CG-Pathfinder

Team Members –

- Raj Singh (1804109)
- Saurav Uppoor (1804119)
- Ritwik Vaidya (1804120)

Project Description –

This project, CG-Pathfinder (Dijkstra's Visualiser), is a visual representation of how Dijkstra's renown shortest path algorithm works behind the scene. The input for the project is a graph that could symbolise any real – life scenario (for e.g. in our project we have set the perspective of graph as cities and the edge weight as the traffic density). We also input the source node where we'd like the algorithm to find the shortest path to all the other nodes and in the end display the path to the destination node. This algorithm has various applications like shortest path finding, cycle detection in a graph etc.

User defined functions –

- `table(int n, int a[], int b[]) :`
  This function helps to create table of the desired size that represents the following data:
  1) `d(v)`-the weight from the source node to the current node.
  2) `v` - the current node.
  3) `u` - predecessor.

- `nodeVal(int i) :`
  The `nodeVal` function is a utility function and is used to print onto the screen the node value of each node represented in the graph section. It has just one parameter and that is the node number that it needs to print. The colour of the text that'd get printed is black since in our graphs the nodes would we either white in colour or green.

- `edgeVal(int I, int j) :`
  The edgeVal function is similar to nodeVal function except it prints out the edge weight between 2 nodes. Thus, it has 2 parameters that is the `(u,v)` edge belonging to graph `G` that is entered. Default colour of the edge weight is white.

- `sourceDestination(int source, int dest) :`
  This function is used to print the source and destination when both are available and is printed in the printing section of the output page. It has 2 parameter – `source` and `dest` which is pretty self-explanatory. The main objective of this function is just with the designing aspect.

- `source(int source) :`
  This function is similar to `sourceDestination` except this just prints out the source that it receives from the parameter. The main objective of this function too is the designing aspect of the text.

- `boxes() :`
  This function was used to give structure to the output page and make it presentable. Rectangle shapes are used to differentiate the three sections namely: Graph section, Table section and Printing section. This function makes it easy for the user to separately look into different

sections and analyse accordingly. 2/3rd of the window 640x480 is used for graph and table section and for the printing section 1/3rd of the vertical length is used.

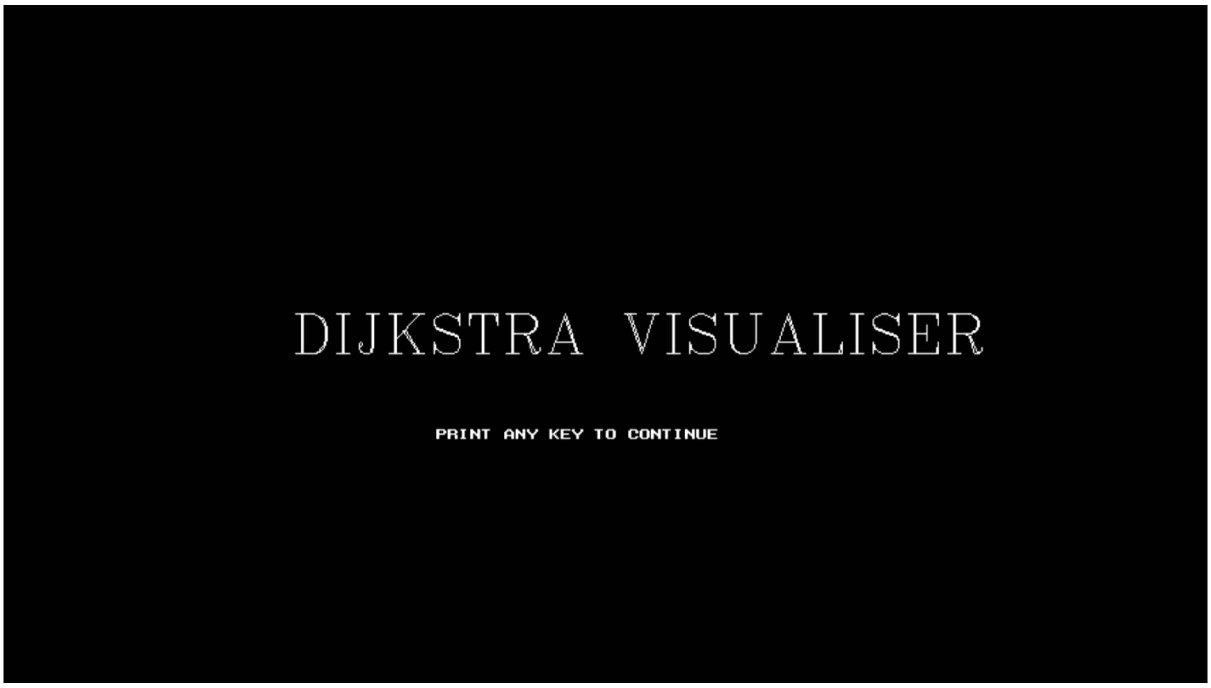
- `processing(int u, int v, int status) :`
  This function is used to while the Dijkstra's algorithm is running. It highlights the edge that is currently being processed. Now there 2 cases possible for the edges – 1. The edge is being processed (which will occur for all the edges) 2. The edge is relaxed i.e. the distance of the second edge from first edge has been changed/decreased. So, this is highlighted by the status parameter. If the edge gets relaxed a message with Green coloured text gets printed in the printing section.

- `printPath(int path[], int pp, int length) :`
  This function is called when the Dijkstra's algorithm has been executed and the path is to be printed. The path is saved in the path array. Now for representation purpose we are highlighting the path in the graph one edge at a time. Thus, pp parameter is used to keep track of that. The length parameter is actually the size of the array path.

Inbuilt functions –

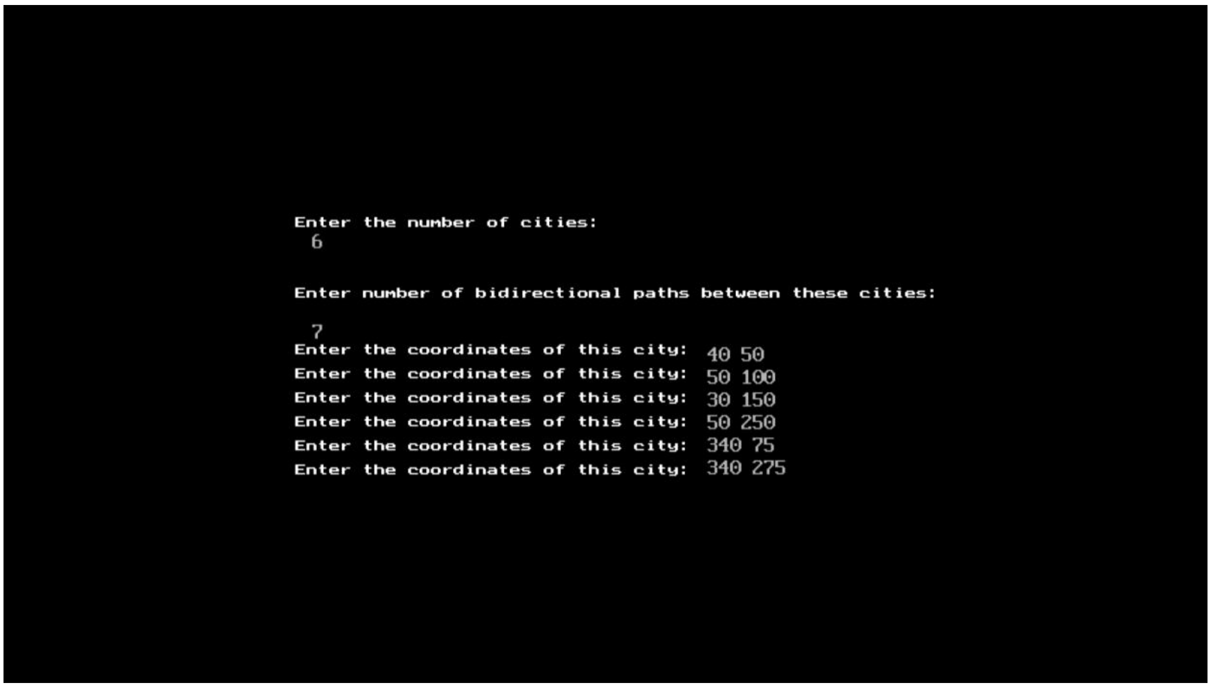
- rectangle(int left, int top, int right, int bottom)
- line(int x1, int y1, int x2, int y2)
- circle(int x, int y, int radius)
- outtextxy(int x, int y, char *string)
- gotoxy(int x, int y)
- setcolor(int color)
- settextstyle(int font, int direction, int charsize)
- setlinestyle(int linestyle, unsigned pattern, int thickness)
- textcolor(int color)
- floodfill(int x, int y, int border)
- setfillstyle(int pattern, int color)

Working with Screenshots –



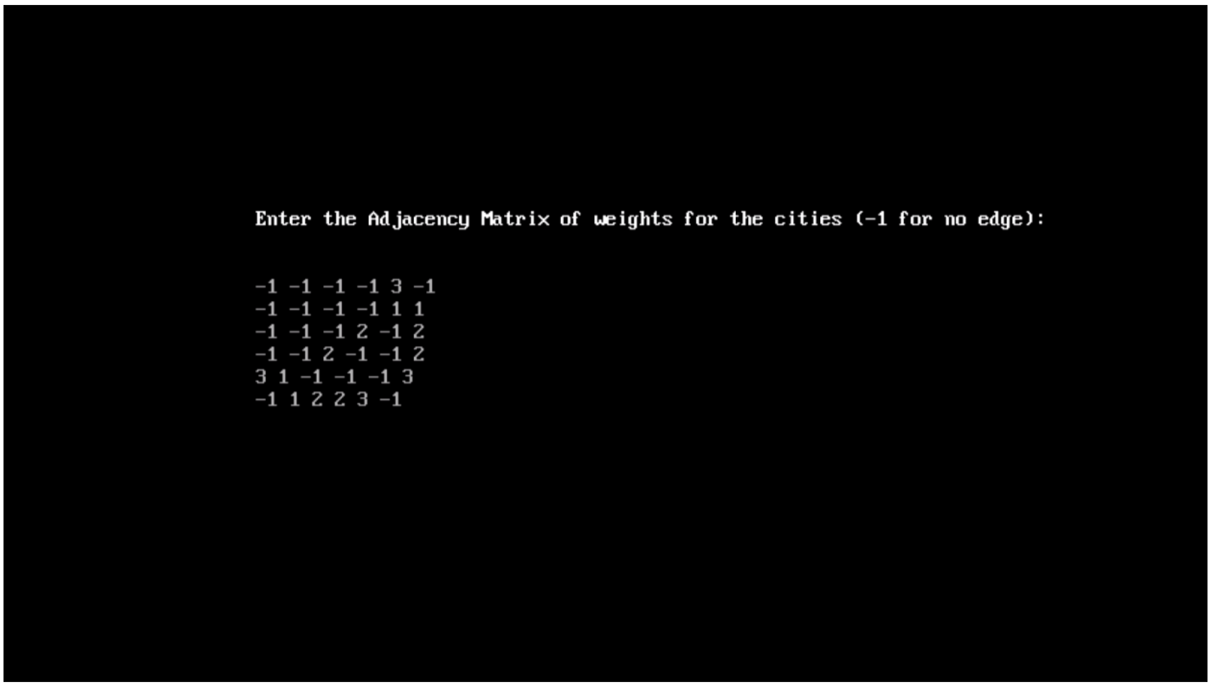
```
DIJKSTRA VISUALISER

PRINT ANY KEY TO CONTINUE
```

The first page displaying the title of the project.



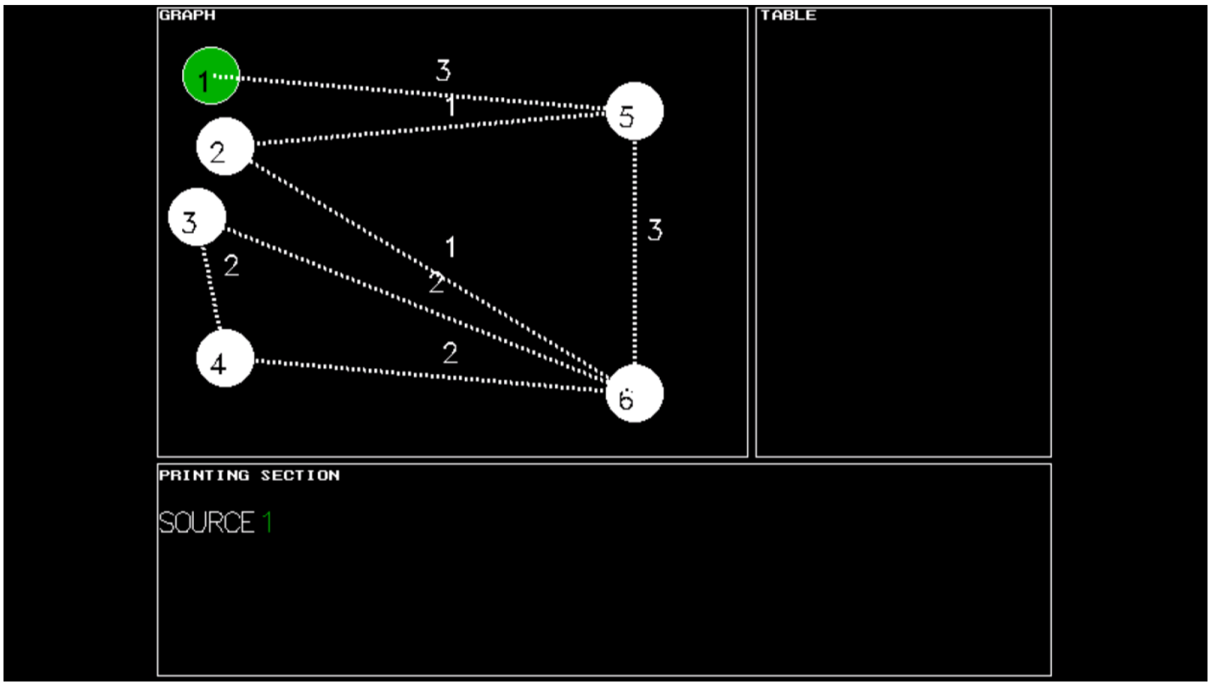
```
Enter the number of cities:
  6

Enter number of bidirectional paths between these cities:

  7
Enter the coordinates of this city:  40 50
Enter the coordinates of this city:  50 100
Enter the coordinates of this city:  30 150
Enter the coordinates of this city:  50 250
Enter the coordinates of this city:  340 75
Enter the coordinates of this city:  340 275
```

Next, we take input from the user regarding the graph ie, the number of the cities(nodes) and bidirectional path (representing the bidirectional edges of a graph). Then we also take input, the coordinate of each city so that we could plot the nodes and the paths.



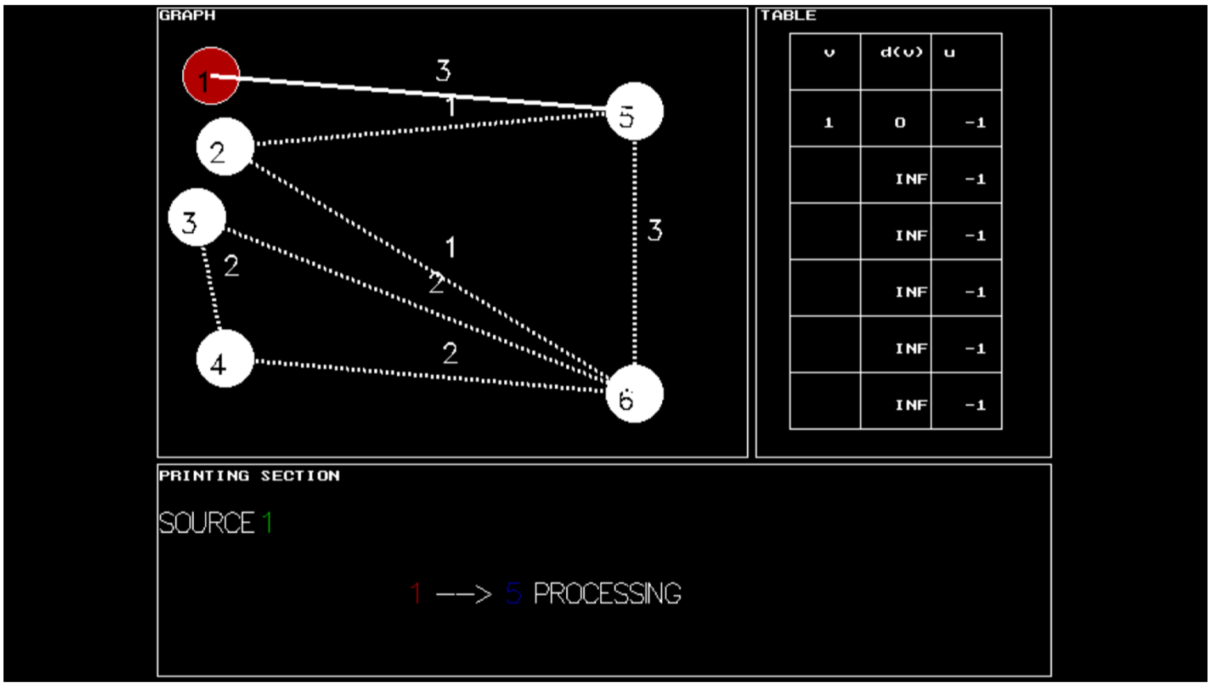
```
Enter the Adjacency Matrix of weights for the cities (-1 for no edge):

-1 -1 -1 -1 3 -1
-1 -1 -1 -1 1 1
-1 -1 -1 2 -1 2
-1 -1 2 -1 -1 2
3 1 -1 -1 -1 3
-1 1 2 2 3 -1
```

Then we proceed to get the traffic density between cities (representing the edge weights in a graph) using the adjacency matrix structure. In case there is no road between 2 cities it is represented using -1.



```
SOURCE 1
```

After taking all the input, we display the entered graph by the user. Highlighting the source node with green colour. You might notice that we have divided the page into 3 sections namely – Graph, Table, and the Printing section. Graph section is were we will represent the visual effects and animations on the graph. Table is were we display the memory status of nodes' properties ie distance and predecessor. And the printing section is were we display which process is currently going on. As of now, in printing section all we display is the information about the source node.

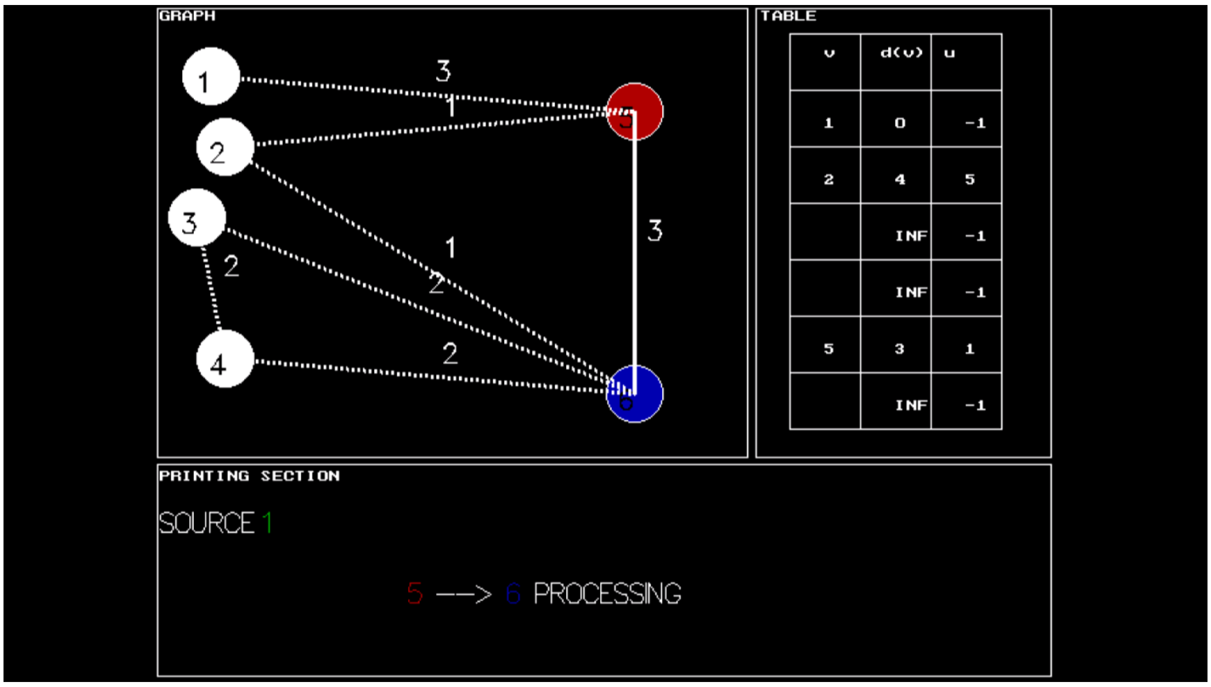


v	d(v)	u
1	0	-1
	INF	-1
	INF	-1
	INF	-1
	INF	-1
	INF	-1

SOURCE 1

1 —> 5 PROCESSING

This is when the Dijkstra’s algorithm finally begins. The table section has been updated to show the values in distance and the predecessor array. The printing section displays which edge is currently being processed.

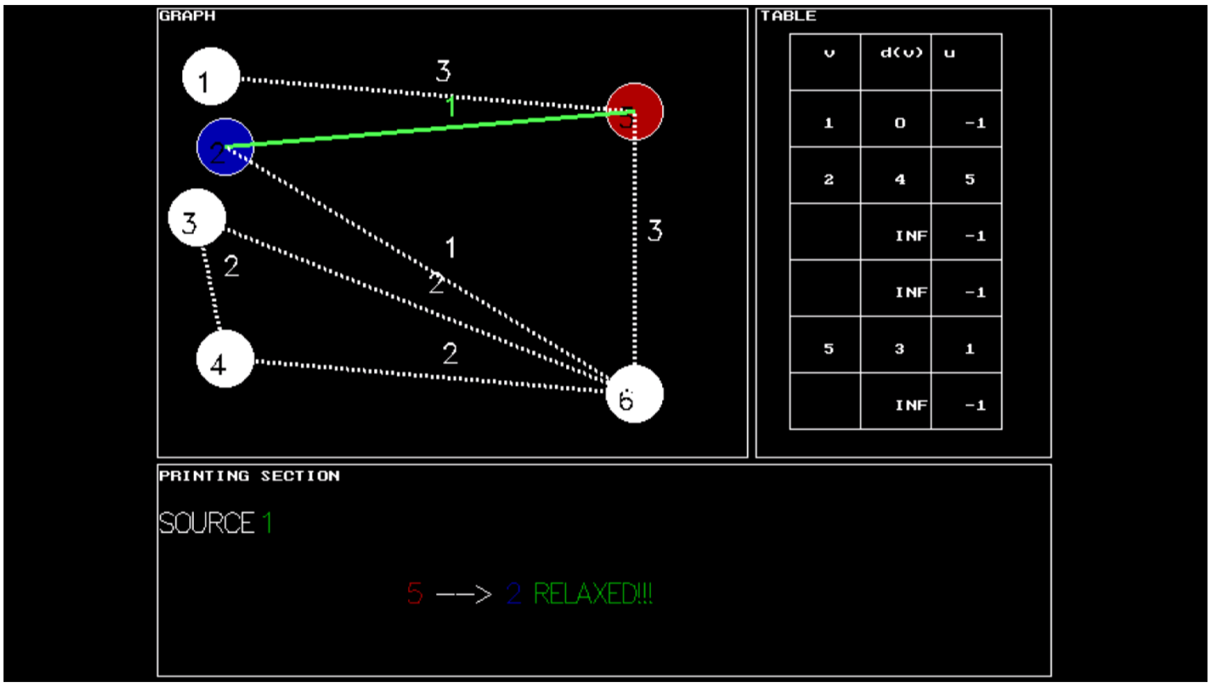


v	d(v)	u
1	0	-1
2	4	5
	INF	-1
	INF	-1
5	3	1
	INF	-1

SOURCE 1

5 —> 6 PROCESSING

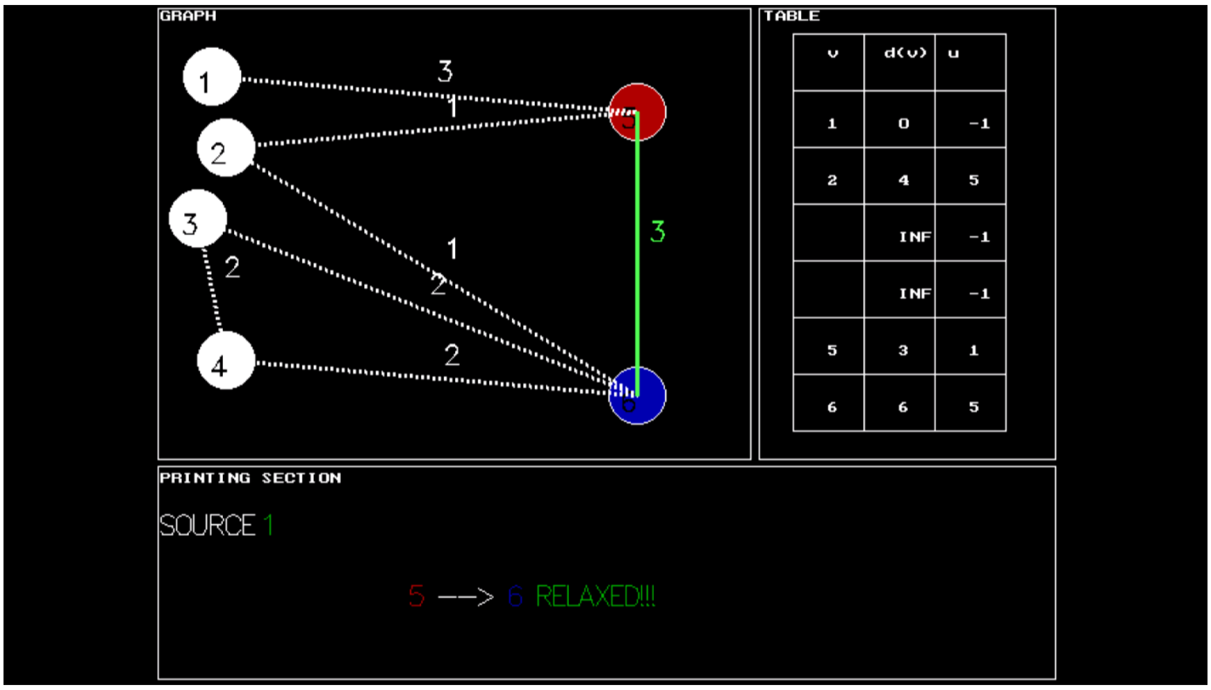
This is image shows an edge getting processed. The red colour signifies that it is the first node in edge evaluation and blue signifies the latter one. This is useful since the blue coloured node is the one which, if the edge gets relaxed, would be changing its distance value. NOTE! In case you are running this algorithm, don't forget to keep hitting any key on the keyboard since for user's convenience we have added getch() before each edge evaluation procedure.



v	d(v)	u
1	0	-1
2	4	5
	INF	-1
	INF	-1
5	3	1
	INF	-1

SOURCE 1

5 ---> 2 RELAXED!!!

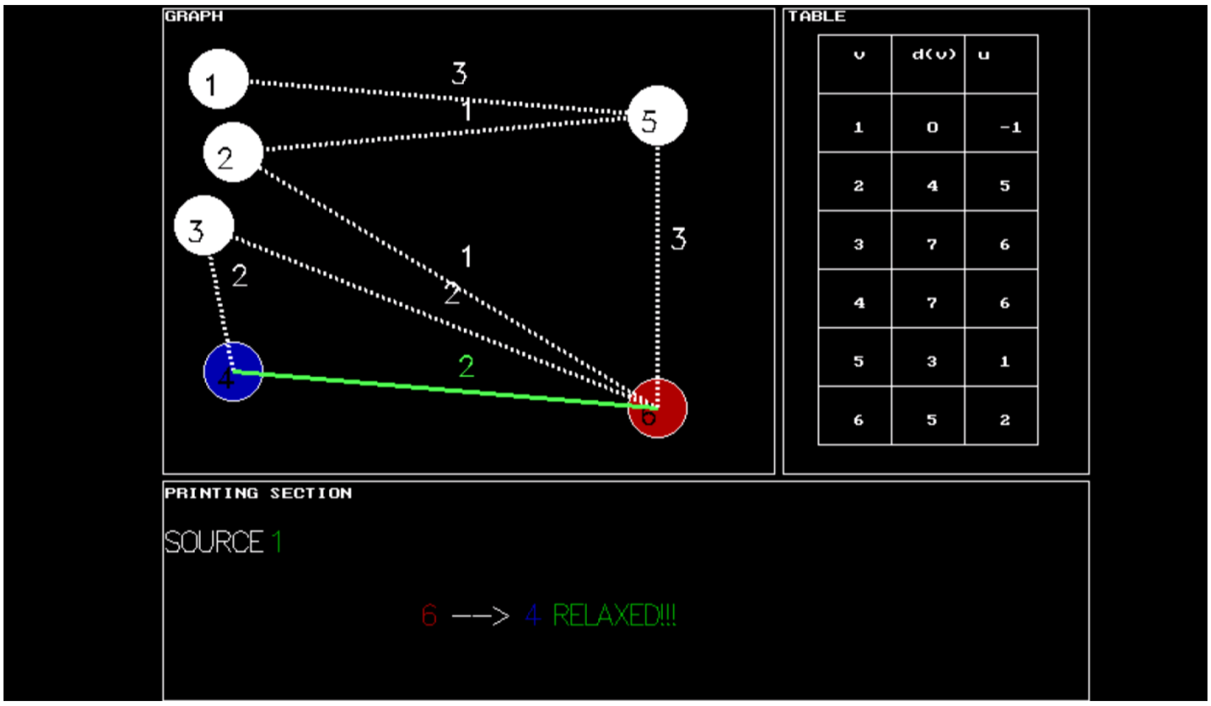


v	d(v)	u
1	0	-1
2	4	5
	INF	-1
	INF	-1
5	3	1
6	6	5

SOURCE 1

5 ---> 6 RELAXED!!!

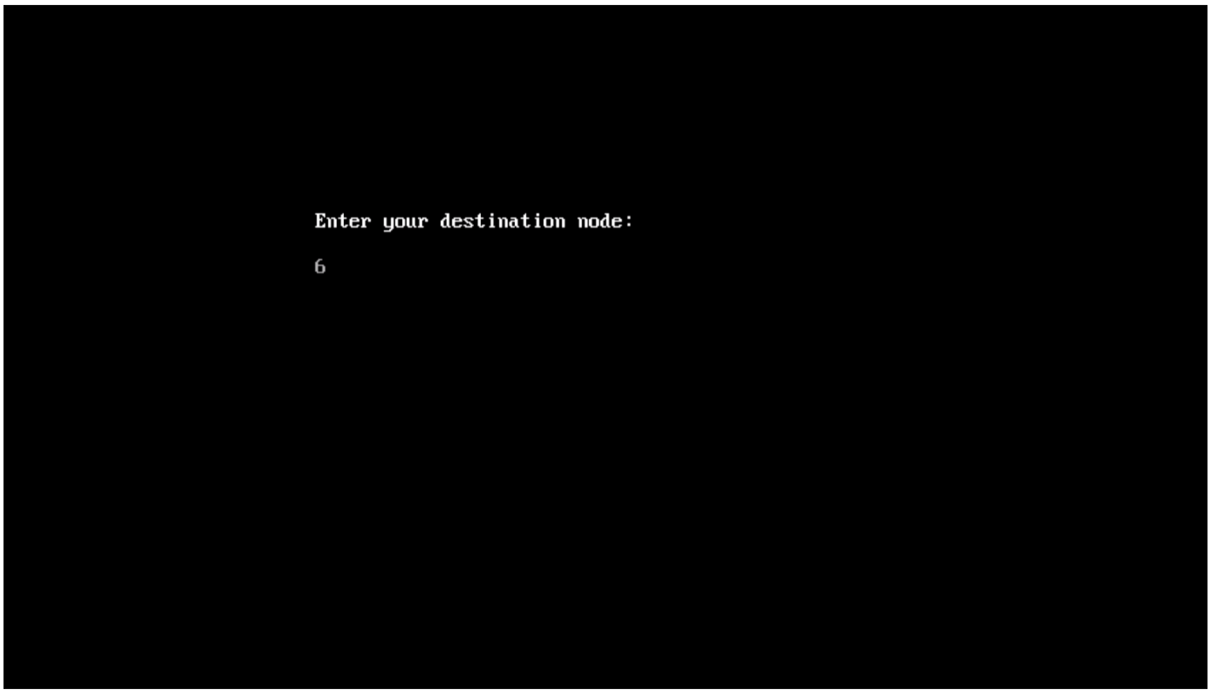
Both the above images show the edge getting relaxed, i.e. when the distance of the blue edge is reduced to sum of distance of red edge and edge weight of the current edge. It gets reflected on the table on the right as well.



v	d(v)	u
1	0	-1
2	4	5
3	7	6
4	7	6
5	3	1
6	5	2

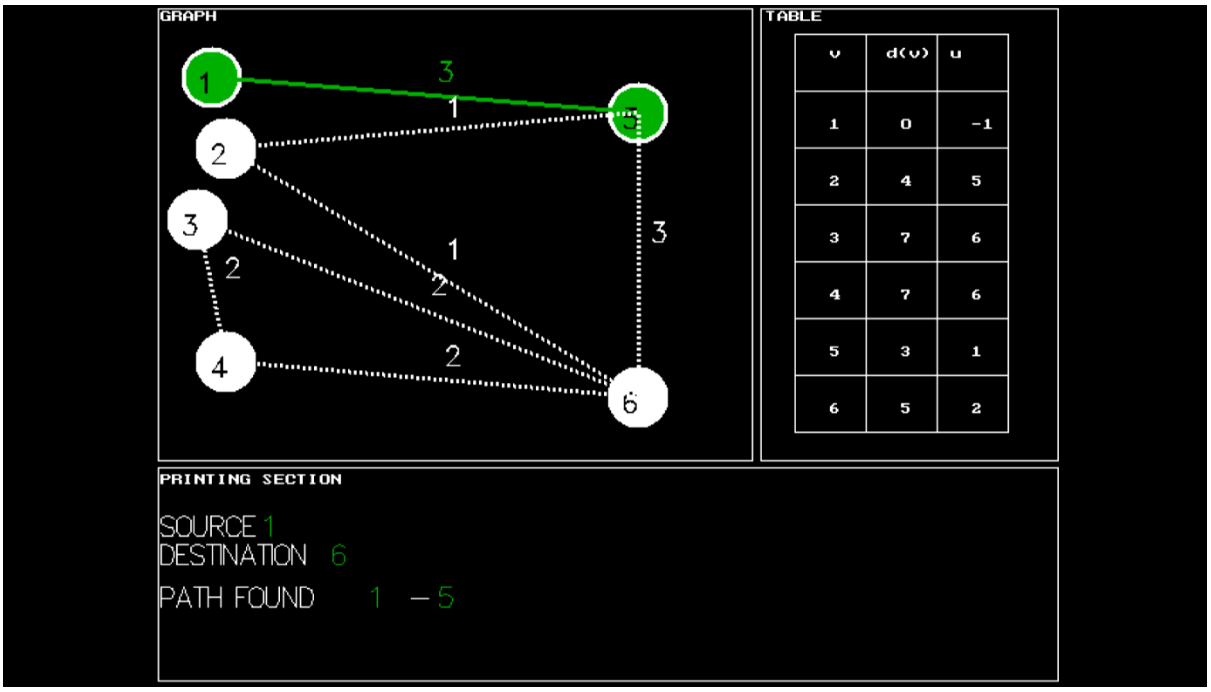
```
SOURCE 1

6 ---> 4 RELAXED!!!
```

This is the when all the edges have been evaluated and the Dijkstra's algorithm has been completed. Now we have the shortest distance from the source node to all the other nodes. It can be seen in the d(v) column of the table.



```
Enter your destination node:
6
```

Taking input from the user about the destination city(node).

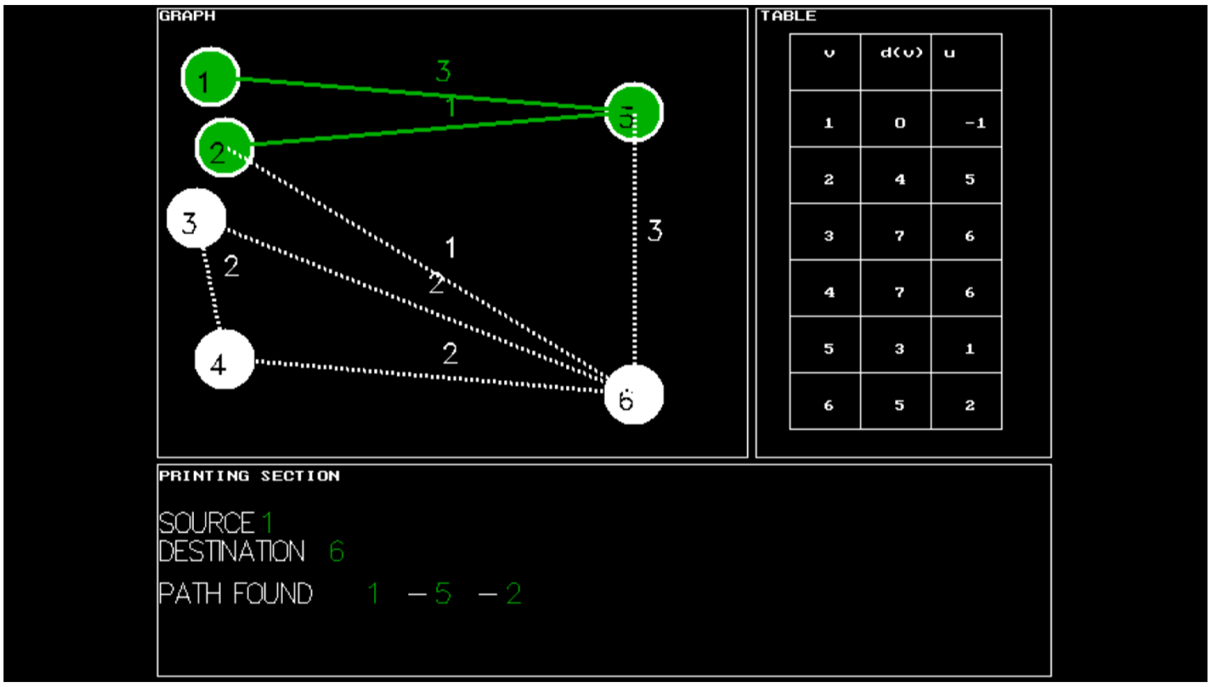


TABLE

v	d(v)	u
1	0	-1
2	4	5
3	7	6
4	7	6
5	3	1
6	5	2

PRINTING SECTION

SOURCE 1
DESTINATION 6
PATH FOUND 1 – 5



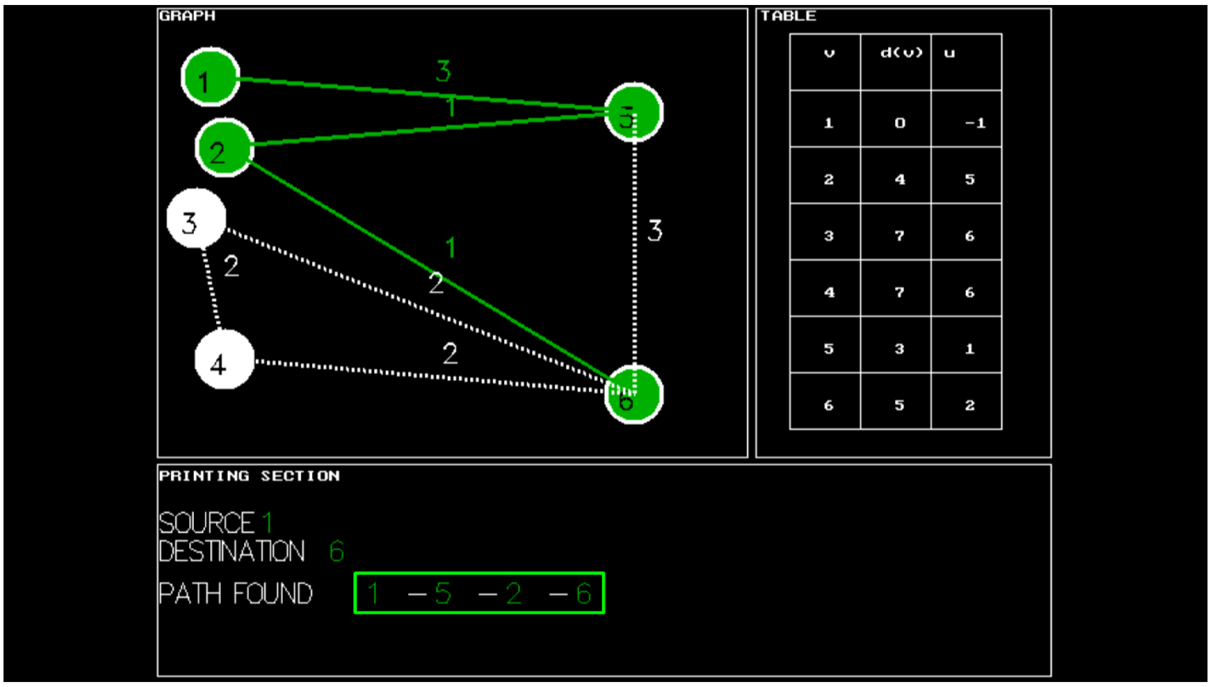
TABLE

v	d(v)	u
1	0	-1
2	4	5
3	7	6
4	7	6
5	3	1
6	5	2

PRINTING SECTION

SOURCE 1
DESTINATION 6
PATH FOUND 1 – 5 – 2

The above screenshots are of path printing stage. For convenience, we have added `getch()` before each edge highlighting process. We also display the path currently followed in the printing section.



TABLE

v	d(v)	u
1	0	-1
2	4	5
3	7	6
4	7	6
5	3	1
6	5	2

PRINTING SECTION

```
SOURCE 1
DESTINATION  6
PATH FOUND    1  – 5  – 2  – 6
```

The final path to be followed has been printed and highlighted and this is the shortest path possible to reach destination city from the source city!