```
int main()
{
    int graph[v][v] = { { 0 , 4 , INF , 5 , INF },
                        { INF, 0 , 1 , INF, 6 },
                        { 2 , INF, 3 , INF , },
                        { INF, INF, 4 , 0 } };
    floydWarshall(graph);
    return 0 ;
}
```

o/p :-

The following matrix shows the shortest distance b/w every pair of vertices

```
0    4    5    5
3    0    1    4    6
2    6    0    3    5
3    7    1    0    2
1    5    5    4    0
```

2/6/24

4/7/24

1) prims :-

```
#include<stdio.h>
#include<limits.h>
#define MAX 100
void prims(int n, int cost[MAX][MAX], int INF){
    int s[MAX], d[MAX], p[MAX], T[MAX][2], sum=0, i, j, k,
                                                min, u;

    min = INF ;
    int source = 0;
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            if(cost[i][j] != 0 && cost[i][j] < min){
                min = cost[i][j];
                source = i;
```

```
for(i=0;  i<n;  i++){
        s[i] = 0;
        d[i] = cost[source][i];
        p[i] = source;
    }
    s[source] = 1;
    k=0
    for(i=1;  i<n;  i++){
            min = INF;
            u = -1;
            for(j=0;  j<n;  j++){
                if(s[j] == 0 && d[j] <= min){
                        min = d[j];
                        u = j;
                }
            }
            T[k][0] = u;
            T[k][1] = p[u];
            k++;
            sum += cost[u][p[u]];
            s[u] = 1;
            for(j=0; j<n; j++){
                if(s[j] == 0 && cost[u][j] < d[j]){
                    d[j] = cost[u][j];
                    p[j] = u;
                }
            }
    }
    if(sum >= INF){
        printf("Spanning tree does not exist\n");
    }else{
        printf("Spanning tree exists & MST is:\n");
        for(i=0;  i<n-1;  i++){
            printf("%d, %d\n", T[i][0], T[i][1]);
        }
        printf("The cost of spanning tree (mst) is:
                %d\n", sum);
    }
}
int main(){
    int n, cost[MAX][MAX], i, j;
    int INF = INT_MAX;
    printf("Enter the number of vertices");
    scanf("%d", &n);
```

```
printf("Enter the cost adjacency matrix (enter 0 for no edges, enter
 INF for infinite distance):\n");
for(i=0; i<n; i++){
    for(j=0; j<n; j++){
        scanf("%d", &cost[i][j]);
        if(cost[i][j]==9999)
            cost[i][j] = INF;
    }
}
prims(n, cost, INF);
return 0;
}
```

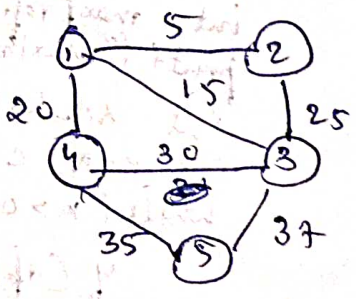
o/p:-

Enter the number of vertices: 5
Enter the cost adjacency matrix (enter 0 for no edge, enter INF for infinite distance);

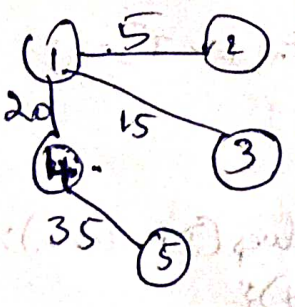
0	5	15	20	9999
5	0	25	9999	9999
15	25	0	30	37
20	9999	30	0	35
9999	9999	37	35	0

Spanning tree edges and MST is:

(0, 1)
(0, 2)
(0, 3)
(0, 4)



The cost of spanning tree (MST) is: 75



2) Knapsack :

```
#include <stdio.h>
#define N 4
#define CAPACITY 7

int max(int a, int b){
    if (a>b){
        return a;
    }
    return b;
}

void knapsack(int weights[], int profits[]){
    int i, w;
    int dp[N+1][CAPACITY+1];
    for(i=0; i<=N; i++){
        for(w=0; w<=CAPACITY; w++){
            if (i==0 || w==0)
                dp[i][w] = 0;
            else if (weights[i-1] <= w)
                dp[i][w] = max(profits[i-1] + dp[i-1]
                    [w-weights[i-1]], dp[i-1][w]);
            else
                dp[i][w] = dp[i-1][w];
        }
    }
    int maxProfit = dp[N][CAPACITY];
    printf("Maximum profit: %d \n", maxProfit);
    int selectedObjects[N];
    int k=N, c=CAPACITY;
    while (k>0 && c>0){
        if (dp[k][c] != dp[k-1][c]){
            selectedObjects[k-1] = 1;
            c = c - weights[k-1];
        } else{
            selectedObjects[k-1]=0;
        }
        k--;
    }
    printf("\n Table values (DP Table): \n");
    for(i=0; i<=N; i++){
        for(w=0; w<=CAPACITY; w++){
            printf("%4d ", dp[i][w]);
        }
```

```
    printf("\n");
}
printf("\n Objects Selected in the Knapsack :\n");
for(i=0; i<N; i++){
    if(selectedObjects[i] == 1)
        printf("Objects :%d (weight : %d, Profit : %d)\n",
               i+1, weights[i], profits[i]);
    }
}
int main(){
    int weights[N];
    int profits[N];
    printf("Enter the weights :\n");
    for(int i=0; i<N; i++){
        scanf("%d", &weights[i]);
    }
    printf("Enter the profits :\n");
    for(int i=0; i<N; i++){
        scanf("%d", &profits[i]);
    }
    printf("Knapsack Capacity : %d\n", CAPACITY);
    printf("Objects :\n");
    for(int i=0; i<N; i++){
        printf("Object %d- weight : %d, Profit : %d\n", i+1,
               weights[i], profits[i]);
    }
    knapsack(weights, profits);
    return 0;
}
```

o/p :- Enter the weights : 1 3 4 5
Enter the profits : 1 4 5 7
Knapsack capacity : 7
Object :
object 1 - weight : 1 , profit 1
object 2 - weight : 3 , profit 4
object 3 - weight : 4 , profit 5
object 4 - weight : 5 , profit 7

Table values (TOP Task):

0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1
0	1	1	4	5	5	5	5
0	1	1	4	5	6	6	9
0	1	1	4	5	7	8	9

objects selected in knapsack
object 2 (weight : 3, profit : 4)
object 3 (weight : 4, profit : 5)