Premier University

Assignment

Assignment Title	Applying Dijkstra and Knapsack Algorithm in Real Life Scenario
Course code	CSE 225
Course name	Algorithm Design and Analysis
Date of Submission	13/02/2024

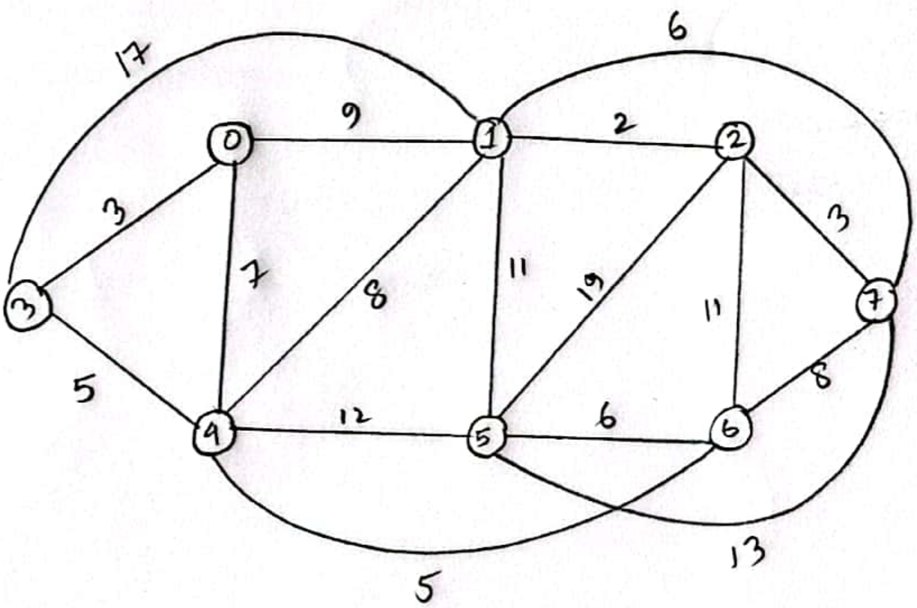
Submitted to
Fairuz Bilquis Khan Lecturer Department of CSE

Submitted by	
Name	Mohammad Hafizur Rahman Sakib
ID	0222210005101118
Section	C
Semester	4th
Session	Fall 2023

Objective:

To design a intelligent transportation optimization System for a smart city using dijkstra and 0/1 Knapsack algorithm. Finding best route for time efficiency to visit from a single city to every city for the inhabitants and maximize sight seeing value for the tourists in a Limited budget.

Design best route from Single City:



Here, In This graph (0-7) → 8 nodes/vertex representing 8 city and each edge(17) representing 17 different Paths with timing Cost to visit from one city to another City. In this graph I will apply, Single Source Shortest Path algorithm (Dijkstra) to find the best route. As, there is no negative Cost. It's an Undirected graph also.

Algorithm choice for Shortest Path:

I will choose dijkstra algorithm for finding minimum cost for the inhabitants. It's an fundamental Shortest Path from a single Source Vertex to all other vertices in a weighted graph. It employs a greedy Strategy, iteratively Selecting the vertex and updating the distances to its neighbors accordingly.

Graph representation:

I've choosen adjecency list for the graph representation, because it's an memory efficient approach, rather than adjecency matrix. An adjacency list is a data structure used to represent a graph where each vertex maintains a list of it's neighboring Ventices.

Data Structure:

I will use C++ STL vectors ~~to~~ and Pairs to store the graph, to keep track of visited vertices, and the distances.

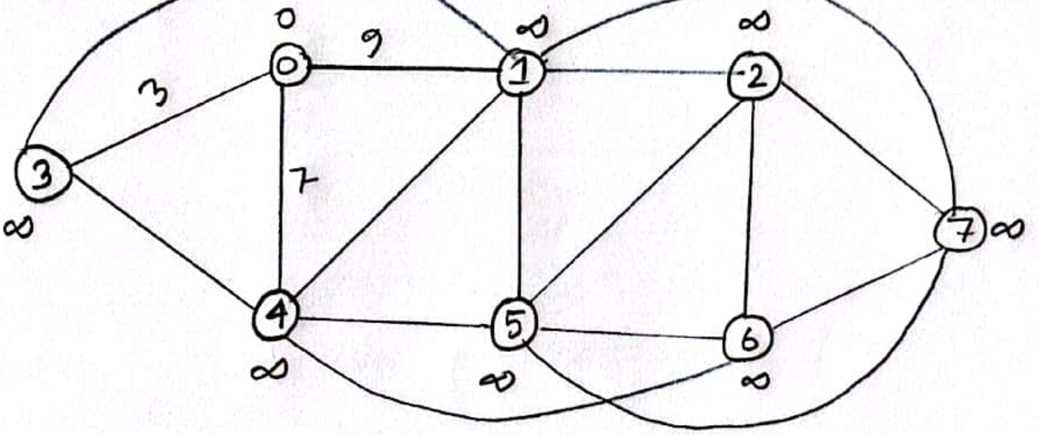
And, I will use C++ STL set to find the minimum cost containing nodes/ventices. I've choose set, because, it's by default can Provide me the minimum weight in (logn) time. I might choose Priority_queue, but declaration is little bit complex (for small→ greater) values. Overall worst case complexity for dijkstra is $O(n^2)$.

Step 01:

* Here, for the drawing Complexity, I will draw only the edges of current vertex in each step

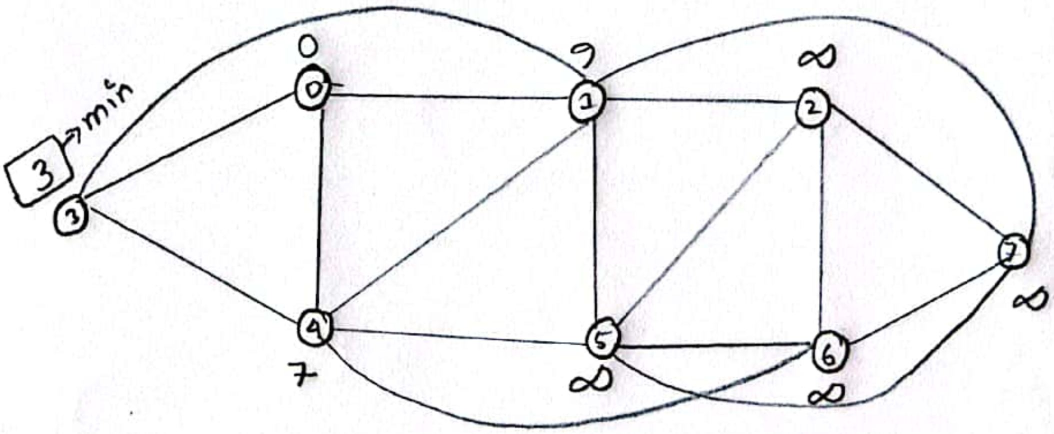
Source node = 0

* Taking all inf.



Step 02:

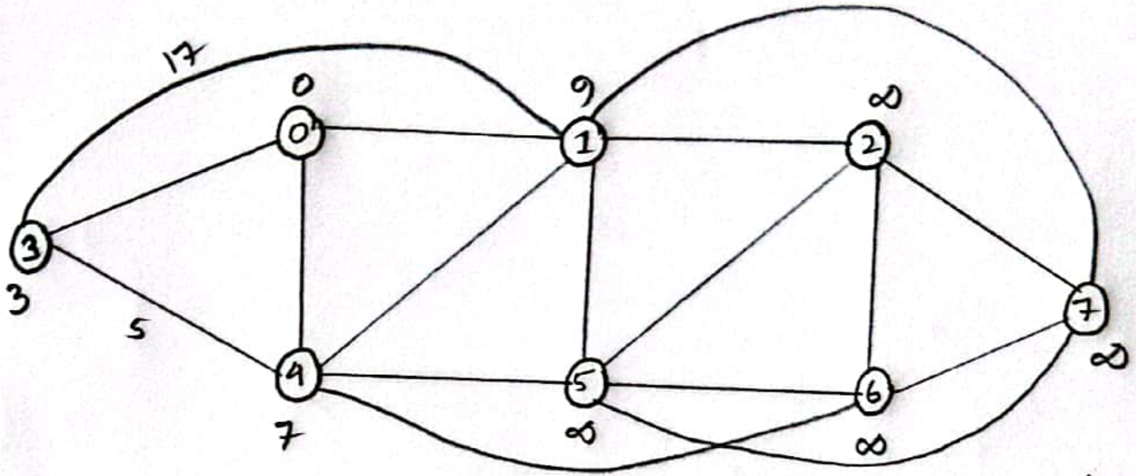
Current vertex = 0



In this step, taking minimum distance from Source Vertex "0", we visit to the new vertex "3", because 3 is the minimum distance here.

Step 03:

Current Vertex = 3



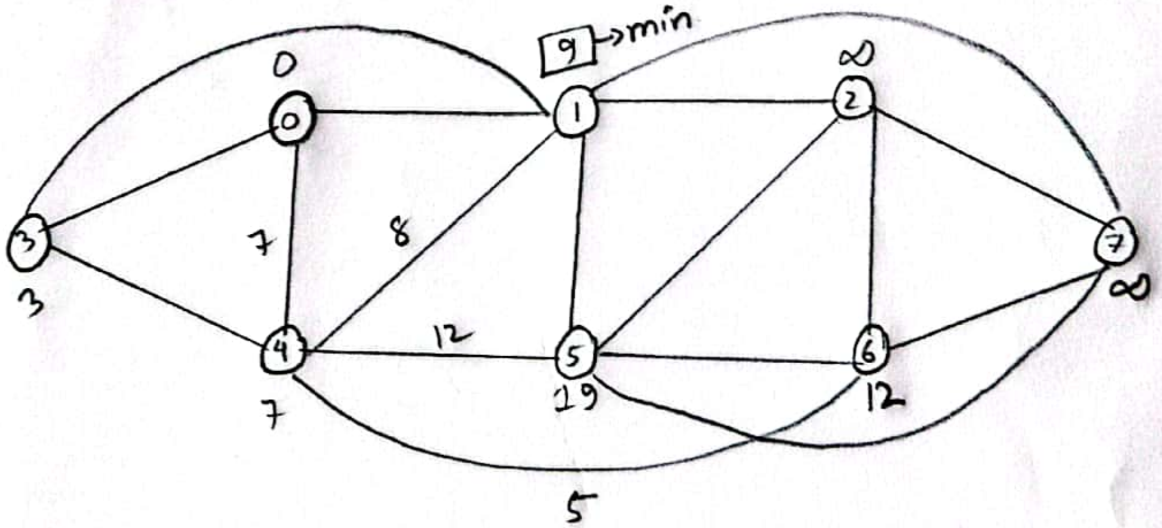
as, $(0 \rightarrow 3 \rightarrow 7) \rightarrow$ Total cost is 8, we apply this-

$\boxed{d(A,B) = \min(d(A) + d(A,B),\ \underset{\text{from } 0}{d(B)})}$ → relexation formula

now, we travel to vertex to 4, because, it's minimal way.

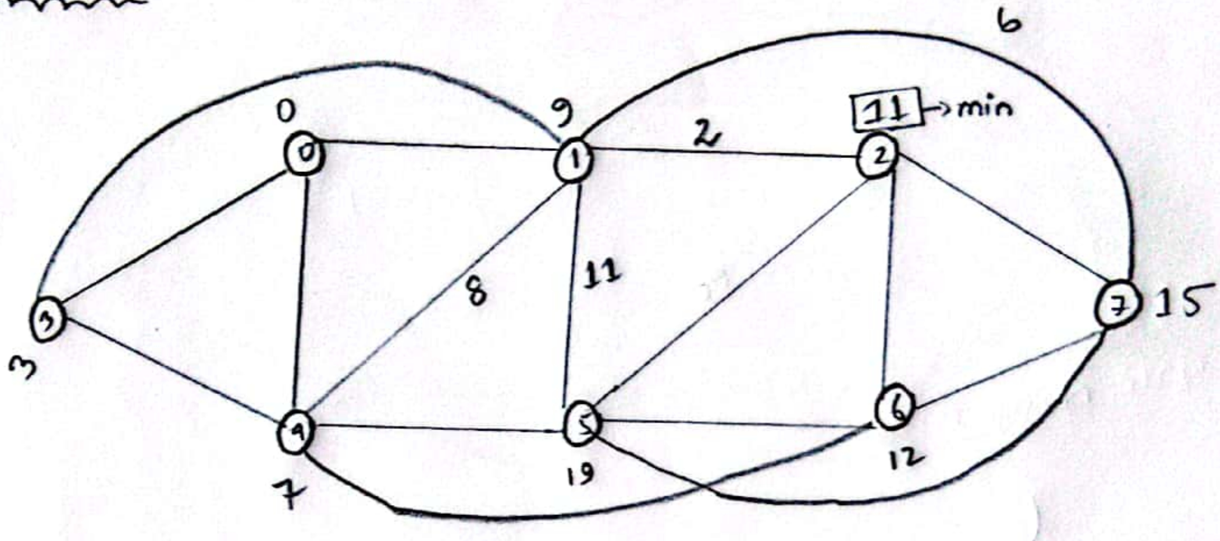
Step 04:

Current Vertex = 4

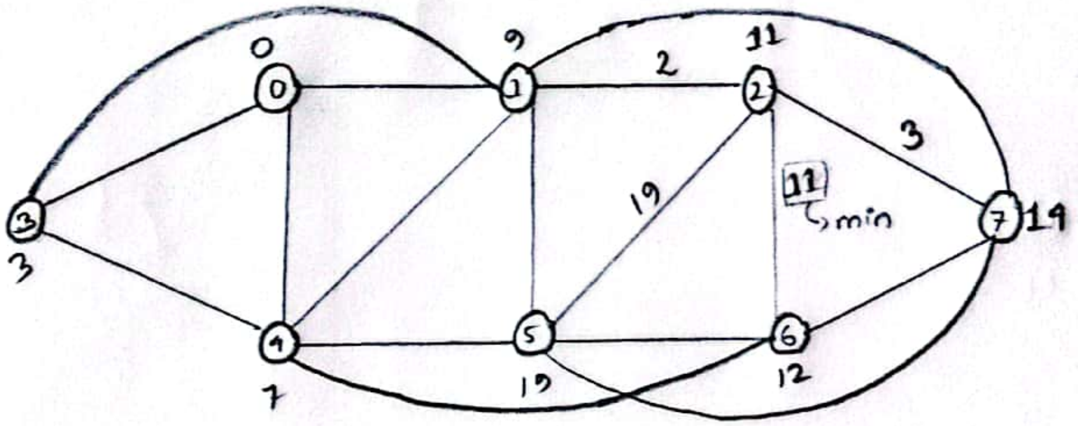


Here, Vertex 1 has minimum cost. So, we traverse to vertex 1, for next relexation.

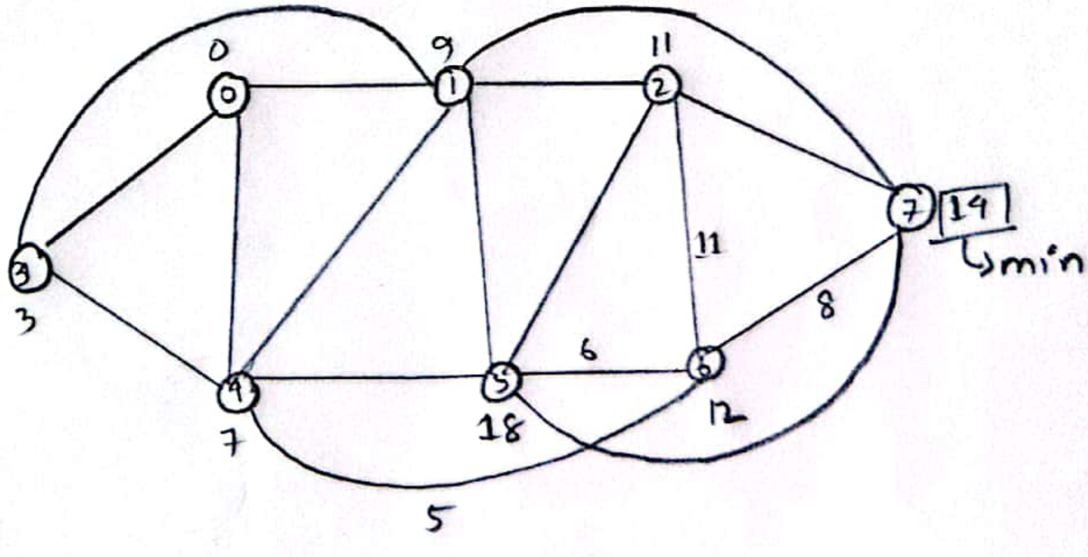
Step 05: Current vertex = 1



Step 06: Current vertex = 2

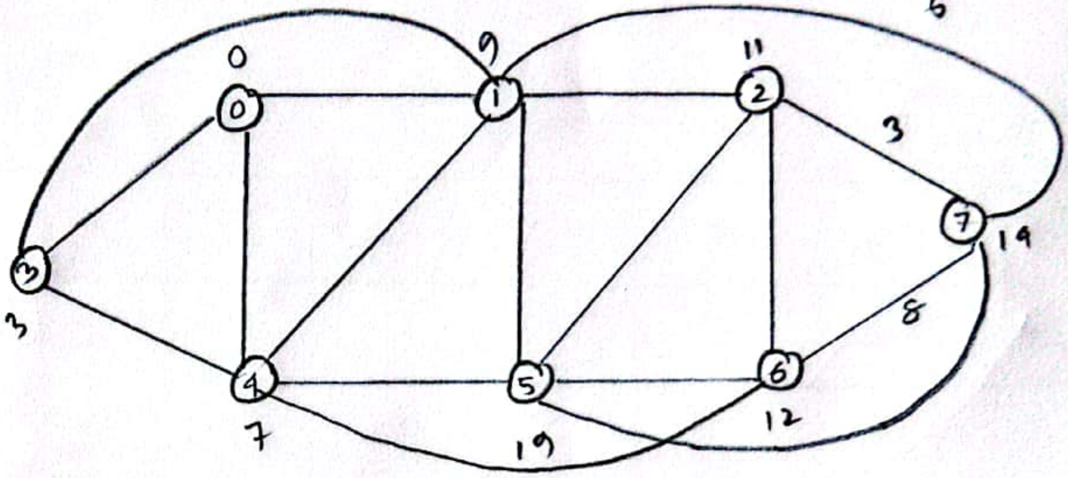


Step 07: Current vertex = 6.



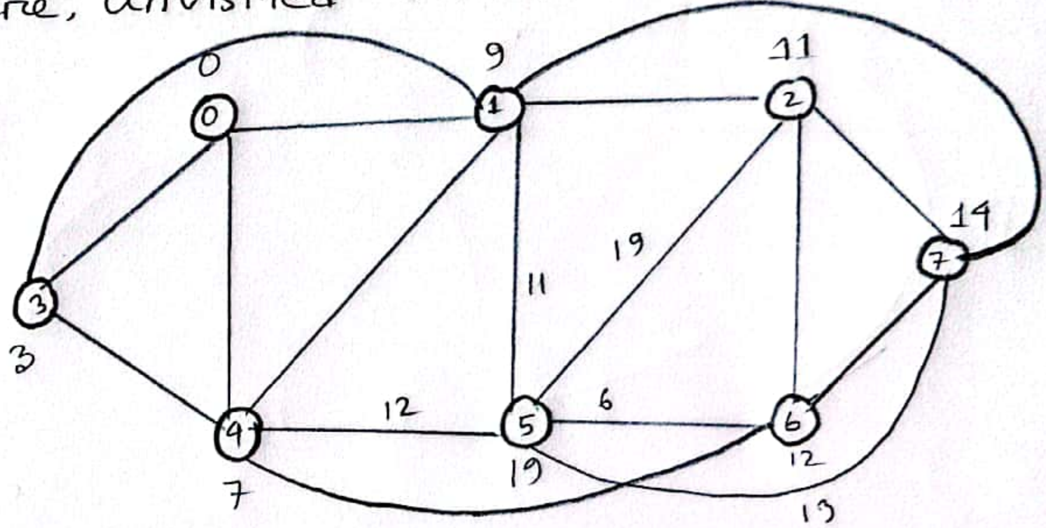
Step 08:

Current vertex = 7



Step 09: Current vertex = 5.

as there, unvisited Vertex is Only 5 now.



Final distances: From city 0 to other city —

City 0 : 0

City 01 : 9

City 2 : 11

City 3 : 3

City 4 : 7

City 5 : 18

City 6 : 12

City 7 : 14

Computation Table:

	0	1	2	3	4	5	6	7
	0	∞	∞	∞	∞	∞	∞	∞
0	$\boxed{0}$	9	∞	3	7	∞	∞	∞
3	$\underline{0}$	9	∞	$\boxed{3}$	7	∞	∞	∞
4	$\underline{0}$	9	∞	$\underline{3}$	$\boxed{7}$	19	12	∞
1	$\underline{0}$	$\boxed{9}$	11	$\underline{3}$	$\underline{7}$	19	12	15
2	$\underline{0}$	$\underline{9}$	$\boxed{11}$	$\underline{3}$	$\underline{7}$	19	12	14
6	$\underline{0}$	$\underline{9}$	$\underline{11}$	$\underline{3}$	$\underline{7}$	18	$\boxed{12}$	14
7	$\underline{0}$	$\underline{9}$	$\underline{11}$	$\underline{3}$	$\underline{7}$	18	$\underline{12}$	$\boxed{14}$
5	$\underline{0}$	$\underline{9}$	$\underline{11}$	$\underline{3}$	$\underline{7}$	$\boxed{18}$	$\underline{12}$	$\underline{14}$

In the computation table –
Marked with box (☐) are the current minimum cost for the vertex/city and underlined (—) costs are representing already taken/visited city/vertex, which should not be visit again. to avoid unnecessary computation.

Code for the Inhabitants :

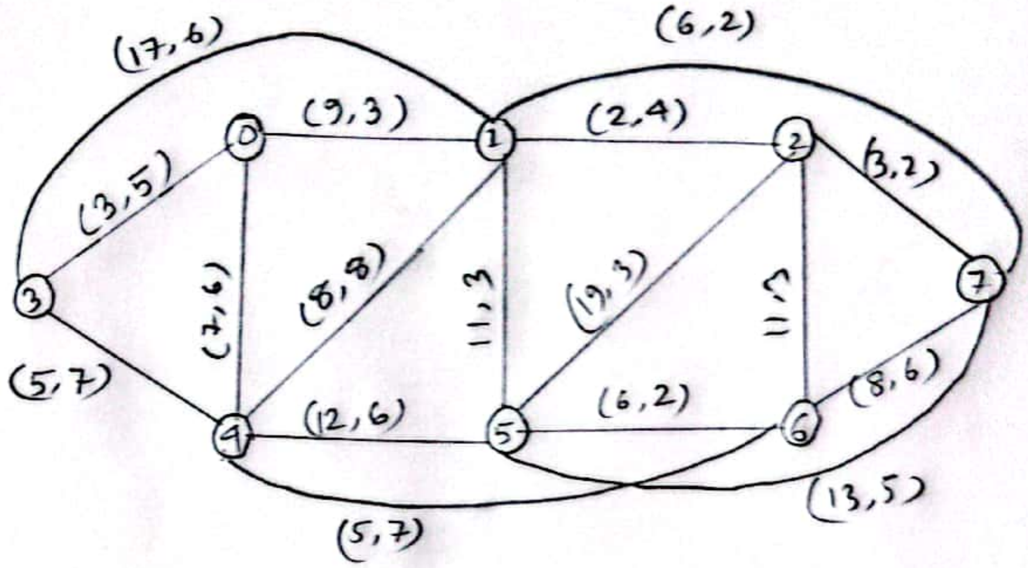
```cpp
#include <bits/stdc++.h>
using namespace std;
const int inf = 1e7;
const int n = 1e3;
vector<pair<int, int>> g[n];
// { node , cost}
void dijkstra(int source)
{
    vector<int> distance(n, inf);
    vector<bool> visited(n, 0);
    set<pair<int, int>> st;
    // {cost , node} => kept cost on first value to sort based on lowest cost
    st.insert({0, source});
    distance[source] = 0;
    while (!st.empty())
    {
        auto node = *st.begin();
        // will give the minimum weighted pair {cost , node}
        int parent_node = node.second;
        int parent_node_cost = node.first;
        st.erase(st.begin());
        if (visited[parent_node])
        {
            continue;
        }
        visited[parent_node] = 1;
        // Traverse to the child of v,for Relaxation
        for (auto child : g[parent_node])
        {
            int child_node = child.first;
            int edge_cost = child.second;

            // Relaxation
            if ((parent_node_cost + edge_cost) < distance[child_node])
            {
                distance[child_node] = (parent_node_cost + edge_cost);
                st.insert({distance[child_node], child_node});
            }
        }
    }
    cout << "Node\tDistance from " << source << endl;
    for (int i = 0; i < n; ++i)
    {
        if (distance[i] != inf)
        {
            cout << i << "\t" << distance[i] << endl;
        }
    }
}
int main()
{
    int node, edge;
    cin >> node >> edge;
    for (int i = 0; i < edge; i++)
    {
        int u, v, cost;
        cin >> u >> v >> cost;
        g[u].push_back({v, cost});
        g[v].push_back({u, cost});
        // u/v indexed node connected with v/u node with cost
    }
    dijkstra(0);

    return 0;
}
```

Minimum costs output for 0 number city's Inhabitants to travel into other cities using C++ :

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
8 17
0 1 9
0 4 7
0 3 3
1 4 8
1 5 11
1 7 6
1 3 17
1 2 2
2 7 3
2 6 11
2 5 19
3 4 5
4 6 5
4 5 12
5 7 13
5 6 6
6 7 8
City     Distance from 0
0        0
1        9
2        11
3        3
4        7
5        18
6        12
7        14
SAKIB  AA Test
```

Maximize Site Seeing for tourists:



Here, (cost, SS_value) Pair taken.

Algorithm choice:

I choose 0/1 Knapsack algorith for maximizing Sig Site-Seeing Values. This algorithm involves Selecting a subset of items with maximum value while respecting a weight consistant. Each item can either be included in the Knapsack (1) or excluded (0), hence the name. It's a classic optimization Problem often solved using dynamic Programming techniques to efficiently explore all Possible Combinations of items and weights.

Data Structures:

I will use C++ STL (vectors) to keep the values of (weight and cost) and a 2D vector for store the results of subproblems in the dynamic Programming Solution to the 0/1 KnapSack Problem.

Site-Seeing values and costs from graph:

Budget of Tourist: 15

Nodes	SS-Value	cost
0→1	3	9
0→4	6	7
0→3	5	3
1→4	8	8
1→5	3	11
1→7	2	6
1→3	6	17
1→2	4	2
2→7	2	3
2→6	3	11
2→5	3	19
3→4	7	5
4→6	7	5
4→5	6	12
5→7	5	13
5→6	2	6
6→7	6	8

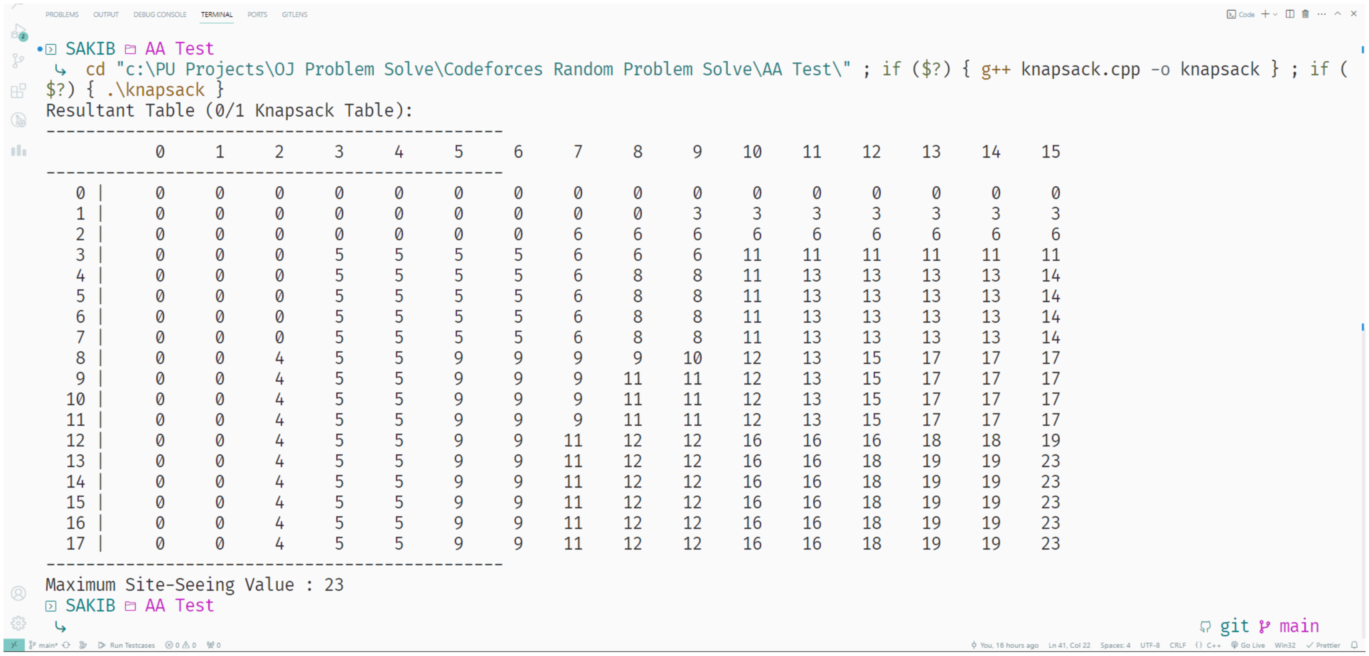
Resultant Table:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3
2	0	0	0	0	0	0	0	6	6	6	6	6	6	6	6	6
3	0	0	0	5	5	5	5	6	6	6	11	11	11	11	11	11
4	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
5	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
6	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
7	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
8	0	0	4	5	5	5	9	9	9	10	12	13	15	17	17	17
9	0	0	4	5	5	5	9	9	11	11	12	13	15	17	17	17
10	0	0	4	5	5	5	9	9	11	11	12	13	15	17	17	17
11	0	0	4	5	5	5	9	9	11	11	12	13	15	17	17	17
12	0	0	4	5	5	5	9	9	12	12	16	16	16	18	18	19
13	0	0	4	5	5	5	9	9	12	12	16	16	18	19	19	23
14	0	0	4	5	5	5	9	9	12	12	16	16	19	19	19	23
15	0	0	4	5	5	5	9	9	12	12	16	16	19	19	19	23
16	0	0	4	5	5	5	9	9	12	12	16	16	19	19	19	23
17	0	0	4	5	5	5	9	9	12	12	16	16	19	19	19	23

Code for maximizing site-seeing values for tourists:

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> ss_value = {3, 6, 5, 8, 3, 2, 6, 4, 2, 3, 3, 7, 7, 6, 5, 2, 6};
vector<int> cost = {9, 7, 3, 8, 11, 6, 17, 2, 3, 11, 19, 5, 5, 12, 13, 6, 8};
int budget = 15; // Tourists Budget

int knapsack(int W)
{
    int n = cost.size();
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));
    for (int i = 1; i <= n; i++)
    {
        for (int w = 1; w <= W; w++)
        {
            if (cost[i - 1] <= w)
            {
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - cost[i - 1]] + ss_value[i - 1]);
            }
            else
            {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    // Printing the resultant table
    cout << "Resultant Table (0/1 Knapsack Table):" << endl;
    cout << "----------------------------------------------" << endl;
    cout << setw(6) << " ";
    for (int w = 0; w <= W; w++)
    {
        cout << setw(6) << w;
    }
    cout << endl;
    cout << "----------------------------------------------" << endl;
    for (int i = 0; i <= n; i++)
    {
        cout << setw(4) << i << " |";
        for (int w = 0; w <= W; w++)
        {
            cout << setw(6) << dp[i][w];
        }
        cout << endl;
    }
    cout << "----------------------------------------------" << endl;

    return dp[n][W];
}

int main()
{
    cout << "Maximum Site-Seeing ss_valueue : " << knapsack(budget) << endl;

    return 0;
}
```

Resultant table and output for the tourists :



```
SAKIB AA Test
↳ cd "c:\PU Projects\OJ Problem Solve\Codeforces Random Problem Solve\AA Test\" ; if ($?) { g++ knapsack.cpp -o knapsack } ; if ($?) { .\knapsack }
Resultant Table (0/1 Knapsack Table):
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3
2	0	0	0	0	0	0	0	6	6	6	6	6	6	6	6	6
3	0	0	0	5	5	5	5	6	6	6	11	11	11	11	11	11
4	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
5	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
6	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
7	0	0	0	5	5	5	5	6	8	8	11	13	13	13	13	14
8	0	0	4	5	5	9	9	9	9	10	12	13	15	17	17	17
9	0	0	4	5	5	9	9	9	11	11	12	13	15	17	17	17
10	0	0	4	5	5	9	9	9	11	11	12	13	15	17	17	17
11	0	0	4	5	5	9	9	9	11	11	12	13	15	17	17	17
12	0	0	4	5	5	9	9	11	12	12	16	16	16	18	18	19
13	0	0	4	5	5	9	9	11	12	12	16	16	18	19	19	23
14	0	0	4	5	5	9	9	11	12	12	16	16	18	19	19	23
15	0	0	4	5	5	9	9	11	12	12	16	16	18	19	19	23
16	0	0	4	5	5	9	9	11	12	12	16	16	18	19	19	23
17	0	0	4	5	5	9	9	11	12	12	16	16	18	19	19	23

```
Maximum Site-Seeing Value : 23
SAKIB AA Test
↳
```

From the resultant table, the maximum site-seeing value for tourist is 23, in the budget of 15.

Complex - Problem Solving Questions:

(a) Does the Solution need in-depth engineering Knowledge?

=> yes, it requires deep expertise in transportation Systems, algorithms, data analysis, and Software development.

(b) Does the Solution involve wide-ranging or conflicting technical, engeneering and other issues?

=> Yes. it's involves a variety of technical challenges, including optimizing routes, integrating real-time data and balancing user Preferences.

(c) Is the Solution well-Known, or does it require abstract thinking and analysis to formulate?

=> while Components are known, integrating them into a smart city context requires abstract thinking and analysis.

(d) Does the Solution involve infrequently encountered issues?

=> Yes, especially in dynamically updating routes and balancing Conflicting user Preferences.

11

(e) Does the Solution need adherence to Standards and Codes of Practise?

=> Yes. adherence to data Privacy, Security and transportation regulations is necessary.

(f) Does the Solution involve Stakeholders with Conflicting technical requirements?

=> Yes. inhabitants and tourists may have conflicting Priorities in route optimization.

(g) Does the Solution involve Interdependence between Sub-Problems or Parts?

=> Yes, various Components of the System are Interdependent, requiring careful coordination