**JAYPEE INSTITUTE OF INFORMATION AND TECHNOLOGY**

DEPARTMENT OF CSE AND IT

B. Tech - SEMESTER-III

# ALGORITHM AND PROLEM SOLVING PROJECT REPORT

SUBMITTED TO-

1. PROF. HIMANSU SEKHAR
2. PROF. PRATISHTHA VERMA

GROUP MEMBERS-

1. ANSHUMAN SINGH- 21103115
2. PRERIT SHARMA- 21103121
3. SURYANSH RATHORE- 21103137

BATCH- B5

# PROJECT SYNOPSIS

# TOPIC- METRO MANAGEMENT SYSTEM

## INTRODUCTION

AIM - Our program aims to bring out a proper solution to travel shortest distance in metro using Dijkstra’s algorithm.

## ABOUT THE PROJECT

We have developed a metro management system that uses three algorithms to optimize the travel time and payment process for metro passengers. Firstly, we use backtracking to find all possible paths between the starting and destination stations. Secondly, we use Dijkstra's algorithm to identify the shortest path among the possible routes. Finally, we apply the Coinage Problem (a dynamic programming algorithm) to facilitate cash payment using the available denominations. This involves determining the optimal combination of denominations to provide the exact amount needed to pay for the fare. By combining these algorithms, our system can provide efficient and reliable solutions for passengers using the metro network.

## OBJECTIVES

- To improve the efficiency of the metro system by reducing travel time and increasing capacity.
- To improve the reliability of the metro system by minimizing delays and improving the accuracy of train schedules.
- To provide passengers with an optimized travel route that minimizes the time and distance required for their journey, while considering various factors such as congestion, transfers between lines, and train frequency.
- To increase the safety of the metro system by reducing the risk of accidents and improving emergency response times.
- To improve the overall user experience of the metro system by providing passengers with accurate and up-to-date information about train schedules, delays, and other relevant information.
- To reduce the environmental impact of the metro system by optimizing travel routes to minimize energy consumption and reduce carbon emissions.
- To reduce the operating costs of the metro system by optimizing train schedules and routes, reducing maintenance costs, and increasing overall efficiency.

## ALGORITHMS TO BE USED-

- BACKTRACKING
- DIJKSTRA’S ALGORITHM
- COINAGE PROBLEM

## BACKTRACKING

- Backtracking is a systematic algorithmic technique used to solve various computational problems.
- It works by incrementally building up candidates to the solutions and discarding them as soon as they are found to be invalid.
- The algorithm tries out various combinations of elements and backtracks if a particular combination is found to be invalid.
- Backtracking continues until a valid solution is found or all possible combinations have been tried and found to be invalid.
- Backtracking can be used to solve various problems such as the N-Queens problem, Sudoku, and many others.
- It is a powerful technique that can be used in combination with other algorithms such as Dijkstra's algorithm and dynamic programming to solve more complex problems.

## COINAGE PROBLEM
## TIME COMPLEXITY : O(n^3)

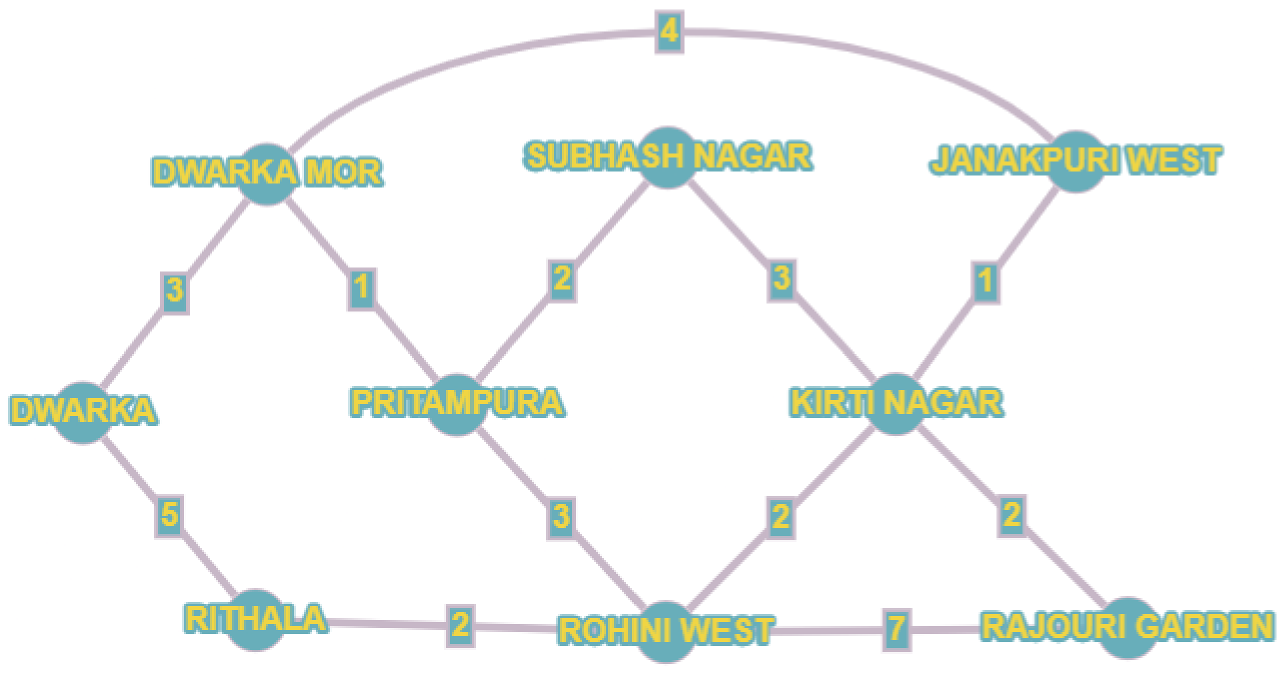
- The Coinage Problem is a dynamic programming (DP) problem that involves finding the minimum number of coins required to make a given amount of money.
- In this problem, the user has a set of available coin denominations, and they need to pay a certain amount using the minimum number of coins.
- The DP approach to solving this problem involves breaking down the problem into subproblems and solving them in a bottom-up manner.
- We start by finding the minimum number of coins required to make each smaller amount of money using the available denominations.
- We then use these solutions to find the minimum number of coins required to make larger amounts of money.
- We repeat this process until we have found the minimum number of coins required to make the full amount.
- The solution involves creating a table that stores the minimum number of coins required to make each amount of money using the available denominations.
- We can then use this table to determine the specific coins that are required to make the full amount.
- The Coinage Problem can be extended by considering additional constraints such as the number of available coins for each denomination or the ability to provide change.
- This problem is commonly used in various applications such as vending machines, cash registers, and financial planning tools

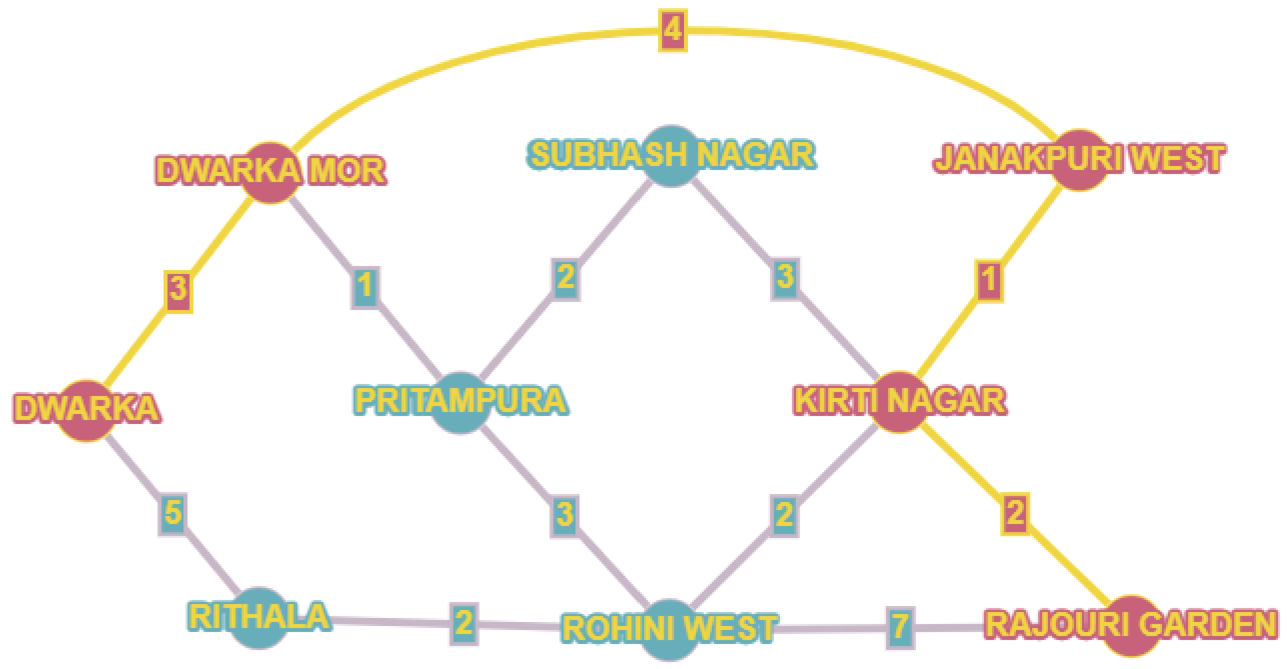
# DIJKSTRA’S ALGORITHM

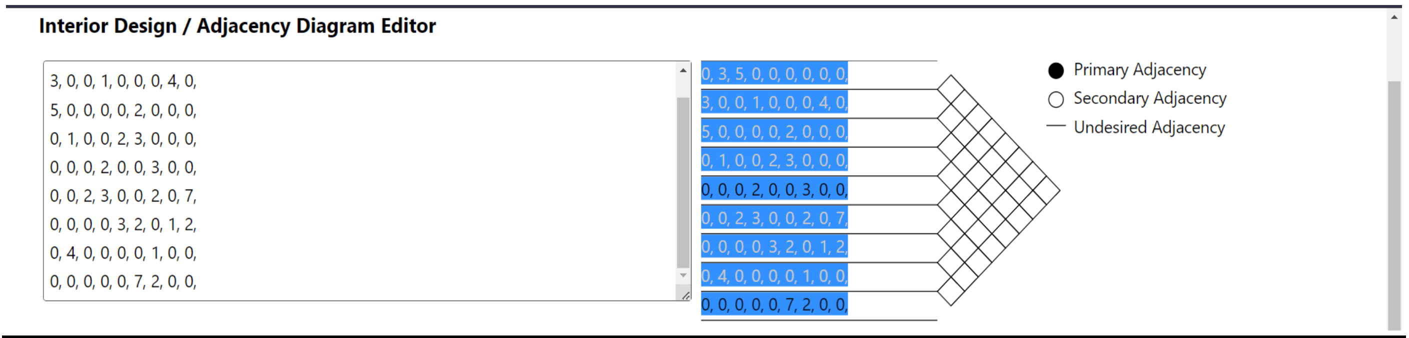
## TIME COMPLEXITY : O(V*E)

## V=vertices, E=edges

- Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyses the graph to find the shortest path between that node and all the other nodes in the graph.
- The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
- Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path. The process continues until all the nodes in the graph have been added to the path.
- This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.
- Dijkstra's Algorithm can only work with graphs that have positive weights. This is because, during the process, the weights of the edges must be added to find the shortest path. If there is a negative weight in the graph, then the algorithm will not work properly.
- Once a node has been marked as "visited", the current path to that node is marked as the shortest path to reach that node. And negative weights can alter this if the total weight can be decremented after this step has occurred.






**Interior Design / Adjacency Diagram Editor**

```
3, 0, 0, 1, 0, 0, 0, 4, 0,
5, 0, 0, 0, 0, 2, 0, 0, 0,
0, 1, 0, 0, 2, 3, 0, 0, 0,
0, 0, 0, 2, 0, 0, 3, 0, 0,
0, 0, 2, 3, 0, 0, 2, 0, 7,
0, 0, 0, 0, 3, 2, 0, 1, 2,
0, 4, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 7, 2, 0, 0,
```

```
0, 3, 5, 0, 0, 0, 0, 0, 0,
3, 0, 0, 1, 0, 0, 0, 4, 0,
5, 0, 0, 0, 0, 2, 0, 0, 0,
0, 1, 0, 0, 2, 3, 0, 0, 0,
0, 0, 0, 2, 0, 0, 3, 0, 0,
0, 0, 2, 3, 0, 0, 2, 0, 7,
0, 0, 0, 0, 3, 2, 0, 1, 2,
0, 4, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 7, 2, 0, 0,
```

● Primary Adjacency
○ Secondary Adjacency
— Undesired Adjacency

# <u>CODE OF THE PROJECT</u>

```
#include <iostream>
#include <unordered_map>
#include <queue>
#include <string>
#include <limits>
#include <vector>
#include <algorithm>

using namespace std;

const int MAX_N = 100; // maximum amount
const int MAX_M = 10;  // maximum number of coin denominations

int no_of_denominations;
int c[MAX_M];              // array of coin denominations
int dp[MAX_N + 1][MAX_M]; // dynamic programming table
int inf = 1e9;             // a large value to represent infinity


// Station struct to store station information
struct Station
{
```

```cpp
    bool visited;
    int distance;
    string prev;
    unordered_map<string, int> neighbors;
};


// Function to search for a station and return its index
int searchStation(string stationName, vector<string> &stations)
{
    for (int i = 0; i < stations.size(); i++)
    {
        if (stations[i] == stationName)
        {
            return i;
        }
    }
    return -1;
}


// Dijkstra's algorithm to find shortest path
int dijkstra(string start, string end, unordered_map<string, Station> &stations)
{
    // Set all station distances to infinity
    for (auto &s : stations)
    {
        s.second.distance = numeric_limits<int>::max();
        s.second.visited = false;
    }

    // Set starting station distance to 0
    stations[start].distance = 0;

    // Create a priority queue to store unvisited stations

    priority_queue<pair<int, string>, vector<pair<int, string>>, greater<pair<int,
string>>> pq;

    // Add starting station to priority queue

    pq.push(make_pair(0, start));

    while (!pq.empty())
    {
        // Get station with shortest distance from priority queue

        pair<int, string> current = pq.top();
        pq.pop();

        // If current station is the end station, we have found the shortest path
        if (current.second == end)
        {
            break;
        }
        // Mark current station as visited
        stations[current.second].visited = true;

        // Go through all neighbors of current station

        for (auto &neighbor : stations[current.second].neighbors)
        {
            string neighborName = neighbor.first;
            int neighborDistance = neighbor.second;

            // If neighbor station has already been visited, skip it
            if (stations[neighborName].visited)
            {
                continue;
            }

            // Calculate new distance to neighbor station

            int distance = stations[current.second].distance + neighborDistance;

            // If new distance is shorter than previous distance, update it

            if (distance < stations[neighborName].distance)
            {
                stations[neighborName].distance = distance;
                stations[neighborName].prev = current.second;

                // Add neighbor station to priority queue

                pq.push(make_pair(distance, neighborName));
            }
        }
    }

    // Print shortest path

    if (stations[end].prev.empty())
    {
        cout << "No path found" << endl;
    }
    else
    {
        string station = end;
        while (station != start)
        {
            cout << station << " <-- ";
            station = stations[station].prev;
        }
        cout << start << endl;
        cout<<endl;
        cout << "TOTAL DISTANCE: " << stations[end].distance <<" KMs"<< endl;
    }

    return stations[end].distance;
```

```
}
//


void backtracking(string start, string end, unordered_map<string, Station> &stations,
vector<string> &path, vector<vector<string>> &allPaths)
{
    path.push_back(start);
    if (start == end)
    {
        allPaths.push_back(path);
    }
    else
    {
        Station &station = stations[start];
        for (auto &neighbor : station.neighbors)
        {
            string neighborName = neighbor.first;
            if (find(path.begin(), path.end(), neighborName) == path.end())
            {
                backtracking(neighborName, end, stations, path, allPaths);
            }
        }
    }
    path.pop_back();
}


void coinage(int target_value, int no_of_denominations)
{
    // initialize dynamic programming table
    for (int i = 1; i <= target_value; i++)
    {
        for (int j = 0; j < no_of_denominations; j++)
        {
            if (i == c[j])
            {
                dp[i][j] = 1;
            }
            else
            {
                dp[i][j] = inf;
            }
        }
    }

    // fill dynamic programming table
    for (int i = 1; i <= target_value; i++)
    {
        for (int j = 0; j < no_of_denominations; j++)
        {
            for (int k = 0; k <= j; k++)
            {
                if (i - c[k] >= 0)
                {
                    dp[i][j] = min(dp[i][j], dp[i - c[k]][k] + 1);
                }
            }
        }
    }

    // print denominations required to make change for n
    vector<int> d(no_of_denominations, 0); // vector to store denominations
    int i = target_value, j = no_of_denominations - 1;
    while (i > 0 && j >= 0)
    {
        if (j > 0 && dp[i][j - 1] <= dp[i][j])
        {
            // exclude coin denomination c[j]
            j--;
        }
        else
        {
            // include coin denomination c[j]
            d[j]++;
            i -= c[j];
        }
    }

    // print results
    if (i > 0)
    {
        cout << "No solution found." << endl;
    }
    else
    {
        for (int j = 0; j < no_of_denominations; j++)
        {
            cout << "Rs."<< c[j] << " : " << d[j] <<" units"<< endl;
        }
    }
}


int main()
{
    // Create vector of station names
    vector<string> stations = {"Rithala", "Rohini West", "Pitampura", "Netaji Subhash
Place", "Kirti Nagar", "Rajouri Garden", "Janakpuri West", "Dwarka Mor", "Dwarka"};

    // Create map of stations
    unordered_map<string, Station> stationMap;

    // Add stations to map
    for (auto &station : stations)
    {
        stationMap[station] = Station();
    }

    // Add neighbors and distances for each station
    stationMap["Rithala"].neighbors = {{"Rohini West", 2}, {"Dwarka", 5}};
    stationMap["Rohini West"].neighbors = {{"Rithala", 2}, {"Pitampura", 3}, {"Rajouri
Garden", 7}, {"Kirti Nagar", 3}};
    stationMap["Pitampura"].neighbors = {{"Rohini West", 3}, {"Netaji Subhash Place", 2},
{"Dwarka Mor", 1}};
    stationMap["Netaji Subhash Place"].neighbors = {{"Pitampura", 2}, {"Kirti Nagar", 3}};
    stationMap["Kirti Nagar"].neighbors = {{"Netaji Subhash Place", 3}, {"Rajouri Garden",
2}, {"Janakpuri West", 4}, {"Rohini West", 2}};
    stationMap["Rajouri Garden"].neighbors = {{"Kirti Nagar", 2}, {"Janakpuri West", 4},
{"Rohini West", 7}};
    stationMap["Janakpuri West"].neighbors = {{"Rajouri Garden", 4}, {"Dwarka Mor", 4},
{"Kirti Nagar", 1}};
    stationMap["Dwarka Mor"].neighbors = {{"Janakpuri West", 4}, {"Dwarka", 3},
{"Pitampura", 1}};
    stationMap["Dwarka"].neighbors = {{"Dwarka Mor", 3}, {"Rithala", 5}};


        cout << "\t\t\t******************************" << endl;
        cout << "\t\t\t**                          DELHI-NOIDA METRO
SYSTEM                              **" << endl;
        cout << "\t\t\t******************************";
        cout<<endl;
        cout<<endl;
        cout << "\t!!!! Welcome to Metro Management !!!!\n";


    // Get source and destination stations from user
    string source, destination;
    cout<<endl;
    cout<<endl;
    cout << "ENTER THE SOURCE STATION: ";
    getline(cin, source);
    cout<< endl;
    cout << "ENTER THE DESTINATION STATION: ";
    getline(cin, destination);
    cout<<endl;
    cout<<endl;
    cout<<endl;


    // Search for source and destination stations in station vector
    int sourceIndex = searchStation(source, stations);
    int destinationIndex = searchStation(destination, stations);

    // If either station is not found, print error message and return
    if (sourceIndex == -1 || destinationIndex == -1)
    {
        cout << "Error: Station not found" << endl;
        return 0;
    }


 // PRINTING ALL POSSIBLE PATHS USING BACKTRACKING
    cout<<"***********************************";
    cout<<endl;
    cout << "DEAR PASSENGER! ALL POSSIBLE ROUTES BETWEEN THE SOURCE AND DESTINATION
STATIONS ARE AS FOLLOWS:";
    cout<<endl;
    cout<<"**********************************";
    cout<<endl;


    vector<string> path;
    vector<vector<string>> allPaths;
    backtracking(stations[sourceIndex], stations[destinationIndex], stationMap, path,
allPaths);


    if (allPaths.empty())
    {
        cout << "No path found\n";
    }
    else
    {
        int count=1;
        for (vector<string> p : allPaths)
        { cout<<count <<".) ";
        count++;
            for (string station : p)
            {
                if (station == destination)
                    cout << station << endl;
                else
                    cout << station << " -> ";
            }
            cout << endl;
        }
    }

    cout << endl;
    cout<<endl;


    // Find shortest path using Dijkstra's algorithm
    cout<< "********************";
    cout<<endl;
    cout << "DEAR PASSENGER! YOUR SHORTEST PATH WILL BE THE FOLLOWING:";
    cout<<endl;
    cout<< "********************";
    cout << endl;
    int distance = dijkstra(stations[sourceIndex], stations[destinationIndex],
stationMap);
    int cost = 1.4 * distance + 10;
    cout << endl;
    cout << endl;
    cout<<endl;
    cout<<endl;
    cout<<endl;
```

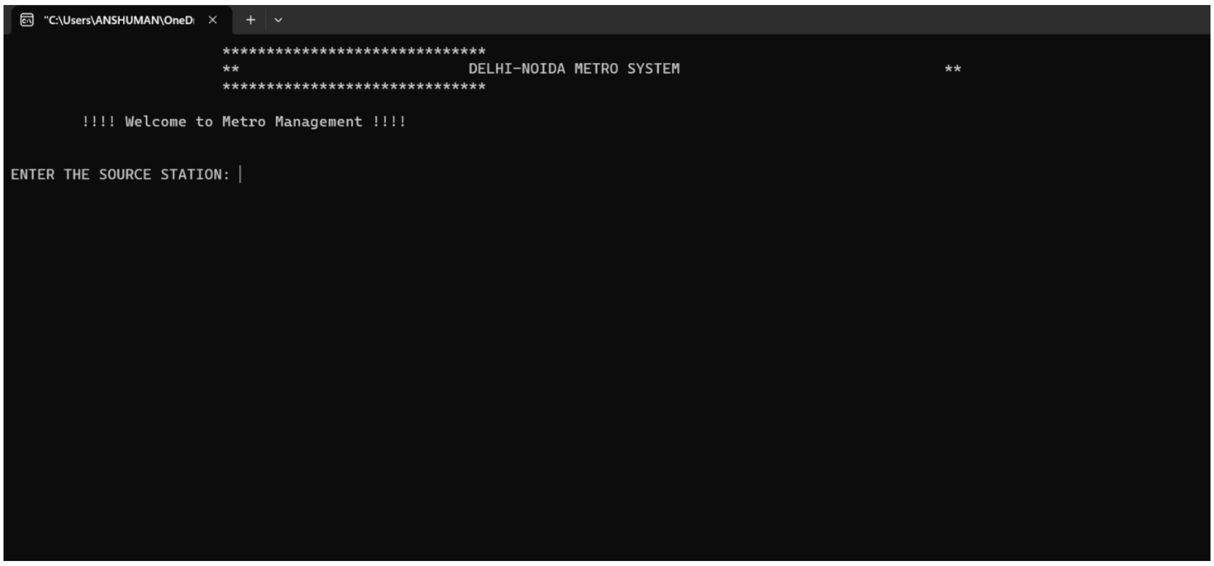
```
       // Calculate the Total Cost for the journey
    cout<< "****************";
    cout<<endl;
    cout << "DEAR PASSENGER! YOUR COST FOR THE JOURNEY:";
    cout<<endl;
    cout<< "****************";
    cout << endl;
    cout<<"COST: Rs."<< cost;
    cout<<endl;
    cout<<"----";
    cout<<endl;
    cout<<endl;
    label:
    cout<<"For payment by card, PRESS: 1 (10% effective discount)";
    cout<<endl;
    cout<<"For payment by cash, PRESS: 2";
    cout<<endl;
    cout<<endl;
    int input;
    cout<<"ENTER YOUR CHOICE: ";
    cout<<endl;
    cout<<"-----------------";
    cin >> input;
    cout<<endl;
    cout<<endl;

switch(input) {
  case 1:
    cout<<"DISCOUNTED COST = Rs."<< 0.9*cost;
    cout<<endl;
    cout<<"---------------";
    cout<<endl;
    cout<<"SAVINGS = Rs."<< 0.1*cost;
    cout<<endl;
    cout<<"-------";
    break;
  case 2:
    // Give the no. of denominations required.
    cout<<"Enter The No. Of Denominations You Have: ";
    cin >> no_of_denominations;
    cout<<endl;
    cout<<"Enter Your Denominations: ";
    for (int i = 0; i < no_of_denominations; i++)
    {
        cin >> c[i];
    }
    cout<<endl;
    cout<<endl;
    cout<<"YOU HAVE TO PAY: ";
    cout<<endl;
    cout<<"---------------";
    cout<<endl;
    coinage(cost, no_of_denominations);

    break;
```

```cpp
  default:
    goto label;
}
cout<<endl;
cout<<endl;
cout<<endl;
cout<<"THANK YOU FOR YOUR JOURNEY WITH US. SEE YOU AGAIN SOON!";
    return 0;
}
```

---

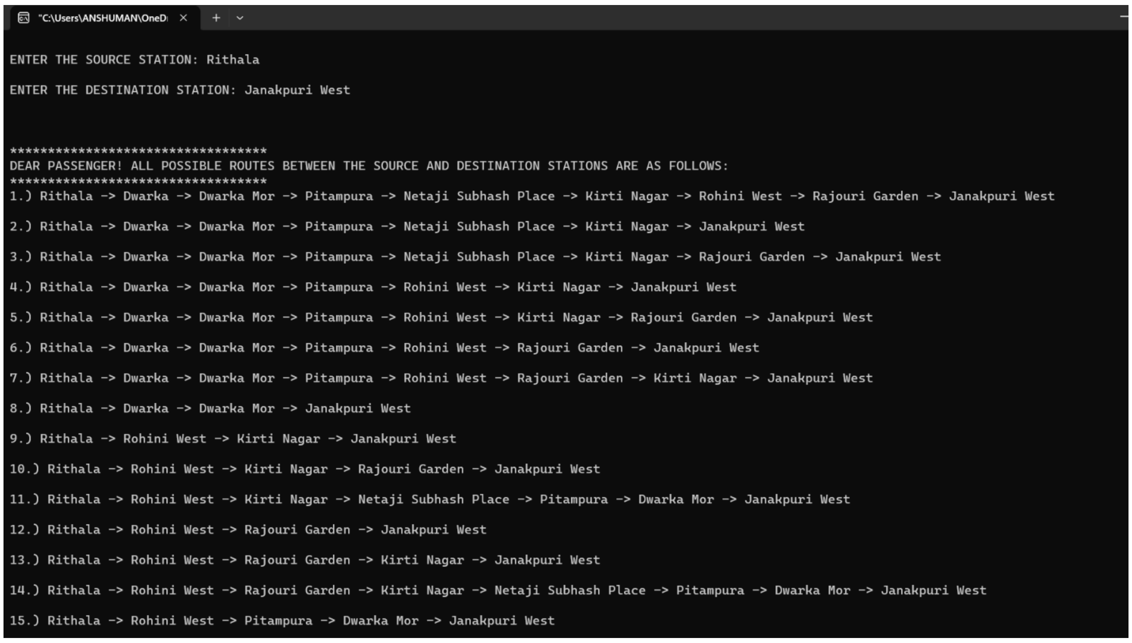
## SCREENSHOTS OF OUTPUT

1. WELCOME SCREEN-



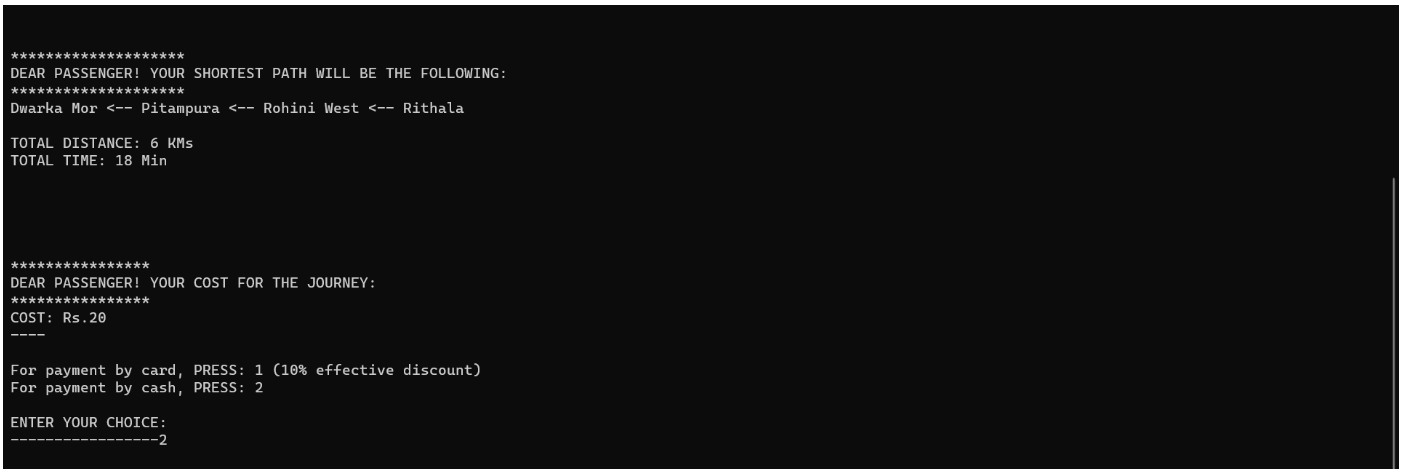
```
                            ********************************
                            **                              DELHI-NOIDA METRO SYSTEM                              **
                            ********************************

          !!!! Welcome to Metro Management !!!!


ENTER THE SOURCE STATION: |
```

2. LIST OF AVAILABLE PATHS



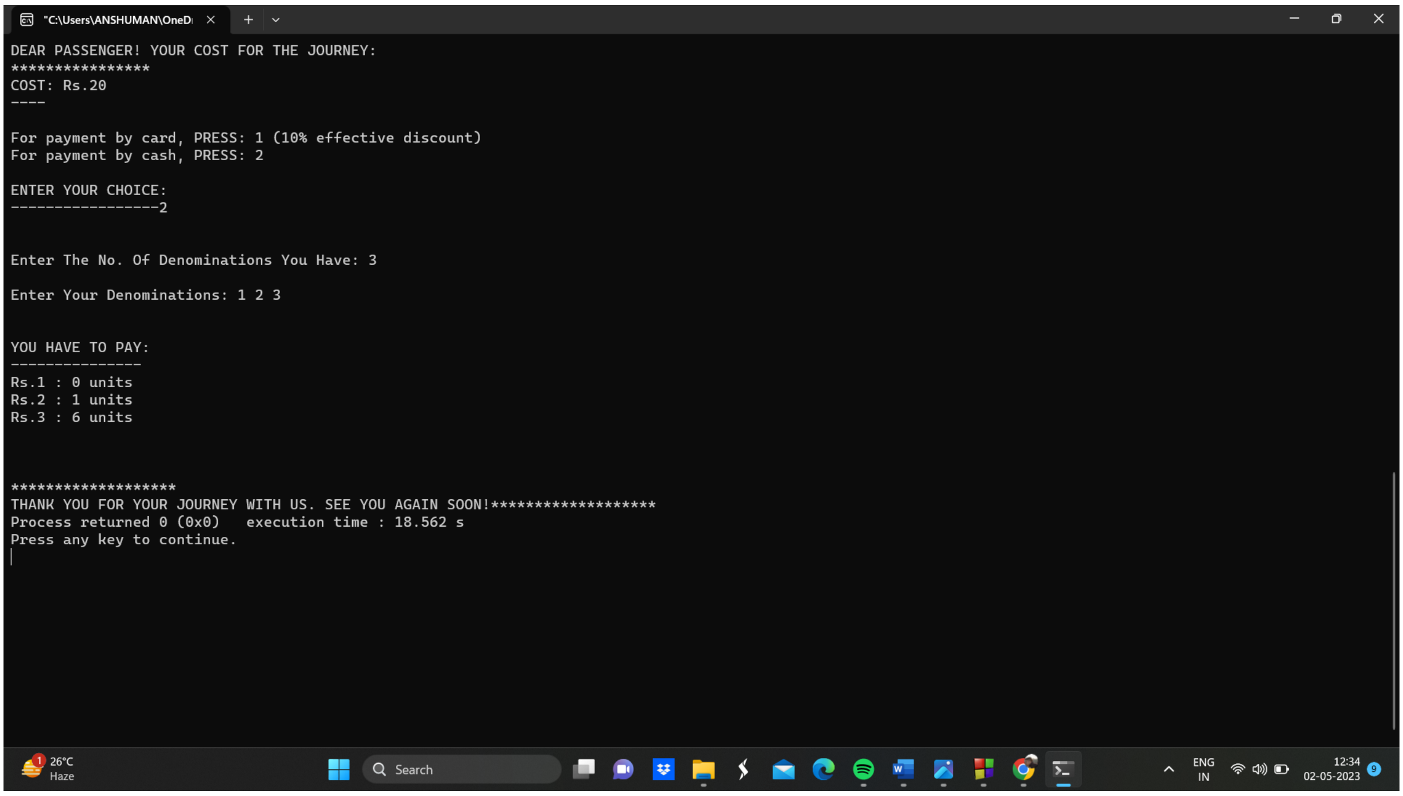
```
ENTER THE SOURCE STATION: Rithala

ENTER THE DESTINATION STATION: Janakpuri West


**********************************
DEAR PASSENGER! ALL POSSIBLE ROUTES BETWEEN THE SOURCE AND DESTINATION STATIONS ARE AS FOLLOWS:
**********************************
1.) Rithala -> Dwarka -> Dwarka Mor -> Pitampura -> Netaji Subhash Place -> Kirti Nagar -> Rohini West -> Rajouri Garden -> Janakpuri West

2.) Rithala -> Dwarka -> Dwarka Mor -> Pitampura -> Netaji Subhash Place -> Kirti Nagar -> Janakpuri West

3.) Rithala -> Dwarka -> Dwarka Mor -> Pitampura -> Netaji Subhash Place -> Kirti Nagar -> Rajouri Garden -> Janakpuri West

4.) Rithala -> Dwarka -> Dwarka Mor -> Pitampura -> Rohini West -> Kirti Nagar -> Janakpuri West

5.) Rithala -> Dwarka -> Dwarka Mor -> Pitampura -> Rohini West -> Kirti Nagar -> Rajouri Garden -> Janakpuri West

6.) Rithala -> Dwarka -> Dwarka Mor -> Pitampura -> Rohini West -> Rajouri Garden -> Janakpuri West

7.) Rithala -> Dwarka -> Dwarka Mor -> Pitampura -> Rohini West -> Rajouri Garden -> Kirti Nagar -> Janakpuri West

8.) Rithala -> Dwarka -> Dwarka Mor -> Janakpuri West

9.) Rithala -> Rohini West -> Kirti Nagar -> Janakpuri West

10.) Rithala -> Rohini West -> Kirti Nagar -> Rajouri Garden -> Janakpuri West

11.) Rithala -> Rohini West -> Kirti Nagar -> Netaji Subhash Place -> Pitampura -> Dwarka Mor -> Janakpuri West

12.) Rithala -> Rohini West -> Rajouri Garden -> Janakpuri West

13.) Rithala -> Rohini West -> Rajouri Garden -> Kirti Nagar -> Janakpuri West

14.) Rithala -> Rohini West -> Rajouri Garden -> Kirti Nagar -> Netaji Subhash Place -> Pitampura -> Dwarka Mor -> Janakpuri West

15.) Rithala -> Rohini West -> Pitampura -> Dwarka Mor -> Janakpuri West
```

## 3. SHORTEST PATH AVAILABLE



```
********************
DEAR PASSENGER! YOUR SHORTEST PATH WILL BE THE FOLLOWING:
********************
Dwarka Mor <-- Pitampura <-- Rohini West <-- Rithala

TOTAL DISTANCE: 6 KMs
TOTAL TIME: 18 Min




****************
DEAR PASSENGER! YOUR COST FOR THE JOURNEY:
****************
COST: Rs.20
----

For payment by card, PRESS: 1 (10% effective discount)
For payment by cash, PRESS: 2

ENTER YOUR CHOICE:
------------------2
```

## 4. COIN DENOMINATIONS TO BE PAID



```
DEAR PASSENGER! YOUR COST FOR THE JOURNEY:
****************
COST: Rs.20
----

For payment by card, PRESS: 1 (10% effective discount)
For payment by cash, PRESS: 2

ENTER YOUR CHOICE:
------------------2


Enter The No. Of Denominations You Have: 3

Enter Your Denominations: 1 2 3


YOU HAVE TO PAY:
---------------
Rs.1 : 0 units
Rs.2 : 1 units
Rs.3 : 6 units



*******************
THANK YOU FOR YOUR JOURNEY WITH US. SEE YOU AGAIN SOON!*******************
Process returned 0 (0x0)   execution time : 18.562 s
Press any key to continue.
```

# REFERENCES

- https://www.freecodecamp.org/news/backtracking-algorithms-explained/
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_dynamic_programming.htm
- Introduction to Algorithms" by T. Cormen, C. Leiserson, R. Rivest, and C. Stein