Practical 3 (COVID-19 Modified Assessment): Routing Algorithm Implementation Assignment

Due Jun 5 by 17:00 **Points** 200

Before you begin

Although this practical is marked automatically, all marks will be moderated by a marker using the following information. The practical is marked out of 200 marks.

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 plagiarisms 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 **will 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 and 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 an implementation of the Distance Vector routing protocol. You will be required to implement the algorithm in its basic form, and then with **poisoned reverse** to improve the performance of the protocol. You will be required to demonstrate that your programs can correctly determine the routing paths for a network.

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.

Algorithm

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
```

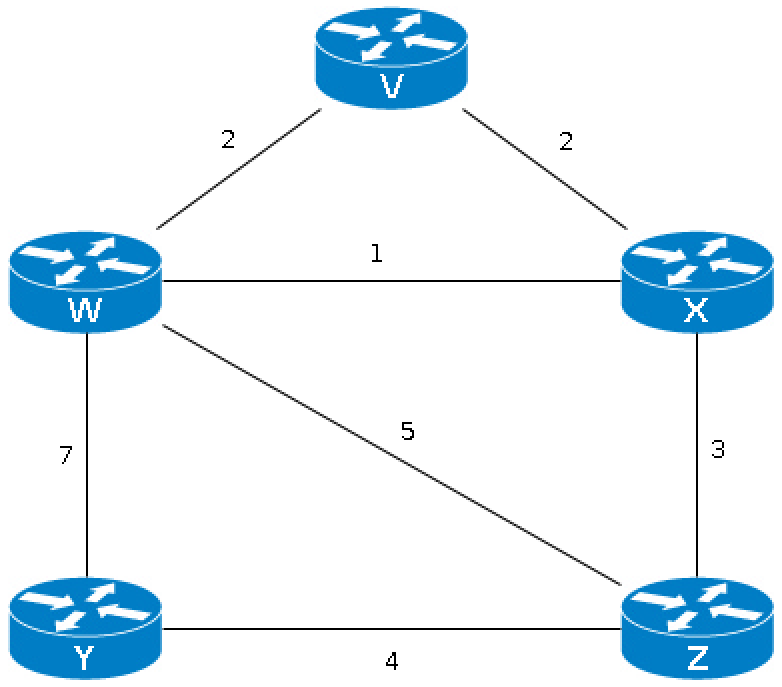
```
      send distance vector D_x = [D_x(y): y in N] to all neighbours.

FOREVER

Note: Infinity is a number sufficiently large that no legal cost is greater than or equal to infinity.
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.

Sample Topology

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



Your Tasks

Part 1 (DV algorithm): You are to produce a program that: i) reads network configuration files specified as a command-line argument; ii) uses the distance vector algorithm to calculate the routing tables whilst outputting the updated distances at each step. In part 1 your distance vector implementation is simple and **will not use** poison reverse.

Part 2 (DV with PR algorithm): The second part of the assignment is to write a second version of the program that is the same as the first, except that this program employs poisoned reverse.

The two programs you are to provide should be named: 1) `DistanceVector` and 2) `PoisonedReverse`.

Both programs act in the same way, except that one uses Poisoned Reverse. Both must perform the following actions:

- Take **two command line input arguments: 1)** a file name representing the current configuration (*`configFileName`*); and **2)** a file name representing the changed configuration (*`changedConfigName`*). The second parameter is a configuration file containing changes to link weights. Link weights are bidirectional.
- Handle bad or missing arguments.
  - Print string `Bad or missing arguments`
  - Terminate the program
- Use DV/ DV with PR, as appropriate, to calculate the routing tables for the configuration in the first parameter (*configFileName)*.
  - Output the updated distances at each step.
  - Output the routing tables for each router for the configuration in the first parameter (*configFileName)*
- Starting from the routing tables set from processing the first parameter (*configFileName*), the links in the network will be set to those of the second configuration file (*changedConfigName*). The DV/ DV with PR algorithm should then be run until such a time as the network converges. Again you are required to
  - Output the updated distances at each step.
  - Output the routing tables for each router for the configuration in the first parameter (*configFileName)*

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 table. 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.

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, if they have an update to send.
- The next set of updates will only be sent once all routers have received and processed the updates from the previous round.
- Where a router receives a number of updates, the router will process the updates in alphabetical order by router name (of the router that sent the update).
- When choosing between paths of equal cost, the router will use the path it learned of first (in other words, keep the known path).
- When a direct link is updated 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.
- Use `INF` to represent infinity values in your output.

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.)

Software/Language

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`
  - 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
- 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 format:

  ```
  make
  ./DistanceVector configFileName changedConfigName
  ./PoisonedReverse configFileName changedConfigName
  ```

  or for java:

  ```
  make
  java DistanceVector configFileName changedConfigName
  java PoisonedReverse configFileName changedConfigName
  ```

Input: Configuration File Format

As seen above your program will be be run with ***two filenames as command line arguments***.

The first file describes the topology and consists of 4 sections:

1. The first line is a number, ***x***, that indicates the number of routers/nodes in the topology.
   - e.g.

     ```
     3
     ```

2. The following ***x*** lines contain the names of each router/node in the topology.
   - e.g.

     ```
     X
     Y
     Z
     ```

3. The the next line is a number, ***y***, that indicates the number of links/edges in the topology.
   - e.g.

     ```
     2
     ```

4. The following ***y*** lines contain 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.

```
X Y 10
Y Z 8
```

The second file describes the the changes to the link/edge weights and consists of 2 sections:

1. The first next line is a number, ***y***, that indicates the number changed links/edges in the topology.
   - e.g.

   ```
   1
   ```

2. The following ***y*** lines contain the the new weight of each changed link/edge in the topology.
   - Written the same as in the first file.
   - Only contains changed links/edges. Where a link is omitted, there is no change to its weight.
   - No new links will be added in this section.
   - e.g.

   ```
   Y Z 6
   ```

You are provided a **sample config file** that matches the sample topology above. You are also provided a **sample change config file**.

Note that there is a number to give the number of changed links and then the link weight change(s).

Output Format

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

Your output should consist of 4 sections:

1. The first section
   1. Begins with the line #START
   2. Is followed by lines of the format `t=n distance from A to B via C is d` , describing the routing updates at each router at during each step towards convergence, where

   - `n` is the number the current step.
     - A value of `0` is the initial state as specified in the input file.
     - A value of `1` is the updated state after all routers have sent their first update to their neighbours.
     - etc.
   - `A` is the name of the source router/node
   - `B` is the name of the destination router/node
   - `C` is name of an immediate neighbour of `A`
     - For directly connected nodes, `C` will have the same value as `B`
   - `d` is a the current total distance from `A` to `B` routing via `C`
2. The 2nd section
   1. Begins with the line #INITIAL
   2. Is followed by lines of the format `router A: B is d routing through C` , describing the current routing table of each router/node in the system, where
      - `A` is the name of the source router/node
      - `B` is the name of the destination router/node
      - `C` is name of an immediate neighbour of `A`
      - `d` is a the current total distance from `A` to `B` routing via `C`
      - Only reachable/known routers should be shown here.
3. The 3rd section
   1. Begins with the line `#UPDATE`
   2. Is followed by lines of the same format as described in section 1 as the changed weights are applied.
4. The 4th section
   1. Begins with the line `#FINAL`
   2. Is followed by lines of the same format as described in section 2, but after the system has reached convergence with the changed weights.

Below is an example of what this output should look like (shortened).

```
#START

t=0 distance from V to W via W is 2
t=0 distance from V to X via X is 2
```

```
...
t=2 distance from V to Z via W is 10
t=2 distance from V to Z via X is 4
...

#INITIAL

router V: X is 1 routing through X
router V: W is 1 routing through X
router V: Y is 2 routing through X
router V: Z is 4 routing through X
...
router Z: V is 5 routing through X
router Z: W is 2 routing through W
router Z: X is 3 routing through Y
router Z: Y is 2 routing through Y

#UPDATE

t=0 distance from V to W via W is 12
t=0 distance from U to X via X is 11
...

#FINAL

router V: W is 3 routing through X
...
router Z: Y is 2 routing through Y
```

Example

The following files are the expected inputs/outputs for the basic example shown in the **Lecture Slides**

- **config file**
- **changed config file**
- **output**

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 files 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

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: **45 marks**
  - DistanceVector correctly calculates the routing table after the link weights are changed: **25 marks**
- Full Testing (***applied only after the deadline and grace period***; see below for details)
  - PoisonedReverse correctly calculates the routing table for sample configuration: **15 marks**
  - PoisonedReverse calculates the routing table after the link weights are changed: **15 marks**
- Both programs *DistanceVector* and *PoisonedReverse* and correctly calculate tables for arbitrary unseen networks: **70 marks for DV and 30 marks for DV with PR (total of 100 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 config0.
    - Use this as a sanity check to ensure your code runs on the WebSubmission system and works as expected for config0.
    - Be sure to add a logbook entry for each submission you make here.
2. After the deadline **your most recent submission (before the deadline+grace period)** 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.