Assignment No: 10

Aim ⇒

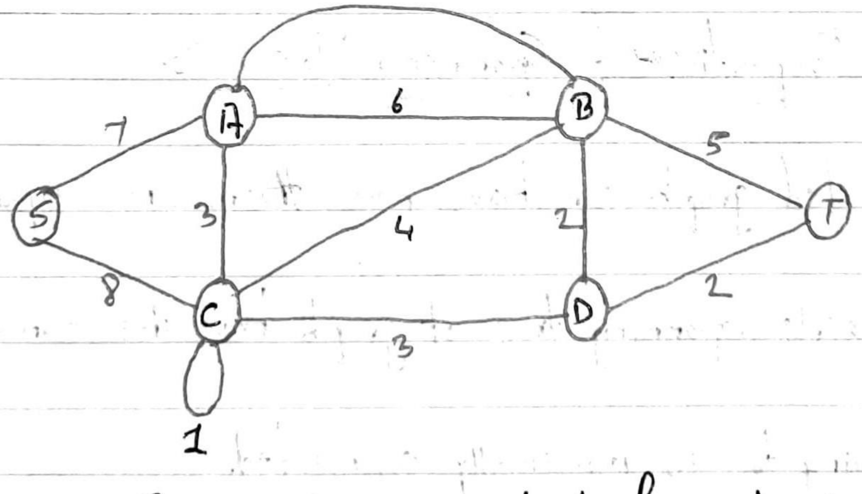
You have a bussiness with several offices you want to lease phone lines to connect them up with each other and phone company charges different amount of money to connect different pairs of cities you want to set af lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropiate data structure.

Theory ⇒

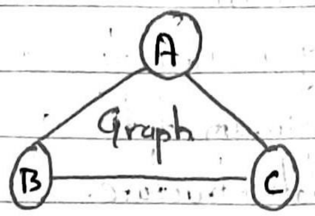
Prim's algorithm to find minimum cost spanning tree uses the greedy approach prim's algorithm shares a similarity with the shortest path first algorithm.

In contrast with kruskal's algorithm treats the node as a single tree and keeps on adding new nodes to the spanning tree form the given graph.

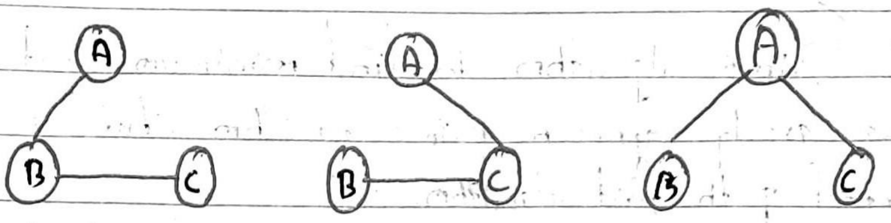
To construct with kruskal's algorithm and to understand prim algorithm better we shall we the same example.



Spanning tree is subset of graph G which has all the vertices covered with minimum possible number of edges. Hence a spanning tree does have cycles and it cannot be disconnected by this defination. We can draw conclusion that every connected and undirected graph has at least one spinning tree. A disconnecte graph doesnot have any spanning tree as it cannot be spanned to its vertices.



Spanning tree ⇒



We found three spanning trees off one complete graph; Complete undirected graph can have maximum n^{n-2} number of spanning tree where n is no.of nodes. In above example n is 3, so $3^{3-2} = 3^1 = 3$ spanning tress are possible

Properties of Spanning Tree ⇒

I) Connected graph can have more than one spanning tree.

II) All possible spanning tree of graph have the same no.of edges.

III) Spanning tree is minimally connected.

IV) Adding one edge to spanning tree will create a ~~cf~~ circuit or loop i.e spanning tree is maximal acyclic

Program Code:-

```
#include <iostream>
#include <iomanip>
using namespace std;

const int MAX = 10;
class EdgeList; // forward
declaration
class Edge      // USED IN
KRUSKAL
{
    int u, v, w;

public:
    Edge() {} // Empty
Constructor
    Edge(int a, int b, int weight)
    {
        u = a;
        v = b;
        w = weight;
    }
    friend class EdgeList;
    friend class PhoneGraph;
};
//---- EdgeList Class ----------
class EdgeList
{
    Edge data[MAX];
    int n;

public:
    friend class PhoneGraph;
    EdgeList()
    {
        n = 0;
    }
    void sort();
    void print();
};
//----Bubble Sort for sorting
edges in increasing weights'
order ---//
void EdgeList::sort()
{
    Edge temp;
    for (int i = 1; i < n; i++)
        for (int j = 0; j < n - 1;
j++)
            if (data[j].w > data[j +
1].w)
            {
                temp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = temp;
            }
}
void EdgeList::print()
{
    int cost = 0;
    for (int i = 0; i < n; i++)
    {
        cout << "\n"
             << i + 1 << " " <<
data[i].u << "--" << data[i].v
<< " = " << data[i].w;
        cost = cost + data[i].w;
    }
    cout << "\nMinimum cost of
```

```
Telephone Graph = " << cost;
}
//------------ Phone Graph
Class---------------
class PhoneGraph
{
    int data[MAX][MAX];
    int n;

public:
    PhoneGraph(int num)
    {
        n = num;
    }
    void readgraph();
    void printGraph();
    int mincost(int cost[], bool
visited[]);
    int prim();
    void kruskal(EdgeList
&spanlist);
    int find(int belongs[], int
vertexno);
    void unionComp(int
belongs[], int c1, int c2);
};
void PhoneGraph::readgraph()
{
    cout << "Enter
Adjacency(Cost) Matrix : \n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cin >> data[i][j];
    }
}
```

```
void PhoneGraph::printGraph()
{
    cout << "\nAdjacency
(COST) Matrix : \n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << setw(3) <<
data[i][j];
        }
        cout << endl;
    }
}
int PhoneGraph::mincost(int
cost[], bool visited[]) // finding
vertex with minimum cost
{
    int min = 9999, min_index;
// initialize min to MAX
value(ANY) as temporary
    for (int i = 0; i < n; i++)
    {
        if (visited[i] == 0 &&
cost[i] < min)
        {
            min = cost[i];
            min_index = i;
        }
    }
    return min_index; // return
index of vertex which is not
visited and having minimum
cost
}
int PhoneGraph::prim()
```

```
{
    bool visited[MAX];
    int parents[MAX];
    int cost[MAX]; // saving
minimum cost
    for (int i = 0; i < n; i++)
    {
        cost[i] = 9999; // set cost
as infinity/MAX_VALUE
        visited[i] = 0; // initialize
visited array to false
    }
    cost[0] = 0;     // starting
vertex cost
    parents[0] = -1; // make first
vertex as a root
    for (int i = 0; i < n - 1; i++)
    {
        int k = mincost(cost,
visited);
        visited[k] = 1;

        for (int j = 0; j < n; j++)
        {
            if (data[k][j] &&
visited[j] == 0 && data[k][j] <
cost[j])
            {
                parents[j] = k;
                cost[j] = data[k][j];
            }
        }
    }
    cout << "Minimum Cost
Telephone Map : \n";
    for (int i = 1; i < n; i++)
    {
        cout << i << " -- " <<
parents[i] << " = " << cost[i]
<< endl;
    }
    int mincost = 0;
    for (int i = 1; i < n; i++)
        mincost += cost[i]; //
data[i][parents[i]];
    return mincost;
}
//------- Kruskal's Algorithm
void
PhoneGraph::kruskal(EdgeList
&spanlist)
{
    int belongs[MAX]; //
Separate Components at start
(No Edges, Only vertices)
    int cno1, cno2;   //
Component 1 & 2
    EdgeList elist;
    for (int i = 1; i < n; i++)
        for (int j = 0; j < i; j++)
        {
            if (data[i][j] != 0)
            {
                elist.data[elist.n] =
Edge(i, j, data[i][j]); //
constructor for initializing edge
                elist.n++;
            }
        }
    elist.sort(); // sorting in
increasing weight order
    for (int i = 0; i < n; i++)
```

```
        belongs[i] = i;

    for (int i = 0; i < elist.n; i++)
    {
        cno1 = find(belongs,
elist.data[i].u); // find set of u
        cno2 = find(belongs,
elist.data[i].v); ////find set of v
        if (cno1 != cno2)
// if u & v belongs to different
sets
        {
            spanlist.data[spanlist.n]
= elist.data[i]; // ADD Edge to
spanlist
            spanlist.n = spanlist.n +
1;
            unionComp(belongs,
cno1, cno2); // ADD both
components to same set
        }
    }
}
void
PhoneGraph::unionComp(int
belongs[], int c1, int c2)
{
    for (int i = 0; i < n; i++)
    {
        if (belongs[i] == c2)
            belongs[i] = c1;
    }
}
int PhoneGraph::find(int
belongs[], int vertexno)
{
    return belongs[vertexno];
```

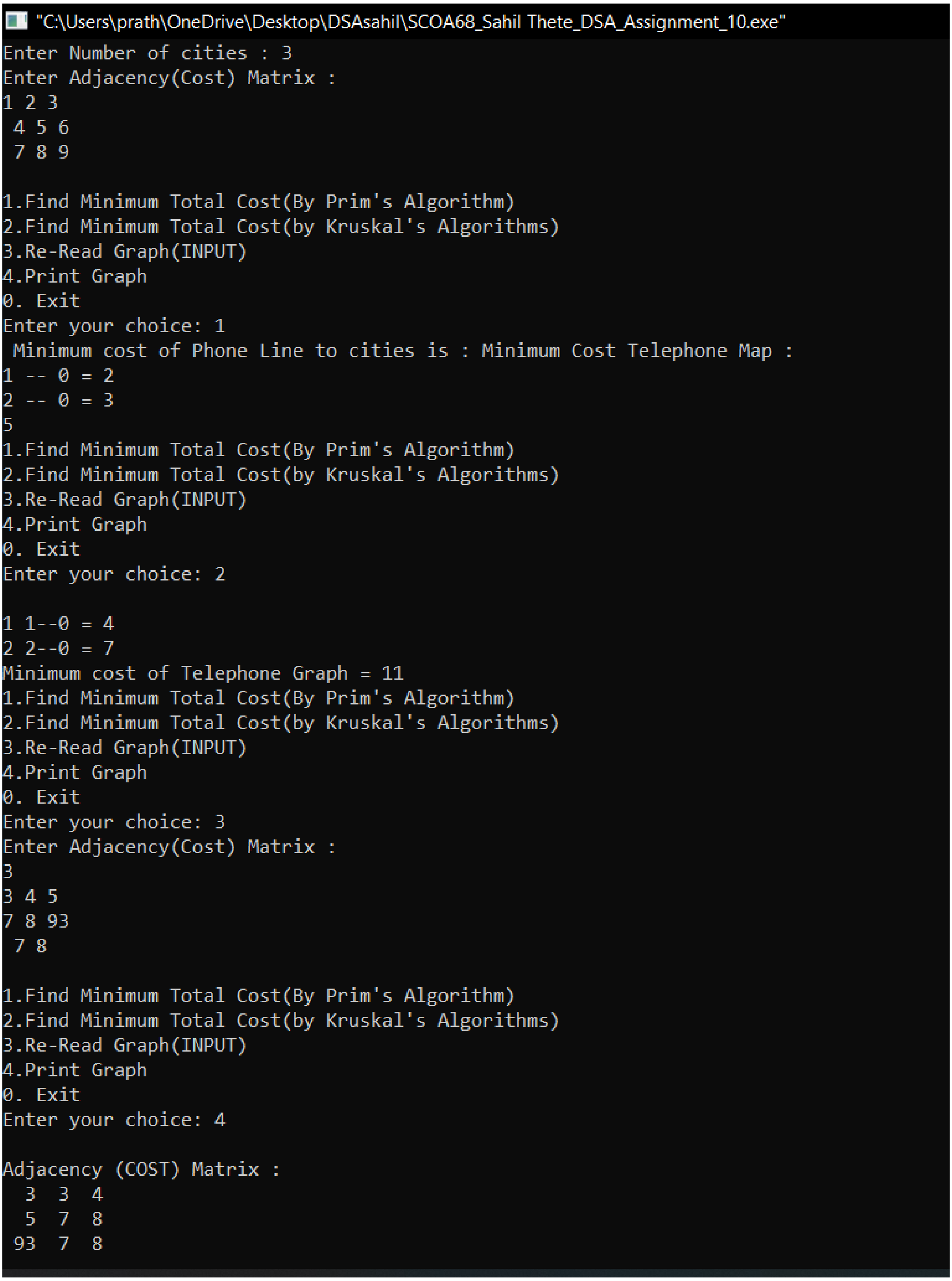
```
}


//--------- MAIN PROGRAM---
--------------------------------
int main()
{


    int vertices, choice;
    EdgeList spantree;
    cout << "Enter Number of
cities : ";
    cin >> vertices;
    PhoneGraph p1(vertices);
    p1.readgraph();
    do
    {
        cout << "\n1.Find
Minimum Total Cost(By Prim's
Algorithm)"
             << "\n2.Find Minimum
Total Cost(by Kruskal's
Algorithms)"
             << "\n3.Re-Read
Graph(INPUT)"
             << "\n4.Print Graph"
             << "\n0. Exit"
             << "\nEnter your
choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << " Minimum
cost of Phone Line to cities is :
" << p1.prim();
            break;
        case 2:
            p1.kruskal(spantree);
            spantree.print();
            break;
        case 3:
            p1.readgraph();
            break;
        case 4:
            p1.printGraph();
            break;
        default:
            cout << "\nWrong
Choice!!!";
        }
    } while (choice != 0);

    return 0;
}
```

Program Output :



"C:\Users\prath\OneDrive\Desktop\DSAsahil\SCOA68_Sahil Thete_DSA_Assignment_10.exe"

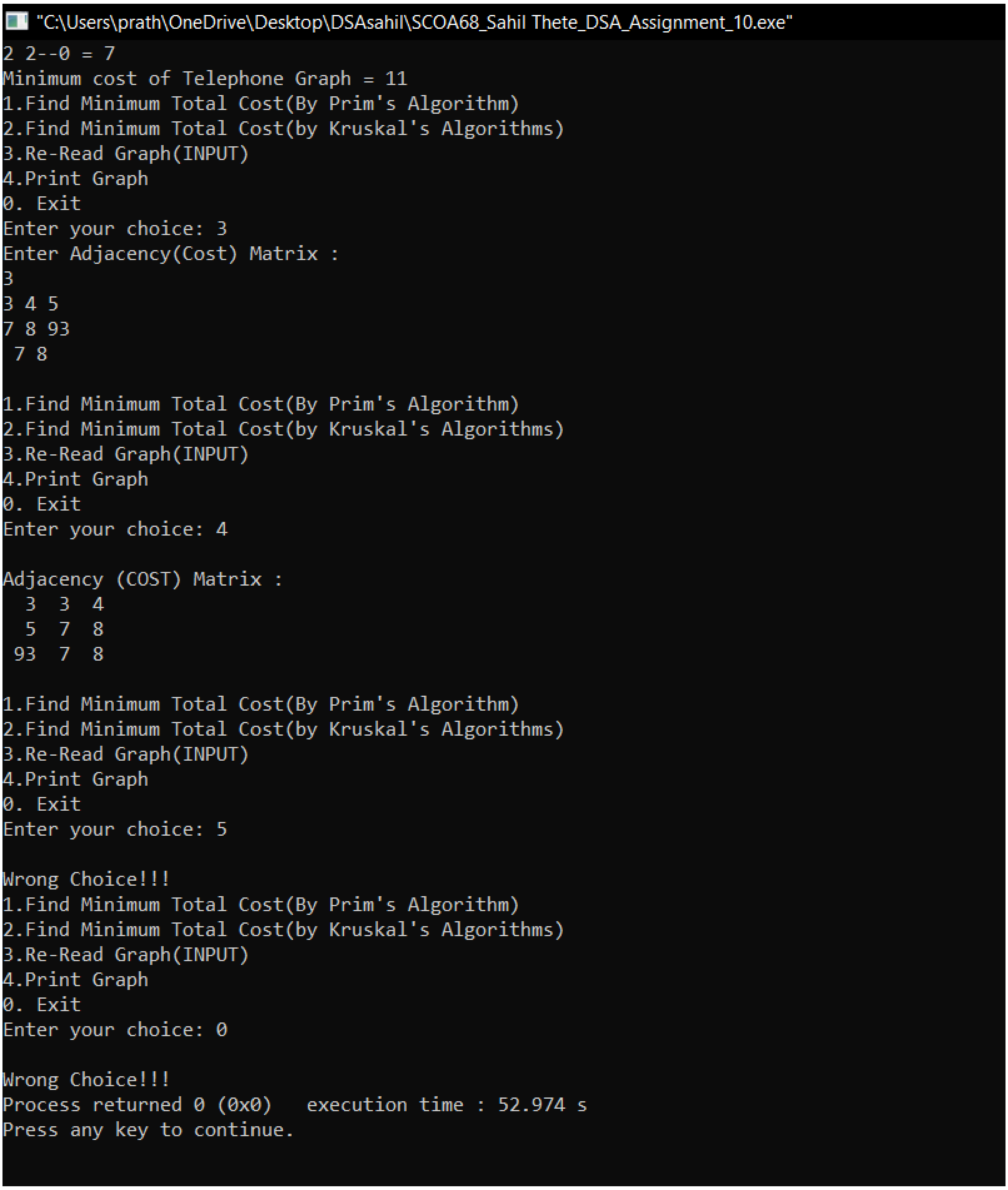
```
Enter Number of cities : 3
Enter Adjacency(Cost) Matrix :
1 2 3
 4 5 6
 7 8 9

1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 1
 Minimum cost of Phone Line to cities is : Minimum Cost Telephone Map :
1 -- 0 = 2
2 -- 0 = 3
5
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 2

1 1--0 = 4
2 2--0 = 7
Minimum cost of Telephone Graph = 11
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 3
Enter Adjacency(Cost) Matrix :
3
3 4 5
7 8 93
 7 8

1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 4

Adjacency (COST) Matrix :
  3  3  4
  5  7  8
 93  7  8
```



"C:\Users\prath\OneDrive\Desktop\DSAsahil\SCOA68_Sahil Thete_DSA_Assignment_10.exe"

```
2 2--0 = 7
Minimum cost of Telephone Graph = 11
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 3
Enter Adjacency(Cost) Matrix :
3
3 4 5
7 8 93
 7 8

1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 4

Adjacency (COST) Matrix :
  3  3  4
  5  7  8
 93  7  8

1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 5

Wrong Choice!!!
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 0

Wrong Choice!!!
Process returned 0 (0x0)   execution time : 52.974 s
Press any key to continue.
```