Project-2

(CSE-220-Data Structure and Algorithm)

Title: *Solving a Maze-Game, using Graph-Theory, and various Traversal Algorithms.*

Made By:
Ankit Vadehra(12BCE0282)
Poorva Arora(12BCE0277)
Rohan Kumar(12BCE0622)

Abstract:

Implementing a game called "**The Batman and Joker**", based on the traditional maze game. It uses Graph Theory, an Adjacency Matrix and Traversal algorithm according to choice, to traverse and find a path between the source-and-destination entered by the user .

Constrution:

- We ask user the number of runs for the program, and then ask for the map by asking for 1's and and 0's, where every 1=Path_Available; and; 0=Obstruction.
- Two players, The Batman (B), and The Joker(J), compete on a fixed, finite undirected graph H.
- First, (B)-The Batman starts by placing himself at a node of his choice; then (J)-The Joker does the same. After that, the graph(adjacency_list) is passed to the BFS function to calculate a proper path through which the **B** can reach **J**.

Code:

1) Finding number of cells passed using BFS.

```
#define PB push_back
#define SZ size()

#include <vector>
#include <iostream>
#include <utility>
#include <stdlib.h>
#include <string>
using namespace std;

template <typename T>
void graph_from_maze(int cols, int rows, std::vector<T> &maze, T obstacle, std::vector<i
{
    for (int y = 0; y < rows; ++y)
      {  for (int x = 0; x < cols; ++x)
        {
            int pos = x + y*cols, up = pos - cols, down = pos + cols, left = pos -1, rig
            if(x != 0 && (maze[left] != obstacle) && (maze[pos] != obstacle))
                adjacency_list[pos].push_back(left);
            if( y != 0 && (maze[up] != obstacle) && (maze[pos] != obstacle))
                adjacency_list[pos].push_back(up);
            if( x < (cols-1) && (maze[right] != obstacle) && (maze[pos] != obstacle))
                adjacency_list[pos].push_back(right);
            if(y < (rows-1) && (maze[down] != obstacle) && (maze[pos] != obstacle))
                adjacency_list[pos].push_back(down);
        }

      }


}
using namespace std;

int N;                  // number of nodes
vector<int> A[10001];   // adjacency lists

int BFS(int s, int t) //BFS to find route..
{ // distance between s and t
  if (s == t)
  {
    cout<<"\n Same Position Occupied by Start and End cell.";
    return 0;
  }

  int l[10001], k = 0;  //initialize queue max condition 1001, 100x100.
  vector<int> v(N, -1); // initialize distances(vector-array)->each value for N is=-1, i
  l[k++] = s;           // Add to the queue; added then k incremented.

  v[s] = 0;             // distance to s is 0
  for (int i = 0; i < k; i++)
    // for all adjacent nodes l [i]
    for (int j = 0; j < (int)A[l[i]].SZ; j++)
      // if the neighbor has not been visited
      if (v[A[l[i]][j]] < 0)
      {
        // update its distance
```

```
        v[A[l[i]][j]] = v[l[i]] + 1;
        // find t and, return result
    if (A[l[i]][j] == t) return v[t];
        // added to queue
    l[k++] = A[l[i]][j];
      }
    return -1;
}

int main()
{cout<<"\n \t\t\tWelcome to Batman-and-Joker..\n";
cout<<" \t\t\t\t  LOADING..\n\n";
//LOADING BAR, JUST FOR THE PRETTY-NESS ;)
float progress = 0.0;
while (progress < 1.0)
{
    int barWidth = 50;
    cout << "\t\t[";
    int pos = barWidth * progress;
    for (int i = 0; i < barWidth; ++i) {
        if (i < pos) std::cout << "|";
        else if (i == pos) std::cout << ">";
        else std::cout << " ";
    }
    cout << "] " << int(progress * 100.0) << " %%\r";
    cout.flush();
    progress += 0.00003; // for demonstration only
}
system("clear");
cout<<"\n \t\t\tWelcome to Batman-and-Joker..\n\n\n";
cout<<"    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::::::::::::::-'     `-::::::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::::::::::::-'          `::::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::::::::::-  '   /(_M_)\\  `::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::::::::-'        |      |  ::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::::::-         .   \\|/~V~\\|/  ,::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::::-'              .          ,::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::::-'                 `-.    .-::::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::::-'                  _,,-::::::::::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::::-'               _,--::::::::::::::::::::::::::::\n";
cout<<"    ::::::::::::::::::::::::::::-'             _.--:::::::::::::::::::::::#####:::::::::\n";
cout<<"    ::::::::::::::::::::::::::-'          _.--:::::::::::::::::::::::::::#####:::::####\n";
cout<<"    ::::::::::::::::::::::::-'    ##     ###.-:::::###:::::::::::::::::::::#####:::::####\n";
cout<<"    ::::::::::::::::::::::-'      ###_.::######::::###:::::::::::::::#####:##########:::####\n";
cout<<"    ::::::::::::::::::::-'        .:###::########::::###:::::::::::::::#####:##########:::####\n";
cout<<"    ::::::::::::::::::-'      ...--:::###::########::::###::::######:::#####:##########:::####\n";
cout<<"    ::::::::::::::::-'     _.--:::##:::###:########::::###::::######:::#####:################\n";
cout<<"    ::::::::::::::-'       _.--:::##:::###:########::::###::::######:::#####:################\n";
cout<<"    ::::::::::::-'         _.--:::##:::###:########::::###::::######:::#####:################\n";
cout<<"    ::::::::::-'           '#########:::###:########::########::######:::#####:################\n";
cout<<"    ::::::::-'             :#########:::#############::########::######:::#####################\n";
cout<<"    ::::::-'               ##########:::#########################################################\n";
cout<<"    ::::-'                 ##########:::#########################################################\n";
cout<<"    ::-'                   ##########:::#########################################################\n";
cout<<"    -'                     ################### BATMAN ## AND ## THE ## JOKER ####################\n";
cout<<"                           ##########################################################################\n";
cout<<"                           ### BY: ###### 12BCE0282 ## 12BCE0277 ## 12BCE0622#####################\n";
cout<<"                           ##########################################################################\n";

cout<<"\n\n\t\t Preparing Format:";
int choice;
```

```
cout<<"\nPlease enter the number correspoding to your choice:\n1.To Enter Game\
      t2.To Exit";
cin>>choice;
switch(choice)
{
  case 1: cout<<" \t\t\tWelcome to Batman-and-Joker..\n\n\n";
          system("clear");
          break;
  case 2: exit(0);
          break;
  default: cout<<"\n Please enter a correct number :/";

}

cout<<"\n How many times do you want to run the program ??";
  int runs;
  cin >> runs;
  for(int run = 0; run < runs; run++)
  {
    int col, row;
    vector<int> map;
    cout<<"\nPlease enter the number of Rows in the Matrix\n";
    cin>>row;
    cout<<"\nPlease enter the number of Columns in the Matrix\n";
    cin >> col;
    N = col*row;//adjaceny list..
    int display_array[row][col];
    int display_array2[row][col];
    int number=0;
    for(int i=0;i<row;i++)
    {
      for(int j=0; j<col; j++)
        display_array[j][i]=number++;
    }

    cout<<"\nHence the matrix is numbered as:\n";
    for(int a=0;a<row;a++)
    {
      for(int b=0; b<col; b++)
        {cout<<display_array[a][b]<<"\t";}
      cout<<"\n\n";
    }
    cout<<"\nPlease enter 1||0, respetively for an Obstacle or an Empty cell:";
    cout<<"\n0=Empty Space..1=Obstacle in path";
    cout<<"\nSpecified Format: \nFor a 2x2 matrix enter y y y y; where y=0||1\n";
    for (int y = 0; y < row; ++y)
    {
      for (int x = 0; x < col; ++x)
      {
        int cell;
        cin >> cell;
        map.PB(cell);
        display_array2[y][x]=cell;
      }
    }
    cout<<"\nEnter Batman's position from 0 to "<<N-1<<" ";
    int start, end;
    //int coordX, coordY;
    //cin >> coordX >> coordY;
    cin>>start; //= coordX + coordY*col ;
    cout<<"\nEnter The Jokers position from 0 to "<<N-1<<" ";
    //cin >> coordX >> coordY;
```

```
    cin>>end; //= coordX + coordY * col;

    //Character array so 0 and position of 1 can be shown..
    //Making Map
    //To Add B & J..
    char made_array[row][col];
    for(int q=0;q<row;q++)
    {
      for(int w=0; w<col;w++)
      {
        if(display_array[q][w]==start)
        made_array[q][w]='B';
      else
        if(display_array[q][w]==end)
          made_array[q][w]='J';
      else
        if(display_array2[q][w]==0)
        made_array[q][w]='-';
      else
        if(display_array2[q][w]==1)
          made_array[q][w]='x';
      }

    }

    //Printing maze:
    cout<<"\n Hence the Maze Created Is:\n";
    for(int a=0;a<row;a++)
    {
      for(int b=0;b<col;b++)
      {
        cout<<made_array[a][b]<<" ";

      }
      cout<<"\n";
    }

    graph_from_maze<int>(col, row, map, 1, A);
    cout<<"\n Hence by BFS the amount of cells Batman has to traverse is:\n";
    cout << BFS(start, end) + 1 << endl;
    for (int i = 0; i < 10001; ++i)
    {
      A[i].clear();
    }

  }
}
```

Working:

The program finally passes out the number of cells **B** has to traverse to reach **J**. Since the code uses Breadth_First_Search, we can only find the number of cells, and if there is a path. It is not sure that the path chosen would be the smalles path or a larger path since BFS has better complexity than Dijikstras or other variants of MST finding the smallest

path, hence e use Dijikstras as a small only_code variant to show that it could also be included in the code to also find the smallest path an then to print the path it has to trverse to reach source to destination.

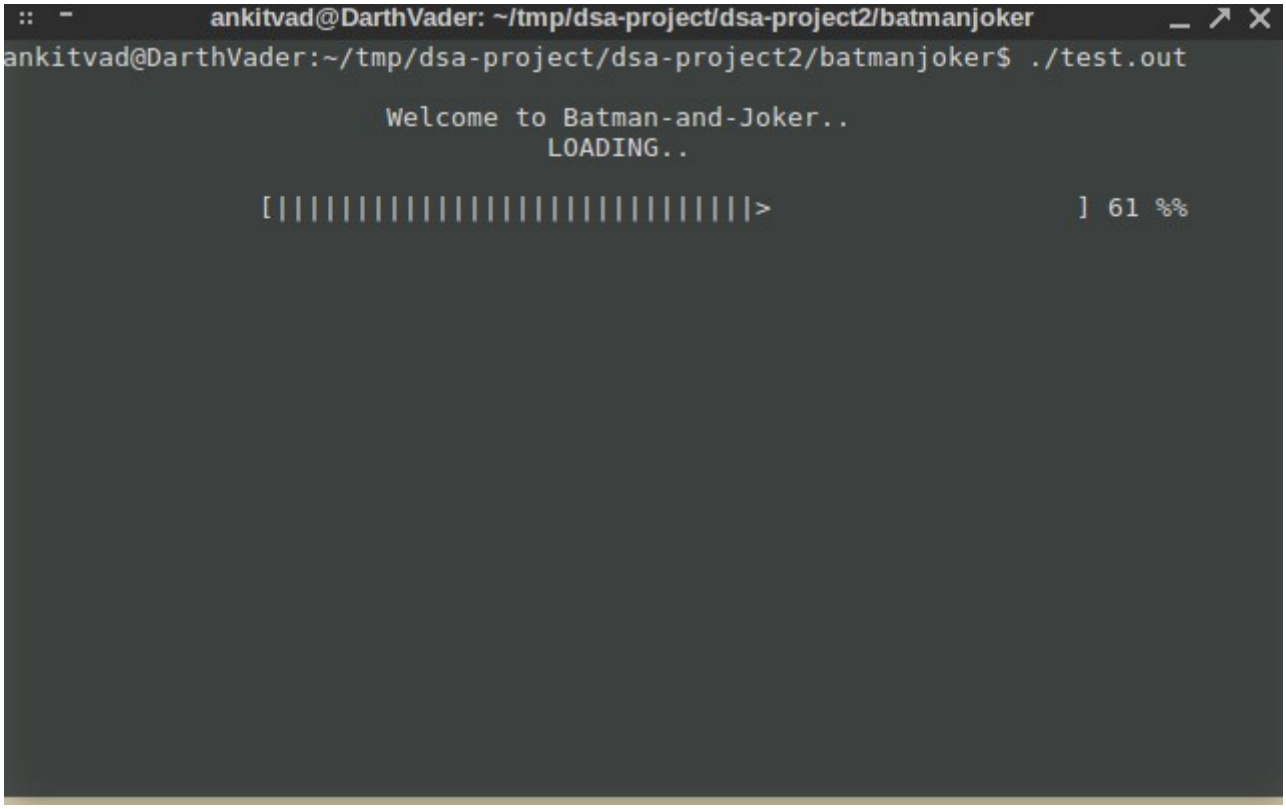
Dijikstras Algorithm:

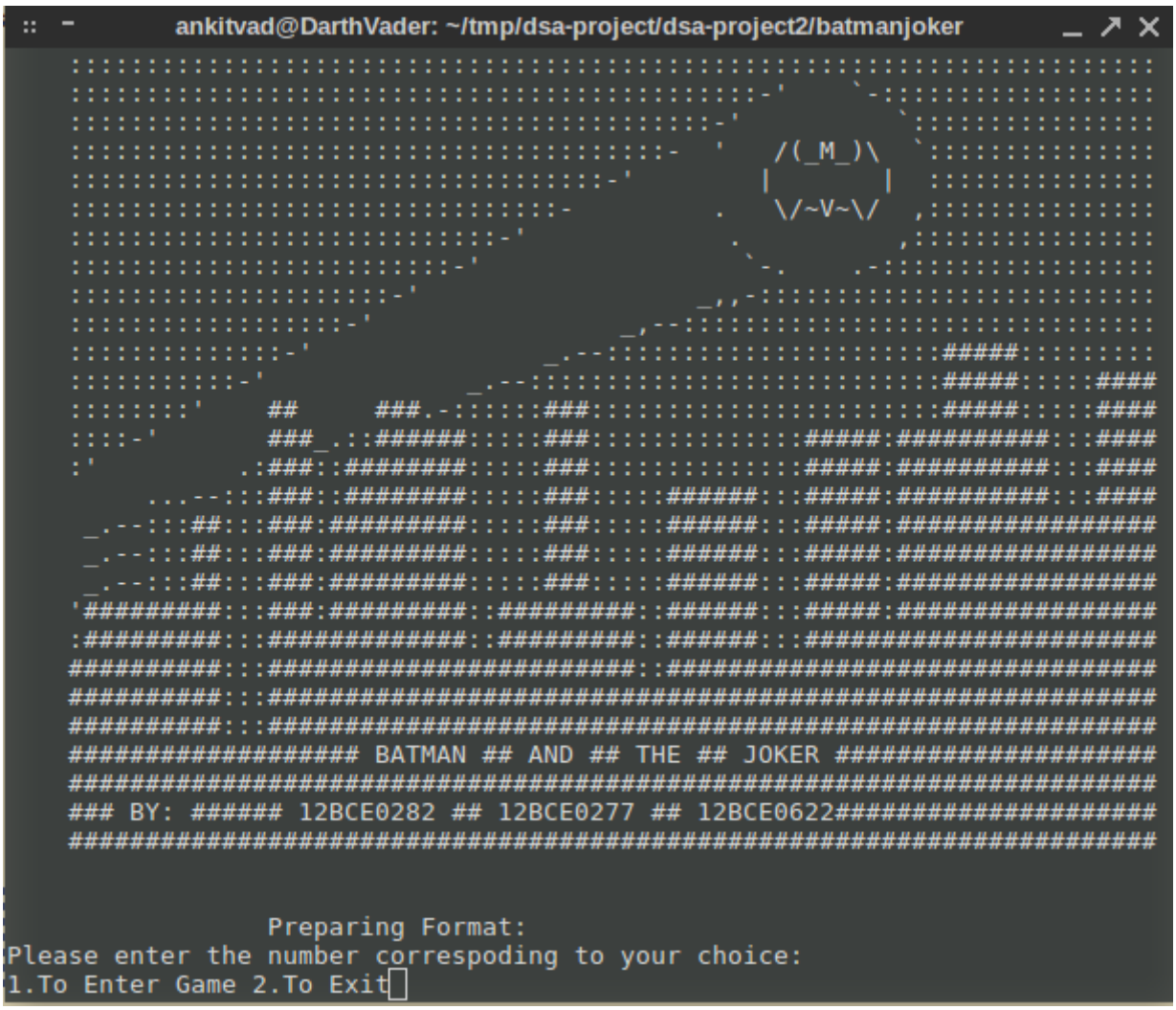
```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>
#include <queue>
#include <iterator>
using namespace std;

typedef vector<vector<pair<int,float> > > Graph;//So that the grph can store 2 numbers c
class Comparator
{
public:
 int operator() ( const pair<int,float>& p1, const pair<int,float> &p2)
 {
 return p1.second>p2.second;
 }
};

void dijkstra(const Graph &G,const int &source,const int &destination,vector<int> &path)
{
vector<float> d(G.size());
vector<int> parent(G.size());
for(unsigned int i = 0 ;i < G.size(); i++)
{
 d[i] = std::numeric_limits<float>::max();
 parent[i] = -1;
}
priority_queue<pair<int,float>, vector<pair<int,float> >, Comparator> Q;
d[source] = 0.0f;
Q.push(make_pair(source,d[source]));
while(!Q.empty())
{
 int u = Q.top().first;
 if(u==destination) break;
 Q.pop();
 for(unsigned int i=0; i < G[u].size(); i++)
 {
  int v= G[u][i].first;
  float w = G[u][i].second;
  if(d[v] > d[u]+w)
  {
   d[v] = d[u]+w;
   parent[v] = u;
   Q.push(make_pair(v,d[v]));
  }
 }
}
path.clear();
int p = destination;
path.push_back(destination);
```

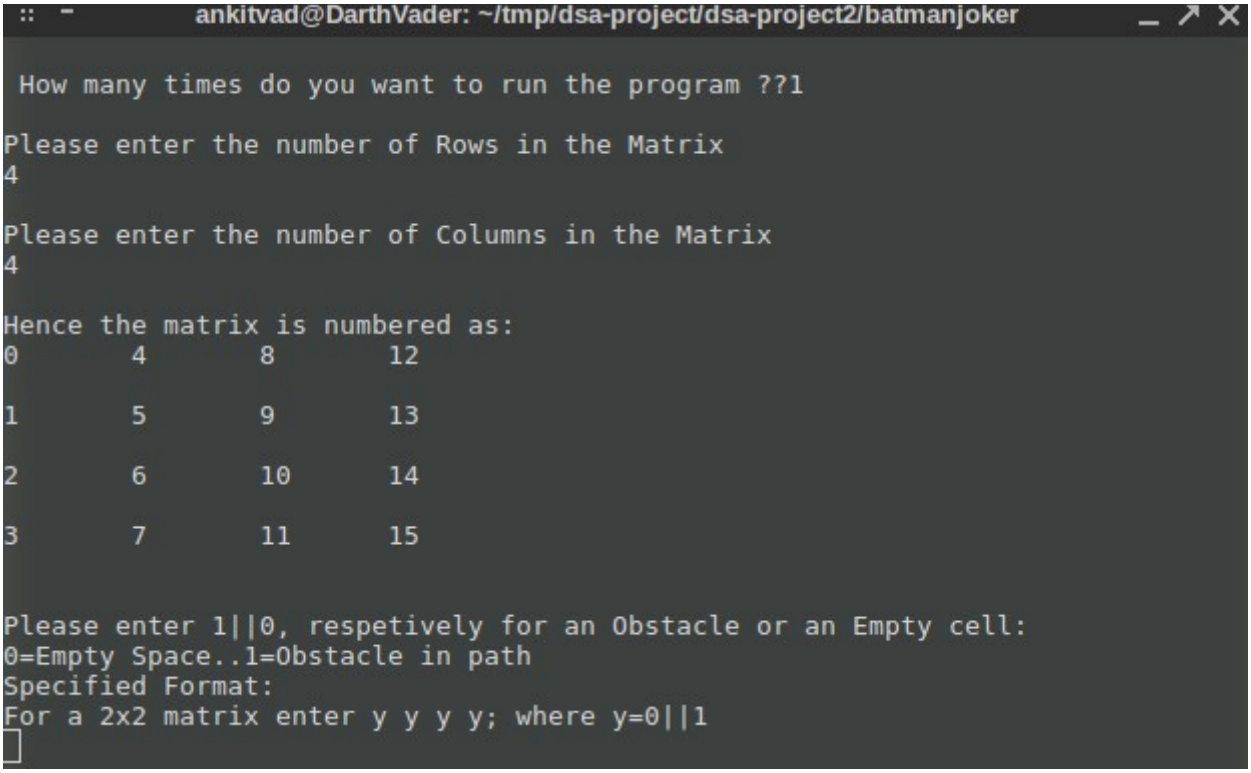
```
while(p!=source)
{
 p = parent[p];
 path.push_back(p);
}
}

int main()
{
    /* Graph
GRAPH TYPE = UNDIRECTED
NUMBER OF VERTICES = 6 indexed from 0 to 5
NUMBER OF EDGES = 9
edge 0->1 weight = 7
edge 0->2 weight = 9
edge 0->5 weight = 14
edge 1->2 weight = 10
edge 1->3 weight = 15
edge 2->5 weight = 2
edge 2->3 weight = 11
edge 3->4 weight = 6
edge 4->5 weight = 9
*/
    Graph g;
    int node;
    int start;
    int end;
    cout<<"\nPlease Insert the number of nodes..\n";
    cin>>node;
    g.resize(node);

    for(int i=0;i<node;i++)
    { int connecting_node;
      int weight;
      int connections;

      cout<<"\nEnter details for node "<<i+1;
      cout<<"\nHow many connections does it have: ";
      cin>>connections;
      for(int j=0;j<connections;j++)
      {
        cout<<"\nEnter details for "<<j+1<<" conection";
        cout<<"\nIt is connected to node: ";
        cin>>connecting_node;
        cout<<"\nWhat is the weight:";
        cin>>weight;

        g[i].push_back(make_pair(connecting_node-1,weight));
        g[connecting_node-1].push_back(make_pair(i,weight));

      }
    }

    /*
    g[0].push_back(make_pair(1,7));
    g[1].push_back(make_pair(0,7));

    g[0].push_back(make_pair(2,9));
    g[2].push_back(make_pair(0,9));

    g[0].push_back(make_pair(5,14));
    g[5].push_back(make_pair(0,14));
```

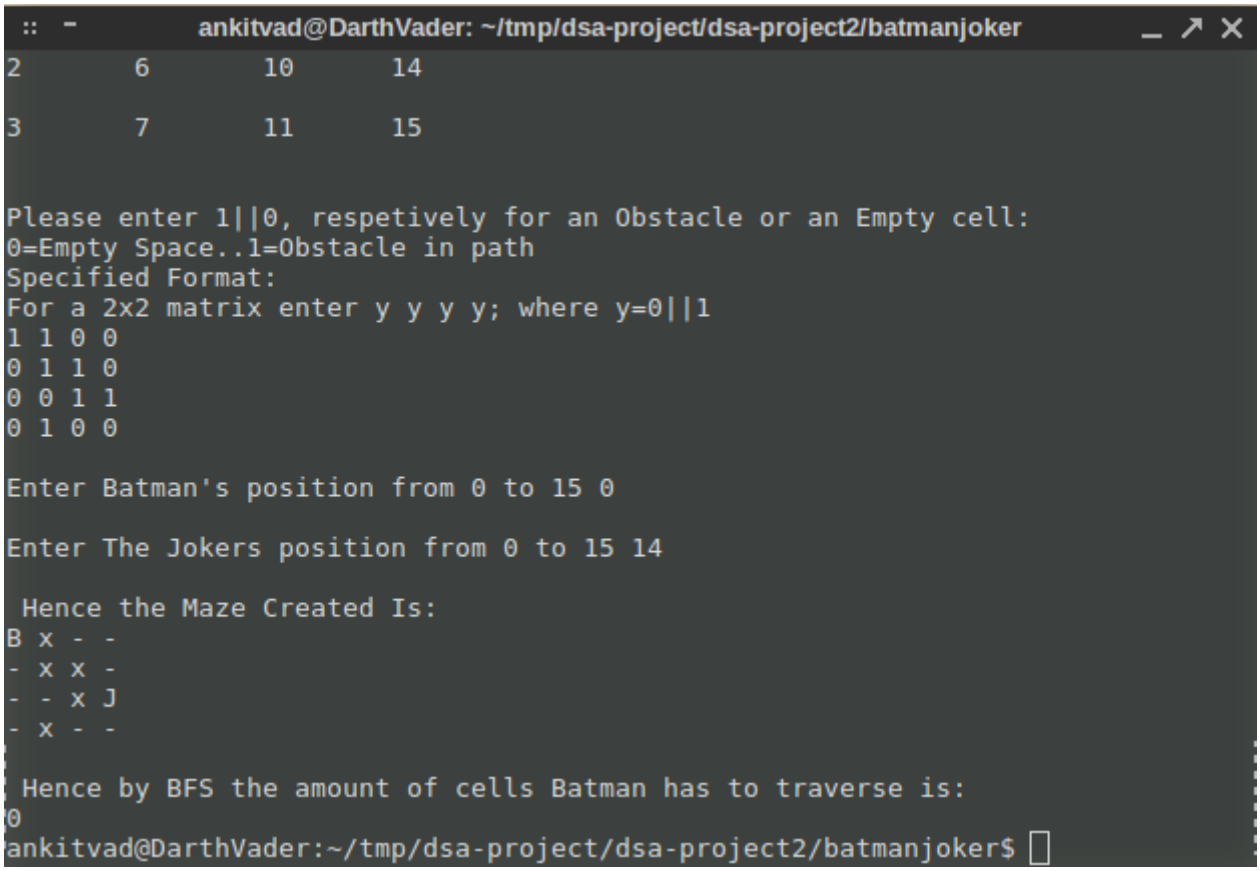
```
    g[1].push_back(make_pair(2,10));
    g[2].push_back(make_pair(1,10));
    g[1].push_back(make_pair(3,15));
    g[3].push_back(make_pair(1,15));
    g[2].push_back(make_pair(5,2));
    g[5].push_back(make_pair(2,2));
    g[2].push_back(make_pair(3,11));
    g[3].push_back(make_pair(2,11));
    g[3].push_back(make_pair(4,6));
    g[4].push_back(make_pair(3,6));
    g[4].push_back(make_pair(5,9));
    g[5].push_back(make_pair(4,9));
    */

    vector<int> path;//Matrix to store the shortest path..
    cout<<"\nPlease enter the start node : ";
    cin>>start;
    cout<<"\nPlease enter the end node : ";
    cin>>end;
    dijkstra(g,start-1,end-1,path);
    cout<<"\n Hence the shortest path is:\n";

    for(int i=path.size()-1;i>=0;i--)
        cout<<path[i]+1<<"->";
    return 0;
}
```

Outputs:

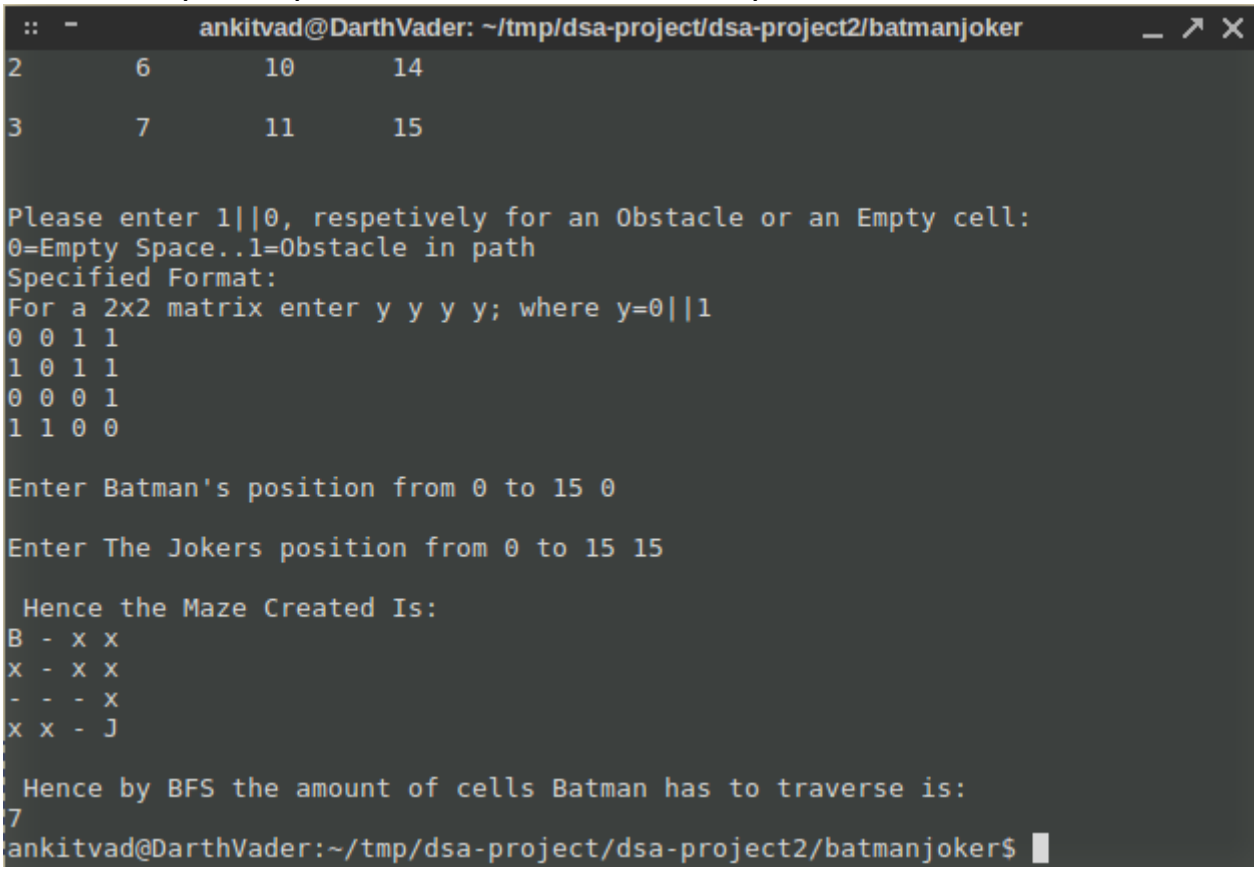



ankitvad@DarthVader: ~/tmp/dsa-project/dsa-project2/batmanjoker
BATMAN ## AND ## THE ## JOKER
BY: ###### 12BCE0282 ## 12BCE0277 ## 12BCE0622
Preparing Format:
Please enter the number correspoding to your choice:
1.To Enter Game 2.To Exit


ankitvad@DarthVader: ~/tmp/dsa-project/dsa-project2/batmanjoker
How many times do you want to run the program ??1
Please enter the number of Rows in the Matrix
4
Please enter the number of Columns in the Matrix
4
Hence the matrix is numbered as:
0 4 8 12
1 5 9 13
2 6 10 14
3 7 11 15
Please enter 1||0, respetively for an Obstacle or an Empty cell:
0=Empty Space..1=Obstacle in path
Specified Format:
For a 2x2 matrix enter y y y y; where y=0||1

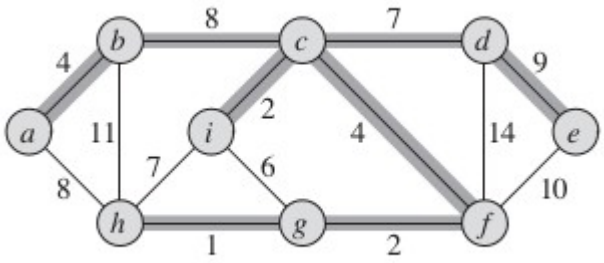


As we see, since **B** was put on 1, which is an obstacle, he can not reach to Batman, irrespective of **J's** position.
Now, we input anpother matrix, which has space to travel:



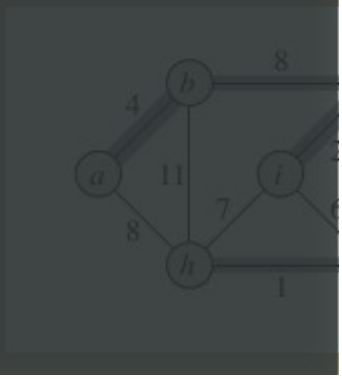
Hence we see that **B** has to move 7 ceells to reach **J.**
IN BFS we can't print path and it's not certain that the path printed is the shortest distance or not.
So we give an alternative algorithm(Dijikstras) which can print the shortest path too.

Consider this particular map: (undirectioned) We create it.

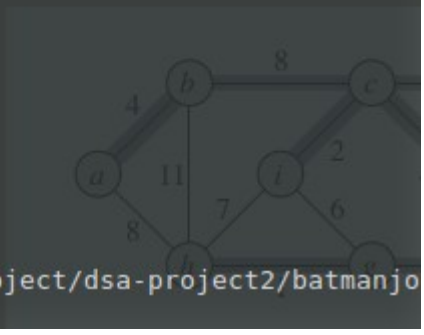



```
ankitvad@DarthVader: ~/tmp/dsa-project/dsa-project2/batmanjoker
Please Insert the number of nodes..
9

Enter details for node 1
How many connections does it have: 2

Enter details for 1 conection
It is connected to node: 2

What is the weight:4

Enter details for 2 conection
It is connected to node: 8

What is the weight:8

Enter details for node 2
How many connections does it have: 2

Enter details for 1 conection
It is connected to node: 3

What is the weight:8
```

```
ankitvad@DarthVader: ~/tmp/dsa-project/dsa-project2/batmanjoker
What is the weight:2

Enter details for node 7
How many connections does it have: 2

Enter details for 1 conection
It is connected to node: 8

What is the weight:1

Enter details for 2 conection
It is connected to node: 9

What is the weight:6

Enter details for node 8
How many connections does it have: 1

Enter details for 1 conection
It is connected to node: 9

What is the weight:7

Enter details for node 9
How many connections does it have:
```



```
ankitvad@DarthVader: ~/tmp/dsa-project/dsa-project2/batmanjoker
What is the weight:1

Enter details for 2 conection
It is connected to node: 9

What is the weight:6

Enter details for node 8
How many connections does it have: 1

Enter details for 1 conection
It is connected to node: 9

What is the weight:7

Enter details for node 9
How many connections does it have: 0

Please enter the start node : 1

Please enter the end node : 5

 Hence the shortest path is:
1->8->7->6->5->ankitvad@DarthVader:~/tmp/dsa-project/dsa-project2/batmanjo
ker$
```



Hence we see that dijikstras can be used to print shortest path, from an adjacency list_matrix.
