Swinburne University of Technology

Faculty of Information & Communication Technologies (FICT)

HIT3037/ HIT7037 Programming in Java

Assessment 2, 2012 Semester 1

Due: 5 pm Tuesday, May 15, 2012 (Week 11)

NOTE

1. This assignment can be done by ***one person*** *or* ***a pair.*** It is worth 25% of the total subject score.
2. You are required to electronically submit your work in a zip file that includes a user manual, detailed class diagram, and program source code (.java files only - excluding .class and project files). The submission system is located at https://esp.ict.swin.edu.au/.
3. Your program will be marked using the laptops or PCs of FICT. Before submitting your work, you are strongly recommended to test your program in the labs of FICT.
4. You may be required to attend an oral assessment.

Background:

A telecommunication company is going to set up a wireless communication network among cities in Australia. One and only one transmitter/receiver (transceiver) tower will be set up at each city to establish a wireless connection with one transceiver tower at another city. Each city must be connected to any other city either directly or indirectly through the transceiver towers at other cities. The cost of maintaining a wireless connection is in direct proportion to the distance between the associated two transceiver towers. To minimize the total maintenance cost, the telecommunication company aims to connect all cities into a *loop-free* wireless network with the shortest distance.

As an example, Table 1 shows the distances among 6 cities in Australia. There are many possible ways (called "network topologies") to create a loop-free network for these cities. Figure 1 and Figure 2 show two possible topologies. The transceiver tower in each city is represented as a node (circle) while the wireless connection is represented as an edge (line). Five edges are deployed to build Network Topology 1 (Figure 1) and Network Topology 2 (Figure 2). In this example, Network Topology 2 has a shorter total distance (8411 km), hence, is more economical to maintain.

A network topology is basically a *spanning tree* [1], where nodes and edges are put together. Kruskal's [2] and Prim's [3] algorithms are two well-known algorithms for searching a reasonably minimum spanning tree. In this assignment, you are asked to create a program that implements both Kruskal's and Prim's algorithms.

Table 1. Distance among six cities in Australia

	City 1	City 2	City 3	City 4	City 5	City 6
City 1 Adelaide	0	725	2707	1424	1534	2315
City 2 Melbourne	725	0	3432	873	2259	1915
City 3 Perth	2707	3432	0	4131	3937	4435

City 4 Sydney	1424	873	4131	0	2958	1122
City 5 Alice Springs	1534	2259	3937	2958	0	3210
City 6 Brisbane	2315	1915	4435	1122	3210	0

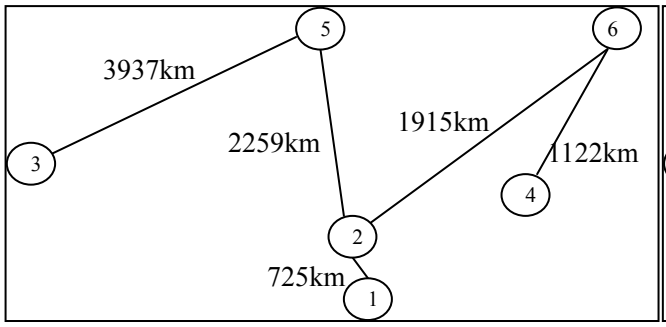


Figure 1. Total distance = 9958 km for Network Topology 1.

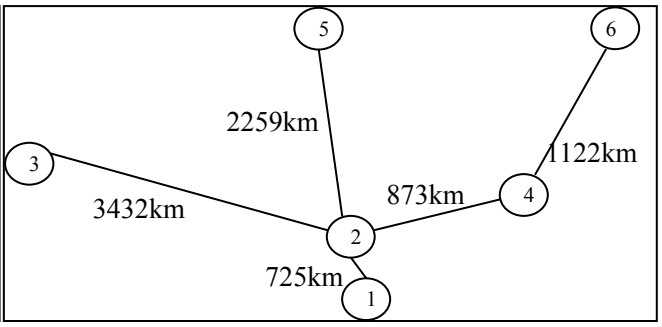


Figure 2. Total distance = 8411 km for Network Topology 2.

Program Input:

One essential input to your program is a file (with file extension *.txt*), which gives the *total number of city records* (in the first line) and summarizes the *connection distances* among cities (in the remaining lines). When two cities are forbidden to be directly connected due to geography, topography and climate factors, their distances will be marked as "N" in the file. Two input files are provided to you for testing purposes. Cities6.txt records every distance in Table 1, implying that each city can be directly connected to any other city. Cities12.txt has records for 12 cities, but some pairs of these 12 cities are forbidden to be directly connected. More different input files (those with unusual and invalid records) are worth to be included in your software testing. Your program will run slow if the number of cities is large, so in this assignment, we will only test your program with 12 cities at maximum.

Design Requirements:

Your program must have at least the following six classes: Assign2, WirelessNet, SpanningTreeAlg and its two subclasses, and InvalidUserInputException.

Class Assign2 is a JFrame (a GUI window).

1. contains a main() function. Executing "*java Assign2 directory_name/input_file_name.txt*" or "*java Assign2*" in DOS should make the JFrame visible.
2. lets WirelessNet to read and process the file after obtaining a valid directory and file name. If no input file is given in the command line or if the given directory or input file name is invalid, your program should pop up a warning message and ask the user to provide a valid directory and file name.
3. has the following basic properties.
   a. Title**:** WirelessNet + your student ID(s) – no "student name" should be shown in the title
   b. Some GUI components**:** for the user to invoke different functions of the system, such as
      - finding spanning trees for many inputs files given with a chosen algorithm
      - presenting each search step/result of the chosen algorithm in graphical and descriptive forms. Tables 2 and 3 are given here to illustrate these. Your program should present at least their 1st, 2nd, 4th, 5th columns. This task demands you to decide how to index each city, where to

position a city in a drawing, and how to colour the edges based on their properties (selected, unselected, cycle) so the presented can be sensible to the user.

Class WirelessNet should have at least the following methods:

1. *getTopologies*
   a *public* method with one possible parameter (*char* algoChoice).
   If algoChoice is 'k', run Kruskal's algorithm. If algoChoice is 'p', run Prim's algorithm. It will call the *solve* method in SpanningTreeAlg to run the associated algorithm.
2. *read*
   a *public* method to read data from the input file.
   If the input file does not follow the required format (omission or wrong placement of data in the file) or if it contains any invalid data (for example, mismatch data type), this method will throw a user-defined exception (called InvalidUserInputException) with a meaningful error message.
3. *Write*
   a *public* method to write each search step/result into a file. Your program should write the content of 1st, 4th and 5th columns of Tables 2 and 3 to the file. Each row of Tables 2 and 3 makes up a line in the file. The output format for each line is suggested to be: (Step no){list of SE}{list of UE} for Kruskal's algorithm; and (Step no.){list of SE}{list of UNE} for Prim's algorithm.
   This *write* method should be called within the getTopologies method. In other words, each time an input file is used to produce a topology, an output file should be generated. This output file must be named as "input_file_ name-ddmmyy_hhmm.txt", where ddmmyy and hhmm represent the file saving date and time, respectively.

Class SpanningTreeAlg:

1. a public class that contains some useful *protected* methods and *protected* instance variables to be shared by its two subclasses, called SpanningTreeAlgKruskal (implementing the Kruskal's algorithm) and SpanningTreeAlgPrim (implementing the Prim's algorithm).
2. has an *abstract* method, called "solve". Its subclasses must implement the *solve* method according to the algorithm.
3. provides some public *get* method(s), allowing WirelessNet to obtain the result of each search step taken to derive the solution.
4. In some occasions, Kruskal's and Prim's algorithms are required to make arbitrary choices of edges or nodes. See [2][3] for detail. You must make use of the functions in java.util.Random class to make such choices, but, do not call setSeed(long) method or Random(long) constructor anywhere in your program.

Class InvalidUserInputException

1. a user-defined exception. It extends the Exception class.
2. must not have unnecessary "method overloading" or "variable shadowing".

Other requirements

1. Your program must avoid or catch all kinds of exceptions, so its execution won't be interrupted and it won't enter any abnormal state. After catching an exception (regardless of whether it is a user-defined exception or system-based exception, such as IOException), your system should

output some meaningful error message related to this exception, so the user knows what went wrong.

2. Your program should use the Generic Collection classes (Vector, ArrayList, HashMap or Stack) to store data (perhaps, from an input file).
3. GUI related requirements
   a. Your system must have at least (i) JList or JComboBox, (ii) JTextfield or JTextArea, (iii) JLabel, (iv) JButton and (v) some kind of dialog components. It must also use Layout Manager(s) to well place these GUI components inside the JFrame.
   b. Code for all GUI classes must be hand-written, that is, not generated by any IDE.
   c. Your system should have good GUI and event handling design, so the user finds the system user-friendly (easy to learn/navigate the system, and fast to receive system responses).
4. Good OO design
   a. Your software must separate the GUI classes from non-GUI classes (where the core functions are implemented) as much as possible.
   b. Each class has meaningful methods and instance variables associated with it, so its design can be *reused* easily in other similar applications.
   c. Each method must have a single purpose and should be kept small.
   d. When a method is only invoked internally within its class, it should be declared as private.
   e. Each class cannot have public variables, but it can have public constants.
5. Good coding style standards must be followed:
   a. Indentation is consistent.
   b. No "magic number" (unexplained numbers) will be used, except for counters in a *for-loop*.
   c. Identifiers (i.e. naming of classes, methods, variables) are meaningful (not too abbreviated).
   d. Every Java file has a Javadoc class header comment showing its main purpose, version and author (including the student ID).
   e. Every method has a Javadoc method header comment indicating its purpose, as well as the purpose of individual parameters and the return value. Important pre-/post- conditions of the method (if any) are also described in its method header comment.
   f. Line/block comments are placed in front of any block of code that does something not obvious (Do not overdo line/block comments; e.g. do not comment normal language usage. Assume that these comments are aimed at proficient Java programmers).

User manual

Create a user manual to explain how the user should use the system. You can include some system snapshots in the user manual. This manual should not be long (10 pages at maximum). This user manual will not be given any marks because it is used by the tutor to run and mark your program.

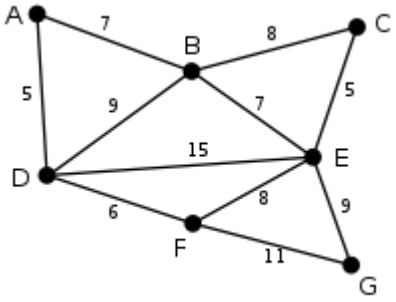
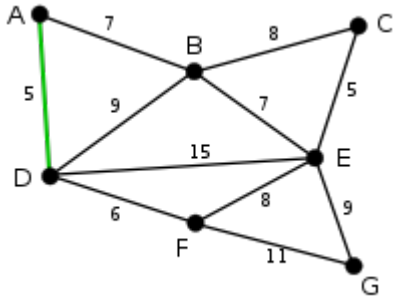
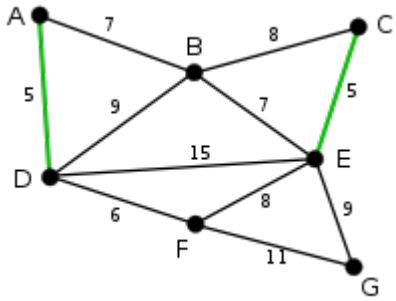
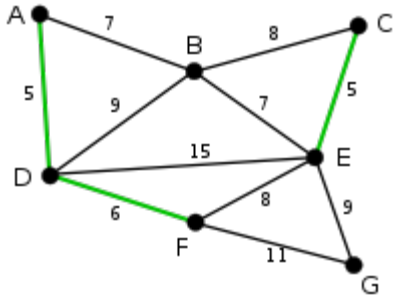
Detailed Class Diagram

Create a detailed class diagram showing (i) relationships among classes, and (ii) variables, methods and their visibility modifiers in each class.

References:

[1] http://en.wikipedia.org/wiki/Spanning_tree
[2] http://en.wikipedia.org/wiki/Kruskal%27s_algorithm
[3] http://en.wikipedia.org/wiki/Prim%27s_algorithm

Table 2. Result of each search step in Kruskal's algorithm

Step no.	Graph	Selected nodes (SN)	Selected edges (SE)	Unselected edges (UE)	Unselected nodes (UN)
1		{}			{A, B, C, D, E, F, G}
2		{A, D}	(A, D)	(A, B) = 7 (A, D) = 5 **V** (B,C) = 8 (B, D) = 9 (B, E) = 7 (C, E) = 5 **V** (D, E) = 15 (D, F) = 6 (E, F) = 8 (E, G) = 9 (F, G) = 11	{B, C, E, F, G}
3		{A, C, D, E}	(A, D) (C, E)	(A, B) = 7 (B,C) = 8 (B, D) = 9 (B, E) = 7 (C, E) = 5 **V** (D, E) = 15 (D, F) = 6 (E, F) = 8 (E, G) = 9 (F, G) = 11	{B, F, G}
4		{A, C, D, E, F}	(A, D) (C, E) (D, F)	(A, B) = 7 (B,C) = 8 (B, D) = 9 (B, E) = 7 (D, E) = 15 (D, F) = 6 **V** (E, F) = 8 (E, G) = 9 (F, G) = 11	{B, G}

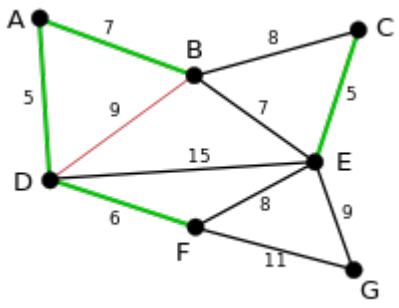
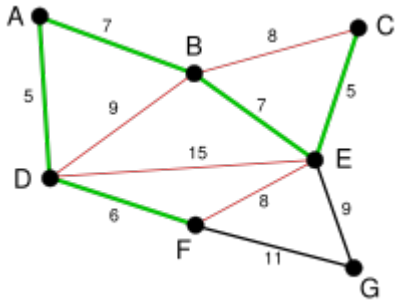
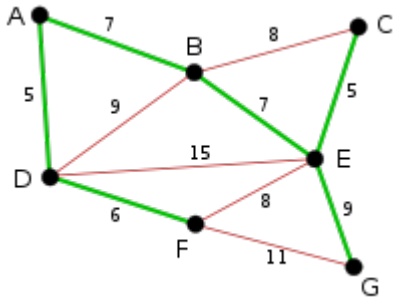
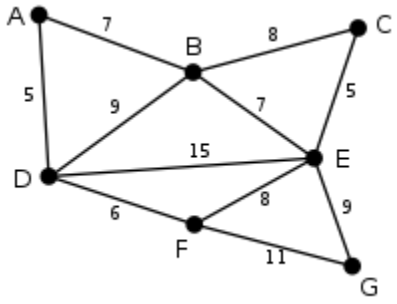
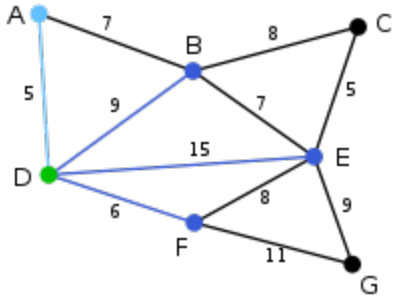
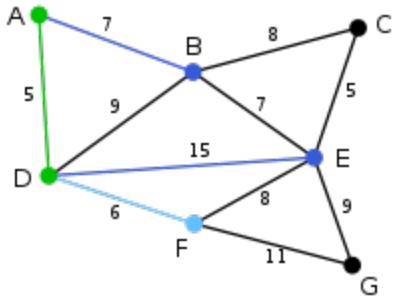
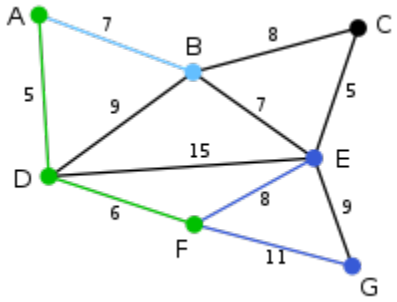
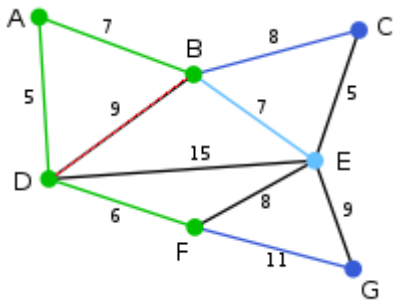
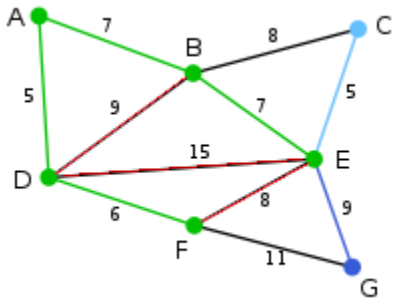
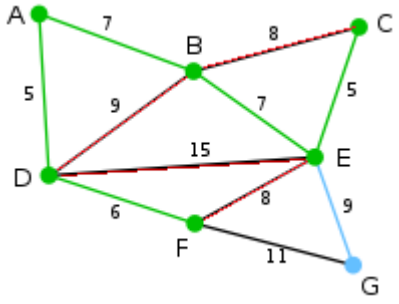
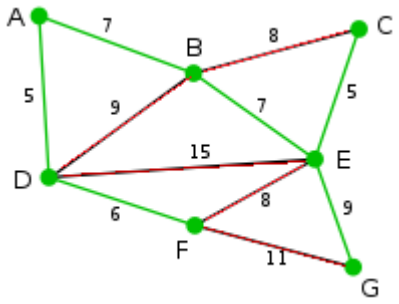
5		{A, B, C, D, E, F}	(A, D) (C, E) (D, F) (A, B)	(A, B) = 7 **V** (B,C) = 8 (B, D) = 9 cycle (B, E) = 7 (D, E) = 15 (E, F) = 8 (E, G) = 9 (F, G) = 11	{G}
6		{A, B, C, D, E, F}	(A, D) (C, E) (D, F) (A, B) (B, E)	(B,C) = 8 cycle (B, D) = 9 cycle (B, E) = 7 **V** (D, E) = 15 cycle (E, F) = 8 cycle (E, G) = 9 (F, G) = 11	{G}
7		{A, B, C, D, E, F, G}	(A, D) (C, E) (D, F) (A, B) (B, E) (E, G)	(B,C) = 8 cycle (B, D) = 9 cycle (D, E) = 15 cycle (E, F) = 8 cycle (E, G) = 9 **V** (F, G) = 11 cycle	{}

Table 3. Result of each search step in Prim's algorithm

Step no.	Graph	Selected nodes (SN)	Selected edges (SE)	Unselected neighbouring edges (UNE)	Unselected nodes (UN)
1		{}			{A, B, C, D, E, F, G}
2		{D}		(D, A) = 5 **V** (D, B) = 9 (D, E) = 15 (D, F) = 6	{A, B, C, E, F, G}
3		{A, D}	(D, A)	(D, B) = 9 (D, E) = 15 (D, F) = 6 **V** (A, B) = 7	{B, C, E, F, G}
4		{A, D, F}	(D, A) (D, F)	(D, B) = 9 (D, E) = 15 (A, B) = 7 **V** (F, E) = 8 (F, G) = 11	{B, C, E, G}

5		{A, B, D, F}	(D, A) (D, F) (A, B)	(B, C) = 8 (B, E) = 7 **V** (D, B) = 9 cycle (D, E) = 15 (F, E) = 8 (F, G) = 11	{C, E, G}
6		{A, B, D, E, F}	(D, A) (D, F) (A, B) (B, E)	(B, C) = 8 (D, B) = 9 cycle (D, E) = 15 cycle (E, C) = 5 **V** (E, G) = 9 (F, E) = 8 cycle (F, G) = 11	{C, G}
7		{A, B, C, D, E, F}	(D, A) (D, F) (A, B) (B, E) (E, C)	(B, C) = 8 cycle (D, B) = 9 cycle (D, E) = 15 cycle (E, G) = 9 **V** (F, E) = 8 cycle (F, G) = 11	{G}
8		{A, B, C, D, E, F, G}	(D, A) (D, F) (A, B) (B, E) (E, C) (E, G)	(B, C) = 8 cycle (D, B) = 9 cycle (D, E) = 15 cycle (F, E) = 8 cycle (F, G) = 11 cycle	{}

HIT3037/ 7037 Feedback to Assessment 2

(worth 25% of the total subject score)

Student ID/ name:	Lab class:

Items	Marks
Non-GUI functionality and computation correctness (25 marks)	+
GUI/Usability design and event handling (45 marks)	+
Exception handling and avoidance (10 marks)	+
Class diagram, OO design and implementation (14 marks)	+
Coding styles and comments (6 marks)	+
Total marks (100 marks)	+

Late penalty (10% penalty per working day) ***Formula***: Number of days late × 10% × Total marks	-

FINAL MARKS (= Total marks – Late penalty)	

COMMENTS:

Special cases: (Remark: Any of the following cases will override the marks given above)

(a) Detection of plagiarism (YES or NO); Set-up of an oral assessment (YES or NO)