ASSIGNMENT 10

Aim : A business house has several offices in different countries ; they want to lease Phone lines to connect them with each other and the phone company charges different rent to connect different pairs of cities. Business house want to connect all its offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

Problem specifications :

· Data structures used :

Array : 2D array to store the adjacent vertices and the weight associated edges.
1D array to store whether the vertex is visited or not.

· Concepts used : Arrays, Functions to construct, display generate minimum spanning tree.

Theory : · A spanning tree of a graph $G = (V, E)$ is a sub graph of G having all vertices of G and no cycles in it.

· ~~A mi~~ Minimal spanning tree : Any tree, which consists solely of edges in graph G and includes all the vertices in G, is called as spaning tree.

Scanned with CamScanner

Thus for a given connected graph, there are multiple spanning trees possible. A spanning tree is called minimal ~~sp~~ cost spanning tree or simply minimal spanning tree of G if its cost is minimum.

- Cycle: If any edge from set B of graph G is introduced into the corresponding spanning tree T of graph G then cycle is formed. This cycle consists of edge (v, w) from the set B and all edges on the path from w to v in T.

- Prim's Algorithm: It is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that form a tree that includes every vertex, where the total weight of all the edges in the tree is minimised. The algorithm was discovered in 1930 by mathematician Vojtech Jarnik and later independently by computer scientist Robert C. Prim in 1957 and rediscovered by Edsger Dijkstra in 1959.

An arbitary node is chosen initially as the tree root. The nodes of the graph are then appended to the tree one at a time until all the nodes of the graph are included.

• Algorithm m:

Structure used

```
typedef struct edges
{
    int v1, v2, wt;
} edge;
```

① Accept graph (int G[][MAX], int n)

```
    declare int i, j;
    print ("Enter 0 if no edge present");
    for i <- 0 to n do
        for j = i+1 to n do
            print (Enter weight) :-
            read (G[i][j]);
            G[j][i] = G[i][j];
        end
    end
end procedure.
```

Ⓘ Algorithm int Adj to Edges (int G[][MAX], int n, int edge[]

```
    declare int i, j, k = 0;
    for int i = 0 to n do
        for j = i+1 to n to
            if G[i][j]
                E[k], v1 = i
                E[k]. v2 = j
                E(K++) . wt = G[i][j];
        end
    end
```

Scanned with CamScanner

(iii) Algorithm PrintEdges (Edge E[], int noe)

```
int i;
for i=0 to n do
    print ( E[i].v1, E[i].v2 , E[i].wt);
```

(iv) Algorithm SortEdges (edge E[], int noe)

```
int i,j;
edge t;

for i=0 to n do
    for j= i+1 to n do
        if (E[i].wt > E[j].wt)
            t= E[i];
            E[i] = E[j];
            E[j] = t;
        end if
    end
end
end procedure.
```

(v) Algorithm int total (edge E[], int noe)

```
int i, sum=0;

for i=0 to n
        sum += E[i].wt;
end for
return sum;
end procedure.
```

Scanned with CamScanner

(vi) algorithm int search (int tr[], int v, int n)

```
    declare int i;

    for i=0 to n do
        if (tr[i] == v)
            return 1;
        end if
        return 0;
    end for
end procedure.
```

(vii) Algorithm printAdj (int G[][MAX], int nov)

```
    declare int i,j;

    for i=0 to n do
        print v[i]
    end for
    for i=0 to n do
        print "v[i]"
        for j=0 to nov do
            print G[i][j]
        end
    end
end procedure.
```

Scanned with CamScanner

(viii) Algorithm prims (int s[][MAX], edge E[], int noe)

```
int TV [MAX], visited [MAX] = {0};
int i, j, k=0, l, vt1, vt2, v;
int edgeT[20];
TV [k++] = 0;

for (i=0 to n do)
    for j=0 to n do
        if (!visited [j])
            vt1 = E[j].v1;
            vt2 = E[j].v2;
            if (!search (TV, vt1, k) and !search (TV, vt2, k))
                s [vt1] [vt2] = E [j]. wt;
                s [vt2] [vt1] = E [j]. wt;
                TV [k++] = vt2;
                visited [j] = 1;
                break;
            end
            else if (search (TV, vt2, k) and !search
                                        (tv, vt1, k))
                s[vt1] [vt2] = E [j].wt;
                s [vt2] [vt1] = E [j]. wt;
                TV [k++] = vt1;
                visited [j] = 1;
                break;
            end
        end if
    end for
end for
end Prim.
```

Scanned with CamScanner

Conclusion: The cost of spaning tree of graph G is the sum of the costs of the edges in that tree. Any connected graph with n vertices must have at least n-1 edges and all connected graphs with n-1 edges are trees.

Scanned with CamScanner