CS4150: Computer Networks Lab

Lab4

111901030
Mayank Singla

Q1. Write a program that uses Dijkstra's algorithm to compute the least cost path from a given node to every other node in the network. Your program should take 3 arguments: the first argument is n — the number of nodes in the network, the second argument is s — the source node, and the third argument is the name of the file that contains the cost (a positive integer less than 10, 000) between every pair of nodes in the network. Note that every node need not be connected to every other node in the network. Your program should have a run-time complexity of at most O(n^2) and output the least cost path (along with the path price) from s to every node in the network.

First of all, I am reading the inputs we received from the command line and doing general input validation and other validation as per the constraints mentioned in the question.

I created a class **Edge** which will denote an edge in the network that has three public fields, the vertices **u** and **v**, and the edge weight **w**. Next, I created a function **stringToLong** which converts a string to a long long integer and if the string contains any non-digit character, it will throw an error. Next, I created a function **readEdges** which takes a **fileName** as input and reads the edges of the network from the given file if it exists and fills those edges in a vector of objects of **Edge** class.

Next, I created a function **buildGraph** which takes the above edges as input and creates the adjacency list representation of the graph. This function also creates a map and a reverse map between node names represented as strings and their normalized integer number starting from **0**. For this, I simply maintained a counter and assign its value to every new node name found and increments its value.
This way, all of my node names for calculations will be from **0** to **numNodes - 1**.

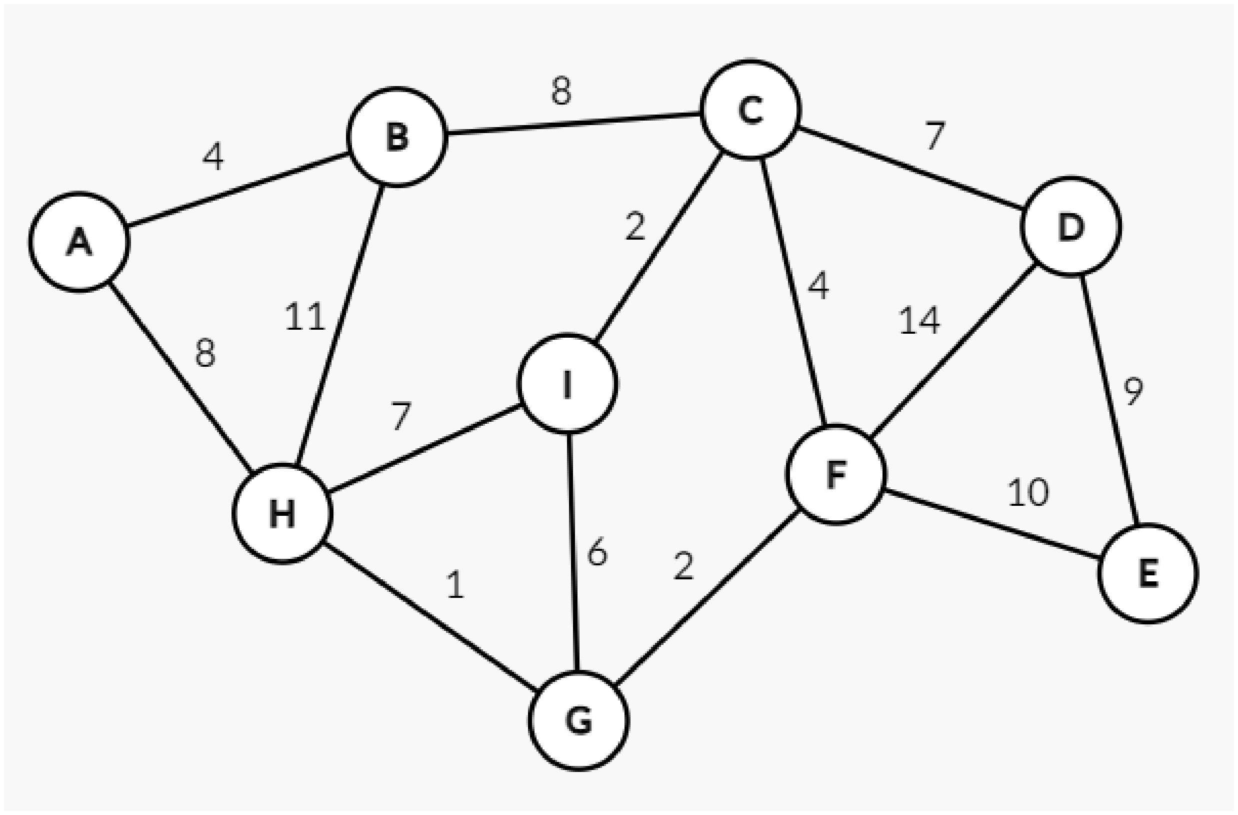
Then I created the function **dijkstra** to perform the Dijkstra algorithm on the network which takes as input a **sourceNode** and the adjacency list representation of a graph. I am initializing the **distance** and the **parent** vectors with **INFINITY** and **-1** respectively. I am marking the distance of the source node from itself as **0**. Then, I am iterating over all the nodes, and in each iteration, I am finding the node having the minimum distance among all the unvisited nodes. I am checking if the distance of this node found is INFINITY. If yes, then that means that all the remaining unvisited nodes in the algorithm are not reachable from the source node and I am exiting from the loop.
If not, then I am iterating over all the edges of the node found. I am checking if the distance of the current node from the source node plus the weight of the current edge with its neighbor is less than the distance of the neighbor node from the source node then updating the distance and the parent

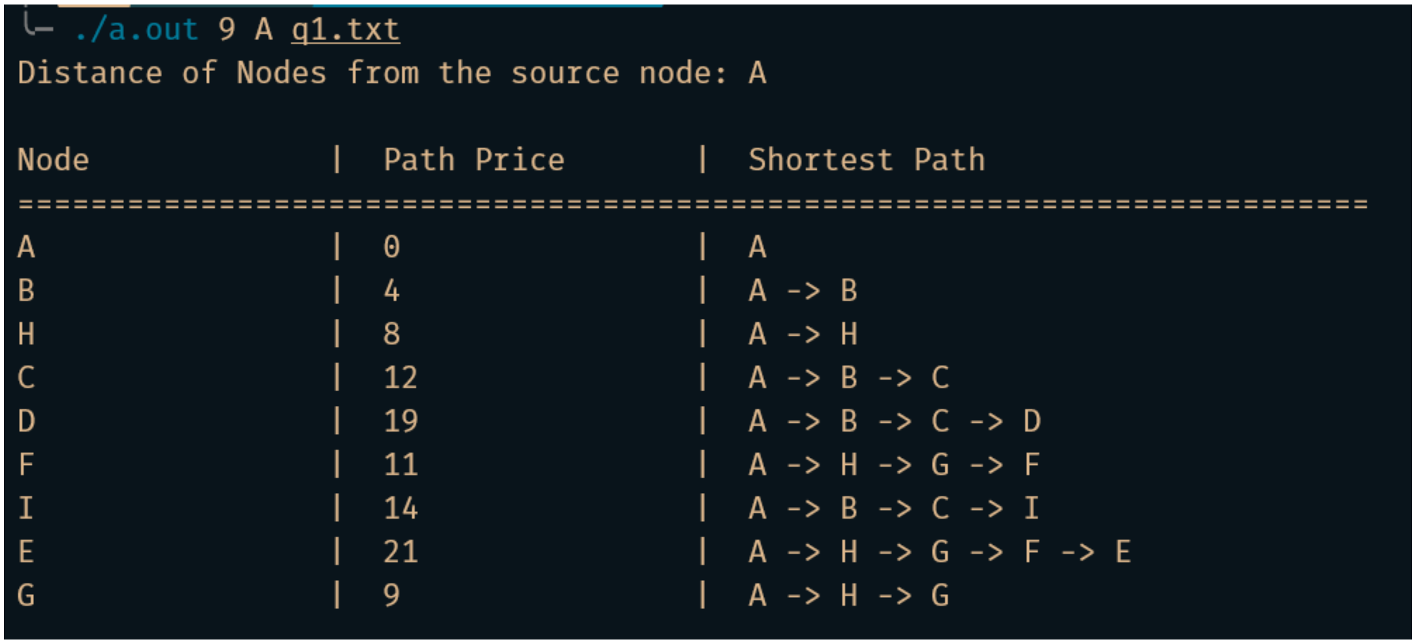
for the neighbor node. At the end of the loop, we will have the least cost path price stored in the **distance** vector and the parents of all the nodes on that least cost path in the **parent** vector.
To note, the distance and parent of the nodes unreachable from the source node in case of a disconnected graph will remain **INFINITE** and **-1** respectively. Also, the parent of the source node is also **-1**.
We are iterating over all the nodes exactly **n** times in the worst case to find the unvisited node having the minimum distance which contributes to **O(n^2)**. Also, collectively I am iterating over all the edges in the graph which contributes to **O(E)**. So the overall time complexity is **O(n^2 + E)**. The value of **E** in the worst case is **n(n-1)/2** for the case of a fully connected graph, so the worst-case time complexity is **O(n^2)**

Next, I created the function **prettyPrint** which prints any input with a defined width, alignment and fill character. Finally, I created the function **printResults** to print the results obtained from the Dijkstra algorithm. It outputs the least cost path from the source node to every node in the network along with the path price. To extract the path, I am starting from a node and using the **parent** vector I am backtracking all the parents until I reach the source node and then printing the path obtained.

Example Tests:

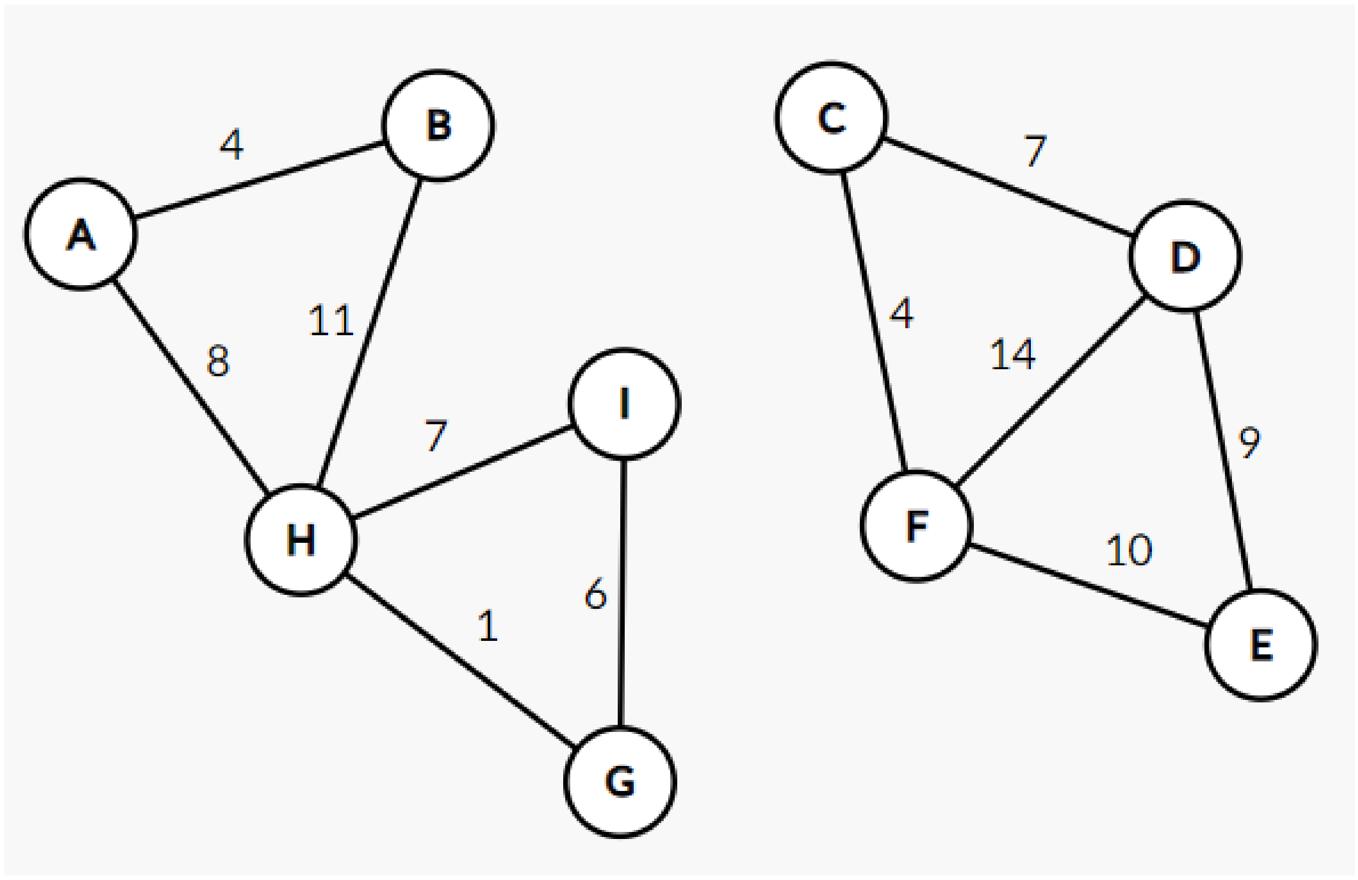
1.

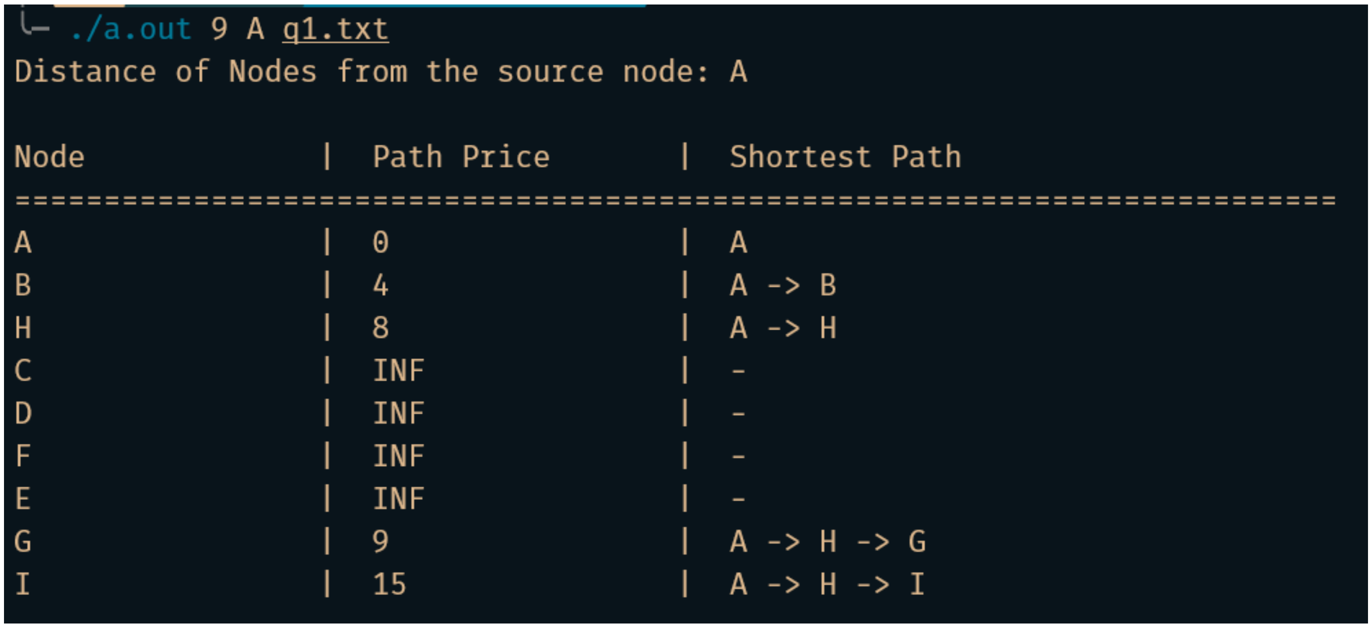




```
└─ ./a.out 9 A q1.txt
Distance of Nodes from the source node: A

Node                |  Path Price         |  Shortest Path
===========================================================================
A                   |  0                  |  A
B                   |  4                  |  A -> B
H                   |  8                  |  A -> H
C                   |  12                 |  A -> B -> C
D                   |  19                 |  A -> B -> C -> D
F                   |  11                 |  A -> H -> G -> F
I                   |  14                 |  A -> B -> C -> I
E                   |  21                 |  A -> H -> G -> F -> E
G                   |  9                  |  A -> H -> G
```

2.





```
└─ ./a.out 9 A q1.txt
Distance of Nodes from the source node: A

Node                |  Path Price        |  Shortest Path
==========================================================================
A                   |  0                 |  A
B                   |  4                 |  A -> B
H                   |  8                 |  A -> H
C                   |  INF               |  -
D                   |  INF               |  -
F                   |  INF               |  -
E                   |  INF               |  -
G                   |  9                 |  A -> H -> G
I                   |  15                |  A -> H -> I
```

Q2. Write a program to simulate the Distance Vector (DV) routing algorithm. Your program should take 2 arguments: the first argument is n — the number of nodes in the network, and the second argument is the name of the file which contains the cost (a positive integer less than 10, 000) between every pair of nodes in the network. Note that every node need not be connected to every other node in the network. Note that your program should be able to demonstrate the possibly random local routing table updates that can happen in DV, and should terminate and print the local routing table at every node after convergence has occurred.

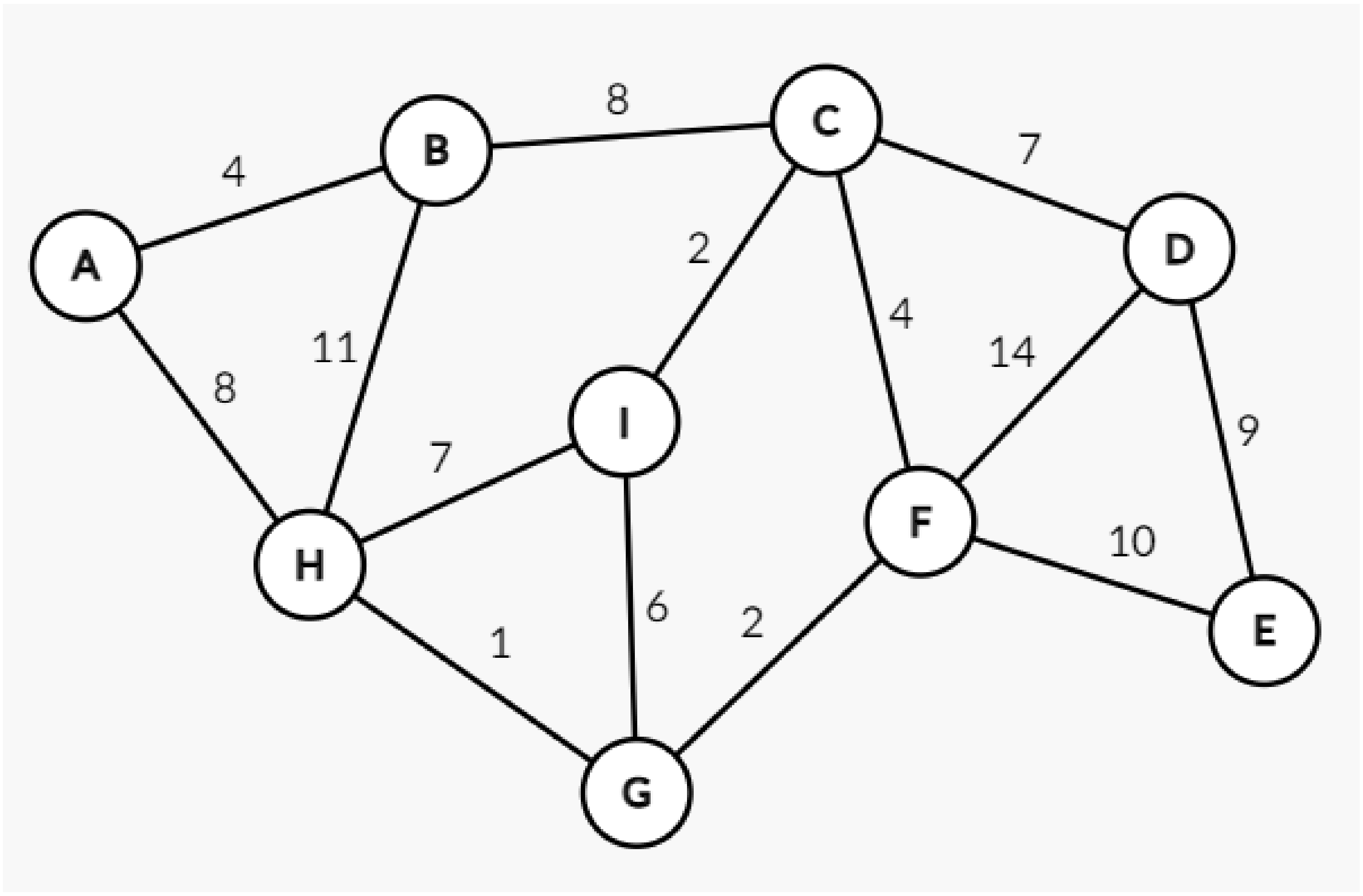
I am using the same class **Edge** and the same functions **stringToLong**, **readEdges,** and **prettyPrint** as in the previous question. I am overloading the **stream insertion operator (<<)** for generic pairs for their pretty printing. My **buildGraph** function is mostly the same as that in the previous question except it is now also initializing the distance vectors for each node with the initial conditions. The initial conditions are that the distance of each node from itself is **0** and they know their edge weights with their neighbor nodes. The **printResults** function is quite simple and it is simply pretty-printing the distance vectors of all the nodes in a tabular representation.

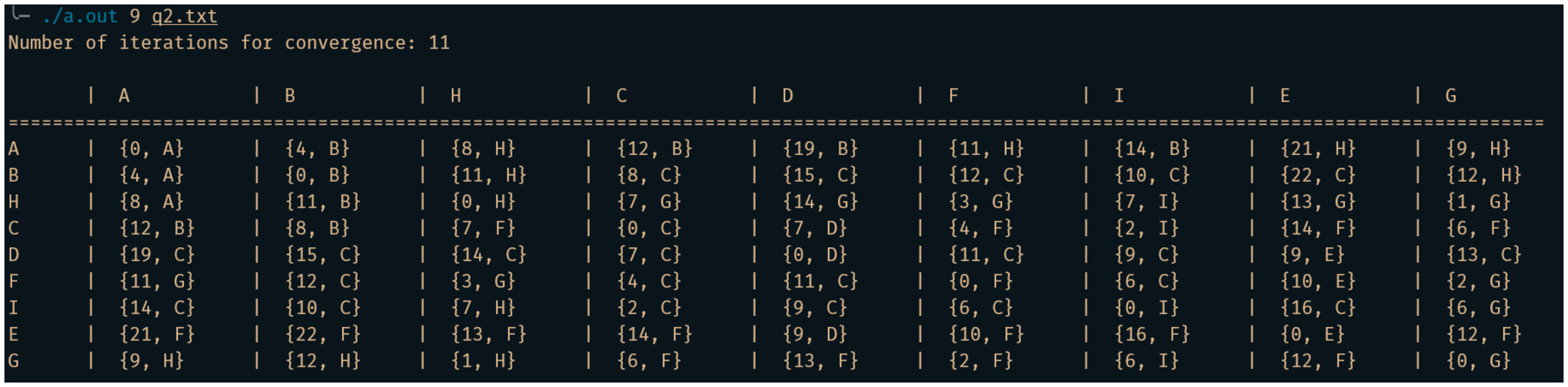
I am creating the function **distVecRouting** which executes the distance vector routing algorithm on the input graph till convergence. I am creating a random number generator to randomly shuffle the nodes and pick a random neighbor for a node in the algorithm. I am keeping count of the number of iterations for printing and the number of continuous iterations when there was no update to the distance vector of any of the nodes in the network.
Now, in an infinite loop, I am shuffling the nodes of the network and then iterating over them. Next, for a particular node, I am picking a random number generated using a uniform distribution so that all the neighbors have an equal probability of picking. Now using the distance vector of the selected neighbor, I am trying to update the distance vector of the current node.

I am keeping a flag **isUpdated** which will be marked as **true** if there is any update in this iteration of the while loop. I am incrementing the number of iterations count and if there was no update, increasing the count for the number of continuous iterations for no update. If my number of continuous iterations with no update reaches a max no updated value, then I am assuming that the convergence has occurred and I am terminating the algorithm. This was done because there could be some graphs in which there is no update in an iteration but still we haven't attained the minimum distance in the distance vectors due to randomness. This max no update convergence value I am storing it in a directive called **MAX_NO_UPDATE_CONV**, and keeping it to a high value so that the probability of an edge not being selected is reduced.

Example Tests:

1. MAX_NO_UPDATE_CONV = 100

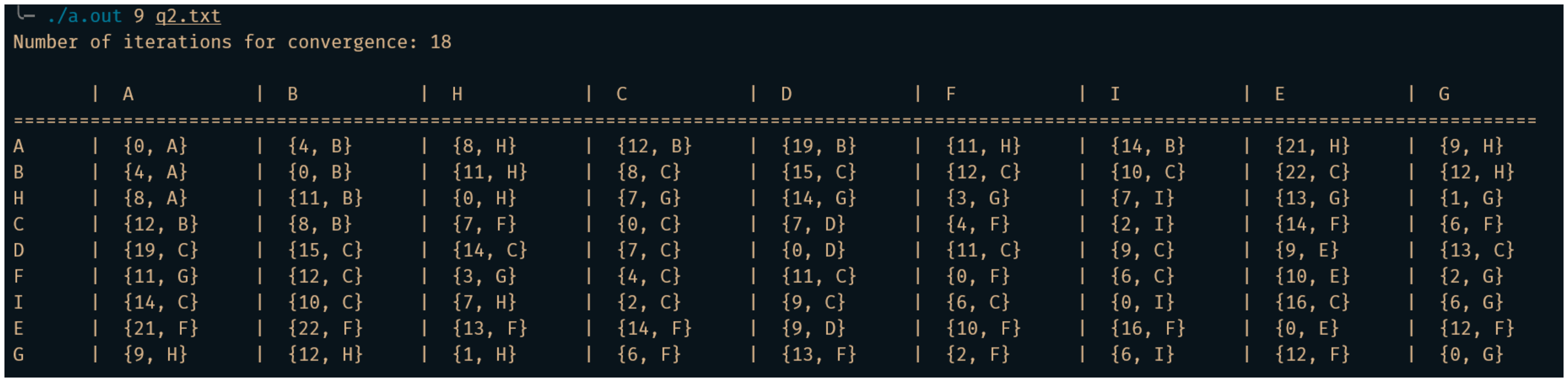




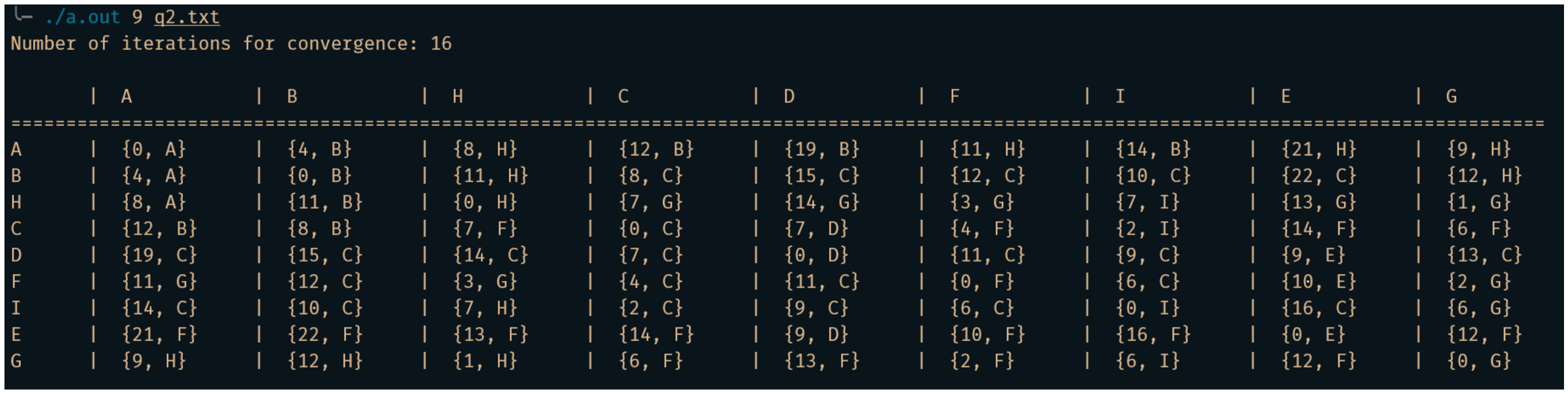
```
╰─ ./a.out 9 q2.txt
Number of iterations for convergence: 11

       |  A          |  B          |  H          |  C          |  D          |  F          |  I          |  E          |  G
===========================================================================================================================
A      |  {0, A}     |  {4, B}     |  {8, H}     |  {12, B}    |  {19, B}    |  {11, H}    |  {14, B}    |  {21, H}    |  {9, H}
B      |  {4, A}     |  {0, B}     |  {11, H}    |  {8, C}     |  {15, C}    |  {12, C}    |  {10, C}    |  {22, C}    |  {12, H}
H      |  {8, A}     |  {11, B}    |  {0, H}     |  {7, G}     |  {14, G}    |  {3, G}     |  {7, I}     |  {13, G}    |  {1, G}
C      |  {12, B}    |  {8, B}     |  {7, F}     |  {0, C}     |  {7, D}     |  {4, F}     |  {2, I}     |  {14, F}    |  {6, F}
D      |  {19, C}    |  {15, C}    |  {14, C}    |  {7, C}     |  {0, D}     |  {11, C}    |  {9, C}     |  {9, E}     |  {13, C}
F      |  {11, G}    |  {12, C}    |  {3, G}     |  {4, C}     |  {11, C}    |  {0, F}     |  {6, C}     |  {10, E}    |  {2, G}
I      |  {14, C}    |  {10, C}    |  {7, H}     |  {2, C}     |  {9, C}     |  {6, C}     |  {0, I}     |  {16, C}    |  {6, G}
E      |  {21, F}    |  {22, F}    |  {13, F}    |  {14, F}    |  {9, D}     |  {10, F}    |  {16, F}    |  {0, E}     |  {12, F}
G      |  {9, H}     |  {12, H}    |  {1, H}     |  {6, F}     |  {13, F}    |  {2, F}     |  {6, I}     |  {12, F}    |  {0, G}
```

The table shows the distance vector of each of the nodes after the convergence has occurred. Each pair shows the minimum distance and the neighbor which should be taken to reach the other node with that minimum distance.

We can see that on different executions the number of iterations for convergence differs due to randomness.

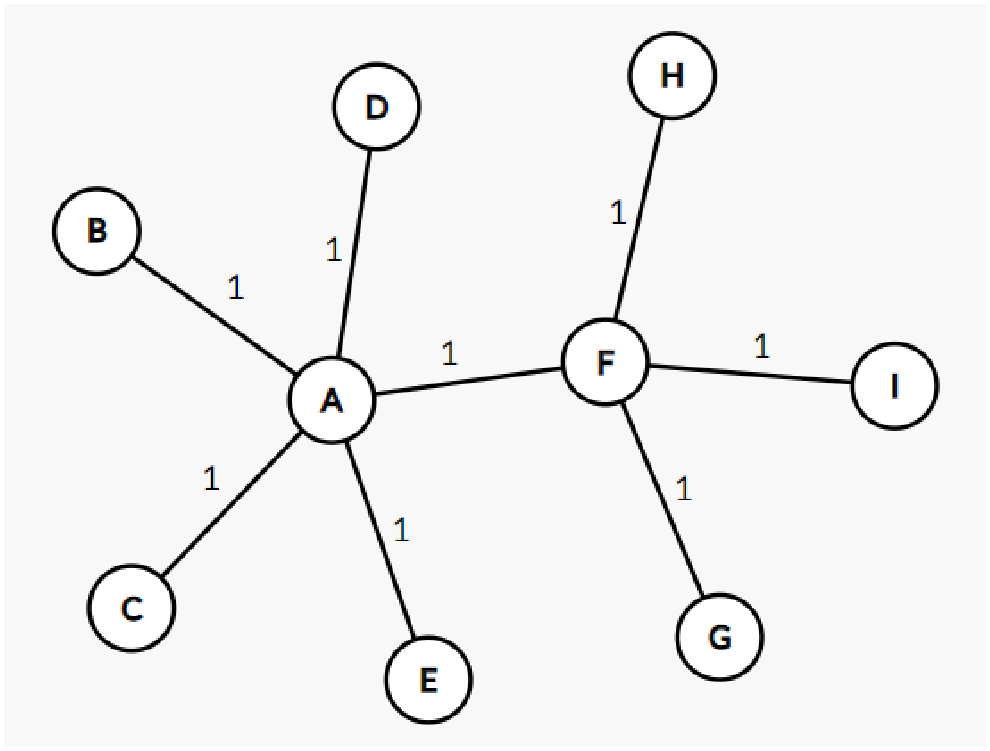


```
╰─ ./a.out 9 q2.txt
Number of iterations for convergence: 18

       |  A          |  B          |  H          |  C          |  D          |  F          |  I          |  E          |  G
===========================================================================================================================
A      |  {0, A}     |  {4, B}     |  {8, H}     |  {12, B}    |  {19, B}    |  {11, H}    |  {14, B}    |  {21, H}    |  {9, H}
B      |  {4, A}     |  {0, B}     |  {11, H}    |  {8, C}     |  {15, C}    |  {12, C}    |  {10, C}    |  {22, C}    |  {12, H}
H      |  {8, A}     |  {11, B}    |  {0, H}     |  {7, G}     |  {14, G}    |  {3, G}     |  {7, I}     |  {13, G}    |  {1, G}
C      |  {12, B}    |  {8, B}     |  {7, F}     |  {0, C}     |  {7, D}     |  {4, F}     |  {2, I}     |  {14, F}    |  {6, F}
D      |  {19, C}    |  {15, C}    |  {14, C}    |  {7, C}     |  {0, D}     |  {11, C}    |  {9, C}     |  {9, E}     |  {13, C}
F      |  {11, G}    |  {12, C}    |  {3, G}     |  {4, C}     |  {11, C}    |  {0, F}     |  {6, C}     |  {10, E}    |  {2, G}
I      |  {14, C}    |  {10, C}    |  {7, H}     |  {2, C}     |  {9, C}     |  {6, C}     |  {0, I}     |  {16, C}    |  {6, G}
E      |  {21, F}    |  {22, F}    |  {13, F}    |  {14, F}    |  {9, D}     |  {10, F}    |  {16, F}    |  {0, E}     |  {12, F}
G      |  {9, H}     |  {12, H}    |  {1, H}     |  {6, F}     |  {13, F}    |  {2, F}     |  {6, I}     |  {12, F}    |  {0, G}
```

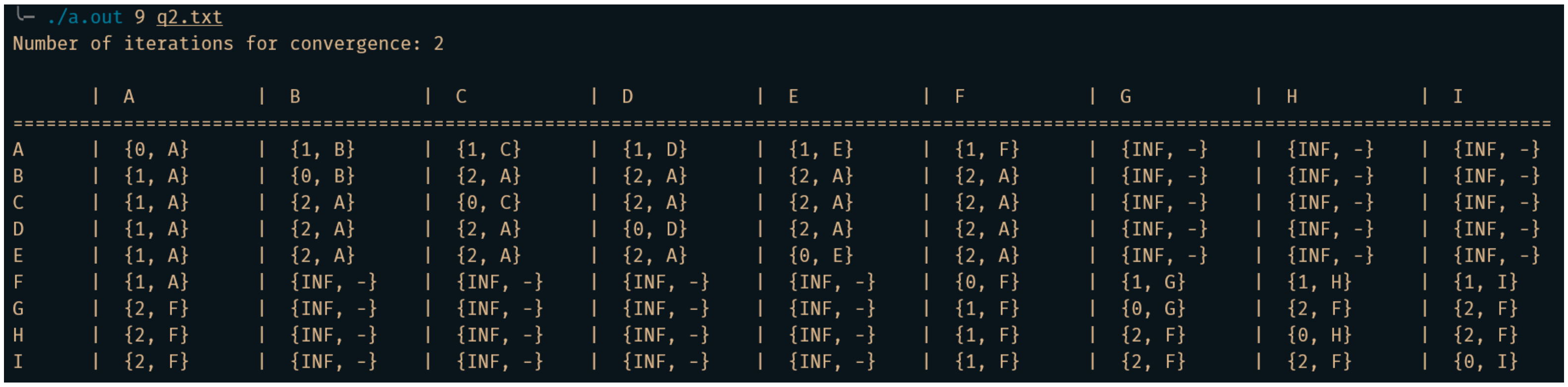


```
╰─ ./a.out 9 q2.txt
Number of iterations for convergence: 16

       |  A          |  B          |  H          |  C          |  D          |  F          |  I          |  E          |  G
===========================================================================================================================
A      |  {0, A}     |  {4, B}     |  {8, H}     |  {12, B}    |  {19, B}    |  {11, H}    |  {14, B}    |  {21, H}    |  {9, H}
B      |  {4, A}     |  {0, B}     |  {11, H}    |  {8, C}     |  {15, C}    |  {12, C}    |  {10, C}    |  {22, C}    |  {12, H}
H      |  {8, A}     |  {11, B}    |  {0, H}     |  {7, G}     |  {14, G}    |  {3, G}     |  {7, I}     |  {13, G}    |  {1, G}
C      |  {12, B}    |  {8, B}     |  {7, F}     |  {0, C}     |  {7, D}     |  {4, F}     |  {2, I}     |  {14, F}    |  {6, F}
D      |  {19, C}    |  {15, C}    |  {14, C}    |  {7, C}     |  {0, D}     |  {11, C}    |  {9, C}     |  {9, E}     |  {13, C}
F      |  {11, G}    |  {12, C}    |  {3, G}     |  {4, C}     |  {11, C}    |  {0, F}     |  {6, C}     |  {10, E}    |  {2, G}
I      |  {14, C}    |  {10, C}    |  {7, H}     |  {2, C}     |  {9, C}     |  {6, C}     |  {0, I}     |  {16, C}    |  {6, G}
E      |  {21, F}    |  {22, F}    |  {13, F}    |  {14, F}    |  {9, D}     |  {10, F}    |  {16, F}    |  {0, E}     |  {12, F}
G      |  {9, H}     |  {12, H}    |  {1, H}     |  {6, F}     |  {13, F}    |  {2, F}     |  {6, I}     |  {12, F}    |  {0, G}
```

2.



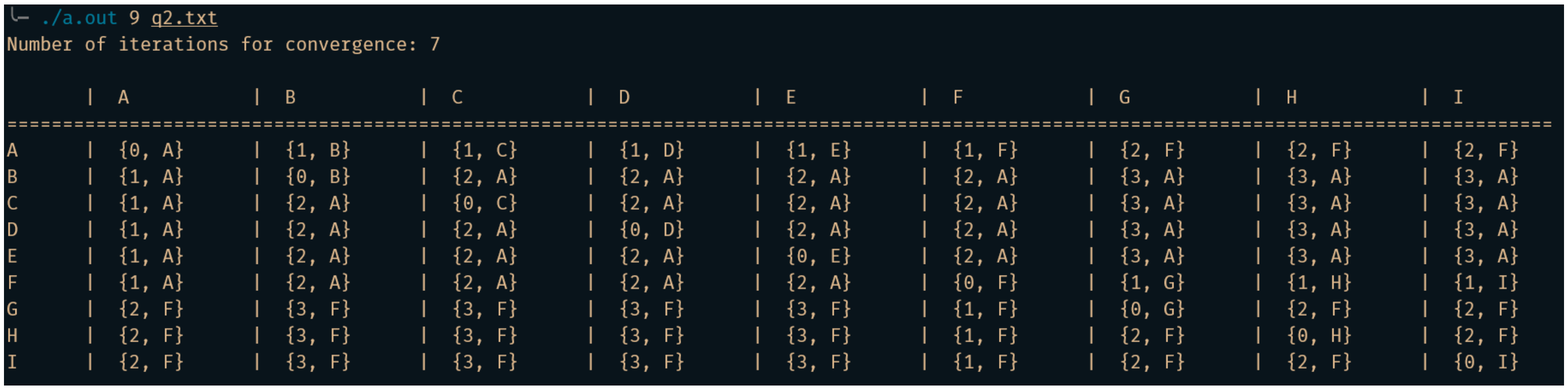
2.1 MAX_NO_UPDATE_CONV = 1



```
╰─ ./a.out 9 q2.txt
Number of iterations for convergence: 2

        |  A         |  B         |  C         |  D         |  E         |  F         |  G         |  H         |  I
=====================================================================================================================================
A       |  {0, A}    |  {1, B}    |  {1, C}    |  {1, D}    |  {1, E}    |  {1, F}    |  {INF, -}  |  {INF, -}  |  {INF, -}
B       |  {1, A}    |  {0, B}    |  {2, A}    |  {2, A}    |  {2, A}    |  {2, A}    |  {INF, -}  |  {INF, -}  |  {INF, -}
C       |  {1, A}    |  {2, A}    |  {0, C}    |  {2, A}    |  {2, A}    |  {2, A}    |  {INF, -}  |  {INF, -}  |  {INF, -}
D       |  {1, A}    |  {2, A}    |  {2, A}    |  {0, D}    |  {2, A}    |  {2, A}    |  {INF, -}  |  {INF, -}  |  {INF, -}
E       |  {1, A}    |  {2, A}    |  {2, A}    |  {2, A}    |  {0, E}    |  {2, A}    |  {INF, -}  |  {INF, -}  |  {INF, -}
F       |  {1, A}    |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {0, F}    |  {1, G}    |  {1, H}    |  {1, I}
G       |  {2, F}    |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {1, F}    |  {0, G}    |  {2, F}    |  {2, F}
H       |  {2, F}    |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {1, F}    |  {2, F}    |  {0, H}    |  {2, F}
I       |  {2, F}    |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {INF, -}  |  {1, F}    |  {2, F}    |  {2, F}    |  {0, I}
```

We can observe that the algorithm converges but still the distances are not correctly computed

2.1 MAX_NO_UPDATE_CONV = 100



```
╰─ ./a.out 9 q2.txt
Number of iterations for convergence: 7

        |  A         |  B         |  C         |  D         |  E         |  F         |  G         |  H         |  I
=====================================================================================================================================
A       |  {0, A}    |  {1, B}    |  {1, C}    |  {1, D}    |  {1, E}    |  {1, F}    |  {2, F}    |  {2, F}    |  {2, F}
B       |  {1, A}    |  {0, B}    |  {2, A}    |  {2, A}    |  {2, A}    |  {2, A}    |  {3, A}    |  {3, A}    |  {3, A}
C       |  {1, A}    |  {2, A}    |  {0, C}    |  {2, A}    |  {2, A}    |  {2, A}    |  {3, A}    |  {3, A}    |  {3, A}
D       |  {1, A}    |  {2, A}    |  {2, A}    |  {0, D}    |  {2, A}    |  {2, A}    |  {3, A}    |  {3, A}    |  {3, A}
E       |  {1, A}    |  {2, A}    |  {2, A}    |  {2, A}    |  {0, E}    |  {2, A}    |  {3, A}    |  {3, A}    |  {3, A}
F       |  {1, A}    |  {2, A}    |  {2, A}    |  {2, A}    |  {2, A}    |  {0, F}    |  {1, G}    |  {1, H}    |  {1, I}
G       |  {2, F}    |  {3, F}    |  {3, F}    |  {3, F}    |  {3, F}    |  {1, F}    |  {0, G}    |  {2, F}    |  {2, F}
H       |  {2, F}    |  {3, F}    |  {3, F}    |  {3, F}    |  {3, F}    |  {1, F}    |  {2, F}    |  {0, H}    |  {2, F}
I       |  {2, F}    |  {3, F}    |  {3, F}    |  {3, F}    |  {3, F}    |  {1, F}    |  {2, F}    |  {2, F}    |  {0, I}
```