COP 5536 Spring 2015

Programming Project

Due Date: April 15, 2015, 11:55 PM Eastern Time Submission Website: Sakai

1. Problem description

Part 1

You are required to implement Dijkstra's Single Source Shortest Path (*ssp*) algorithm for undirected graphs using Fibonacci heaps. You must use the adjacency list representation for graphs.

Part 2

You are required to implement a routing scheme (*routing*) for a network. Each router has an IP address and packets are forwarded to the next hop router by longest prefix matching using a binary trie (See Lectures 27-30). For each router R in the network, call *ssp* implemented in Part 1 to obtain shortest path from R to each destination router Y. To construct the router table for R, for each destination Y, examine the shortest path from R to Y and determine the router Z just after R on this path. This gives you a set of pairs <IP address of Y, next-hop router Z>. Insert these pairs into a binary trie. Finally, do a postorder traversal, removing subtries in which the next hop is the same for all destinations. Thus, multiple destinations having a prefix match and the same next hop will be grouped together in the trie.

2. Input/Output Requirements

You may implement this assignment in Java or C++. Your program must be compilable and runable on the Thunder CISE server using gcc/g++ or standard JDK. You may access the server using Telnet or SSH client on thunder.cise.ufl.edu.

You must write a makefile document which creates two executables. The names of your executables must be ssp and routing.

Part 1

Your program has to support redirected input from a file "file-name" which contains undirected graph representation. The command line is:

$ssp file_name source_node destination_node

Read the input from a file ‘file-name’. The arguments ‘source_node’ and ‘destination_node’ are integers representing the vertex numbers of the source node and the destination node for Dijkstra’s algorithm respectively.

Input format

---

n m

v1 v2 w1

v2 v3 w2

...

vi vj wm

The number of vertices and edges, respectively, in the first line

The undirected edge (v1, v2) with weight w1

The undirected edge (v2, v3) with weight w2

...

The undirected edge (vi, vj) with weight wm

Assume that vertices are labeled from 0 to n-1 (n<1000000) and weights are positive integers (0<wi<1000).

An example input is shown below:

3 2

0 1 5

1 2 8

The graph consists of three vertices {0, 1, 2} and two undirected edges <0,1> and <1,2> with costs 5 and 8 respectively.

Output format

The output of this part consists of two lines.

The first line has one integer, the total weight.

The second line shows the path starting with source_node and ending with destination_node separated by a space between nodes. Don’t use any character other than space between nodes.

An example for the given input above when the source is 0 and the destination is 2:

13

0 1 2

Print the output to the standard output stream.

Part 2

Your program has to support redirected input from the files "file-name" which contains an undirected graph and the IP addresses. The command line is:

$routing file_name_1 file_name_2 source_node destination_node

Input format

Read the input graph from a file ‘file-name_1’. This has a similar format as shown in Part 1. The weights represent already existent traffic in the network from other applications. The arguments ‘source_node’ and ‘destination_node’ are integers representing the vertex number of the source node and the destination node respectively for a packet.

The argument ‘file_name_2’ has the list of IP addresses, for each of the nodes. The IP addresses are given in the form of 4 1-byte unsigned integers as shown below.

IP_0

IP_1

...

IP_n-1

Example:

192.168.1.1

192.168.4.1

192.168.1.27

Output format

The output of this part consists of two lines.

The first line has one integer, the total weight.

The second line shows the path starting with source_node and ending with destination_node. But this time you need to print the matched prefixes separated by a space between prefixes for each node on the path. Don't use any character other than space between nodes.

Print the output to the standard output stream.

3. Submission

The following contents are required for submission:

1. Makefile: Your makefile must be directly under the zip folder. No nested directories.

2. Source Programs: Provide comments.

3. REPORT: The report must be in PDF format. State which compiler you used. Present function prototypes showing the structure of your programs. Include the structure of your program. A list of function prototypes is not enough.

To submit, Please compress all your files together using a zip utility and submit to the Sakai system. You should look for Assignment->Project for the submission. Your submission should be named LastName_FirstName.zip.

Please make sure the name you provided is the same as the same that appears on the Sakai system. DO NOT submit directly to a TA. All email submission will be ignored without further notification. Please note that the due day is a hard deadline. No late submission will be allowed. Any submission after the deadline will not be accepted.

4. Grading Policy

Grading will be based on the correctness and efficiency of algorithms. Below are some details of the grading policy.

Correct and efficient implementation and execution: 60%. There will be a threshold for the running time of your program in Part1. If your program runs slow, we will assume that you have not implemented the Fibonacci heap properly. Links to larger graphs (1 million nodes) will be posted soon. For the time being, try testing you program with the sample input and output provided. Sample input and output files for part 2 will be posted soon.

Comments and readability: 15%

Report: 25%

Note: If you do not follow the input/output or submission requirements above, 25% of your score will be deduced. In addition we may ask you to demonstrate your projects.

If you have any question, please contact Eyup Serdar Ayaz at ayaz@cise.ufl.edu.