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.

A

B

C

D

E

F

G

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

**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.

A

B

D

C

E

F

G

H

I

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

S.

e

c

d

b

a

2

9

3

16

5

6

5

0

12

4

6

S

∞

∞

∞

∞

∞

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

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

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

a

b

c

d

e

f

g

15

6

19

12

5

9

17

8

10

20

4

7

**Problem 4-(1)**. Show the sequence of nodes in the order they are brought into the “cloud.” **Problem 4-(2)**. Show the minimum spanning tree T, generated by the algorithm, as a set of edges.

**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.

A

D

B

C

G

E

F

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:

A

B

C

D

E

F

G

B

C

F

B

C

E

B

F

F

This is ArrayList

These are

ArrayLists

null

null

G

* 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.