Data Structure Project 5

Graph

Introduction

Design a program to find the ***minimum spanning tree (MST)*** of a graph, and find the ***shortest path*** between two nodes in the graph.

Requirements

Project requirements

Write a program implementing basic graph algorithms with the following requirements:

- **Minimum spanning tree**
  - Find the minimum-cost spanning tree of the input graph using *Kruskal's algorithm* or *Prim's algorithm.*
  - Output all edges in the minimum spanning tree.
- **Shortest path**
  - Find the single-source shortest path between two nodes in the graph using *Dijkstra's algorithm*.
  - Output all edges from source to destination.

Program requirements

- Standard C++ language, without any external or OS-dependent library
  - Visual C++ 2010 or GCC 4.5/4.6 are recommended
- C++ STL Containers (vector, deque, list, stack, queue, …) are not allowed

- Do not include any system command functions (for example, *system("pause");*) in your program

Input and Output

Input

The input data are stored in a text file. There may be more than one testing cases in the file. For each testing case, the following lines describe the problem:

- **1st line**: a positive integer v indicating total number of nodes in the undirected graph. You should establish a graph with vertices numbered from 0 to $v-1$. v is always smaller than 1,000.
- **2nd line**: a positive integer e indicating the total number of edges in the undirected graph.
- **3rd to $(\boldsymbol{e}+\mathbf{2})^{\text{th}}$ line**: edge information. In each line, there are three integers v_1, v_2, and w; the v_1 and v_2 indicate the vertex numbers, the w is the weight/length/distance between v_1 and v_2. Three numbers are separated by a space character. The w is always larger than 0, and v_1, v_2 are always smaller than v, you don't have to check the correctness of the data.
- **$(\boldsymbol{e}+\mathbf{3})^{\text{th}}$ to $(\boldsymbol{e}+\mathbf{3}+\boldsymbol{s})^{\text{th}}$ line**: shortest path problem. Given s pairs of vertices, you should find the shortest path of each pair. In each line, there are two numbers vs_1 and vs_2 indicating the source and destination vertices in the graph; two numbers are separated by a space character.
- **Last line**: a symbol indicating the end of the testing case. Process another testing case if the character in this line is "z", or exit your program if the character is "x". Output a line-change character ('\n').

The filename of the input file will be passed through command line argument. You should use the filename string passed to your main function.

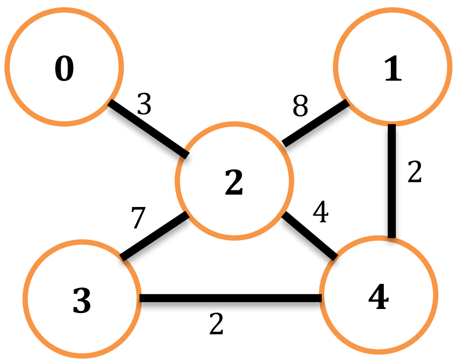
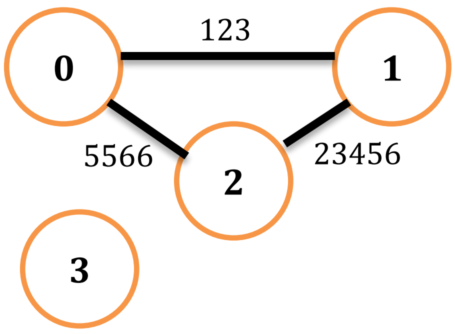
Output

The output of both minimum spanning tree and shortest path are sequences of edges. Print an edge in (v_1, v_2) format, where v_1 and v_2 are the vertex numbers; and for a sequence of edges, put a semicolon and a space character after each edge. For example *(1,2); (2,3); (3,4);* indicating a tree/path with three edges. Output the result to *stdout* using *std::cout* or *printf*.

The outputting information is described below:

- After reading the $(e+2)^{\text{th}}$ line, print the minimum spanning tree of the graph. If the minimum spanning tree does not exist, output "*No spanning tree*".
- Print the shortest path after reading each line from $(e+3)^{\text{th}}$ to $(e+3+s)^{\text{th}}$ line. The outputting edge sequence should follow the order of the path. That is, the second number in an edge should be the same as the first number in the next one. Output "*Not exist*" if the path is not found.

Example

Input.txt	5 6 0 2 3 1 2 8 1 4 2 2 4 4 2 3 7 3 4 2 0 4 1 4 2 3 z 4 3 0 1 123 1 2 23456 2 0 5566 0 0 0 3 x	Illustration:  
Command arguments	Input.txt	

Output	(0,2); (2,4); (4,3); (4,1);
	(0,2); (2,4);
	(1,4);
	(2,4); (4,3);
	
	No spanning tree
	(0,0);
	Not exist	

Note: your program should output **exactly the same** format described above. Additional letters, punctuations, spaces, or characters will be treated as wrong results.

Scoring Criteria

- Correctness 80%
  - TA will test your program with some basic data
- Extreme cases 10%
  - More difficult input data
- Source code quality and readability 10%
  - Comments, variable/function naming, consistency, …

Project Submission

Please pack all your **source code** files (*.cpp and/or *.h) in the studentID.zip file, ex: *0016789.zip*, and upload to e3. <u>DO NOT UPLOAD/CONTAIN OTHER FILES</u>. The dead line is 23:59, 25 Dec. 2012.

Specify the compiler you used in the very beginning (first/second line) of your codes using comments is recommended. Also, if you finished the bonus function, leave some messages about this in the beginning of your codes.

Note

Plagiarism / copy others work are not allowed.

If you have any problem about this project, please contact TA through email or come to EC 637 for more information.