**AIM** :

An X-ray telescope (XRT) is a telescope that is designed to observe remote objects in the X-ray spectrum. In order to get above the Earth's atmosphere, which is opaque to X-rays, X-ray telescopes must be mounted on high altitude rockets, balloons or artificial satellites. Planets, stars and galaxies and the observations are to be made with telescope. Here the process of rotating equipment into position to observe the objects is called slewing. Slewing is a complicated and time consuming procedure handled by computer driven motors. The problem is to find the tour of the telescope that moves from one object to other by observing each object exactly once with a minimum total slewing time.

**DESCRIPTION** :

To solve the above problem we need to follow the approach of travelling salesman problem which says , if we have a graph G=(V,E) where V is the set of vertices and E is the set of edges, we need to travel all the vertices only once such that we can obtain a minimum cost. Here, vertices are nothing but the objects present in the sky such as satellites and cost is nothing but the time taken to move from one satellite to another. And we have to minimise this time called as slewing time.

**ALGORITHM** :

Algorithm minimum\_cost(object)

//where object contains the name of city or it’s index

{

travelled[object]=1

nobject=least(object)

if(nobject==999) then

{

nobject=0;

cost:=cost+a[object][nobject];

return;

}

minimum\_cost(nobject);

}

Algorithm least(c)

{

nc:=999;

min:=999;

for i:=0 to n do{

if(a[c][i]!=0) && (travelled[i]==0)) then{

min=a[i][0]+a[c][i];

kmin:=a[c][i];

nc:=i;

}

}

if(min!=999) then cost+=kmin;

return nc;

}

**CODE** :

#include<bits/stdc++.h>

using namespace std;

int a[10][10], travelled[10], n, cost = 0;

void takeInput()

{

int i, j;

cout << "Enter the number of satellites : ";

cin >> n;

cout << "\nEnter the Cost Matrix\n";

for (i = 0; i < n; i++)

{

cout << "\nEnter time (rows in the matrix) : " << i + 1 << "\n";

for (j = 0; j < n; j++)

cin >> a[i][j];

travelled[i] = 0;

}

cout << "\n\nThe cost list is:";

for (i = 0; i < n; i++)

{

cout << "\n";

for (j = 0; j < n; j++)

cout << "\t" << a[i][j];

}

}

int least(int c)

{

int i, nc = 999;

int min = 999, kmin;

for (i = 0; i < n; i++)

{

if ((a[c][i] != 0) && (travelled[i] == 0))

if (a[c][i] + a[i][c] < min)

{

min = a[i][0] + a[c][i];

kmin = a[c][i];

nc = i;

}

}

if (min != 999)

cost += kmin;

return nc;

}

void minimum\_cost(int object)

{

int i, nobject;

travelled[object] = 1;

cout << object + 1 << "--->";

nobject = least(object);

if (nobject == 999)

{

nobject = 0;

cout << nobject + 1;

cost += a[object][nobject];

return;

}

minimum\_cost(nobject);

}

int main()

{

takeInput();

cout << "\n\nThe Path is:\n";

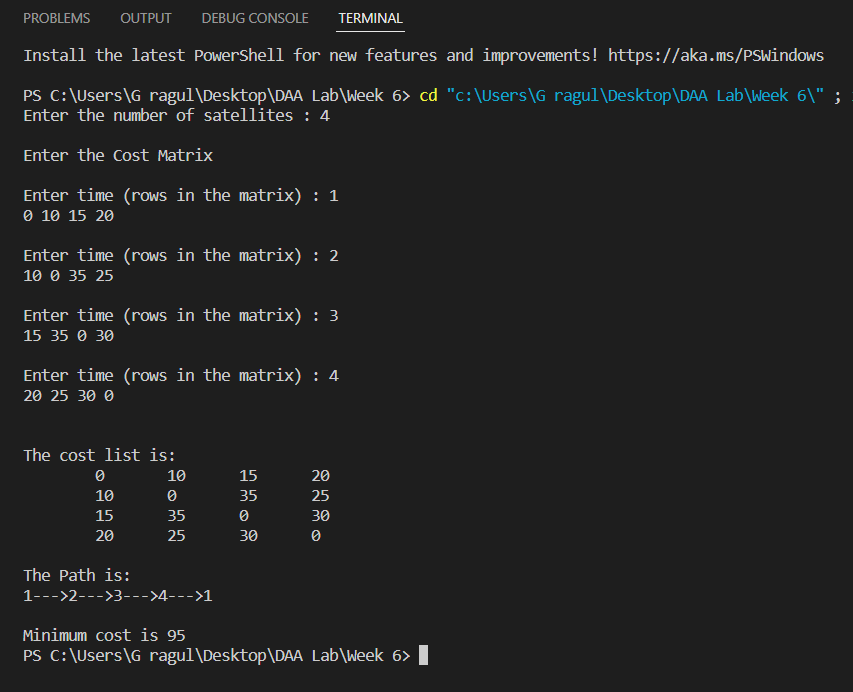
minimum\_cost(0);

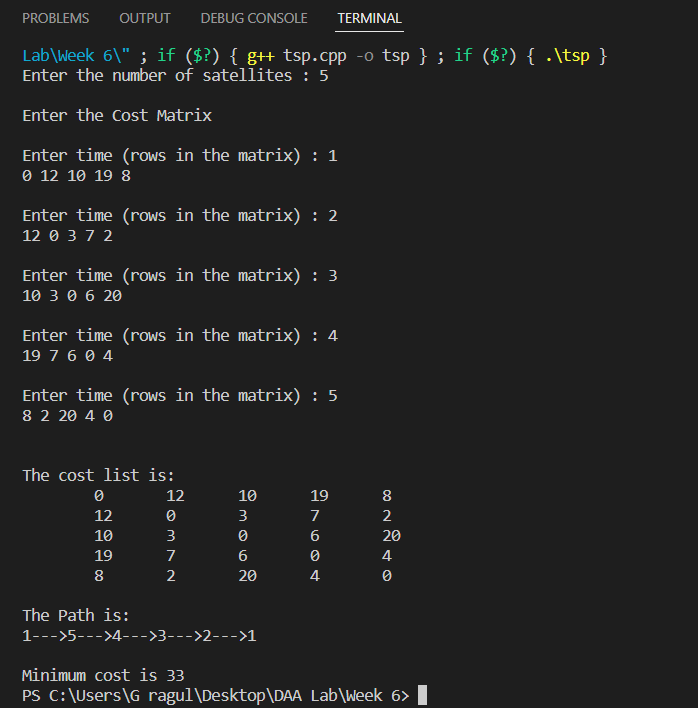
cout << "\n\nMinimum cost is " << cost;

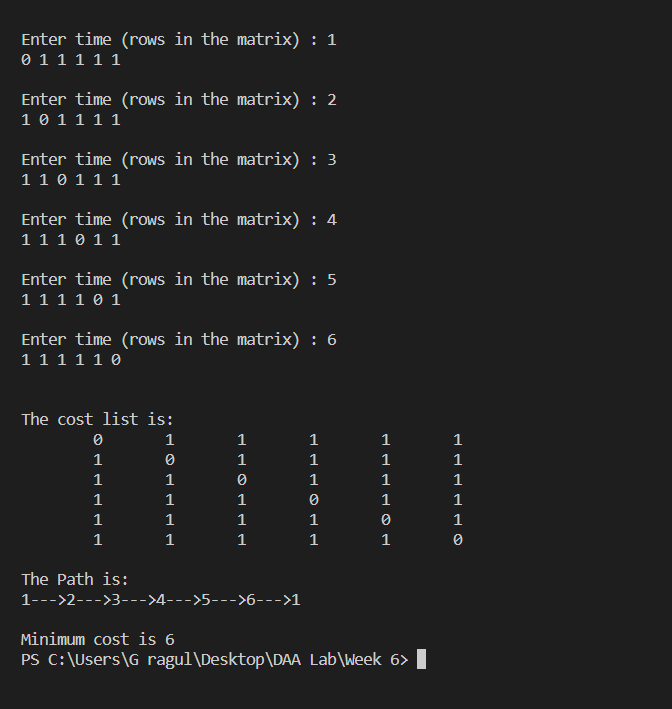
return 0;

}

**RESULT ANALYSIS** :







**Time Complexity** :

Number of sub-problems is : (n-1) \*2^(n-2) sub-problems, which is O (n\*2^n).

Each sub-problem will take O (n) time (finding path to remaining (n-1) nodes).

Therefore total time complexity is O (n2n) \* O (n) = O (n^2\*2^n)

**Space complexity** :

Constant as we have no extra space, space for storing cost is declared as global variable, if it’s local the space complexity would have been O(n^2)

**Conclusion** :

In order to obtain minimal cost if we want to travel ‘n’ number of places exactly once which are connected with each other , we use this method.

**References**

* Geeks for Geeks