Log book for user: a1799298 for assignment: 2024/s1/cna/routing

Log book entries - oldest entries first

Reload Page Read Only | Reload Page Read Write

Datestamp	Entry
22 May 2024 12:54:06 Kind: Text, by: a1799298, from: 118_211_67_70	Initialised logbook.
22 May 2024 13:11:18 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r97 | a1799298 | 2024-05-22 13:11:18 +0930 (Wed, 22 May 2024) | 1 line Changed paths: A /2024/s1/cna/routing routing
22 May 2024 13:23:52 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 1 (1:15pm, May 22, 2024): I started off by reading the assignment spec and logbook requirements. Since the assignment is quite rigorous, I'm going to start off by breaking everything down into digestable chunks. - Logbook entries: Seem straightforward. I tend to be quite forgetful, so I'm sticking a sticky note on my monitor to remind myself to WRITE ENTRIES ALL THE TIME. - Undergrad / Postgrad Reqs: Was a bit confused about whether or not the report / stress tests were included for undergrads, but a quick check of Piazza answered this. - Language choice: Going to pick python as I have the most experience with it. C++ could be another option, but I prefer the simplicity of Python. - Routing familiarity: It's been a few weeks since taking my notes on Routing / Djikstra, so I'm going to review those and go through the examples first. What's next? Working through the routing examples in the specified format BY HAND, uploading my working as logbook entires with annotated notes.
22 May 2024 14:28:33 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 2 (2:27pm, May 22, 2024) Since I wasn't quite grasping the explanation of the inputs and outputs from the assignment spec (im a very visual learner), I decided I should try to work through some examples. For each of the steps, I'll outline the expected output and draw the network topology. Here is one of the examples I worked through:

1. Input:
```
A
B
C
LINKSTATE
A-B 2
B-C 3 B
UPDATE
A-C 1 A,C
END
```

2. Explanation:
- The first three lines define the routers in the topology: A, B, and C.
- The "LINKSTATE" section starts, and the first link is defined:
  - `A-B 2`: There is a link between router A and router B with a cost of 2.
- The second link is defined:
  - `B-C 3 B`: There is a link between router B and router C with a cost of 3. The optional list "B" specifies that the Expected Output should be shown for router B after processing this link.
- The "UPDATE" section starts, and an update is provided:
  - `A-C 1 A,C`: A new link is added between router A and router C with a cost of 1. The optional list "A,C" specifies that the Expected Output should be shown for routers A and C after processing this update.
- The "END" keyword signals the end of the input.

3. Expected Output:
```
B Neighbour Table:
A|2
C|3

B LSDB:
A|B|2
B|C|3

B Routing Table:
A|A|2
C|C|3

A Neighbour Table:
B|2
C|1

A LSDB:
A|B|2
A|C|1
B|C|3

A Routing Table:
B|B|2
C|C|1

C Neighbour Table:
A|1
B|3

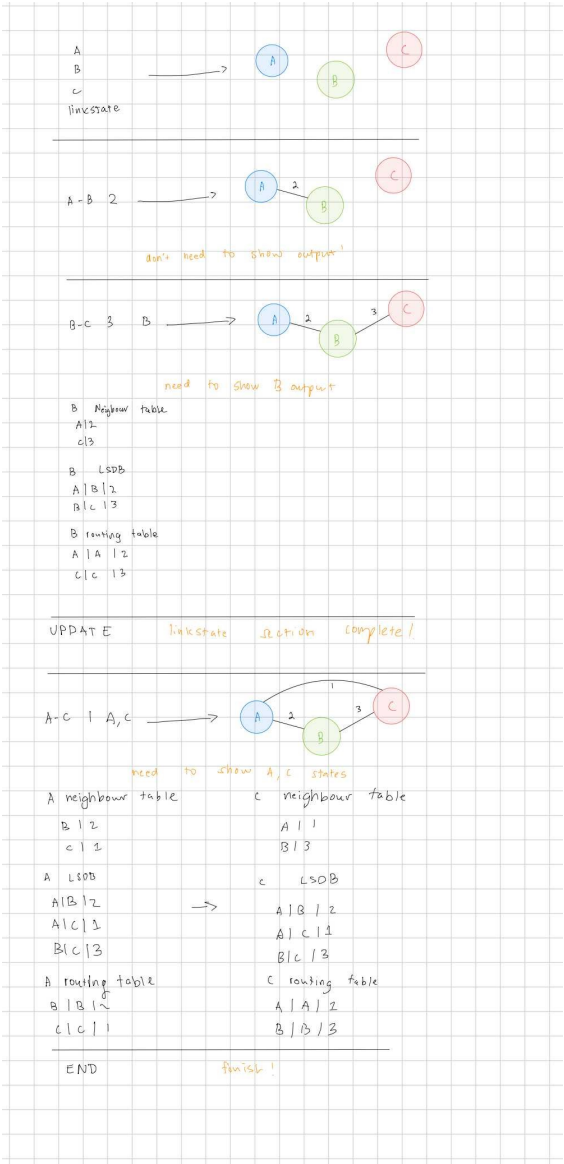
C LSDB:
A|B|2
```

A|C|1
B|C|3

C Routing Table:
A|A|1
B|B|3
```

My drawings of the network topology have been included in the next entry!

22 May 2024 14:31:27
Kind: JPEG Image, by: a1799298, from: 118_211_67_70




22 May 2024 14:40:00
Kind: Text, by: a1799298, from: 118_211_67_70

Logbook Entry 3 (2:32pm, May 22, 2024)

Thoughts / Observations:
- Working through some hand-written examples really helped my understanding of what the expected requirements
```

	are. - This assignment seems perfect for an object-oriented programming approach. Next steps: - Outlining an extensive python implementation
22 May 2024 14:44:46 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 4 (2:40pm, May 22, 2024) Now that I have a pretty good understanding of what I need to do, I can actually start planning it. This class implementation won't be perfect, but it's going to benefit me greatly to have planned this before diving into it. Code Outline: 1. Node Class: - Variables: - `name`: Stores the name of the node. - `neighbors`: A dictionary that stores the neighboring nodes and their associated link costs. - `routing_table`: A dictionary that stores the routing table entries. - Functions: - Constructor, `add_neighbor()`, `remove_neighbor()`, `calculate_routing_table()`, and printing functions for neighbor table, LSDB, and routing table. 2. Network Class: - Variables: - `nodes`: A dictionary that stores all the nodes in the network, with node names as keys and Node objects as values. - `lsdb`: A dictionary that represents the Link-State Database (LSDB) of the network. - Functions: - Constructor, `add_node()`, `add_link()`, `remove_link()`, `process_input()`, and `print_output()`. Implementation Details: - The `Node` class encapsulates the properties and behaviors of individual nodes in the network. It stores the node's name, neighbors, and routing table, and provides methods to update and retrieve this information. - The `Network` class represents the entire network topology and manages the interactions between nodes. It stores all the nodes in the network and maintains the LSDB. It provides methods to add/remove nodes and links, process input commands, and generate the expected output. By separating the responsibilities into two classes, we can achieve a modular and maintainable design. The `Node` class focuses on the individual node's perspective, while the `Network` class handles the overall network operations and coordination between nodes. Next Steps: - Setting up the SVN directory, initialising initial code files with proposed functions and comments.
22 May 2024 14:52:36 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r98 | a1799298 | 2024-05-22 14:52:36 +0930 (Wed, 22 May 2024) | 1 line Changed paths: A /2024/s1/cna/routing/Node.py Initalise 'Node' class in 'Node.py'
22 May 2024 14:53:26 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r99 | a1799298 | 2024-05-22 14:53:26 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Implemented constructor for 'Node'

22 May 2024 14:57:27 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 5 (2:56pm, May 22, 2024) A small entry to explain SVN commit for revision 99: - The message states 'implemented' constructor, but it should read 'initalised'. It contains the function description ONLY, not the actual implementation!
22 May 2024 14:58:17 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r100 | a1799298 | 2024-05-22 14:58:17 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Initialise 'add_neighbour' for 'Node'. Function description added.
22 May 2024 14:58:53 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r101 | a1799298 | 2024-05-22 14:58:53 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Initialise 'remove_neighbour' for 'Node'. Function description added.
22 May 2024 14:59:30 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r102 | a1799298 | 2024-05-22 14:59:30 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Initialise 'calculate_routing_table' for 'Node'. Function description added.
22 May 2024 15:01:30 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r103 | a1799298 | 2024-05-22 15:01:30 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Initialise 'print_routing_table' for 'Node'. Function description added.
22 May 2024 15:02:18 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r104 | a1799298 | 2024-05-22 15:02:18 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Initialise 'print_lsdb' for 'Node'. Function description added.
22 May 2024 15:02:25 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 5 (2:56pm, May 22, 2024) A small entry to explain SVN commit for revision 99: - The message states 'implemented' constructor, but it should read 'initalised'. It contains the function description ONLY, not the actual implementation!
22 May 2024 15:05:43 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 6 (3:02pm, May 22, 2024) Some observations for previous commit (revision 104) - It seems a bit weird to give the `print_lsdb` functionality to the node class instead of the network, but I did it this way because Nodes exist as *variables* of networks. They have access to the instantiated lsdb regardless, and can add the node name for each respective print. - A funny observartion: I'm surprised just how much I remember from OOP. Just goes to show how much I liked that class!

22 May 2024 15:06:09 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 6 (3:02pm, May 22, 2024) Some observations for previous commit (revision 104) - It seems a bit weird to give the `print_lsdb` functionality to the node class instead of the network, but I did it this way because Nodes exist as *variables* of networks. They have access to the instantiated lsdb regardless, and can add the node name for each respective print. - A funny observartion: I'm surprised just how much I remember from OOP. Just goes to show how much I liked that class!
22 May 2024 15:06:15 Kind: Text, by: a1799298, from: 118_211_67_70	Logbook Entry 6 (3:02pm, May 22, 2024) Some observations for previous commit (revision 104) - It seems a bit weird to give the `print_lsdb` functionality to the node class instead of the network, but I did it this way because Nodes exist as *variables* of networks. They have access to the instantiated lsdb regardless, and can add the node name for each respective print. - A funny observartion: I'm surprised just how much I remember from OOP. Just goes to show how much I liked that class!
22 May 2024 15:24:45 Kind: Text, by: a1799298, from: 118_211_67_70	========== Non entry ========== Some previous entries seem to have been duplicated due to a refresh bug. I asked in Piazza about in @208. Just placing this here for the marker to know what happened!
22 May 2024 15:28:27 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r105 | a1799298 | 2024-05-22 15:28:27 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Initialise 'print_routing_table' for 'Node'. Function description added. Node initialisation complete!
22 May 2024 15:33:23 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 7 (3:28pm, May 22, 2024) ========== I've completed the initialisation of the 'Node' class and included extensive docstrings (function explanations) Here's the rundown: - The __init__ method is extensively commented to describe its purpose, parameters, and attributes. - The name parameter represents the name of the node. - The neighbors attribute is a dictionary that stores the neighboring nodes and their link costs. - The routing_table attribute is a dictionary that stores the routing table entries. - Each method has a docstring that describes its purpose and parameters (if any). - The add_neighbor method adds a neighbor node with the given link cost. - The remove_neighbor method removes a neighbor node. - The calculate_routing_table method calculates the routing table based on the current LSDB using Dijkstra's algorithm. - The print_neighbor_table method prints the neighbor table in the required format. - The print_lsdb method prints the LSDB in the required format. - The print_routing_table method prints the routing table in the required format.
22 May 2024 15:34:30 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 7 (3:28pm, May 22, 2024) ==========

	Cont. (I forgot to add it to the prior entry :/) What's Next? - Initialising the Network class
22 May 2024 15:36:09 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r106 | a1799298 | 2024-05-22 15:36:09 +0930 (Wed, 22 May 2024) | 1 line Changed paths: A /2024/s1/cna/routing/Network.py Intialise 'Network' class file
22 May 2024 15:36:50 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r107 | a1799298 | 2024-05-22 15:36:50 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Initialise constructor for 'Network'. Function description added.
22 May 2024 15:41:15 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r108 | a1799298 | 2024-05-22 15:41:15 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Initialise 'add_node' for 'Network'. Function description added.
22 May 2024 15:41:30 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r109 | a1799298 | 2024-05-22 15:41:30 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Initialise 'add_link' for 'Network'. Function description added.
22 May 2024 15:44:13 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r110 | a1799298 | 2024-05-22 15:44:13 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Initialise 'remove_link' for 'Network'. Function description added.
22 May 2024 15:44:52 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r111 | a1799298 | 2024-05-22 15:44:52 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Initialise 'process_input' for 'Network'. Function description added.
22 May 2024 15:47:29 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 8 (3:45pm, May 22, 2024) ========== Some observations for previous commit (revision 111) - I may need to segment `process_input` even further at some point. I foresee a long while loop with lots of ifs in the future. - A new function for each 'line type' seems suitable, but I want to see how I fare first.
22 May 2024 15:48:36 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r112 | a1799298 | 2024-05-22 15:48:36 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Initialise 'print_output' for 'Network'. Function description added.
22 May 2024 15:51:12 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 9 (3:48pm, May 22, 2024) ========== I've completed the initialisation of the 'Network' class and included extensive docstrings (function

explanations)

Here's the rundown:

- The `__init__` method is extensively commented to describe its purpose and attributes.
  - The `nodes` attribute is a dictionary that stores all the nodes in the network, with node names as keys and Node instances as values.
  - The `lsdb` attribute is a dictionary representing the link-state database (LSDB) of the network, where the keys are tuples of node names (representing links) and the values are the link costs.
- Each method has a docstring that describes its purpose and parameters (if any).
  - The `add_node` method adds a new node to the network.
  - The `add_link` method adds a new link between two nodes in the network.
  - The `remove_link` method removes the link between two nodes in the network.
  - The `process_input` method processes the input lines and performs the necessary actions based on the network topology and updates. This may be segmented into multiple functions in future.
  - The `print_output` method prints the expected output for the specified nodes.

What's Next?
- Setting up the main `Dijkstra` file to instantiate a Network object and a function to take input

22 May 2024 15:57:43
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r113 | a1799298 | 2024-05-22 15:57:43 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
A /2024/s1/cna/routing/Dijkstra

Intialise 'Dijkstra' main file

22 May 2024 16:06:14
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 10 (4:05pm, May 22, 2024)
==========

I've initialised all the necessary files and functions, and now it's time to start implementing everything!

I'll start by tackling the input processing first. Here's my plan:

- Implement the `read_input` function in `Dijkstra` to read input lines from the standard input (stdin) until "END" is encountered.
  - Return the list of input lines.

- Implement the `process_input` method in the `Network` class to handle different types of input lines and update the network accordingly.
  - Keep track of the current input section ("INIT", "LINKSTATE", or "UPDATE") using a variable.
  - Iterate over each input line and process it based on the current section.
  - For "INIT" lines, add new nodes to the network.
  - For "LINKSTATE" and "UPDATE" lines:
    - Extract the node names, cost, and optional list of chosen routers.
    - Add or remove links between nodes based on the cost.
    - Update the network topology and print the output for chosen routers, if any.

22 May 2024 16:09:32
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r114 | a1799298 | 2024-05-22 16:09:32 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Dijkstra

Implement read_input function to read input lines from stdin

22 May 2024 16:11:17
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 11 (4:09pm, May 22, 2024)
==========

I've implemented the `read_input` function. Seems to be working pretty well!

```
A
B
C
LINKSTATE
A-B 2
B-C 3 B
UPDATE
A-C 1 A,C
END
::input lines:: ['A', 'B', 'C', 'LINKSTATE', 'A-B 2', 'B-C 3 B', 'UPDATE', 'A-C 1 A,C', 'END']
```

Now on to implementing the `process_input` function!

22 May 2024 16:13:01
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r115 | a1799298 | 2024-05-22 16:13:01 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Dijkstra

Implement main function to run the network simulation

22 May 2024 16:16:14
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r116 | a1799298 | 2024-05-22 16:16:14 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Implement main function to run the network simulation

22 May 2024 16:18:07
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r117 | a1799298 | 2024-05-22 16:18:07 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Implemented 'process_input' function inside Network class

22 May 2024 16:21:34
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 12 (4:18pm, May 22, 2024)
==========

I have implemented the `process_input` function in the `Network` class to handle the input processing for the network simulation. Here's a summary of what the function does:

- It takes a list of input lines as a parameter.
- It iterates over each line and determines the current section based on the keywords "LINKSTATE", "UPDATE", and "END".
- For each line in the "LINKSTATE" or "UPDATE" section:
  - It splits the line into parts and extracts the node names and cost.
  - If the nodes are not already present in the network, it adds them by calling the `add_node` function.
  - If the cost is -1, it removes the link between the nodes by calling the `remove_link` function.
  - Otherwise, it adds the link between the nodes with the given cost by calling the `add_link` function.
  - If there are additional parts in the line, it splits them by comma and passes them to the `print_output` function to print the output for the specified nodes.
- For lines outside the "LINKSTATE" or "UPDATE" section, it adds the node to the network by calling the `add_node` function.

Running the our initial example:

```
A
B
```

C
LINKSTATE
A-B 2
B-C 3 B
UPDATE
A-C 1 A,C
END
add_node function called with argument: A
add_node function called with argument: B
add_node function called with argument: C
add_node function called with argument: B
add_node function called with argument: C
add_link function called with arguments: B, C, 3
add_node function called with argument: A
add_node function called with argument: C
add_link function called with arguments: A, C, 1
```

That doesn't seem right.... I think there was an issue with adding nodes as the final else case. Let's try again.

22 May 2024 16:25:15
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r118 | a1799298 | 2024-05-22 16:25:15 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Implemented 'process_input' function inside Network class

22 May 2024 16:32:38
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 13 (4:32pm, May 22, 2024)
==========
Seems I forgot a major function in our Network implementation - `update_links`. We also need to call the printing functions within the linkstate and update functions.

22 May 2024 16:38:34
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r119 | a1799298 | 2024-05-22 16:38:34 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Initialise new node function - 'update_link' inside Network

22 May 2024 16:42:08
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r120 | a1799298 | 2024-05-22 16:42:08 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Restarting 'process_input' from scratch. Taking it 1 part at a time

22 May 2024 16:43:38
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 14 (4:43pm, May 22, 2024)
==========

`process_input` isn't going that well. I decided to scrap what I had. It was too confusing to continue. Going to approach each type of line one at a time now

22 May 2024 16:48:19
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r121 | a1799298 | 2024-05-22 16:48:19 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Implemented add node part of process_input
```

22 May 2024 17:03:09
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r122 | a1799298 | 2024-05-22 17:03:09 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Implemented link state section of process input

22 May 2024 17:05:48
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r123 | a1799298 | 2024-05-22 17:05:48 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Implemented update and end sections of process input. Process input seems to be working properly now!

22 May 2024 17:13:22
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 15 (5:09pm, May 22, 2024)
==========

Process input seems to be working properly now!!

Observations:
This `process_input` function may be the hardest part of the assignment... Here are a list of issues I've found when parsing.
- We need to define an instruction type for each type of line
- If the line *is* the instruction type, then it needs to be skipped over...
- LINKSTATE and UPDATE instructions are variable, so splitting lines is not trivial
- Nodes need to be split using `-` as a delimiter
- Printed nodes need to be split using `,` as a delimiter, sorted alphabetically and stored in an array

But, it seems to be working properly now!

```
A
B
C
LINKSTATE
A-B 2
B-C 3 B,A,C
UPDATE
A-C 1 A,C
END
Added node: A
Added node: B
Added node: C
add_link function called with arguments: A, B, 2
No nodes chosen for printing output.
add_link function called with arguments: B, C, 3
Chosen routers: A, B, C
update_link function called with arguments: A, C, 1
Chosen routers: A, C
End of input.
```

I will need to revisit this function as I implement the other functions. New lines are still not working properly, but I think I can implement within the lower level functions.

What's next?
Implementing the `add_node` function inside `Network`.

22 May 2024 17:28:16 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 16 (5:27pm, May 22, 2024) ========== Reflection: I forgot to handle removal of links .... oops. Its alright, I can handle that when I get to that function. It works properly as is atm.
22 May 2024 17:33:46 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r124 | a1799298 | 2024-05-22 17:33:46 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Dijkstra M /2024/s1/cna/routing/Network.py Implemented the add_node function to add new nodes to the network.
22 May 2024 17:37:21 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 15 (5:37pm, May 22, 2024) ========== Reflection: Implemented the `add_node` function in the `Network` class. This function now allows for the addition of new nodes to the network by checking if the node already exists and adding it if not. Here's a brief overview of what the function does: - It takes the node name as a parameter. - Checks if the node already exists in the network's nodes dictionary. - If not, it creates a new `Node` instance, adds it to the dictionary, and prints a confirmation message. - If the node already exists, it prints a message indicating this. What's next? Implementing add, update and remove links. As well as print output functions. I am leaving out implementing the routing table calculations so that I can focus on that last.
22 May 2024 17:37:58 Kind: Text, by: a1799298, from: 118_211_67_70	========== Aside ========== Last entry should have been entry 17**
22 May 2024 17:40:29 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r125 | a1799298 | 2024-05-22 17:40:29 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Updated add_neighbor function to add neighboring nodes with specified link costs.
22 May 2024 17:45:38 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r126 | a1799298 | 2024-05-22 17:45:38 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Fixed LSDB updates. Now only adds single link, not both
22 May 2024 17:47:47 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r127 | a1799298 | 2024-05-22 17:47:47 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py M /2024/s1/cna/routing/Node.py Implemented 'remove_neighbor' inside Node

22 May 2024 17:48:38 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r128 | a1799298 | 2024-05-22 17:48:38 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Implemented update_link function to update the cost of an existing link between nodes by removing and re-adding neighbors with updated costs.
22 May 2024 17:50:34 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r129 | a1799298 | 2024-05-22 17:50:34 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Implemented update_link function to update the cost of an existing link or add a new link if it does not exist.
22 May 2024 17:54:00 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 18 (5:52pm, May 22, 2024) ========== Reflection: Since my last entry, I've made significant progress on implementing key functionalities in the Network class, specifically focusing on adding, updating, and managing links between nodes. 1. **add_node Function:** - Implemented the `add_node` function to add new nodes to the network. This function checks if the node already exists before adding it, ensuring no duplicates. If the node exists, it prints a message indicating so. 2. **add_neighbor Function:** - Updated the `add_neighbor` function in the `Node` class to add neighboring nodes with specified link costs. This ensures that each node maintains an accurate dictionary of its neighbors and their respective link costs. 3. **add_link Function:** - Implemented the `add_link` function in the `Network` class to create links between nodes with specified costs. This function adds the link to both nodes' neighbor dictionaries and updates the link-state database (LSDB) with the new link. 4. **remove_neighbor Function:** - Implemented the `remove_neighbor` function in the `Node` class to remove a neighbor node from the neighbors dictionary. This function checks if the neighbor exists before deleting it, ensuring the neighbors list is always up-to-date. 5. **update_link Function:** - Implemented the `update_link` function to handle updating the cost of an existing link or adding a new link if it does not already exist. This function removes the existing neighbors, re-adds them with the new cost, and updates the LSDB accordingly. If the link does not exist, it calls the `add_link` function to create it. What's next? - Implementing the `remove_link` function.
22 May 2024 17:56:01 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r130 | a1799298 | 2024-05-22 17:56:01 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Implemented remove_link function to remove an existing link between nodes and update the link-state database.
22 May 2024 17:57:12 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r131 | a1799298 | 2024-05-22 17:57:12 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py

	Updated process_input function to handle link removal when cost is -1.
22 May 2024 18:01:00 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 19 (5:58pm, May 22, 2024) ========== Reflection: Without much hassle, I've implemented the following: 1. **remove_link Function:** - Implemented the `remove_link` function to remove an existing link between two nodes and update the link-state database (LSDB). This function checks if the link exists before attempting to remove it, ensuring accurate network topology. 2. **process_input Function Update:** - Updated the `process_input` function to handle cases where the cost is -1, indicating a link removal. This ensures that the network can dynamically handle the removal of links based on input. Observations: - I have functionality for removing a link inside 'add link'. This might not be necessary, but it's good to be rigorous. - I'm really really glad I planned this all out beforehand. It's paying dividends What's next? - Implementing print output functions.
22 May 2024 18:03:25 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r132 | a1799298 | 2024-05-22 18:03:25 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Implemented print_output function to print neighbor table, LSDB, and routing table for specified nodes.
22 May 2024 18:08:13 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r133 | a1799298 | 2024-05-22 18:08:13 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Updated process_input function to call print_output for specified nodes and removed placeholder prints.
22 May 2024 18:12:12 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r134 | a1799298 | 2024-05-22 18:12:12 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Implemented print_neighbor_table function to print the neighbor table for the node in the required format.
22 May 2024 18:21:00 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 20 (6:20pm, May 22, 2024) ========== Reflection: I've implemented the `process_input` and `print_output` functions in the `Network` class, as well as `print_neighbour`. However, I've encountered an issue where the `print_lsdb` function within the `Node` class does not have access to the `lsdb` attribute, which is part of the `Network` class. I did touch on this in one of the first few entries, and I mistakenly said that the Node has access to the lsdb. This is wrong. This oversight means that individual nodes cannot directly print the link-state database (LSDB) as they lack access to the global `lsdb` information stored in the `Network` class. As a result, the current implementation of the `print_lsdb` function in the `Node` class is not feasible.

	What's the Fix? - I've decided to move the `print_lsdb` function to the `Network` class and update the `print_output` function accordingly. This approach ensures that the LSDB is printed correctly using the information available in the `Network` class. What's next? - Move `print_lsdb` to the `Network` class and update `print_output` accordingly. - Test the updated implementation to ensure correct functionality.
22 May 2024 18:22:11 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r135 | a1799298 | 2024-05-22 18:22:11 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py M /2024/s1/cna/routing/Node.py Removed 'print_lsdb' function from Node and added to Network
22 May 2024 18:23:02 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r136 | a1799298 | 2024-05-22 18:23:02 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Removed 'Updated 'print_output' inside Network to call 'print_lsdb' directly
22 May 2024 18:25:11 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r137 | a1799298 | 2024-05-22 18:25:11 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Removed printing from add_node function
22 May 2024 18:31:40 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 21 (6:31pm, May 22, 2024) ========== LSDB has now been implemented properly (sort of) Observations - For some reason, the example outputs in the assignment spec seem to have LSDBs instantiated on a node by node basis. My implementation works (and is more space efficient!!!), but ill have to fix this up later. What's next? - Implement Dijkstra in the routing table calculations - Implement the function to print the routing table for each node.
22 May 2024 18:36:18 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r138 | a1799298 | 2024-05-22 18:36:18 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Implemented basic Dijkstra's algorithm in Node class to calculate routing tables.
22 May 2024 18:38:30 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r139 | a1799298 | 2024-05-22 18:38:30 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Updated add_link, remove_link, and update_link functions to recalculate routing tables after changes in network topology.
22 May 2024 18:39:08 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r140 | a1799298 | 2024-05-22 18:39:08 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Implemented calculate_routing_tables function in Network class to recalculate routing tables for all nodes.

22 May 2024 18:41:35
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r141 | a1799298 | 2024-05-22 18:41:35 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Node.py

Implemented print_routing_table function in Node

22 May 2024 18:46:27
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 22 (6:44pm, May 22, 2024)
==========

Reflection:
I've successfully implemented the basic version of Dijkstra's algorithm in the `Node` class to calculate routing tables.

To ensure that the routing tables are recalculated whenever the network topology changes, I updated the `add_link`, `remove_link`, and `update_link` functions in the `Network` class. These functions now call a new function, `calculate_routing_tables`, which recalculates the routing tables for all nodes in the network.

Changes made:
1. **Implemented Dijkstra's Algorithm:**
- Added the `calculate_routing_table` method to the `Node` class to compute the shortest paths and populate the routing table.

2. **Updated Link Functions:**
- Modified `add_link`, `remove_link`, and `update_link` in the `Network` class to call `calculate_routing_tables` after making changes to the network topology.

3. **New `calculate_routing_tables` Function:**
- Created a new method `calculate_routing_tables` in the `Network` class to iterate over all nodes and recalculate their routing tables.

4. **Updated `print_output` Function:**
- Ensured that the `print_output` function in the `Network` class correctly prints the routing tables for the specified nodes.

5. **Implemented `print_routing_table` Function:**
- Added the `print_routing_table` method to the `Node` class to print the routing table in the required format.

It's working beautifully!

```
X
Y
Z
LINKSTATE
X-Z 1 X,Y
X-Y 5
Y-Z 3 X,Z
UPDATE
X-Z -1 X,Y
Y-Z 9 Y,Z
END
X Neighbour Table:
Z|1

X LSDB:
X|Z|1
```

X Routing Table:
Z|Z|1

Y Neighbour Table:

Y LSDB:
X|Z|1

Y Routing Table:

X Neighbour Table:
Y|5
Z|1

X LSDB:
X|Y|5
X|Z|1
Y|Z|3

X Routing Table:
Y|Z|4
Z|Z|1

Z Neighbour Table:
X|1
Y|3

Z LSDB:
X|Y|5
X|Z|1
Y|Z|3

Z Routing Table:
X|X|1
Y|Y|3

X Neighbour Table:
Y|5

X LSDB:
X|Y|5
Y|Z|3

X Routing Table:
Y|Y|5
Z|Y|8

Y Neighbour Table:
X|5
Z|3

Y LSDB:
X|Y|5
Y|Z|3

Y Routing Table:
X|X|5

Z|Z|3

Y Neighbour Table:
X|5
Z|9

Y LSDB:
X|Y|5
Y|Z|9

Y Routing Table:
X|X|5
Z|Z|9

Z Neighbour Table:
Y|9

Z LSDB:
X|Y|5
Y|Z|9

Z Routing Table:
X|Y|14
Y|Y|9
```

What's next?
- Fix up LSDB to be instantiated within nodes, not the network itself.
- Lots and lots and lots of testing
```

22 May 2024 18:50:02 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 23 (6:49pm, May 23, 2024) ========== Plan for implementing LSDB Instantiation within Nodes **Objective:** Move the instantiation of the Link-State Database (LSDB) from the `Network` class to individual `Node` instances. This will allow each node to maintain its own LSDB, reflecting its view of the network topology. **Steps to Implement:** 1. **Modify Node Class:** - Add an `lsdb` attribute to the `Node` class. - Initialize the `lsdb` attribute as a dictionary in the `Node` constructor. 2. **Update Network Class:** - Remove the `lsdb` attribute from the `Network` class. - Update the `add_link`, `remove_link`, and `update_link` methods to update the `lsdb` of each affected node. 3. **Adjust Link-State Database Functions:** - Ensure that the `print_lsdb` function in the `Node` class correctly prints the node's own LSDB. 4. **Recalculate Routing Tables:** - Ensure that routing tables are recalculated based on the updated LSDB information in each node.
22 May 2024 18:53:42 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r142 | a1799298 | 2024-05-22 18:53:42 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Node.py Added LSDB attribute to Node class and initialized it in the constructor.
22 May 2024 18:54:52 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r143 | a1799298 | 2024-05-22 18:54:52 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py Removed lsdb attribute from Network class and updated add_link, remove_link, and update_link methods to update node-specific lsdb.
22 May 2024 19:12:04 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r144 | a1799298 | 2024-05-22 19:12:04 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py M /2024/s1/cna/routing/Node.py Reverted previous changes. See next logbook for details
22 May 2024 19:48:08 Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?	r145 | a1799298 | 2024-05-22 19:48:08 +0930 (Wed, 22 May 2024) | 1 line Changed paths: M /2024/s1/cna/routing/Network.py M /2024/s1/cna/routing/Node.py Fix LSDB printing to include only reachable links
22 May 2024 19:52:31 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 24 (8:22 PM, May 23, 2024) ========== After implementing the routing table calculations using Dijkstra's algorithm, the next challenge was printing the Link-State Database (LSDB) for each node. Despite a comprehensive plan, this task proved more difficult

than anticipated.

Initial Implementation:
- LSDB was instantiated within the Network class, representing the global network topology.
- The expected output suggested that the LSDB should be maintained and printed for individual nodes.

Refactoring:
- Moved LSDB instantiation to the Node class.
- Modified add_link, remove_link, and update_link methods in Network to update the LSDB of affected nodes.
- Updated print_lsdb method to retrieve LSDB from each node instance.

Persistent Issue:
- Output still didn't match the expected format.
- Realized that the LSDB for a node should only include links directly connected to that node, not the entire network topology.

Final Solution:
- Implemented a breadth-first search (BFS) algorithm in print_lsdb to find all reachable links from the specified node.
- This approach ensured that the LSDB output contained only the relevant links for each node.

Next Steps:
- Conduct extensive testing with various network topologies and edge cases.

22 May 2024 20:01:32
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 25 (8:00 PM, May 23, 2024)
==========

Final Reflections: Code Implementation and Development Process

As I reflect on the code implementation and development process for this assignment, there are several aspects that stand out, both in terms of what went as expected and what presented unexpected challenges.

Planning and Design:
The initial planning and design phase (Entry 4) proved to be invaluable. Breaking down the problem into smaller components, such as the Node and Network classes, and outlining their respective responsibilities and methods, provided a solid foundation for the implementation. This approach allowed me to tackle the problem in a modular and organized manner, which paid dividends as the development progressed.

Node Class Implementation:
The implementation of the Node class (Entry 7) went relatively smoothly. The concepts of object-oriented programming were fresh in my mind, and I was able to encapsulate the necessary attributes and methods effectively. The docstrings and comments helped maintain a clear understanding of the code's purpose and functionality.

Network Class Implementation:
The Network class implementation (Entry 9) presented some challenges. While the overall structure was well-planned, the intricate details of handling input processing, link updates, and output formatting required more attention than anticipated. The `process_input` method, in particular, underwent multiple iterations and refinements (Entries 12, 13, 15) to handle the various input scenarios correctly.

Dijkstra's Algorithm Implementation:
Implementing Dijkstra's algorithm (Entry 22) for calculating routing tables was a relatively straightforward process, surpisingly!

LSDB Implementation:
The implementation of the Link-State Database (LSDB) proved to be one of the most challenging aspects of this

assignment. Initially, I had implemented the LSDB within the Network class (Entry 21), but later realized that it should be maintained and printed for each individual node based on the expected output format (Entry 23).

The breakthrough moment came when I implemented the Breadth-First Search (BFS) algorithm in the `print_lsdb` method to find all reachable links from a specified node (Entry 24). This approach ensured that the LSDB output contained only the relevant links for each node, as per the requirements.

Testing and Debugging:
The program has been tested quite minimally (only with the given example), so I still have lots more to do. I will be submitting for the first time now, and seeing how I go!

Overall, the code implementation and development process for this assignment presented a mix of expected and unexpected challenges. While some aspects went as planned, others required more effort and iterative refinement than initially anticipated.

22 May 2024 20:03:04
Kind: Web Submission, by: a1799298, from: 118_211_67_70

Web Submission System: submitted revision 145 to Routing

22 May 2024 20:04:46
Kind: SVN Log Entry, by: a1799298, from: ?_?_?_?

r146 | a1799298 | 2024-05-22 20:04:46 +0930 (Wed, 22 May 2024) | 1 line
Changed paths:
M /2024/s1/cna/routing/Network.py

Program implemented, move to testing / debugging

22 May 2024 20:20:17
Kind: Web Submission, by: a1799298, from: 118_211_67_70

Web Submission System: submitted revision 146 to Routing

22 May 2024 20:22:47
Kind: Text, by: a1799298, from: 118_211_67_70

==========
Logbook Entry 25 (8:00 PM, May 23, 2024)
==========

Final Reflections: Code Implementation and Development Process

As I reflect on the code implementation and development process for this assignment, there are several aspects that stand out, both in terms of what went as expected and what presented unexpected challenges.

Planning and Design:
The initial planning and design phase (Entry 4) proved to be invaluable. Breaking down the problem into smaller components, such as the Node and Network classes, and outlining their respective responsibilities and methods, provided a solid foundation for the implementation. This approach allowed me to tackle the problem in a modular and organized manner, which paid dividends as the development progressed.

Node Class Implementation:
The implementation of the Node class (Entry 7) went relatively smoothly. The concepts of object-oriented programming were fresh in my mind, and I was able to encapsulate the necessary attributes and methods effectively. The docstrings and comments helped maintain a clear understanding of the code's purpose and functionality.

Network Class Implementation:
The Network class implementation (Entry 9) presented some challenges. While the overall structure was well-planned, the intricate details of handling input processing, link updates, and output formatting required more attention than anticipated. The `process_input` method, in particular, underwent multiple iterations and refinements (Entries 12, 13, 15) to handle the various input scenarios correctly.

Dijkstra's Algorithm Implementation:
Implementing Dijkstra's algorithm (Entry 22) for calculating routing tables was a relatively straightforward process, surpisingly!

LSDB Implementation:

	The implementation of the Link-State Database (LSDB) proved to be one of the most challenging aspects of this assignment. Initially, I had implemented the LSDB within the Network class (Entry 21), but later realized that it should be maintained and printed for each individual node based on the expected output format (Entry 23). The breakthrough moment came when I implemented the Breadth-First Search (BFS) algorithm in the `print_lsdb` method to find all reachable links from a specified node (Entry 24). This approach ensured that the LSDB output contained only the relevant links for each node, as per the requirements. Testing and Debugging: The program has been tested quite minimally (only with the given example), so I still have lots more to do. I will be submitting for the first time now, and seeing how I go! Overall, the code implementation and development process for this assignment presented a mix of expected and unexpected challenges. While some aspects went as planned, others required more effort and iterative refinement than initially anticipated.
23 May 2024 11:54:18 Kind: Text, by: a1799298, from: 118_211_67_70	========== Logbook Entry 26 (11:52 AM, May 24, 2024) ========== I've implemented a simple test driver, and included the given test and its expected output! The output is quite nice, it will show me any deviations in the actual output vs expected output in red. I'll add an image for this in the next log! From here on out, I can easily add more test cases and see how my program fares! What's next? - Continue testing - Monitor Piazza for any issues that others might have faced and see if they affect me too.
23 May 2024 11:56:03 Kind: Text, by: a1799298, from: 118_211_67_70	========== aside ========== Seems I made an error somewhere and incremented the date for some reason. For clarity - All code commits occured on 22 May 2024 - Final entry (up to this point) occured on 23 May 2024

23 May 2024 11:59:53
Kind: JPEG Image, by: a1799298, from: 118_211_67_70