EE 450
Lab1: C/C++ Implementation of Distance-Vector (DV) Routing Algorithm

Summer 2019 Nazarian

Student ID: _____ Name: _____
Assigned: May. 19th
Due: Friday, May. 31st, at 11:59pm.
Maximum points: 100
Late submissions will be accepted only in the first two days after deadline with a maximum penalty of 15% per day: For each day, submissions between 12 and 1am: 2%, 1 and 2am: 4%, 2 and 3am: 8% and after 3am: 15%.

- **Lab1 is based on individual work. No collaboration is allowed. We may pick some students in random to demonstrate their design and simulations. Please refer to the syllabus for a summary of AI policies (including the penalties for any violation.) If you have any doubts about what is allowed or prohibited in this course, please contact the instructor.**

A. Background

What is Distance-Vector (DV) Routing Algorithm?

The distance-vector routing algorithm, also known as the distance vector protocol, is one of the two major routing algorithms in the network layer, the other major algorithm being the link-state routing algorithm.

In a computer network connected by many routers, e.g., the Internet, data packets from the original computer needs to travel along many intermediate routers to arrive their destination computer. In a datagram network, two successive packets of the same computer pair may take different routes, and a routing decision is made for each individual link. In a virtual circuit network, a routing decision is made when the virtual circuit is set up and all packets of this virtual circuit use this path. In both cases, the job of routing algorithms is to select the path with the best performance, e.g., lowest delay, best reliability, or smallest monetary cost.

In general, a computer network can modeled as a weighted directed graph (digraph) G=(V,E) where V is the set of nodes representing the routers and computers, and E is the set of directed edges representing the links that connect different nodes. A length metric can be assigned each edge to represent the transmission delay, failing probability, or monetary cost of each link. Thus, the routing decision is indeed the problem of finding a shortest path in a graph, which is a classical problem in algorithm and graph theory.

The distance-vector routing algorithm is an implementation of the Bellman-Ford algorithm developed by Bellman and Ford to compute the shorted path in a weighted digraph. The

Bellman-Ford algorithm is one of the most important applications of the method of dynamic programming in algorithm design.

How does Bellman-Ford Algorithm work?

Let c(x,y) be the length of the directed edge from node x to node y. Let $d_x(y)$ be the length of the shortest path from note x to node y. For all x and y, $d_x(y)$ must satisfy the Bellman-Ford equation given as follows:

$$d_x(y) = \min_{v \in V:(v,y)\ is\ an\ edge} \{d_x(v) + c(v,y)\}$$

The Bellman-Ford equation is quite intuitive. If the shortest path from node x to node y jumps from node v to node y in the last hop, then the residual path, which is from node x to node v, must be the shortest path from node x to node v. Otherwise, we could have replaced the residual path by a better one. Thus, to find the shortest path from node x to node y, we just need to check among all possible first hop and find the one yielding the smallest overall path length.

Now you may ask: "Wait a moment! We know the value of c(v,y), which is the edge length, but how can we know the value of $d_x(v)$ for different v?".

If you have worked with dynamic programming before, you may recall that in dynamic programming the first step is usually to find a recurrence equation, which is similar to the Bellman-Ford equation given above, to relate the original problem with a smaller problem. Once we have the recurrence equation, we will first solve the base problem, which is the smallest one and is usually trivial, and use the recurrence equation iterative and in a reverse order and in the end we are able to solve the original problem.

In the shortest path problem where we aim to find the shortest paths from node 0 to all the other nodes, the base case is easy and is $d_0(0) = 0$. But what is the order of calculating of $d_0(x)$ for $x \neq 0$? This is really not clear. In fact, Bellman and Ford show that we do not need an order of calculating $d_0(x)$ for $x \neq 0$. Instead, we do the following: First, we initialize $d_0(0) = 0$ and $d_0(x) = \infty$ for $x \neq 0$. Then, we use Bellman-Ford equations to update $d_0(x)$ for all node x based on the initialized $d_0(0)$ and repeat the update procedure based on $d_0(x)$ from the last iteration for at most n-1 iterations, where n is the number of nodes in this graph. These $d_0(x)$ gives the length of the shortest path from node 0 to all node x. If we record the intermediate node v giving the updated $d_0(x)$ during the update procedure, we may also recover the corresponding path associated with each $d_0(x)$.

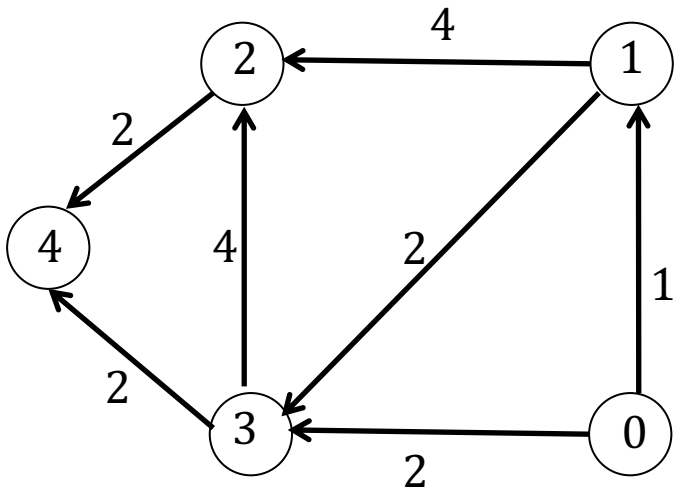
Let n be the number of nodes in the graph. The pseudo-code of Bellman-Ford algorithm to finding the length of the shortest path from node 0 to all the other nodes is described as follows:

- Initialization: $d_0(0) = 0$.
- For i from 1 to n-1
  - $s_0(x) = \min_{v \in V:(v,x)\ is\ an\ edge} \{d_0(v) + c(v,x)\}$ for all $x \neq 0$
  - $d_0(x) = s_0(x)$ for all $x \neq 0$
- Return $d_0(x)$ for all x

Apply Bellman-Ford Algorithm to a Simple Example:

Now let us consider a simple example and use Bellman-Ford algorithm to find the shortest path.

Consider the graph given as follows and we need to find the shortest path from node 0 to all the other nodes.



The Bellman-Ford algorithm works as follows:

Iterations	$d_0(0)$	$d_0(1)$	$d_0(2)$	$d_0(3)$	$d_0(4)$
0(Initialization)	0	Infinity	Infinity	Infinity	Infinity
1	0	1(pre=0)	Infinity	2(pre=0)	Infinity
2	0	1(pre=0)	5(pre=1)	2(pre=0)	4(pre=3)
3	0	1(pre=0)	5(pre=1)	2(pre=0)	4(pre=3)
4	0	1(pre=0)	5(pre=1)	2(pre=0)	4(pre=3)

This example is extremely simple and you may do not need Bellman-Ford algorithm. But we will still have some interesting observations when applying Bellman-Ford algorithm to it.

1) In the pseudo code, we describe that n-1 iterations are required. In the above example with n=5, however, only 3 iterations are enough. This is because row 3 in the table is calculated from row 2 based on the Bellman equation; and row 4 is calculated from row 3 based on the Bellman equation. Since row 3 is identical to row 2, we can predict that row 4 should be identical to row 2. This is known as the early termination property of Bellman-Ford algorithm. That is, if after any iteration the $d_0(x)$ vector is identical to that from the last iteration, then we can terminate our algorithm. In practice, this is a very nice property. (Imagine that in the INTERNET, we could have hundreds of thousands of routers, which means we may require hundreds of thousands of iterations for Bellman-Ford if we do not have the early termination property.)
2) In the table, we also record the predecessor of each node, which is the node that attains the minimum in the Bellman equation during each iteration. This information can be used to find the shortest path. For example, if we want to have the shortest path from node0 to node4. From the table, we know $d_0(4) = 4$. To find the path, we know the predecessor of node 4 is node 3. Check the table again, we

know the predecessor of node 3 is node 0. Thus, the shortest path from node 0 to node 4 is the path 0→3→4.

B. Your tasks for this assignment

1. Implement Bellman-Ford algorithm in either C or C++.
    a. Your compiled C/C++ program should be able to read a graph described in an external text file and find the shortest paths from node 0 to all the other nodes. The file name of the graph should be an input parameter of your complied program. Suppose your compiled C/C++ program is BellmanFord. A command of running your program can be:
    "./BellmanFord graphfile.txt"
    where "graphfile" is the name of the file that describes a graph.
    b. The external file should describe the graph in the above example as follows:
    0 1 inf 2 inf
    inf 0 4 2 inf
    inf inf 0 inf 2
    inf inf 4 0 2
    inf inf inf inf 0

    where the first line is the lengths of the edges from node 0 to nodes 0-4, the second line is the lengths of the edges from node 1 to nodes 0-4; and so on so forth. Note that by conversion, the length from one node to itself is zero. We use "inf" to represent that there is no edge from one node to another, i.e., the length is infinity.
    c. Your code should realize the early termination property of Bellman-Ford algorithm.
    d. Your code should be aware of negative loop.
    e. The output of your program should also be a file, where the first line should output the vector $d_0(x)$; the second line should be the shortest path from node 0 to node 0, which is trivial; the third line should be the shortest path from node 0 to node 1; and so on so forth. The last line should be the number of the required iteration. For example, if the output of your program for the above example should be:

    0,1,5,2,4
    0
    0→1
    0→1→2
    0→3
    0→3→4
    Iteration: 3

    Using commas and arrows in the above example to make your output file human readable.

    If there is a negative loop:

    Output: Negative Loop Detected

f. If there exist negative loops, output any one of the negative loops.

2. Run your implementation on a set of graphs that we have provided, and gather results generated by your program.
3. Prepare a complete report that summarizes codes, key aspects of implementation, experimental results and the output file format.

C. Submission Rules

1. You should submit a zip file that contains all your code files and report as well as a readme.txt file. The name of the zip file should follow this pattern: Firstname_Lastname_lab1.zip. For example for Mohammad Mirza the zip file should be named: Mohammad_Mirza_lab1.zip
2. Along with your code files, include a **readme.txt** with the following information:
   i. Your full name and student ID
   ii. Compilation steps to run your programs (Optional: include the makefile to compile your code)
   iii. Additional info, if you think necessary.