Programming Assignment 3: Routing Algorithm Implementation Assignment

Due 18 Jun by 17:00 **Points** 200 **Available** after 7 May at 8:00

Assessment

Weighting:	20% (200 Marks)
Task description:	In this assignment, you will be writing code to simulate the Distance Vector routing protocol in a network of routers.
Academic Integrity Checklist	Do ✅ Discuss/compare high level approaches ✅ Discuss/compare program output/errors ✅ Regularly commit your work and write in the logbook Be careful ❓ Code snippets from reference pages/guides/Stack Overflow must be attributed/referenced. ❓ Only use code snippets that do not significantly contribute to the exercise solution. Do NOT ❌ Submit code not solely authored by you. ❌ Post/share code on Piazza/online/etc. ❌ Give/show your code to others

Before you begin

Although this practical is marked automatically, all marks will be moderated by a marker using the following information.

You must complete a logbook as you develop your code (this process should be familiar to those that have or are doing the Computer Systems course). You can find details on how to do that and what that should look like **in this guide.**

- During manual marking and plagiarism checks, we will look look at your development process. If we do not see a clear path to a solution (i.e. code changes and regular commits and logbook entries **reflecting** your learning to develop your implementation you **may forfeit up to 200 marks.**

- *An example case of forfeiting all 200 marks would be the sudden appearance of working code with no prior evidence of your development process*.
- It is up to you to provide evidence of your development through regular submissions and logbook entries**.**

This practical requires thought and planning. You need to start early to allow yourself time to think of what and how to test before modifying any code. Failing to do this is likely to make the practical take far more time than it should.
Starting the practical in the final week or days is likely to be an unsuccessful strategy with this practical; further your logbook entries are likely to be overlapped in close succession and, depending on the quality of the entries, is likely to lead to a mark that will be scaled lower (due to possibly poor documentation in the logbook).

Aims

- Learn about routing protocols and route propagation.
- Implement a routing protocol.

Overview

In this assignment, you will be writing code to simulate a network of routers performing route advertisement using a Distance Vector routing protocol.

You will need to implement the algorithm in its basic form, and then with **poisoned reverse/route poisoning** to improve the performance of the protocol. Your implementation will need to ensure that the simulated routers in the network correctly and consistently converge their distance and routing tables to the correct state.

You will find a more detailed description of the Distance Vector algorithm in the course notes and in section 5.2.2 of Kurose and Ross, Computer Networking, 7th Edition.

Your Task

Part 1 (DV algorithm)

You are to produce a program that:

1. Reads information about a topology/updates to the topology from the standard input.
    - Handle bad input:
        - Printing a reasonable error message and
        - Terminating the program with exit code 1;
        - Bad input should not cause your program to crash.
2. Uses DV algorithm or DV with PR algorithm, as appropriate, to bring the simulated routers to convergence.

- Output the distance tables in the required format for each router at each step/round.
- Output the final routing tables in the required format once **convergence** is reached.

3. Repeats the above steps until no further input is provided.

The DV algorithm program you are to provide should be named `DistanceVector`.

Part 2 (DV with PR algorithm)

You will need to modify/write a second version of the program that uses poisoned reverse/route poisoning.

The DV with PR algorithm program you are to provide should be named `PoisonedReverse`.

In Your Task

You will need to craft any internal data structures and design your program in such a way that it will reliably and correctly converge to the correct routing tables. We have *deliberately* not provided you with a code templates and this means that you will have more freedom in your design but that you will have to think about the problem and come up with a design.

You will need to record your progress and development cycle in a logbook as described in the 'Before you Begin' section above.

Programming Language/Software Requirements

You may complete this assignment using the programming language of your choice, **with the following restrictions**:

- For **compiled** languages (Java, C, C++ etc.) you must provide a Makefile.
  - Your software will be compiled with `make` (***Please look at this resource on how to use Makefile build tool: https://makefiletutorial.com/ (https://makefiletutorial.com/)***)
  - Pre-compiled programs will ***not*** be accepted.
- Your implementation must work with the versions of programming languages installed on the Web Submission system, these are the same as those found in the labs and on the **uss.cs** server and include (but are not limited to):
  - **C/C++:** g++ (GCC) 4.8.5
  - **Java:** java version "1.8.0_201"
  - **Python:** python 2.7.5 or python 3.6.8
- Your implementation may use any libraries/classes available on Web Submission system, but **no external libraries/classes/modules**.
- Your programs will be executed with the command examples below:
  - For C/C++

```
make
./DistanceVector
./PoisonedReverse
```

You can find a ***simple*** example makefile for C++ **HERE** (https://myuni.adelaide.edu.au/courses/64831/files/8781526/download?download_frd=1) . ***A good resource is here: https://makefiletutorial.com/*** (https://makefiletutorial.com/)
This will ***need to be customised*** for your implementation. Make sure you use tabs (***actual tab characters***) on the indented parts

- For java:

```
make
java DistanceVector
java PoisonedReverse
```

You can find a ***simple*** example makefile for Java **HERE** (https://myuni.adelaide.edu.au/courses/64831/files/8781519/download?download_frd=1) . ***A good resource is here: https://makefiletutorial.com/*** (https://makefiletutorial.com/)
This will ***need to be customised*** for your implementation. Make sure you use tabs (***actual tab characters***) on the indented parts

- For Python:

```
./DistanceVector
./PoisonedReverse
```

Programs written using an interpreted language such as python:

- Will need to use UNIX line endings (always test on a uni system such as the **uss** cloud instance).
- Will not be built with make (as shown above, because they are not compiled)
- Will require a 'shebang line' at the start of your file to run as above.
  e.g. `#!/usr/bin/env python2` (Python 2) or `#!/usr/bin/env python3` (Python 3).

Algorithm

Distance Vector (DV)

At each node, *x*:

```
D_x(y) = minimum over all v { c(x,v) + D_v(y) }
```

The cost from a node x to a node y is the cost from x to a directly connected node v plus the cost to get from v to y. This is the minimum cost considering both the cost from x to v and the cost from v to y.

```
At each node x:

INITIALISATION:
   for all destinations y in N:
      D_x(y) = c(x,y) /* If y not a neighbour, c(x,y) = Infinity */
   for each neighbour w
      D_w(y) = Infinity for all destinations y in N
   for each neighbour w
      send distance vector D_x = [D_x(y): y in N] to w
```

```
LOOP
   wait (until I see a link cost change to some neighbour w or until
      I receive a distance vector from some neighbour w)
   for each y in N:
      D_x(y) = min_v{c(x,v) + D_v(y)}

   if D_x(y) changed for any destination y
      send distance vector D_x = [D_x(y): y in N] to all neighbours.

FOREVER

Note: Infinity is a number sufficiently large that no legal cost is greater than or equal to i
nfinity.
The value of infinity is left for you to choose.
```

Poisoned Reverse (PR)

In Poisoned Reverse, if a node A routes through another node B to get to a destination C, then A will advertise to B that its distance to C is Infinity. A will continue to advertise this to B as long as A uses B to get to C. This will prevent B from using A to get to C if B's own connection to C fails.

Key Assumptions

In a real DV routing environment, messages are not synchronised. Routers send out their initial messages as needed.

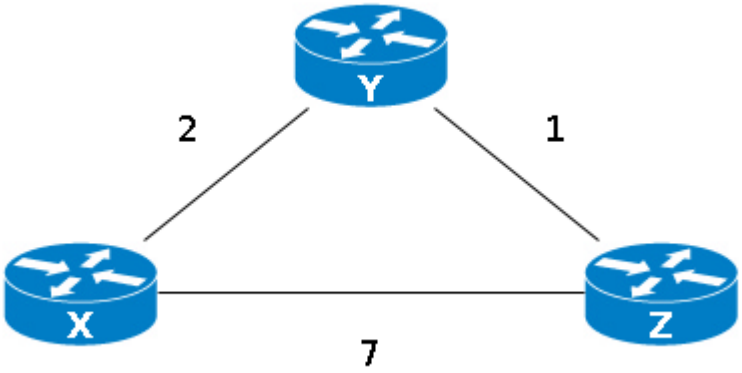
In this environment, to simplify your programs, you should assume:

- All routers come up at the same time at the start.
- If an update needs to be sent at a given round of the algorithm, all routers will send their update at the same time.
- The next set of updates will only be sent once all routers have received and processed the updates from the previous round.
- When a link to a directly connected neighbour is updated or an update is received, and the update affects the routing table:
  - Choose the new best route from the distance table, searching in alphabetical order.
  - Where multiple best routes exist, the first one is used (in alphabetical order, by router name)
  - Send an update (in the next round).

You should confirm for yourself that the assumptions above will not change the least-cost path routing table that will be produced at the nodes. (Although, for some topologies, you may take different paths for the same cost.)

Sample Topology

At its most basic, your program should be able to calculate the correct routing tables for the following network:



As shown in lectures

Input Format

Your program will need to read input from the terminal/command line's standard input.

The expected input format is as follows:

```
X
Y
Z

X Z 7
X Y 2
Y Z 1

X Z 5
Z Y -1
```

1. The input begins with **the name of each router/node in the topology.**
   - Each name is on a new line
   - Router names are case-sensitive
   - Router names may not contain spaces
2. **An empty/blank line** separates each section.
3. The next section of input contains the details of **each link/edge in the topology**.
   - Written as the names of two routers/nodes followed by the weight of that link/edge, all separated by spaces.
   - e.g.

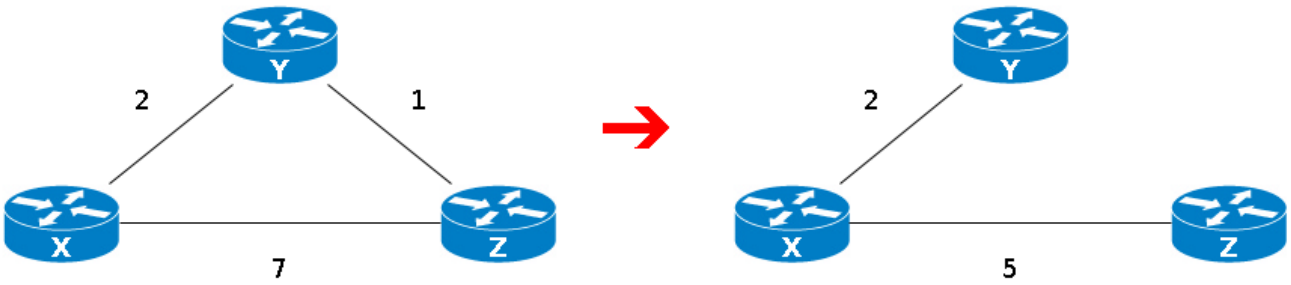
     ```
     Y X 2
     Y Z 1
     ```

   - Weight values should always be integers.
   - A weight value of `-1` indicates a link/edge to remove from the topology if present.
   - Once all values in this section are read in, your algorithm should be run with this topology information to bring your simulated routers to convergence.
4. **An empty/blank line** separates each section.

5. The next section again contains the details of **each link/edge in the topology**.
   - The values in each of these sections should be used to update the topology.
   - If a link is not included in any section, it should remain unchanged.
   - As above, a weight value of `-1` indicates a link/edge to remove from the topology if present.
   - Again, once all values in each of these sections are read in, your algorithm should be run with this topology information to bring your simulated routers' routing tables to convergence.
     A user may input as many of these sections as they like.

     25/05/21 CLARIFICATION: This section may also be omitted, i.e. A user may input 0 or more of these sections.
6. This repeats until **2 empty/blank lines** are received, at which point the program exits normally.

The input above matches the sample topology with the following updates:



Expected Output Format

As this is Distance Vector, a node will only be able to communicate with its neighbours. Thus, node X can only tell if it is sending data to Y or Z. You should indicate which interface the packets will be sent through, as shown below.

Your program should print 2 types of output that repeat:

1. The distance table of each router in the following format:

```
router X at t=0
  Y   Z
Y 2   INF
Z INF 7
```

   1. The name of the router, and the current step (starting at 0)
   2. The name of the destination router
   3. The name of the next hop router
   4. The current known distance (from the current router, to the destination, via the next hop)
      - Use `INF` to represent infinite values

- Use `-` where no link is present
- Include all routers **except the current router** in the rows/columns in the table, even if no direct link is present.
- Rows/columns should be **ordered by router name**.
- Your rows/columns don't have to align, and the amount of white-space you use is your choice, but the easier it is to read the easier your debugging/testing will be.

5. A blank line to separate each table.

When running the algorithm to converge the routing tables, this table should be printed for every router (in alphabetical order, by router name), at each step

2. The converged routing information:
   `router X: Y is n routing through Z`
   where
   1. The name of the source router/node
   2. The name of the destination router/node
   3. The name of an immediate neighbour of the source
   4. The current total distance from the source to the destination routing via the next hop
   5. If a destination is unreachable from the source router/node, your output should look like `router A: B is unreachable`

   This output should be printed for every router, and every destination (in alphabetical order, by router name, then destination name), each time the routers have reached convergence.

Below is an example of what this output should look like for the provided topology and inputs (shortened, full output **HERE** ⬇ **(https://myuni.adelaide.edu.au/courses/64831/files/8785739/download?download_frd=1)**).

```
router X at t=0
  Y   Z
Y 2   INF
Z INF 7

router Y at t=0
  X   Z
X 2   INF
Z INF 1

router Z at t=0
  X   Y
X 7   INF
Y INF 1

...

router X at t=2
  Y   Z
Y 2   8
Z 3   7

router Y at t=2
  X   Z
X 2   4
Z 5   1

router Z at t=2
  X   Y
X 7   3
Y 9   1

router X: Y is 2 routing through Y
```

```
router X: Z is 3 routing through Y
router Y: X is 2 routing through X
router Y: Z is 1 routing through Z
router Z: X is 3 routing through Y
router Z: Y is 1 routing through Y

router X at t=3
  Y   Z
Y 2   6
Z 3   5

router Y at t=3
  X   Z
X 2   -
Z 5   -

router Z at t=3
  X   Y
X 5   -
Y 7   -

...

router X: Y is 2 routing through Y
router X: Z is 5 routing through Z
router Y: X is 2 routing through X
router Y: Z is 7 routing through X
router Z: X is 5 routing through X
router Z: Y is 7 routing through X
```

Recommended Approach

1. Start by ensuring you're familiar with the DV algorithm.
   - Review the course notes, section 5.2.2 of Kurose & Ross (7th Ed.), and the **Wikipedia entry (https://en.wikipedia.org/wiki/Distance-vector_routing_protocol)**.
   - Be sure to add logbook entries as you go.
2. Manually determine the expected distance and routing tables at each step for the sample topology
   - Feel free to ask questions and check your tables with your peers on Piazza.
   - Be sure to add logbook entries as you go.
3. Plan your implementation
   - Determine what data structures you'll need, choose a programming language, plan how you're going to parse the input and generate output, plan your algorithm's implementation.
   - Be sure to add logbook entries as you go.
4. Implementation
   - Develop your implementation, testing as you go.
   - Write a makefile if required.
   - Be sure to add logbook entries as you go.
5. Testing
   - Ensure your code runs on the university systems.
   - Develop additional scenarios and topologies to ensure your systems function as expected.

- Be sure to add logbook entries as you go.

Submission, Assessment & Marking

Submission

Create and checkout a new folder in your SVN repository at `https://version-control.adelaide.edu.au/svn/aXXXXXXX/yyyy/s1/cna/routing`

Assessment

The assignment is marked out of 200. These 200 marks are allocated as follows using automated testing:

- Acceptance Testing (available to each submission before deadline; see below for details)
  - DistanceVector correctly calculates the routing table for sample configuration: **30 marks**
  - DistanceVector correctly calculates the routing table after the link weights are changed: **20 marks**
- Full Testing (***applied only after the deadline***; see below for details) (**30 marks**)
  - PoisonedReverse correctly calculates the routing table for sample configuration
  - PoisonedReverse calculates the routing table after the link weights are changed
- Both programs, *DistanceVector* and *PoisonedReverse,* correctly calculate tables for arbitrary unseen networks: **90 marks for DV and 30 marks for DV with PR (total of 120 marks)**
- **All marks above are from passing Web Submission tests**. No additional marks are given after a manual review.

However, your marks are **scaled and reviewed** based on the following:

1. Up to 10 marks may be deducted for poor code quality. Below is a code quality checklist to help (notably this is not an exhaustive list but describes our expectations):
   - write comments above the header of each of your methods, describing
   - what the method is doing, what are its inputs and expected outputs
   - describe in the comments any special cases
   - create modular code, following cohesion and coupling principles
   - don't use magic numbers
   - don't misspell your comments
   - don't use incomprehensible variable names
   - don't have long methods (not more than 80 lines)
   - don't have TODO blocks remaining

2. As noted earlier, up to 200 marks (all marks) may be deducted for poor/insufficient/missing evidence of development process.

The two above will be assessed manually. To obtain all of the marks allocated from tests, *you will need to ensure your code is of sufficient quality and document your development process using the logbook entries*.

Marking Process

You should not be using Web Submission for debugging. As part of your design phase, you should work out what sequence of updates you expect to happen and what you expect the final distance tables will be. As such your submission will be marked in **3 stages**:

1. All submissions before the deadline will be run against an acceptance testing script.
    - This script will compile your programs, and run your DistanceVector code for the sample config.
    - Use this as a sanity check to ensure your code compiles and runs on the WebSubmission system and works as expected.
    - Be sure to add a logbook entry for each submission you make here.
2. After the deadline **your most recent submission** will be run against the full testing script.
    - This script will run the tests from the acceptance testing script as well as a number of additional tests against both your DistanceVector and PoisonedReverse code.
    - It's important that you've thoroughly tested your implementation to ensure it will be able to pass these, as you will only get 1 chance at them.
3. You code will then be reviewed for quality and evidence of your development process by a marker. Marks will be deducted if your code and/or development process are not of a reasonable standard.
   Note: This is not a review of code functionality, and you will not receive extra testing marks for it