16 Mar, 18

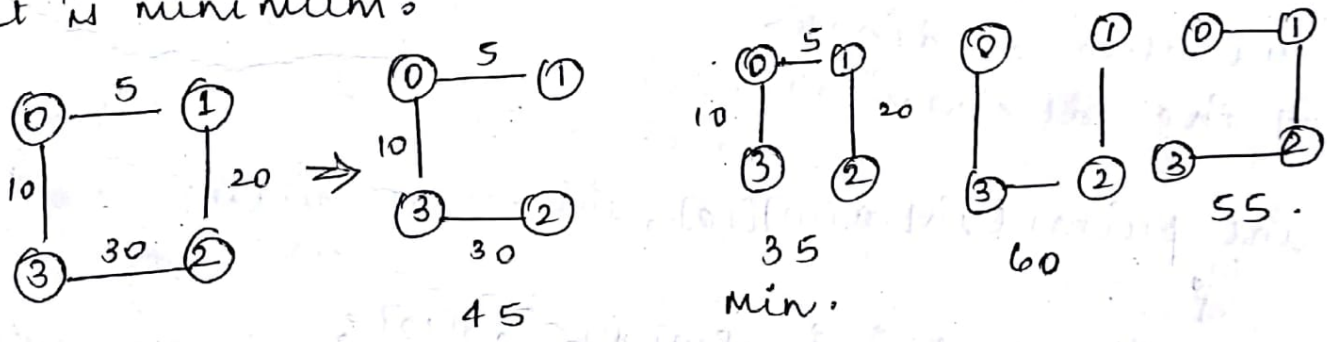
Module : 4
Greedy Programming

A greedy method is a problem solving technique that always tries to find best solution that works in stages considering one input at a time with the hope that we get an optimal solution.

* Minimum spanning tree (MST) :

The spanning tree is a tree in which all nodes are connected without forming cycle or closed path

The minimum spanning tree is a spanning tree whose cost is minimum.

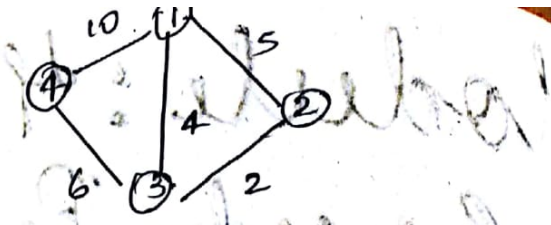


* Methods of obtaining a MST

1) Prim's Algorithm
2) Kruskal's Algorithm.

1) Prim's Algorithm :

	1	2	3	4
1	0	5	4	10
2	5	0	2	∞
3	4	2	0	6
4	10	∞	6	0



selected node 2 as starting node.

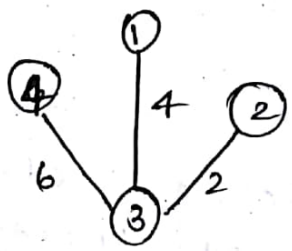
(2,1) = 5 ✗ → forms a cycle instead go to second lowest

(2,3) = 2 ✓

(3,1) = 4 ✓

(3,4) = 6 ✓

(1,4) = 10



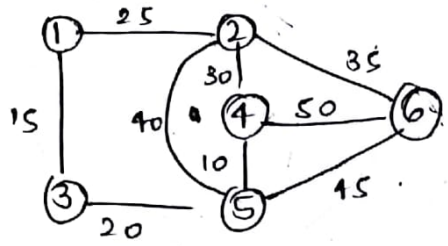
MST = 12

LAB PROGRAM : 8

```
#include <stdio.h>
#include <conio.h>

int prims (int a[10][10], int n, int source)
{
    int u, v, i, j, d[10], sum, s[10];
    for (i = 0; i <= n; i++)
    {
        s[i] = 0;
        d[i] = a[source][i];
    }
    s[source] = 1.
    sum.
    for (i = 1; i <= n-1; i++)
    {
        int min = 999;
        for (j = 1; j <= n; j++)
        if (s[j] == 0)
```

```
        if (d[j] < min)
        {
            min = d[j];
            u = j;
        }
        s[u] = 1;   // making node visited.
        sum += d[u];
        for(v = 1; v<=n; v++)
        {
            if (s[v]==0 && a[u][v]<d[v])
                d[v] = a[u][v];
        }
    }
        return sum;
    }
void main()
{
    int n, a[10][10], sum, i, j, source;
    clrscr();
    pf("Enter the no. of nodes\n");
    sf("%d", &n);
    pf("Enter the cost adjacency matrix");
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            sf("%d", &a[i][j]);
    pf("Enter source\n");
    sf("%d", &source);
    sum = prims(a, n, source);
    if (sum >= 999)
        pf("MST does not exist\n");
    else
        pf("cost of MST is %d\n", sum);
    getch();
}
```

Example:



Source = 1

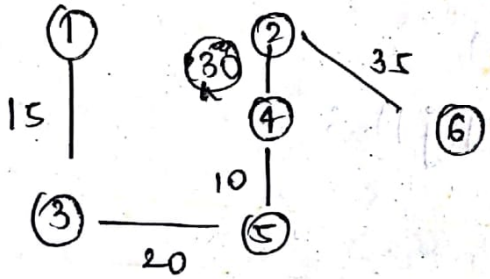
	1	2	3	4	5	6	
→ 1	0	25	15	∞	∞	∞	←
→ 2	25	0	∞	30	40	35	←
→ 3	15	∞	0	∞	20	∞	←
→ 4	∞	30	∞	0	10	50	←
→ 5	∞	40	20	10	0	45	←
6	∞	35	∞	50	45	0	

(2,1) = 25
(2,4)

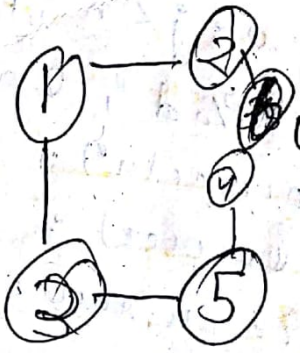
(1,2) = 25
(1,3) = 15 ✓
(3,5) = 20 ✓

(5,2) = 40
(5,4) = 10 ✓
(5,6) = 45

(4,2) = 30 ✓
(4,6) = 50



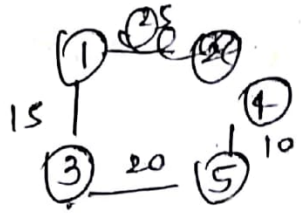
1
15
20
10
30
35
110



(2,1) = 25 ✓ X
(2,5) = 40
(2,6) = 35 ✓

(6,4) = 50 X
(6,5) = 45 X

(1, 2) = 25 ✓
(1, 3) = 15 ✓

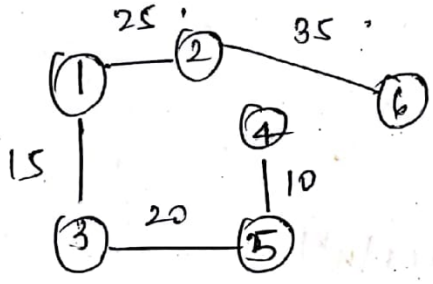


(3, 5) = 20 ✓

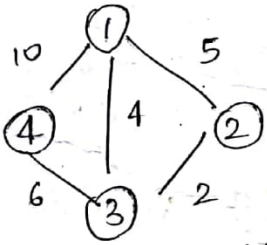
(5, 2) = 40 ✓
(5, 4) = 10 ✓
(5, 6) = 45 ✓

(4, 2) = 30 ✗
(4, 6) = 50

(2, 6) = 35 ✓

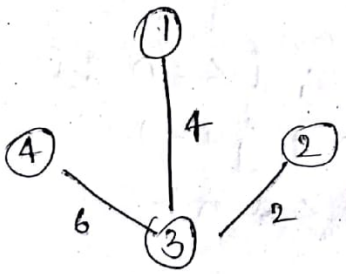


2) Kruskal's Algorithm :-



Step 1: Arrange the edges according to their cost in ascending order
Step 2: Start selecting the edges
Step 3: If the selected edge forms a cycle discard it and choose the next edge

	✓	✓	✗	✓	✗
Edge	(2, 3)	(1, 3)	(1, 2)	(4, 3)	(4, 1)
cost	2	4	5	6	10



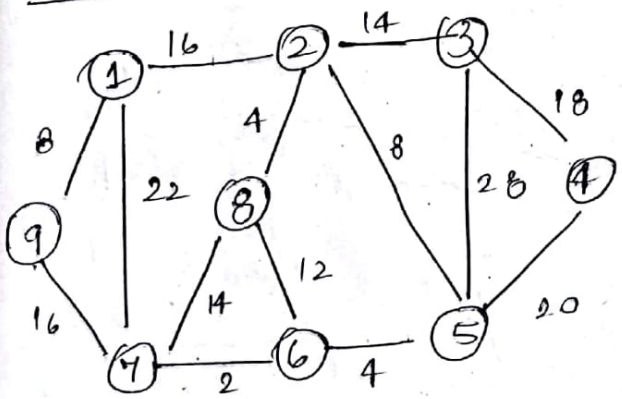
MST = 12

LAB PROGRAM 4

```
#include <stdio.h>
#include <conio.h>
int parent[20] = {0}, min, mincost=0, ne=1, n,
  cost[20][20];
int a, b, i, j, u, v;
void kruskal(void);

void main()
{
  int
  clrscr();
  printf("Enter the no. of nodes\n");
  scanf("%d", &n);
  printf("Enter the cost matrix");
    for(i=0; i<n; i++)
      for(j=0; j<n; j++)
        {
          scanf("%d", &cost[i][j]);
          if(cost[i][j]==0)
            cost[i][j]=999;
        }
    kruskal();
  getch();
}
void kruskal()
{
  while(ne<n)
  {
    for(i=1; min=999; i<=n; i++)
        for(j=1; j<=n; j++)
            if(cost[i][j]<min)
            {
```

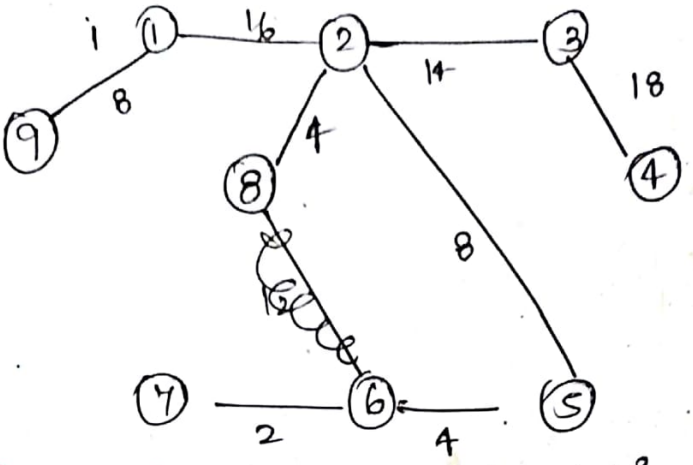
```
        min = cost[i][j];
        a = u = i;
    }   b = v = j;
    while (parent[u])
        u = parent[u];
    while (parent[v])
        v = parent[v];
    if (u != v)
    {
        printf("%d Edge (%d, %d) = %d\n",
                                    ne++, a, b, min);

        mincost += min;
        parent[v] = u;
    }
    cost[a][b] = cost[b][a] = 999;
}
printf("The minimum cost is %d\n", mincost);
}.
```

Examples -



(6,7) (5,6) (2,8) (1,9) (2,5) (6,8) (2,3) (7,8)
2 4 4 8 8 12 14 14

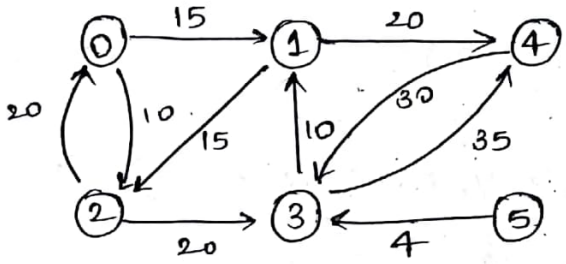
(1,2) (7,9) (3,4) (4,5) (1,7) (3,5)
16 16 18 20 22 28



MST = 44

Dijkstra's Algorithm :- (Single source shortest pa

solve the foll. single source shortest path probl
assuming vertex 5 as the source.



M →

N ↓	0	1	2	3	4	5
0	0	15	10	∞	∞	∞
1	∞	0	15	∞	20	∞
2	20	∞	0	20	∞	∞
3	∞	10	∞	0	35	∞
4	∞	∞	∞	30	0	∞
5	∞	∞	∞	4	∞	0

source = 5

d[0] =	∞	∞	∞	49
d[1] =	∞	14	14	14
d[2] =	∞	∞	29	29
d[3] =	4	4	4	4
d[4] =	∞	39	34	34
d[5] =	0	0	0	0

S	unvisited node	d[v] = min(d[v], d[u] + w[u][v])	u, d[u]
5	0, 1, 2, 3, 4	-	3, 4
5, 3	0, 1, 2, 4	d[0] = min(∞, 4 + ∞) = ∞ d[1] = min(∞, 4 + 10) = 14 d[2] = min(∞, 4 + ∞) = ∞ d[4] = min(∞, 4 + 35) = 39	1, 14
5, 3, 1	0, 2, 4	d[0] = min(∞, 14 + ∞) = ∞ d[2] = min(∞, 14 + 15) = 29 d[4] = min(39, 14 + 20) = 34	2, 29
5, 3, 1, 2	0, 4	d[0] = min(∞, 29 + 20) = 49 d[4] = min(34, 29 + ∞) = 34	4, 34
5, 3, 1, 2, 4	0	d[0] = min(49, 34 + ∞) = 49	0, 49
5, 3, 1, 2, 4, 0	-	-	-

Parallel.

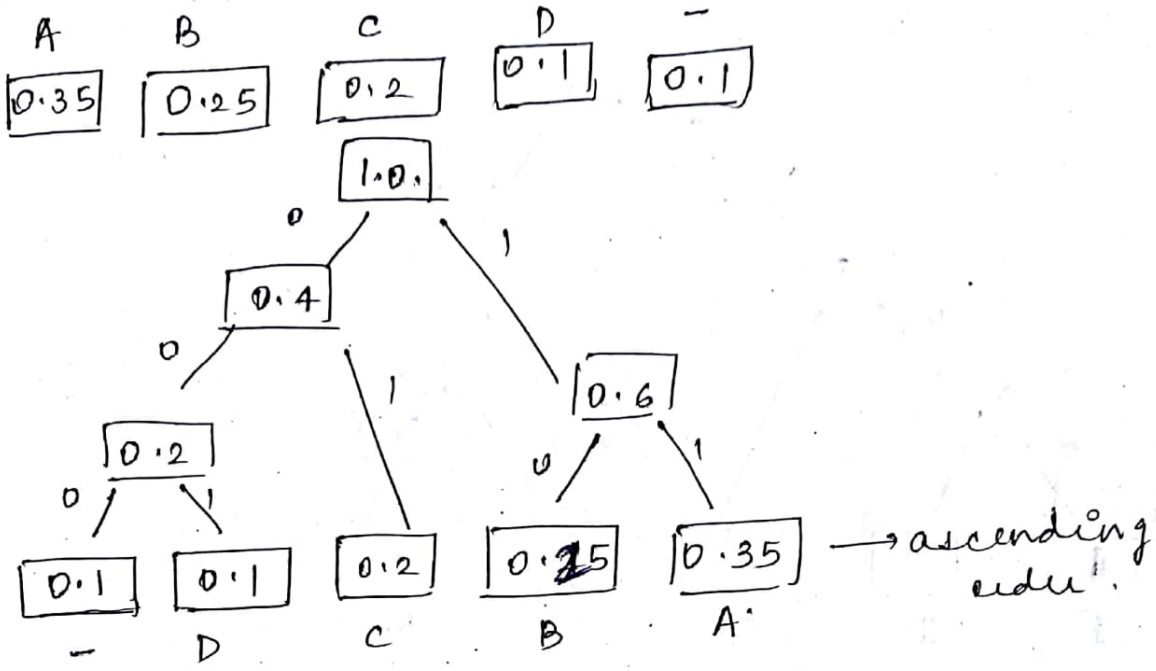
LP : 9

```
#include <stdio.h>
#include <conio.h>
void dijk(int cost[10][10], int n, int source, int v[10]
                                    int d[10]);

void main()
{
  int n, cost[10][10], source, v[10], d[10], i, j;
  clrscr();
  printf("Enter the no. of nodes");
  scanf("%d", &n);
  printf("Enter the cost adjacency matrix");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d", &cost[i][j]);
```

```
pf("Enter the source node\n");
  sf("%d", &source);
for(i=1; i<=n; i++)
{
  d[i] = cost[source][i];
  v[i] = 0;
}
dijk(cost, n, source, v, d);
for(i=0; i<n; i++)
  pf("%d -> %d = %d\n", source, i, d[i]);

void dijk(int cost[10][10], int n, int source, int v
                                    int d[10])
{
  int least, i, j, u;
  v[source] = 1;
  for(i=1; i<=n; i++)
  {
    least = 999;
    for(j=1; j<=n; j++)
    {
      if(v[j]==0 && d[j]<least)
      {least = d[j];
       u = j;
      }
    }
    v[u]=1;
    for(j=1; j<=n; j++)
    {
      if(v[j]==0 && (d[j]>(d[u]+cost[u
        d[j] = d[u]+cost[u][j];
      }
    }
  }
```

pf("The shortest distance is:\n")

Huffman Tree and Prefix codes:



– = 000 C = 01 A = 11 ⇒ Prefix code.

D = 001 B = 10

AB_C = 11100001

step 1: Arrange the symbols in ascending order according to their probability

step 2: Take 2 min. probability add the probability and promote it to new node.

step 3: Repeat step 2 until we get a final tree

step 4: For the Huffman tree obtained, label left child with 0 and right child with 1 and obtain prefix codes (traversed from the root node)

Q construct a Huffman tree and obtain optimal prefix codes for the symbols in the message "engineering." and encode the message.

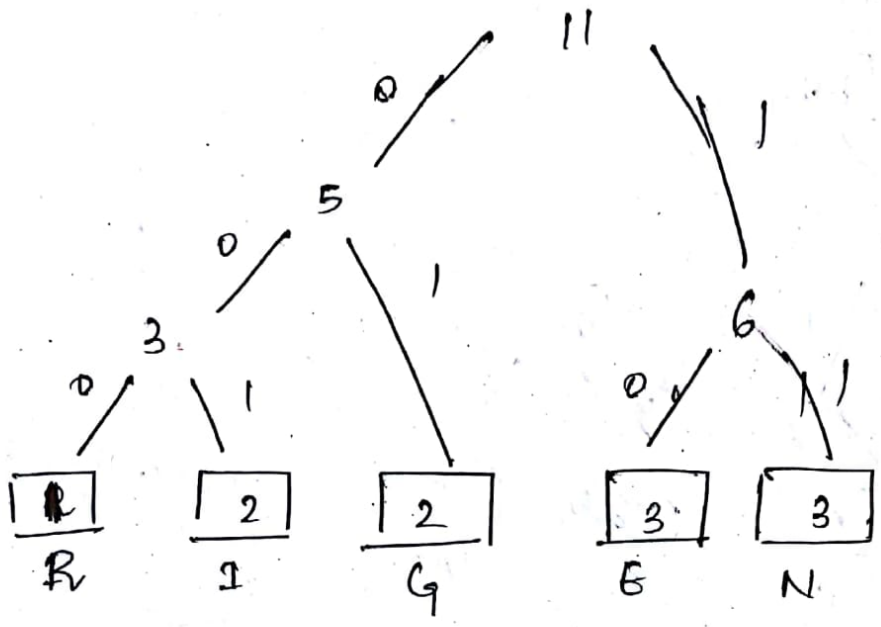
E = 3

N = 3

G = 2

I = 2

R = 1



R = 000 E = 10

I = 001 N = 11

G = 01