Grant Davis
CPE400 Final Project
5/4/2022

Technical Report

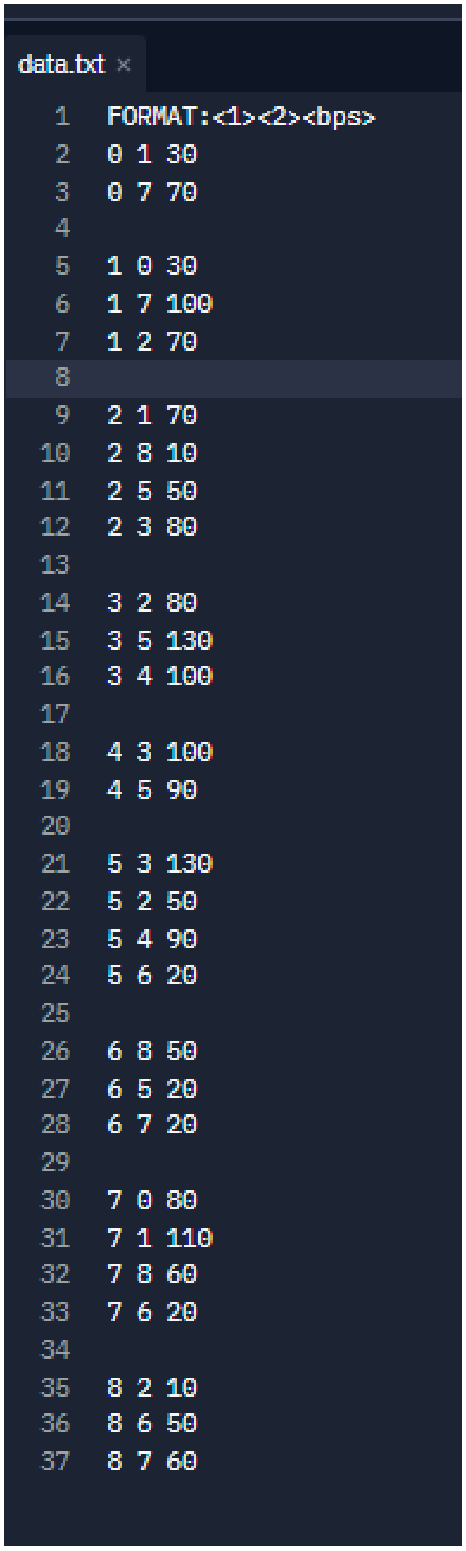
Project Topic (3): Dynamic routing mechanism design with a focus on throughput. Create and simulate a new routing strategy that maximizes the overall throughput of a mesh network. Throughput is affected by many factors that should be considered, such as nodal processing delay, overloaded buffers, loss, etc. The more realistic assumptions you can make for your network, the better it is.

Problem:
The amount of data that can be sent through a network is the throughput, which is measured in bits per second (bps). The higher number of bps allows for faster data transfer between routers. Many factors affect throughput, however in this project bandwidth will be the focus. Bandwidth allows the maximum amount of data to be sent through a network channel. To maximize throughput, route the path with the highest bandwidth.

Solution:
The graph data structure is best to show networks in a program. An adjacency matrix is used to show the graph in this project. An algorithm for finding the maximum bandwidth path will also be needed. Pseudo-Code for the algorithm is as follows:

1. Any two routers in the network, node1, and node2, use a depth-first algorithm to search all paths from node1 to node2.
2. bandwidth = min(all hop bandwidths) The bandwidth of each path is calculated.
3. max(all path bandwidths) is the chosen path.

Program Functionality:
The main function reads data from the “data.txt” file to configure the network, shown in Fig. 1. After the content from the file creates the network the algorithm can be run on it. The program reads the “data.txt” config file and builds an adjacency matrix to show the network in a table format. The output graph table is shown in Fig. 2.



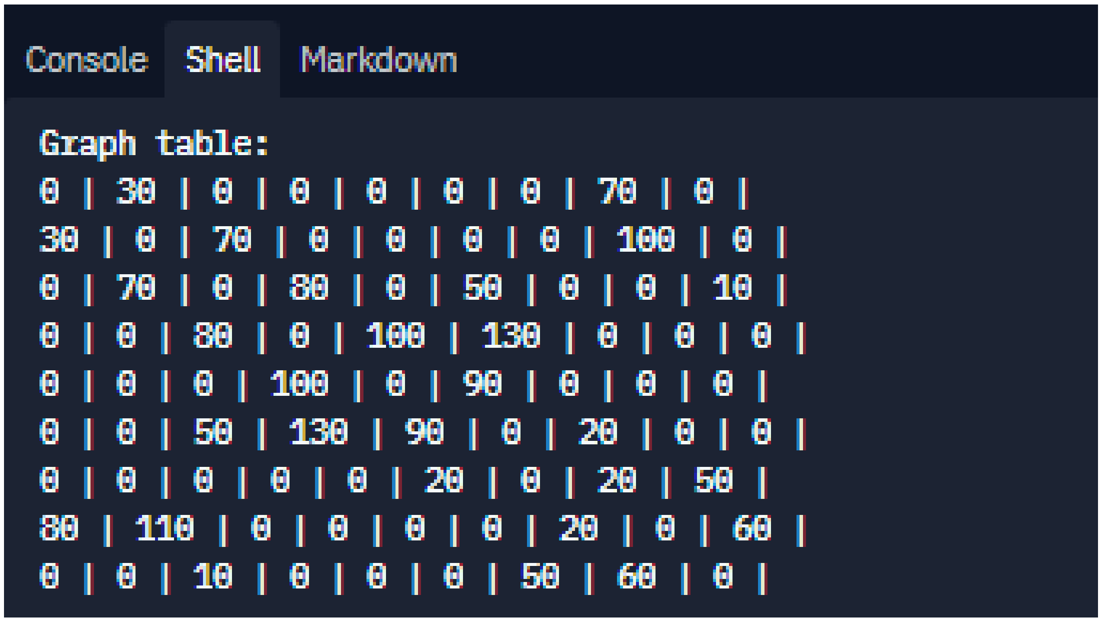
data.txt ×

```
FORMAT:<1><2><bps>
0 1 30
0 7 70

1 0 30
1 7 100
1 2 70

2 1 70
2 8 10
2 5 50
2 3 80

3 2 80
3 5 130
3 4 100

4 3 100
4 5 90

5 3 130
5 2 50
5 4 90
5 6 20

6 8 50
6 5 20
6 7 20

7 0 80
7 1 110
7 8 60
7 6 20

8 2 10
8 6 50
8 7 60
```

Fig 1. “data.txt”

Table 1. Adjacency Matrix for Network Graph

Nodes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	30	0	0	0	0	0	70	0
1	30	0	100	0	0	0	0	70	0
2	0	70	0	10	0	50	0	0	80
3	0	0	80	0	130	100	0	0	0
4	0	0	0	100	0	90	0	0	0
5	0	0	130	50	90	0	20	0	0
6	0	0	0	0	0	50	0	20	20
7	80	110	0	0	0	0	60	0	20
8	0	0	10	0	0	0	50	60	0



```
Graph table:
0 | 30 | 0 | 0 | 0 | 0 | 0 | 70 | 0 |
30 | 0 | 70 | 0 | 0 | 0 | 0 | 100 | 0 |
0 | 70 | 0 | 80 | 0 | 50 | 0 | 0 | 10 |
0 | 0 | 80 | 0 | 100 | 130 | 0 | 0 | 0 |
0 | 0 | 0 | 100 | 0 | 90 | 0 | 0 | 0 |
0 | 0 | 50 | 130 | 90 | 0 | 20 | 0 | 0 |
0 | 0 | 0 | 0 | 0 | 20 | 0 | 20 | 50 |
80 | 110 | 0 | 0 | 0 | 0 | 20 | 0 | 60 |
0 | 0 | 10 | 0 | 0 | 0 | 50 | 60 | 0 |
```

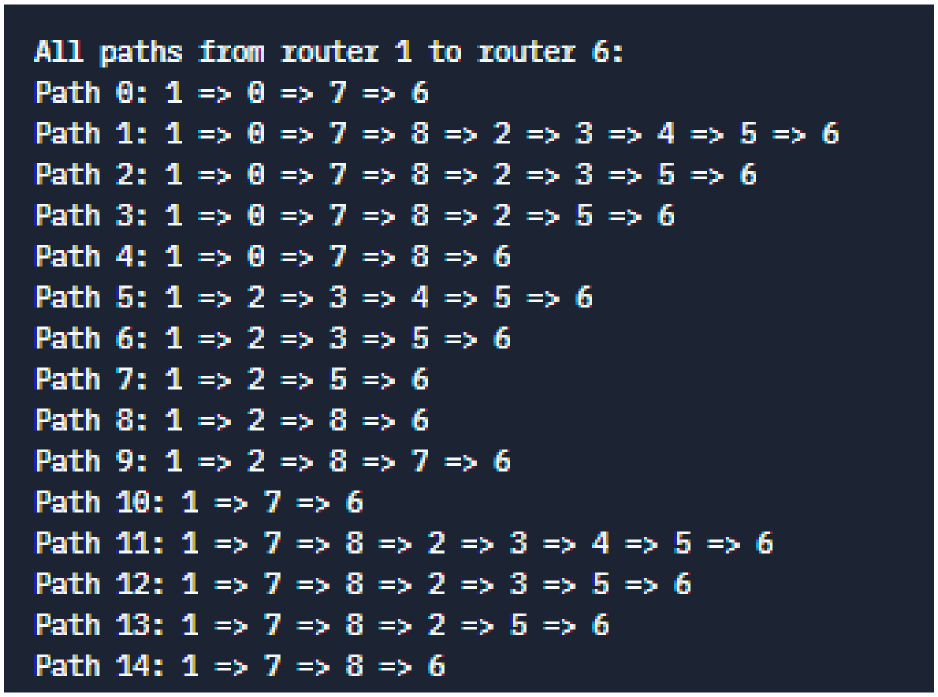
Fig 2. Printed Graph Table

Using any two routers, int x and int y, an adjacency matrix graph, G, is created. G[x][y] will be where bandwidth is stored. After the graph is created the algorithm is run.

The algorithm starts by finding all paths from the chosen source router and the destination router. To find all paths the following depth-first search algorithm is used:

```
void Graph::findAllPaths(int source, int destination, int index, ofstream& fout){
    visited[source] = true;
    path[index] = source;
    index++;

    if(source == destination){
        for(int i=0; i<index ; i++){
            fout << path[i] << " ";
        }
        fout << endl;
    }
    else
    {
        for(int v=0 ; v<numVert ; v++){
            if(!visited[v] && graph[source][v] > 0){
                findAllPaths(v, destination, index, fout);
            }
        }
    }
    index--;
    visited[source] = false;
}
```

The Depth First Search (DFS) algorithm is used to find all nodes in the graph from a source node. In DFS, the base case exits when all nodes have been found. In the modified DFS, “findAllPaths()” is a different base case where the source id is equal to the destination id. When the base case condition is met, all nodes stored in the array “path[]” is printed. The function will use recursion, each recursive step, the destination is removed from visited nodes. The algorithm restarts using the next undiscovered node. When the destination is found the array is printed. If a path cannot be found from removing destination nodes will continue to remove nodes until a new path can be created. The array is printed and saved to “path.txt” every time. Fig 3. Shows the program printing out all paths from source router 1 to destination router 6.



```
All paths from router 1 to router 6:
Path 0: 1 => 0 => 7 => 6
Path 1: 1 => 0 => 7 => 8 => 2 => 3 => 4 => 5 => 6
Path 2: 1 => 0 => 7 => 8 => 2 => 3 => 5 => 6
Path 3: 1 => 0 => 7 => 8 => 2 => 5 => 6
Path 4: 1 => 0 => 7 => 8 => 6
Path 5: 1 => 2 => 3 => 4 => 5 => 6
Path 6: 1 => 2 => 3 => 5 => 6
Path 7: 1 => 2 => 5 => 6
Path 8: 1 => 2 => 8 => 6
Path 9: 1 => 2 => 8 => 7 => 6
Path 10: 1 => 7 => 6
Path 11: 1 => 7 => 8 => 2 => 3 => 4 => 5 => 6
Path 12: 1 => 7 => 8 => 2 => 3 => 5 => 6
Path 13: 1 => 7 => 8 => 2 => 5 => 6
Path 14: 1 => 7 => 8 => 6
```

Fig 3. Paths from router 1 to router 6

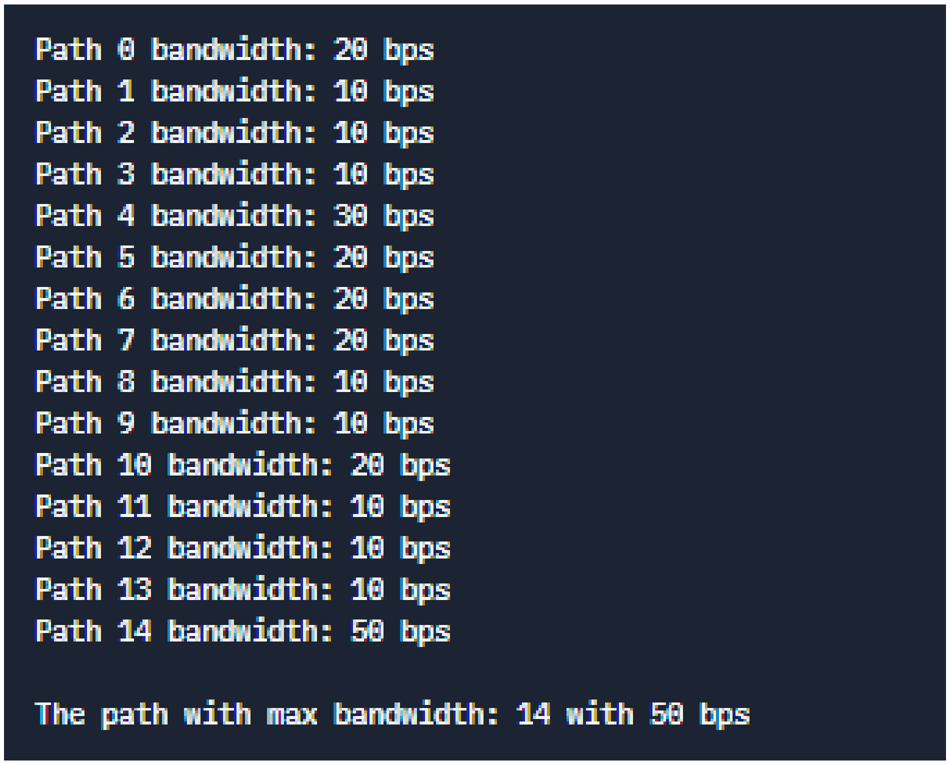
Once all paths between the source and destination routers are found, bandwidth of each path is then found. The following function "findBandwidth()" calculates the bandwidth of a path.

```
int Graph::findBandwidth(int path[], int routers)
{
    int bandwidth = __INT_MAX__;
    for(int i=0 ; i<routers-1 ; i++){
        bandwidth = min(bandwidth, graph[path[i]][path[i+1]]);
    }
    return bandwidth;
}
```

To calculate a path's bandwidth, the minimum bandwidth of all paths is used. For example, using path 9 from Fig 3. The bandwidth of each path will be stored at G[x][y]. X being the source node and y being the destination node.

Path 9: 1 => 2 => 8 => 7 => 6
G[1][2]=70 bps
G[2][8]=10 bps
G[8][7]=60 bps
G[7][6]=20 bps

Since the minimum bandwidth of path 9 is 10 bps, then the bandwidth of the path would be 10 bps. All the bandwidths from source router 1 to destination router 6 paths are shown in Fig. 4.

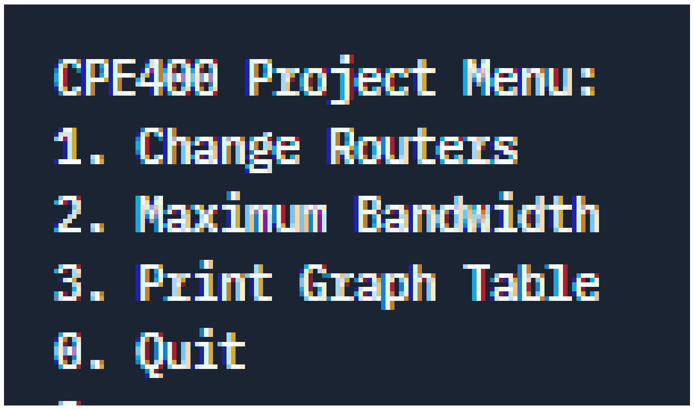


```
Path 0 bandwidth: 20 bps
Path 1 bandwidth: 10 bps
Path 2 bandwidth: 10 bps
Path 3 bandwidth: 10 bps
Path 4 bandwidth: 30 bps
Path 5 bandwidth: 20 bps
Path 6 bandwidth: 20 bps
Path 7 bandwidth: 20 bps
Path 8 bandwidth: 10 bps
Path 9 bandwidth: 10 bps
Path 10 bandwidth: 20 bps
Path 11 bandwidth: 10 bps
Path 12 bandwidth: 10 bps
Path 13 bandwidth: 10 bps
Path 14 bandwidth: 50 bps

The path with max bandwidth: 14 with 50 bps
```

Fig 4. Bandwidths from router 1 to router 6

After all bandwidths for all paths have been found, the max bandwidth is printed to the user. As shown in Fig 4. The max bandwidth for router 1 to router 6 is 50 bps, from path 14.

The program uses a menu system to test maximum bandwidth on multiple different source routers and destination routers. The Menu is shown in Fig 5. Allowing users to change routers, test maximum bandwidth, and view the graph table.



```
CPE400 Project Menu:
1. Change Routers
2. Maximum Bandwidth
3. Print Graph Table
0. Quit
```

Fig 5. Program Menu

Analysis
The algorithm used in the program is dynamic for its ability to switch from any source to any destination on the network. The algorithm is also reactive because if a router is added or removed from the network the algorithm can be run and find the new paths. The algorithm functions as intended, but it could be improved upon. The time efficiency of the algorithm is O(n^2) + O(Recursion). With the small network used in this program, the time efficiency is not a problem. However, with a much larger network, the current algorithm would cause stalling with current run times. The inefficient run time would also cause lag when trying to update in real-time.