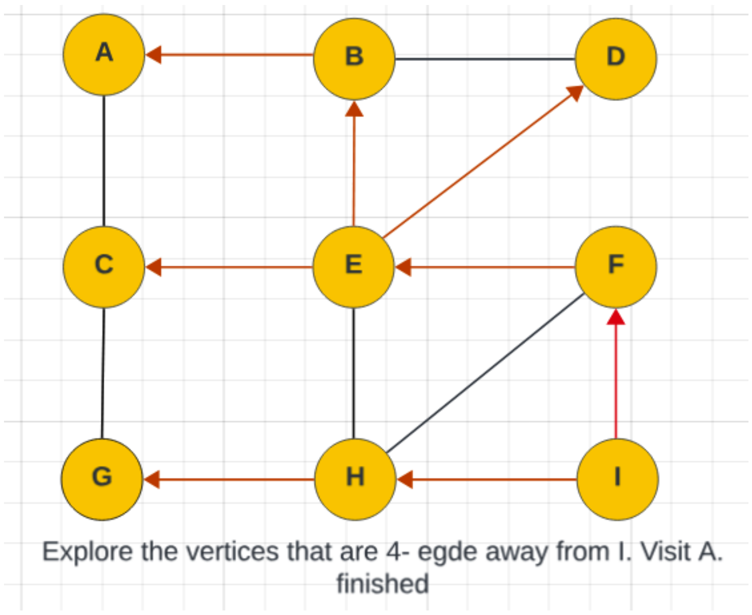
Explore the vertices that are one- egde away from I. Visit F, then H



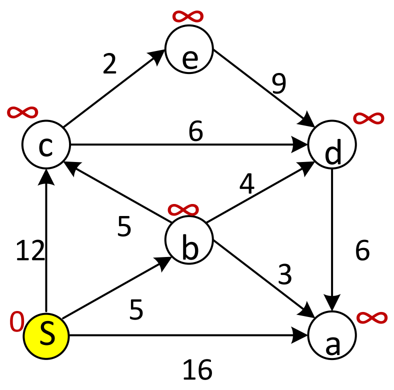
Explore the vertices that are 2- egde away from I. Visit E, then G



Explore the vertices that are 3- egde away from I.
Visit B, then C, then D

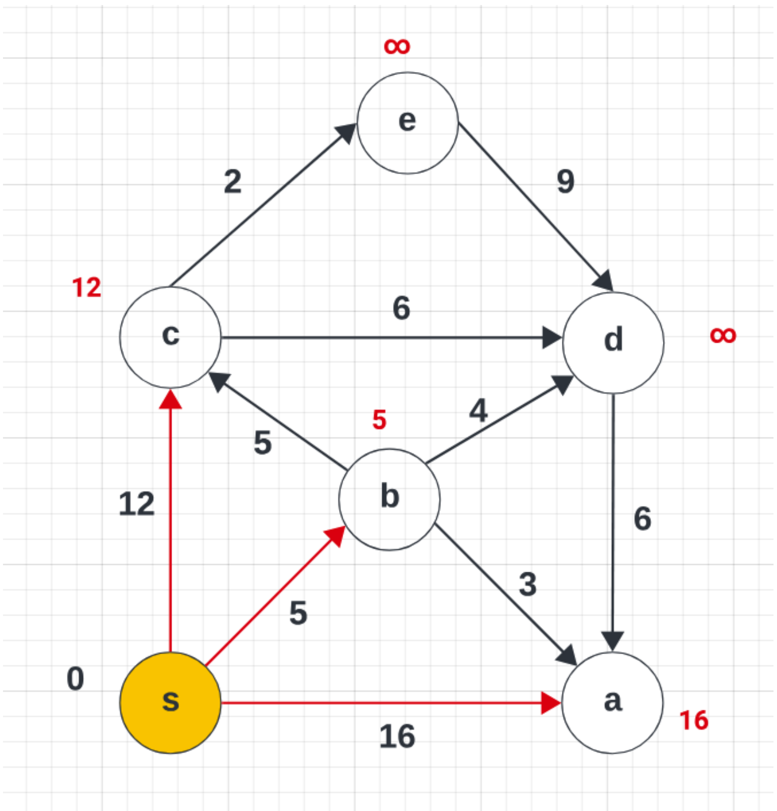


Problem 3 (10 points). Run Dijkstra's algorithm on the following graph beginning at node S.

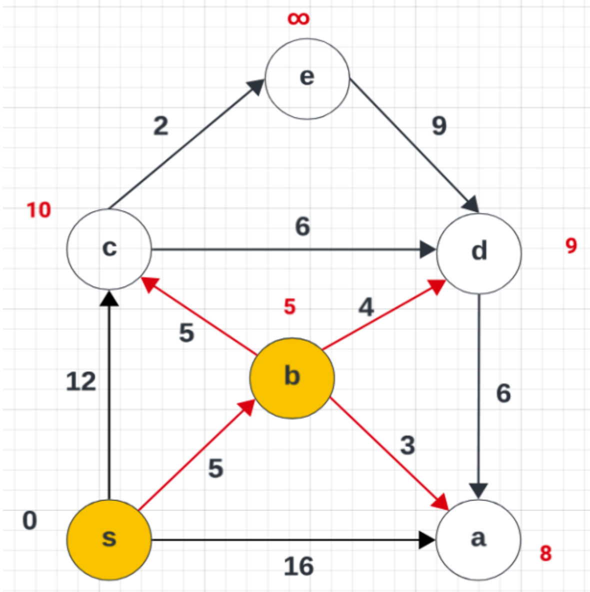


Problem 3-(1). After each iteration, show the D values of all nodes (initial D values are shown above each node in red).

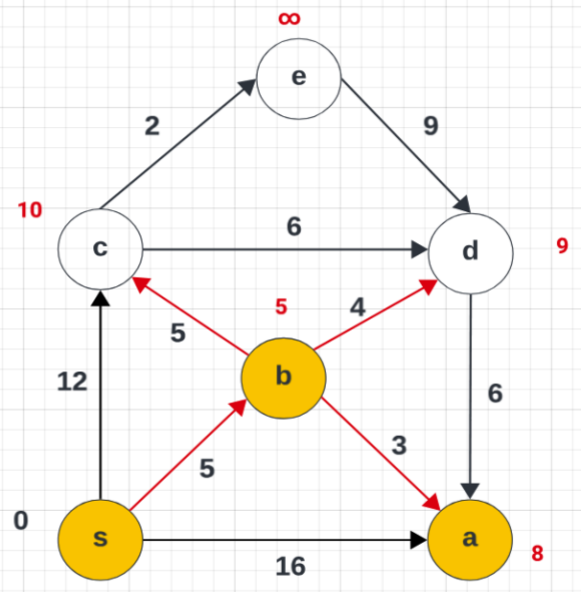
- C is cloud and empty at the beginning



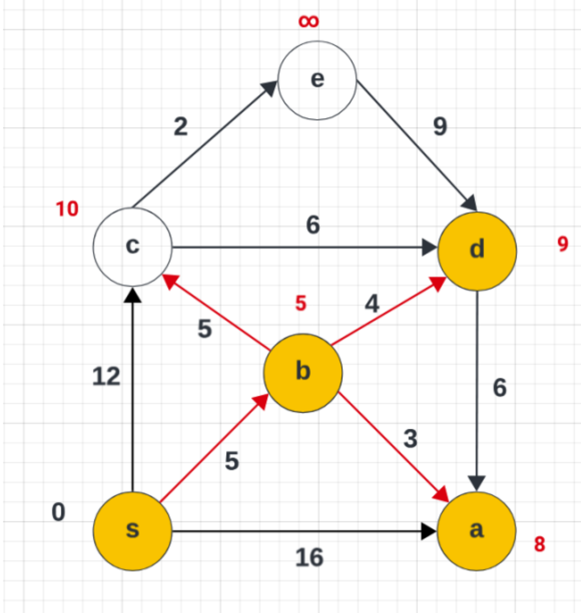
1. s into C, (s,c), (s,b), (s,a) are relaxed



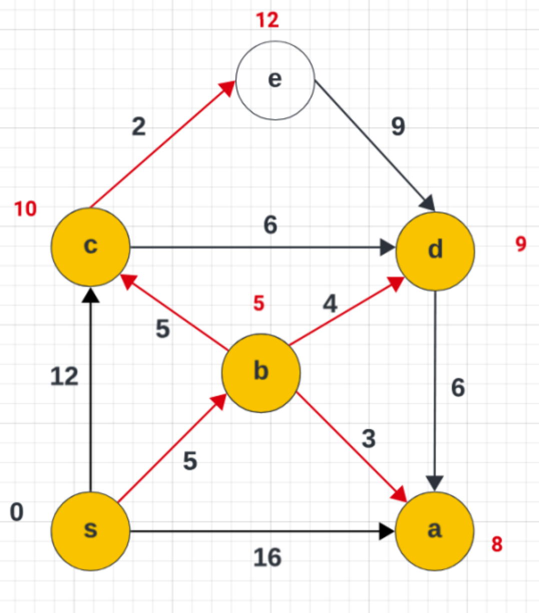
2. b into C, (b,c), (b,d), (b,a) are relaxed



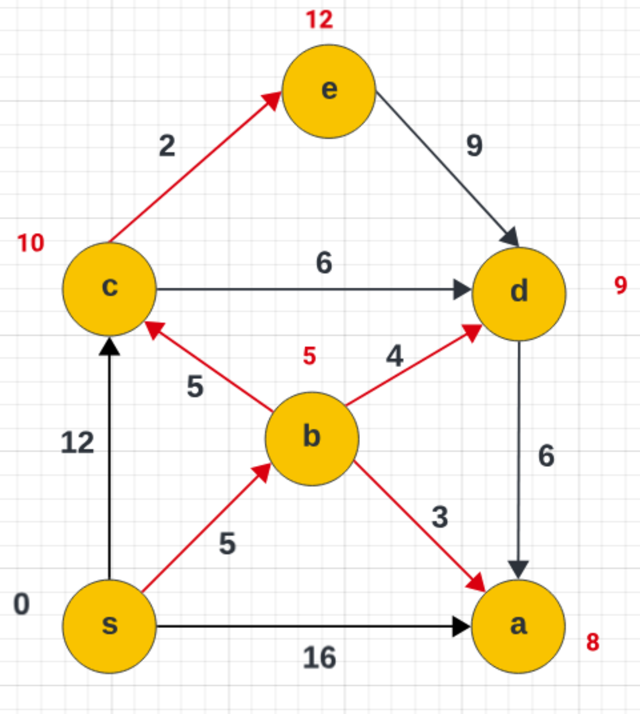
3. a into C, no edge need to be relaxed



4. d into C, no edge need to be relaxed



5. c into C, (c,e) is relaxed



6. e into C, No edge relaxation needed. Finished

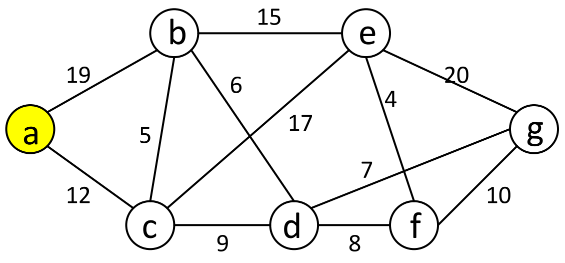
Problem 3-(2). Show the shortest path from S to every other node generated by the algorithm.

Shortest Path from S to every other node

1. s to b: s → b || *Distance: 5*

2. s to a: s → b → a || *Distance: 8*
3. s to c: s → b → c || *Distance: 10*
4. s to d: s →b → d || *Distance: 9*
5. s to e: s → b → c → e || *Distance: 12*

Problem 4 (10 points). Run the Prim-Jarnik algorithm on the following graph beginning at node *a*.



Problem 4-(1). Show the sequence of nodes in the order they are brought into the "cloud."

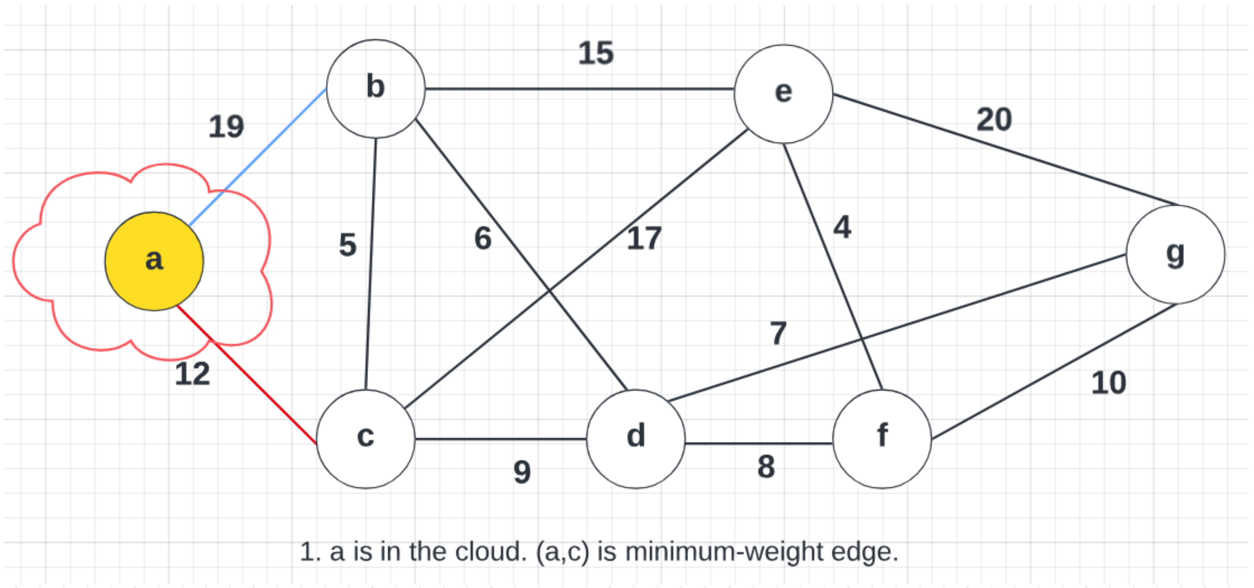
The sequence of nodes in the order they are brought into the "cloud": **{a, c, b, d, g, f, e}**

Problem 4-(2). Show the minimum spanning tree T, generated by the algorithm, as a set of edges.

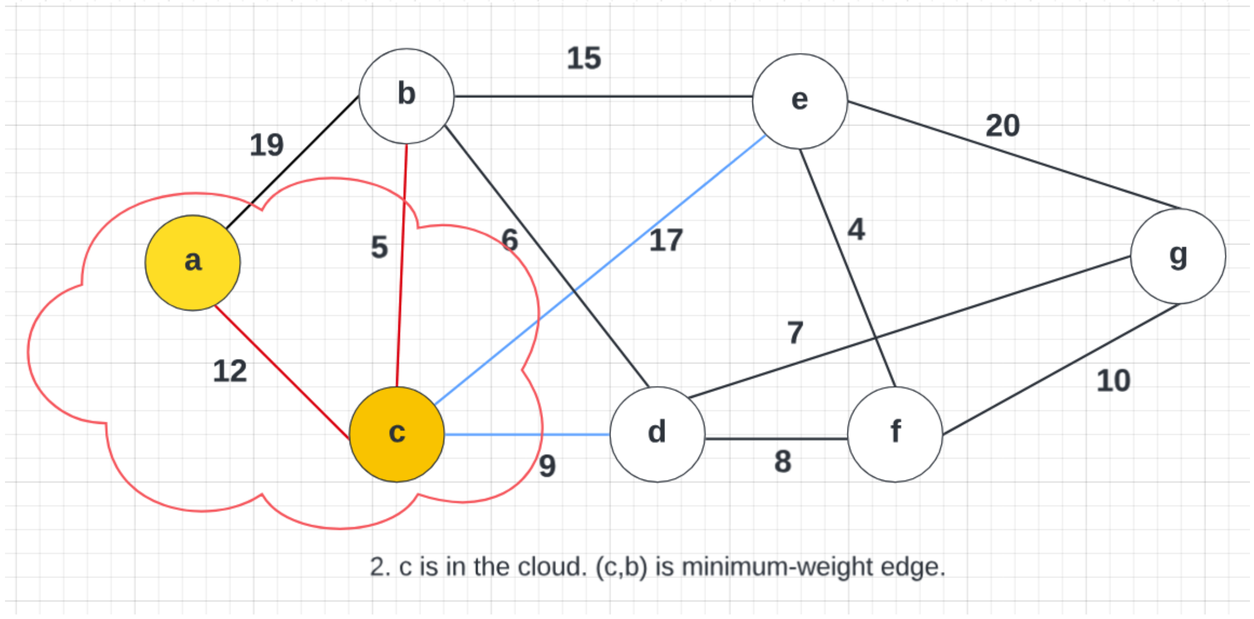
The minimum spanning tree T, generated by the algorithm, as a set of edges:

{(a,c), (c,b), (b,d), (d,g), (d,f), (f,e)}

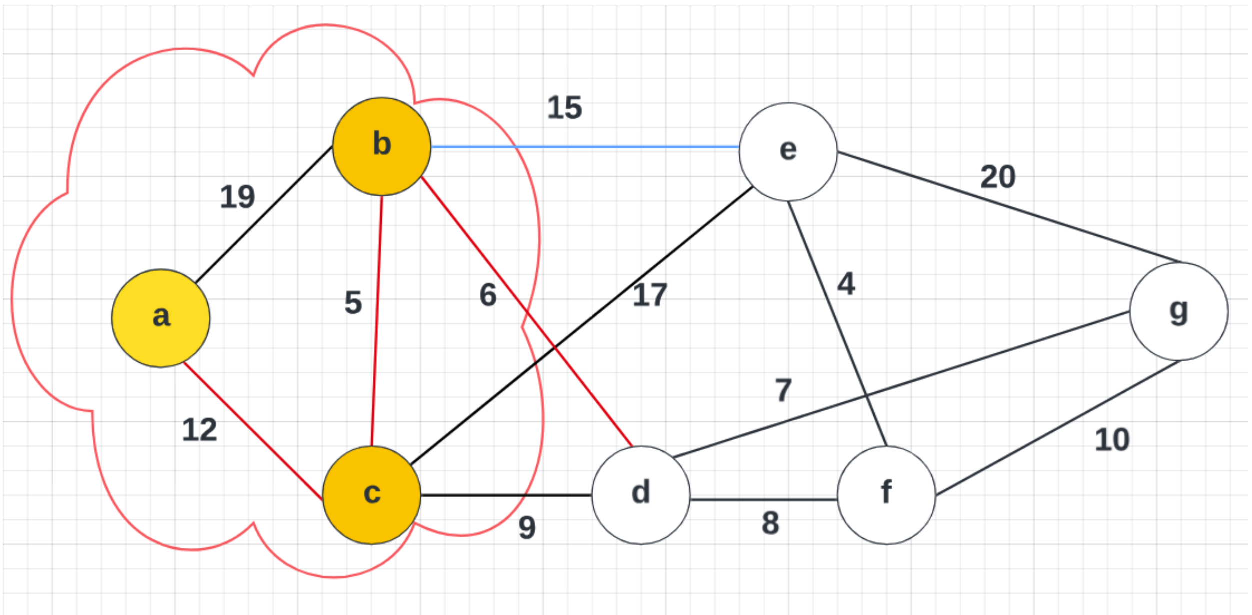
1. W(a,c) = 12
2. W(c,b) = 5
3. W(b,d) = 6
4. W(d,g) = 7
5. W(d,f) = 8
6. W(f,e) = 4



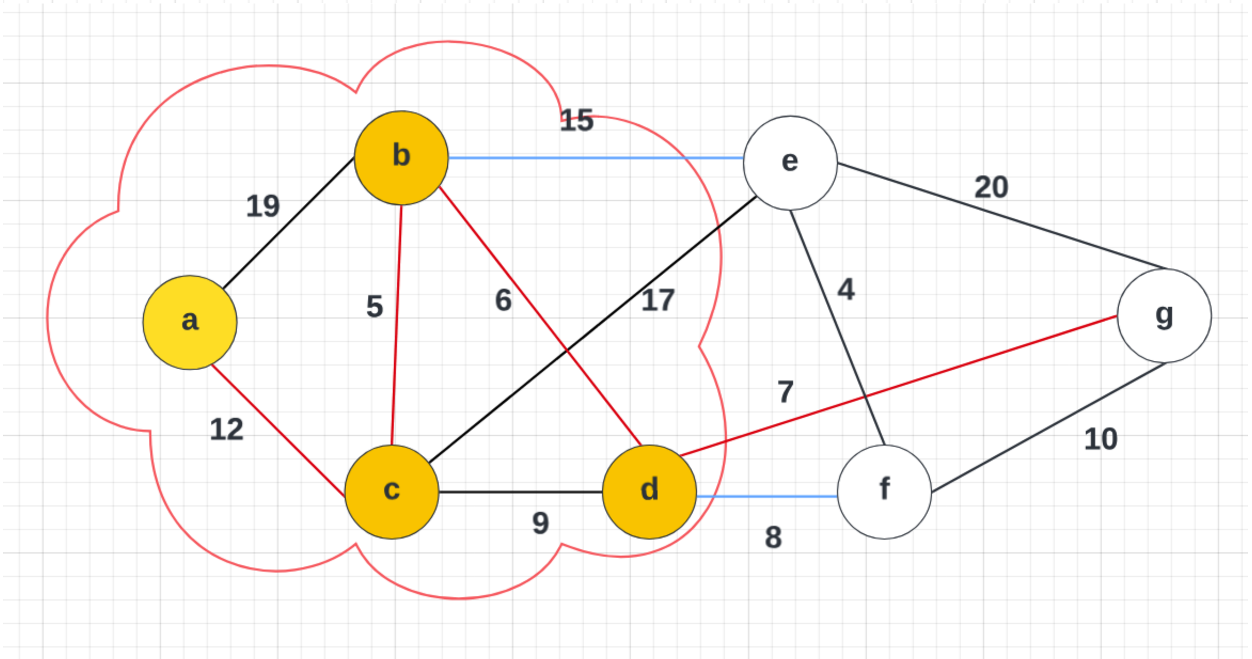
1. a is in the cloud. (a,c) is minimum-weight edge.



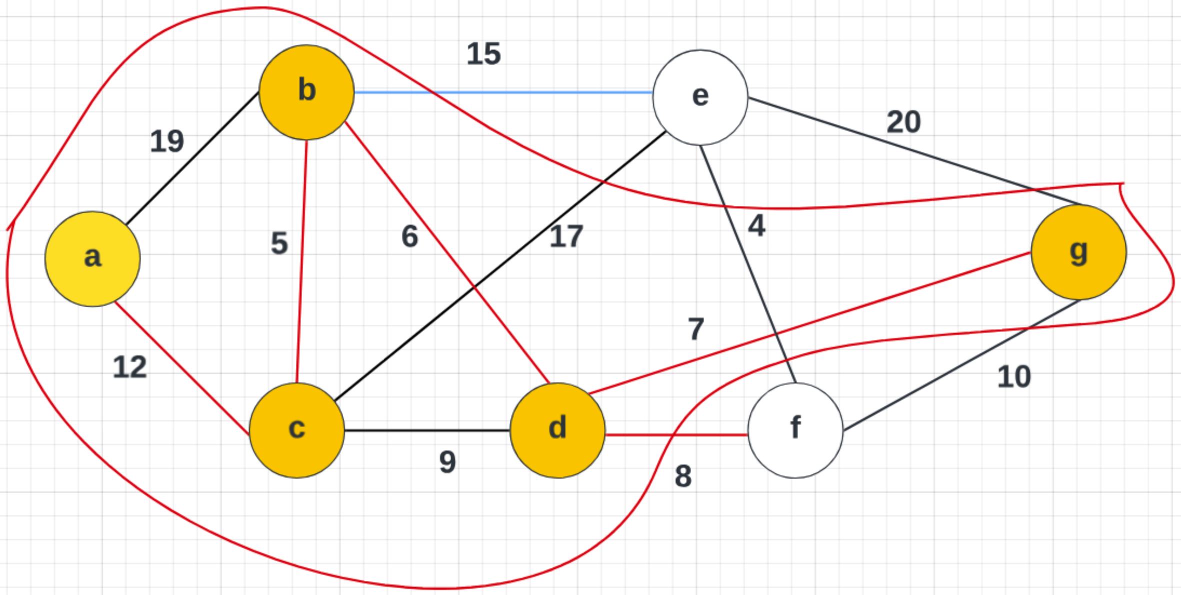
2. c is in the cloud. (c,b) is minimum-weight edge.



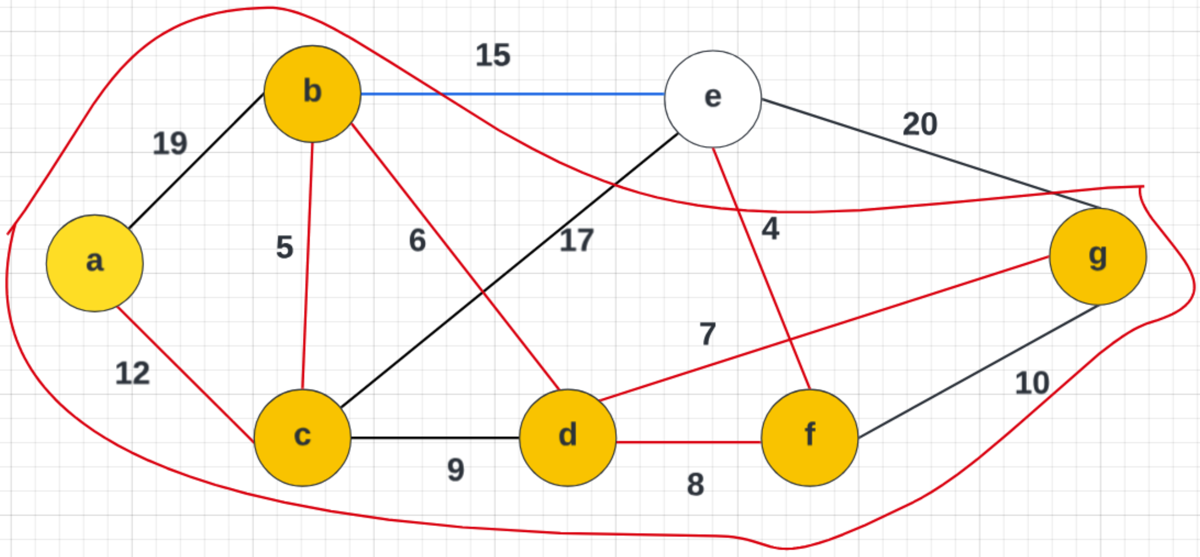
3. b is in the cloud. (b,d) is minimum-weight edge.



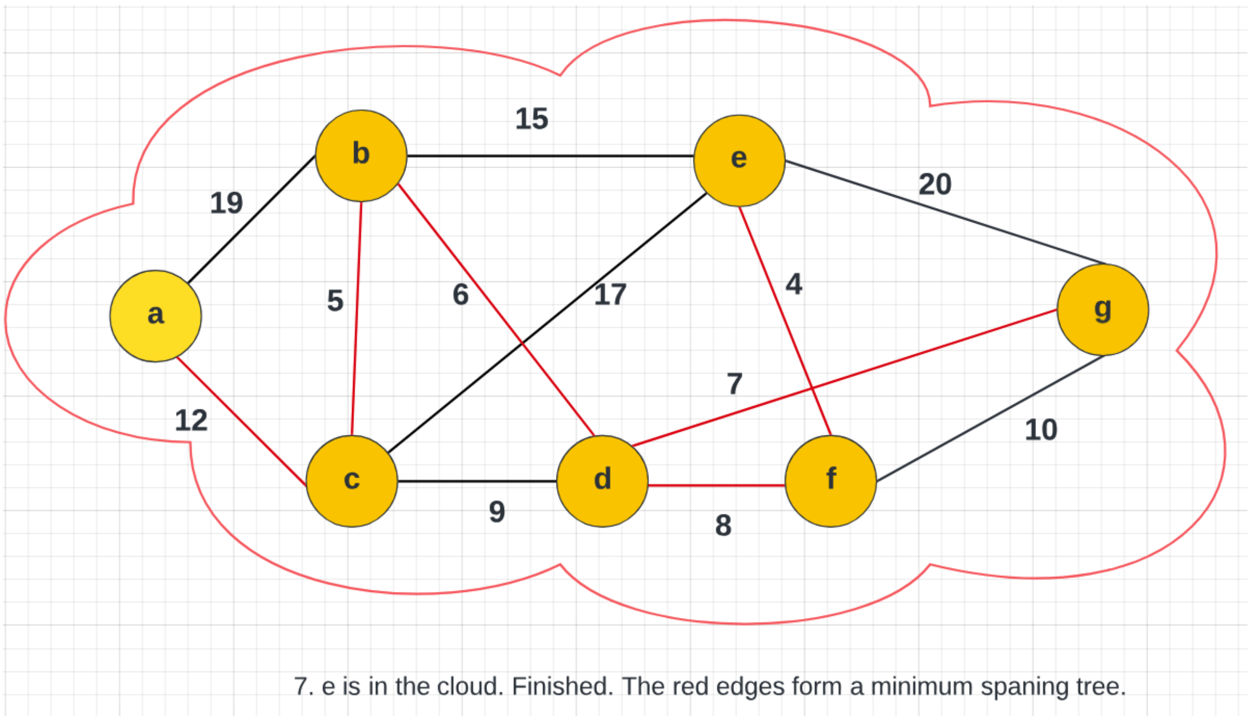
4. d is in the cloud. (d,g) is minimum-weight edge.



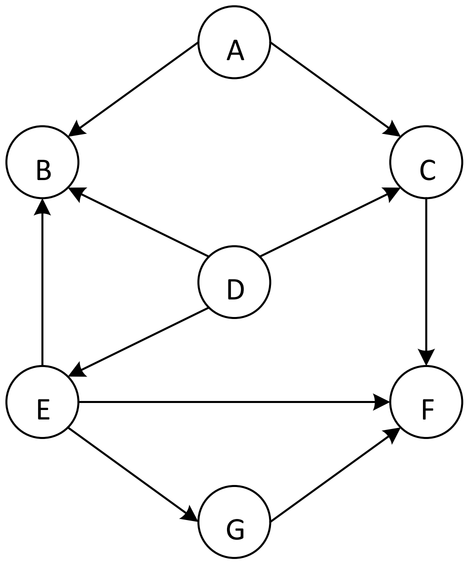
5. g is in the cloud. (d,f) is minimum-weight edge.



6. f is in the cloud. (f,e) is minimum-weight edge.


a
b
c
d
e
f
g
19
15
20
5
6
17
4
12
7
10
9
8
7. e is in the cloud. Finished. The red edges form a minimum spaning tree.

Problem 5 (60 points) This problem is a practice of writing a small program that stores and uses *follows relationship* in a graph. In a social network, people follow other people and such *follows relationship* can be represented as a directed graph, an example of which is shown below. For simplicity, people's names are shown as alphabets.



The *follows relationship* represented by the above graph are:

- A follows B and C
- B does not follow any person
- C follows F
- D follows B, C, and E
- E follows B, F, and G
- F does not follow any person
- G follows F

We will call the above graph *follows relationship graph*.

We distinguish two types of *follows relationship – direct follows* and *indirect follows*.

A person *X directly follows* a person *Y* if there is an edge from *X* to *Y* in the *follows relationship graph*.

A person *X indirectly follows* a person *Y* if there is a path from *X* to *Y* in the *follows relationship graph*.

Given a person *X*, we can form two sets of people. One set include all people *X* directly follows and the other set includes all people *X* indirectly follows.

For example:

- The set of people D directly follows is {B, C, E}
- The set of people D indirectly follows is {F, G}

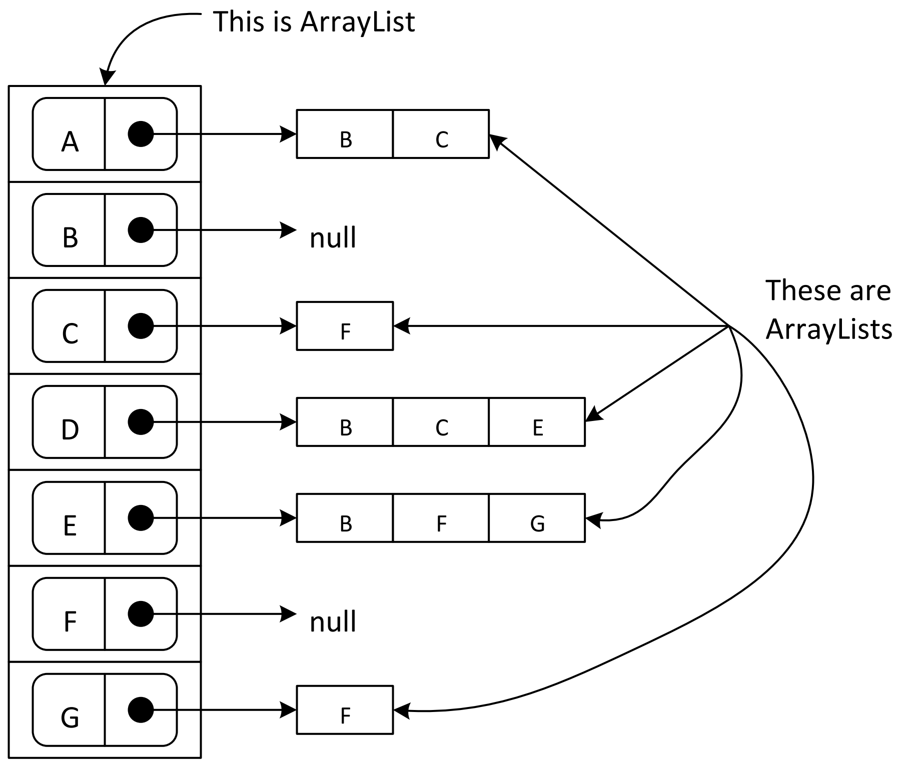
Note that D directly follows B but D also indirectly follows B along the path D -> E -> B. In this case, we include B only in the set of people D directly follows (i.e., we do not include B in the set of people D indirectly follows).

You are required to write a program named *Hw6_p5.java* that implements the following requirements.

- Your program must read *follows relationships* information from an input file named *follows_input.txt*. The input file corresponding to the above graph is:

  ```
  A, B, C
  B
  C, F
  D, B, C, E
  E, B, F, G
  F
  G, F
  ```

- Your program must store the follow relationships in an adjacency list.
- The adjacency list must be implemented as an ArrayList of *nodes*.
- A *node* must have the name of a person, say *X*, and a reference (or a pointer) to an ArrayList. This ArrayList must include all people *X* directly follows.
- You may want to use Java's ArrayList but you need to implement the *node* data structure yourself.
- The adjacency list representing the above graph is:



- In your program, you must write a method satisfying the following requirements:
- The name of the method must be *allFollows*.
- The method must receive two arguments: a person *X* and an adjacency list *adjList*
- Then, the method must print on the screen all people *X* directly follows and all people *X* indirectly follows. For example, if *D* and the above adjacency list are passed as arguments, your output must be:

```
D directly follows {B, C, E}
D indirectly follows {F, G}
```

- You also need to write a *main* method that is used to test the *allFollows* method. In the main method, you may want to invoke the *allFollows* method multiple times with different arguments to test your method.

Note that when the method *allFollows* determines the required output, it must use the *adjList* your program created.

Deliverables

You must submit the following files:

- *Hw6_p1_p4.pdf*: This file must include answers to problems 1 through 4.
  Hw6_p5.java
- Other files, if any.

Combine all files into a single archive file and name it *LastName_FirstName_hw6.EXT*, where *EXT* is an appropriate archive file extension, such as *zip* or *rar*.

Grading

Problem 1 through Problem 4:

- For each problem, up to 6 points will be deducted if your answer is wrong.

Problem 5:

- If your program does not compile, 36 points will be deducted.
- If your program compiles but causes a runtime error, 30 points will be deducted.
- Your program will be tested with three different input arguments and 10 points will be deducted for each wrong output. .