

Programming Assignment 3: Routing Algorithm Assignment (Must Use GitHub Commit Comments as a blog) (DV) (2025)

- Due 10 Jun by 17:00
- Points 150
- Submitting an external tool
- Available until 3 Jul at 23:59

Assessment

Weighting:	15%
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	<p>Do</p> <ul style="list-style-type: none"> ✓ Discuss/compare high level approaches ✓ Discuss/compare program output/errors ✓ Regularly commit your work and write in the logbook <p>Be careful</p> <ul style="list-style-type: none"> ⚠ 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. <p>Do NOT</p> <ul style="list-style-type: none"> ✗ Submit code not solely authored by you. ✗ Post/share code on Piazza/online/etc. ✗ Give/show your code to others

Before you begin

Documenting your code (assessed)

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

You must add a comment explaining the changes you've made to **each** of your repository commits. In other words, each commit message should explain or blog what was changed in the code and why. Examples can include.

- it may be a bug fix or a logical error fix to your routing algorithm after testing.
- it may be that you recognized a mistake after submission and testing, say what it was and what you fixed.
- it can include a test case you tried and passed
- see detailed guide [here \(https://myuni.adelaide.edu.au/courses/101156/pages/commit-logs-slash-messages\)](https://myuni.adelaide.edu.au/courses/101156/pages/commit-logs-slash-messages): <https://myuni.adelaide.edu.au/courses/101156/pages/commit-logs-slash-messages>

Example

28 May 2024 18:07:06	<p>I am going to write this in C++ as I am most familiar with it.</p> <p>I may be naïve, but this seems relatively easy to implement, provided I can plan it correctly from the outset. I don't need to sanitize user inputs as stated in Piazza.</p> <p>I can think of the routers as nodes and the weights between them as edges or links.</p> <p>...</p> <p>So the main function would look something like:</p> <p>...</p>
06 Jun 2024 20:27:03	<p>Need to modify Dijkstra and make it select the node with least traversals and the shortest length. LSDB table needs to be modified to include entries for routers which are not reachable.</p>
06 Jun 2024 20:30:07	<p>Output formatting is wrong. There seems to be parsing errors and extra spaces printed in the output. Printing of new lines and empty lines where necessary are now fixed, output is now consistent among different router table outputs</p>

We should be able to review your development process and see the changes made to code with each version you have committed. **Please keep in mind that we will have access your repository and can check the number of lines of code as well as actual code changes made with each commit. We will be using GitHub classrooms so we can visit your GitHub repositories and review your work.**

During manual marking and plagiarism checks, we will look at your development process. If we do not see a clear path to a solution (i.e. code changes and regular commit entries **reflecting** your learning to develop your implementation you may forfeit up to 150 marks. *An example case of forfeiting all 150 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 commit entries.

Start Early (the task requires time to think and test)

This practical requires thought and planning. You need to start early to allow yourself time to think of what and how to test before writing any code. Failing to do this is likely to make the practical take far more time than it should.

Starting the task in the final week or days is likely to be an unsuccessful strategy with this assignment; further your development process 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).

Setting Up

We will use Visual Studio Code IDE for development & mark your work using the Gradescope submission system.



What is Visual Studio Code?

Visual Studio Code is an advanced text and code editor from Microsoft. It runs across the most common platforms you'll use, which means that whether you're Windows, Mac, or Linux, you can use these tools.

Gradescope

You will be submitting to Gradescope via UAdelaide organisation on GitHub.com.

Guide

1. We assume you should have a GitHub account since you have done the previous assignments.
2. **If you do not have VSCode:** Install VSCode, and the VSCode Remote Development extensions
3. We assume you have setup Git in VSCode. "To use Git and GitHub in VS Code, first make sure you have Git installed on your computer. If Git is missing, the **Source Control** view shows instructions on how to install it. Make sure to restart VS Code afterwards." (copied from VS Code help)
4. Connect to GitHub Classroom using this link
<https://classroom.github.com/a/RVjydhIY> 
https://classroom.github.com/a/RVjydhIY_ 
<https://classroom.github.com/a/r8scNqpb>
 - i. When prompted, log in with your GitHub credentials (username and password)

- ii. [Select your student ID from the list](#). Contact your course coordinator if you can't find your student ID. **Please don't pick someone else's as a joke** - we have to manually reset your connection through a painful process. We don't have an easy way to connect your GitHub to your identity without this link and getting this right is very important. Let us know via [private Piazza message](#) to instructors, if you accidentally pick the wrong one. Mistakes happen.



Join the classroom:
UAdelaide-CNA-2025

To join the GitHub Classroom for this course, please select yourself from the list below to associate your GitHub account with your school's identifier (i.e., your name, ID, or email).

Can't find your name? [Skip to the next step](#) →

Identifiers	
a1631364	>
a1645500	>
a1661736	>
a1671575	>
a1680241	>
a1687565	>
a1694675	>
a1695034	>
a1704579	>
a1706557	>
a1714327	>
a1714717	>

- iii. Accept the assignment by clicking the button.

UAdelaide-CNA-2025

Accept the assignment —
**CNA 2025 Routing Programming
Assignment 3**

Your user
name will
be here

Once you accept this assignment, you will be granted access to the `cna-2025-routing-programming-assignment-3` repository in the [UAdelaide](#) organization on GitHub.

Accept this assignment

- iv. This will create a new repository for your work; copy the link to this new repository (see red arrow).



You're ready to go!

You accepted the assignment, **CNA 2025 Routing Programming Assignment 3**.

Your assignment repository has been created:

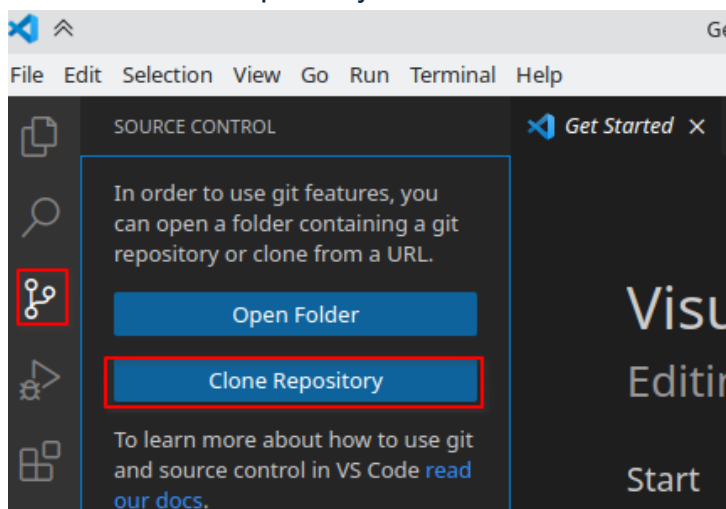
<https://github.com/UAdelaide/cna-2025-routing-programming-assignment-3>

We've configured the repository associated with this assignment.

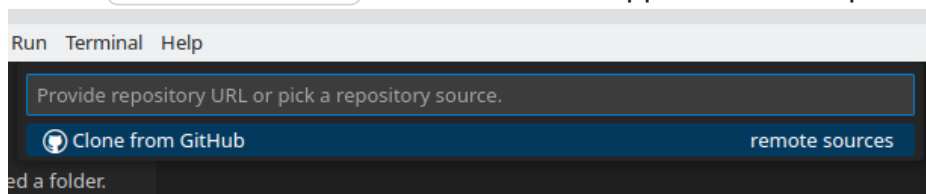


5. Load the repository in VSCode

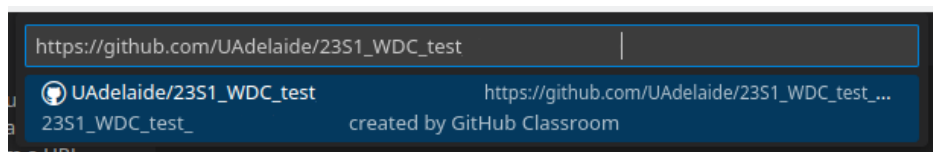
- i. Open a new VSCode window.
- ii. Select "Source Control" from the menu on the left
- iii. Choose Clone Repository



- iv. Select **Clone from Github** in the menu that appears at the top of the screen.




- v. When prompted, Log in with your GitHub credentials
- vi. Paste the link from step 3 iv) above in the menu that appears at the top of the screen and select the matching repository.



- vii. When prompted, choose a location on your computer for the repository.
- viii. When prompted, open the cloned repository.

Congratulations; your now ready to go!

- You're set up! **Cool**  (<https://slate.com/human-interest/2013/10/cool-the-etymology-and-history-of-the-concept-of-coolness.html>). Let's get to the next stage.

- Please use Piazza for all Q&A and not personal emails please. Please tag it to the folder "**routing_assignment**"
- As a third year CS course there are no formal help or programming sessions. The programming assignments are done in your own time.
- But, we are able to provide help when you need it, in a targeted way, to support you in small groups. To request help please kindly fill this form:
[https://forms.office.com/Pages/ResponsePage.aspx?
id=QN_Ns1SWJkqGoXecUfacSGw41kwG_UpEgCXc8uFywohUMEYxRkZFOU9GOUs5QkJLUj](https://forms.office.com/Pages/ResponsePage.aspx?id=QN_Ns1SWJkqGoXecUfacSGw41kwG_UpEgCXc8uFywohUMEYxRkZFOU9GOUs5QkJLUj)
- We will get notified of your requests and then based on help sought organise small group sessions. You TA, Kai will support you in these sessions.

- Please see [Assessment Overview](https://myuni.adelaide.edu.au/courses/101156/pages/assessment-overview) (<https://myuni.adelaide.edu.au/courses/101156/pages/assessment-overview>) and follow the instructions there in.
- **In this assignment, you are given a 3 day grace period past your deadline to submit your work with no questions asked and no penalty.** So before you apply, please consider your case for wanting more than 3 days. After three days past the deadline, you will get a zero.
- If you email the course coordinator or me about extensions, we will simply direct you to the **Assessment Overview** page with the instruction to follow. Doing this ensures that you get timely responses and your case is handled without one of us (Cheryl or myself) being the bottleneck.

- Learn about routing protocols and route propagation.
- Implementing and testing a routing protocol.

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 Distance Vector (DV) algorithm (**Part 1**) in its basic form. *If you select to do the bonus part*, then with poisoned reverse (PR) (Part 2) 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

- **Do Part 1** (150 marks)
- **Do Part 2 if you have time** (Bonus 40 marks in addition to 150 marks)

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.
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 (BONUS) (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++, Rust, 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/>**  **[\(https://makefiletutorial.com/\)](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.adelaide.edu.au** server and include:
 - **C/C++:** g++ (GCC) 4.8.5
 - **Java:** java version "1.8.0_201"
 - **Python:** python 3.6.8
 - **Rust:** rustc 1.62.1
- Your implementation may use any libraries/classes available on the Grade Scope system (same as those on the **uss.cs.adelaide.edu.au** server), but **no external libraries/classes/modules**.
- Your programs will be executed by Gradescope with the commands below:
 - For C/C++ and Rust

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

You can find a **simple** example makefile for C++ **[HERE](#)**



<https://myuni.adelaide.edu.au/courses/101156/files/16613112/download?wrap=1>  **https://myuni.adelaide.edu.au/courses/101156/files/16613112/download?download_frd=1** . **A good resource is here: <https://makefiletutorial.com/>**  **[\(https://makefiletutorial.com/\)](https://makefiletutorial.com/)**

This will **need to be customised** for your implementation. Make sure you use tabs (**actual tab characters**) on the indented parts. If you are using Rust, replace "cc" with "rustc".

- For java:

```
make
java DistanceVector
java PoisonedReverse
```

You can find a **simple** example makefile for Java **[HERE](#)**

<https://myuni.adelaide.edu.au/courses/101156/files/16613236/download?wrap=1>  **https://myuni.adelaide.edu.au/courses/101156/files/16613236/download?download_frd=1** . **A good resource is here: <https://makefiletutorial.com/>**  **[\(https://makefiletutorial.com/\)](https://makefiletutorial.com/)**

This will **need to be customised** for your implementation. Make sure you use

tabs (**actual tab characters**) on the indented parts

(<https://myuni.adelaide.edu.au/courses/101156/files/16613236/download?wrap=1>)

- For Python (please note the lack of a file extension, i.e. `.py` at the end for file names):

```
./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.cs.adelaide.edu.au** 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 python3` (Python 3).

Algorithm

Distance Vector (DV)

At each node, :

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

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.

Key Assumptions

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

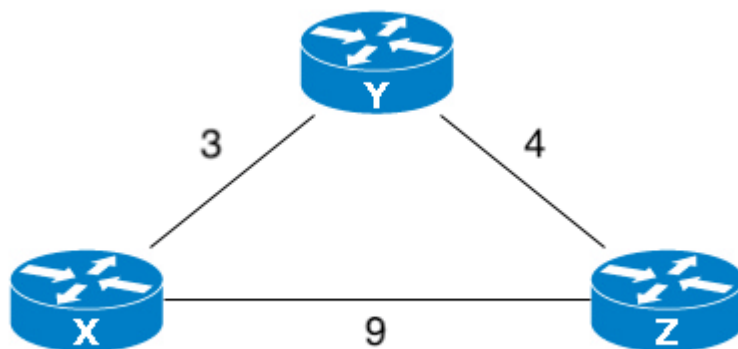
In this environment, to simplify your programs, you should assume:

- 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).

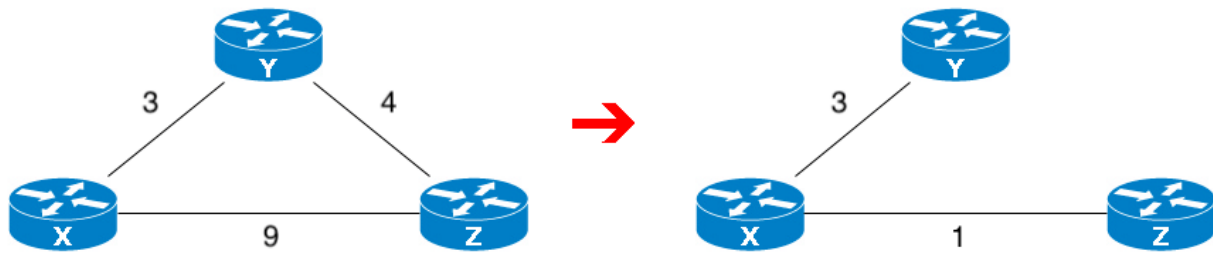
You should confirm for yourself that the assumptions above will not change the least-cost path routing table that will be produced at the nodes. (Although, for some topologies, you may take different paths for the same cost.)

Sample Topology

Consider the following network sample topology in our description of the required input format:



Your program will need to read input from the standard input (terminal/command line) to construct such a given topology. Then, perform the updates, as illustrated below, and also given via the standard input (terminal/command line).



Please refer to this sample topology with the following updates above when looking into the expected input examples below.

Input Format

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

The expected input format ([input file](#)

(<https://myuni.adelaide.edu.au/courses/101156/files/17364613?wrap=1>)_ ↓

(https://myuni.adelaide.edu.au/courses/101156/files/17364613/download?download_frd=1)) is as follows:

```
X
Y
Z
START
X Z 9
X Y 3
Y Z 4
UPDATE
X Z 1
Z Y -1
END
```

1. The input begins with **the name of each router/node in the topology**.
 - Each name is on a new line.
 - Router names can consist of any characters (except numbers) and are case-sensitive.
 - Router names may not contain spaces.
 - This section ends with the keyword "START".
2. The next input section 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.
 - Format: "<router_name_1> <router_name_2> <weight>", 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.
- This section ends with the keyword "UPDATE".
- Your algorithm **should run when the keyword "UPDATE" is entered** and bring your simulated routers' routing tables and distance tables to convergence. Then, show the **Expected Output** (refer to the section below) for each router.

3. The input continues with the update details of each link/edge in the topology given a link and cost.

- The values in each of these sections should be used to update the current 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.
- As an example, if an unseen new router/node name has been input in this section, your program should be able to **add** this new router to the topology.

```
■ Y A 10
```

- From the example input given above, your program should add **A** as a new router into the topology where it has a link with a cost of 10 to **Y**.
- A user may input 0 or more lines in the "UPDATE" section. This section ends with the keyword "END".
 - If there is input in the "UPDATE" section, your algorithm **should be run when the keyword "END" is entered** and bring your simulated routers' routing tables and distance tables to convergence. Then, show the **Expected Output** for each router. After that, the program exits normally.
 - If no input is provided in the "UPDATE" section, the program exits normally when the keyword "END" is entered.

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. When running your algorithm to bring convergence to the routing tables, print the **distance table** of each router at each step in the following format:

```
◦ Distance Table of router X at t=0:
  Y  Z
Y 2  INF
Z INF 7
```

- The name of the router, and the current step (starting at 0).
- The name of the destination router.
- The name of the next hop router.
- The current known distance (from the current router, to the destination, via the next hop).
 - Use **INF** to represent infinite values and 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 in alphabetical order**.
- 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.
- A blank line to separate each table.
- **Note:** 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. Once the routing tables are converged, print the **routing table** of each router in the following format:

- **Routing Table of router X:**
Y,Y,2
Z,Y,5

- The name of the source router/node.
- The name of the destination router/node.
- The name of an immediate neighbour of the source.
- The current total distance from the source router/node to the destination router/node route via the next hop.
- If a destination is unreachable from the source router/node, your output should look like **Y, INF, INF**.
- The destination needs to be **printed in alphabetical order**.
- A blank line to separate each table.
- **Note:** This table 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. Where multiple best routes exist for a destination, the first next hop is used (in alphabetical order, by router name)**.

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/101156/files/17272769/download)

(<https://myuni.adelaide.edu.au/courses/101156/files/17272769/download>) 

(https://myuni.adelaide.edu.au/courses/101156/files/17272769/download?download_frd=1)).

```
Distance Table of router X at t=0:
  Y   Z
Y   3  INF
Z  INF  9
```

Distance Table of router Y at t=0:

	X	Z
X	3	INF
Z	INF	4

Distance Table of router Z at t=0:

	X	Y
X	9	INF
Y	INF	4

Distance Table of router X at t=1:

	Y	Z
Y	3	13
Z	7	9

Distance Table of router Y at t=1:

	X	Z
X	3	13
Z	12	4

Distance Table of router Z at t=1:

	X	Y
X	9	7
Y	12	4

.....

Routing Table of router X:

Y,Y,3
 Z,Y,7

Routing Table of router Y:

X,X,3
 Z,Z,4

Routing Table of router Z:

X,Y,7
 Y,Y,4

Distance Table of router X at t=3:

	Y	Z
Y	3	5
Z	7	1

Distance Table of router Y at t=3:

	X	Z
X	3	INF
Z	10	INF

Distance Table of router Z at t=3:

	X	Y
X	1	INF
Y	4	INF

.....

Routing Table of router X:

Y,Y,3
 Z,Z,1


Routing Table of router Y:

X,X,3
 Z,X,4

Routing Table of router Z:

X,X,1

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)  (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*.
 - Remember, you must document as you go (detailed commit messages).
5. Testing
 - Ensure your code runs on the submission system and not just your personal machine (we cannot mark any code that is not submitted).
 - Use input/output redirection - read the input from a file and redirect the output into a file for easy testing and diff'ing (see: <https://www.geeksforgeeks.org/input-output-redirection-in-linux/> (<https://www.geeksforgeeks.org/input-output-redirection-in-linux/>)).
 - Develop additional scenarios and topologies to ensure your systems function as expected.
 - *I cannot stress enough the importance of creating routing configurations and route update scenarios and testing your code thoroughly. Coming up with good test cases is an exercise that will test and improve your understanding of routing and DV routing.* Already: we have given you: 1) a detailed topology and topology update (acceptance test), 2) another, in your lecture/textbook discussion of DV, 3) in your workshops and, 4) in the Wiki article.
 - Be sure to document as you test.

Submission, Assessment & Marking

Submission

Your work will need to be submitted to **Gradescope** through the submission portal at the bottom of this assignment page.

- Make sure you are submitting via GitHub (1) and (2) select your repo, try typing your username.

Submit Programming Assignment

Upload all files for your submission

Submission Method

☐ Upload
 ☒ **GitHub**
☐ Bitbucket

Repository *

Select a repository...

Branch *

Select a branch...

Student Name (Optional)

Enter student name

Cancel Upload

- Gradescope will perform a static analysis of your code to perform some sanity checks.
- Gradescope will then run Acceptance Testing (See **Assessment** and **Marking Process** tab below).
- Full testing and marking of submissions will be done after the deadline.

Assessment

The assignment is marked out of 150. These 150 marks are allocated as follows using automated testing:

- Acceptance Testing (available to each submission before deadline; see below for details) **(70 marks)**
 - DistanceVector correctly calculates the routing table for sample configuration: 40 marks
 - DistanceVector correctly calculates the routing table after the link weights are changed: 30 marks
- Full Testing (***applied only after the deadline***; see Marking Process for details) **(80 marks)**
 - *DistanceVector* correctly calculates tables for arbitrary unseen networks

Part 2 (Optional: *Bonus*)

- Testing after submission deadline (40 marks)
 - PoisonedReverse correctly calculates the routing table for sample configuration
 - PoisonedReverse calculates the routing table after the link weights are changed
 - *PoisonedReverse*, correctly calculate tables for arbitrary unseen networks.

All marks above are from passing test cases. 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, all marks may be deducted for poor/insufficient/missing evidence of development process.

The two analyses 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 the autograder 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. This is an exercise we expect you would undertake as part of this assignment to gain a better appreciation for routing algorithms. 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 run your Makefile and run your DistanceVector code for the sample config.
 - Use this as a sanity check to ensure your code compiles and runs on the test environment 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 a full testing script.
 - This script will run the tests from the acceptance testing script as well as additional tests against your DistanceVector code. And PoisonedReverse code, *if you did the optional bonus Part 2.*
3. Your code will then be reviewed for quality and evidence of your development process by a marker. We will visit your linked GitHub repositories and review your work. Marks

will be deducted if your code and/or development process are not of a reasonable standard. Please **note**: This is not a review of code functionality, and you will not receive extra testing marks for it.

4. We are sorry, as in all other programming assignments, code that does not compile/run or crash while testing on the Linux image (same as CATS machines, same as the image used by Gradescope and  cloud mentioned in *Programming Language/Software Requirements* tab) will get an automatic zero. The marking team will not fix error to get your code to run and mark your work. Testing and submitting working code is entirely your responsibility.

This tool needs to be loaded in a new browser window

Load Programming Assignment 3: Routing Algorithm Assignment (Must Use GitHub Commit Comments as a blog) (DV) (2025) in a new window