SATYUG DARSHAN INSTITUTE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(SESSION: 2020- 2021)

B.Tech CSE – 4th SEM

Lab File

Of

Design and Analysis of Algorithm (DAA)

Submitted By: - Deepak

Roll No. : - CSE-19/034

Submitted To:-

Ms. Nidhi Sharma

INDEX

Malloc, Calloc and Re-Alloc

```c
#include<stdio.h>
#include<stdlib.h>

void main(){
    int i,a,b,num;
    int *p;
    printf("Enter 1 for malloc or 0 for calloc: ");
    scanf("%d",&num);

    if(num == 1){
         p =(int*)malloc(a*sizeof(int));
         printf("\n\tMalloc Implementation\n\n");
    }else if(num == 0){
        p =(int*)calloc(a,sizeof(int));
          printf("\n\tCalloc Implementation\n\n");
    }else{
        printf("\n \tNot Valid\n\n");
        exit(0);
    }
    printf("Enter the number to have to be print : ");
    scanf("%d",&a);

    if(p==NULL){
        printf("not ___________");
        exit(0);
    }else{
        for(i=0;i<a;i++){
            p[i]= i+1;
        }
        for(i=0;i<a;i++){
            printf("your numbers are : %d\n",p[i]);
        }
        printf("\n\n ----------> Re-allocation <----------\n\n");
        printf("Enter how many numbers you want: ");
        scanf("%d",&b);

         for(i=a;i<b;i++){
            p[i]= i+1;
        }
        for(i=0;i<b;i++){
            printf("your numbers are : %d\n",p[i]);
        }
    }
    return 0;
}
```

```
"D:\COLLAGE\Semester 4\_DAA\Program\DynamicProgram.exe"
Enter 1 for malloc or 0 for calloc: 1

        Malloc Implementation

        Enter the number to have to be print : 5
        your numbers are : 1
        your numbers are : 2
        your numbers are : 3
        your numbers are : 4
        your numbers are : 5


 ----------> Re-allocation <----------

        Enter how many numbers you want: 2
        your numbers are : 1
        your numbers are : 2

Process returned 2 (0x2)   execution time : 6.322 s
Press any key to continue.
```

Depth First Search (DFS)

```c
#include<stdio.h>

void DFS(int);
int G[10][10],visited[10],n;    //n is no of vertices and graph i
void main()
{
    int i,j;
    printf("Enter number of vertices:");

scanf("%d",&n);
    //read the adjecency matrix
printf("\nEnter adjecency matrix of the graph:");

for(i=0;i<n;i++)
       for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
    //visited is initialized to zero
   for(i=0;i<n;i++)
        visited[i]=0;
    DFS(0);
}
void DFS(int i)
{
    int j;
printf("\n%d",i);
    visited[i]=1;
for(j=0;j<n;j++)
       if(!visited[j]&&G[i][j]==1)
            DFS(j);
}
```

```
"D:\COLLAGE\Semester 4\_DAA\Program\DFS (2).exe"
Enter number of vertices:3

Enter adjecency matrix of the graph:
0 1 1
0 1 0
1 0 0

        0
        1
        2
Process returned 3 (0x3)   execution time : 6.425 s
Press any key to continue.
```

Breath First Search (BFS)

```c
#include <stdlib.h>
#include <stdbool.h>
#define MAX 5

struct Vertex {
   char label;
   bool visited;
};
//queue variables
int queue[MAX];
int rear = -1;
int front = 0;
int queueItemCount = 0;

//graph variables
//array of vertices
struct Vertex* lstVertices[MAX];
//adjacency matrix
int adjMatrix[MAX][MAX];
//vertex count
int vertexCount = 0;
//queue functions
void insert(int data) {
   queue[++rear] = data;
   queueItemCount++;
}
int removeData() {
   queueItemCount--;
   return queue[front++];
}
bool isQueueEmpty() {
   return queueItemCount == 0;
}

//graph functions
//add vertex to the vertex list
void addVertex(char label) {
   struct Vertex* vertex = (struct Vertex*) malloc(sizeof(struct Vertex));
   vertex->label = label;
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
}
void addVertex(char label) {
   struct Vertex* vertex = (struct Vertex*) malloc(sizeof(struct Vertex));
   vertex->label = label;
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
}
//add edge to edge array
void addEdge(int start,int end) {
   adjMatrix[start][end] = 1;
   adjMatrix[end][start] = 1;
}
//display the vertex
void displayVertex(int vertexIndex) {
   printf("%c ",lstVertices[vertexIndex]->label);
}
//get the adjacent unvisited vertex
int getAdjUnvisitedVertex(int vertexIndex) {
   int i;

   for(i = 0; i<vertexCount; i++) {
      if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false)
         return i;
   }
   return -1;
}
void breadthFirstSearch() {
   int i;
   //mark first node as visited
   lstVertices[0]->visited = true;
   //display the vertex
   displayVertex(0);
   //insert vertex index in queue
   insert(0);
   int unvisitedVertex;

   while(!isQueueEmpty()) {
      //get the unvisited vertex of vertex which is at front of the queue
      int tempVertex = removeData();
      //no adjacent vertex found
      while((unvisitedVertex = getAdjUnvisitedVertex(tempVertex)) != -1) {
         lstVertices[unvisitedVertex]->visited = true;
         displayVertex(unvisitedVertex);
         insert(unvisitedVertex);
      }
   }
   //queue is empty, search is complete, reset the visited flag
```

```
71      int unvisitedvertex;
72
73      while(!isQueueEmpty()) {
74         //get the unvisited vertex of vertex which is at front of the queue
75         int tempVertex = removeData();
76         //no adjacent vertex found
77         while((unvisitedVertex = getAdjUnvisitedVertex(tempVertex)) != -1) {
78            lstVertices[unvisitedVertex]->visited = true;
79            displayVertex(unvisitedVertex);
80            insert(unvisitedVertex);
81         }
82      }
83      //queue is empty, search is complete, reset the visited flag
84      for(i = 0;i<vertexCount;i++) {
85         lstVertices[i]->visited = false;
86      }
87   }
88
89   int main() {
90      int i, j;
91
92      for(i = 0; i<MAX; i++){ // set adjacency {
93         for(j = 0; j<MAX; j++) // matrix to 0
94            adjMatrix[i][j] = 0;
95      }
96      addVertex('S');   // 0
97      addVertex('A');   // 1
98      addVertex('B');   // 2
99      addVertex('C');   // 3
100     addVertex('D');   // 4
101
102     addEdge(0, 1);    // S - A
103     addEdge(0, 2);    // S - B
104     addEdge(0, 3);    // S - C
105     addEdge(1, 4);    // A - D
106     addEdge(2, 4);    // B - D
107     addEdge(3, 2);    // C - B
108
109     printf("\nBreadth First Search: ");
110
111     breadthFirstSearch();
112
113     return 0;
114  }
115
```

```
"D:\COLLAGE\Semester 4\_DAA\Program\BFS2.exe"

        Breadth First Search: S A B C D


Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

Brute Force (Pattern Matching)

```c
#include<stdio.h>
int match(char*, char*);

int main()
{
    char a[100], b[100];
    int position;
        printf("Enter some text\n");
        gets(a);
    printf("Enter a string to find\n");
    gets(b);

    position = match(a, b);
    if(position!=-1)
       printf("Found at location %d\n", position+1);
    else{   printf("Not found.\n");}

    return 0;
}
int match(char *a, char *b)
{
    int c;
    int position = 0;
    char *x, *y;
    x = a;
    y = b;
    while(*a)
    {
       while(*x==*y)
       {
          x++;
          y++;
          if(*x=='\0'||*y=='\0')
             break;
       }
       if(*y=='\0'){  break;}
         a++;
       position++;
       x = a;
       y = b;
    }
    if(*a)
       return position;
    else
       return -1;
}
```

```
"D:\COLLAGE\Semester 4\_DAA\Program\patternMatch.exe"
Enter some text
My name is Deepak
Enter a string to find
Deepak
Found at location 12

Process returned 0 (0x0)   execution time : 7.976 s
Press any key to continue.
```

Fraction Knapsack Problem

```c
#include <stdio.h>
int n = 5 , num; /* The number of objects */
int c[10] = {12, 1, 2, 1, 4}; /* c[i] is the *COST* of the ith object; i.e. what
                YOU PAY to take the object */
int v[10] = {4, 2, 2, 1, 10}; /* v[i] is the *VALUE* of the ith object; i.e.
                what YOU GET for taking the object */
int W = 15; /* The maximum weight you can take */

void simple_fill() {
    int cur_w;
    float tot_v;
    int i, maxi;
    int used[10];

    for (i = 0; i < n; ++i)
        used[i] = 0; /* I have not used the ith object yet */

    cur_w = W;
    while (cur_w > 0) { /* while there's still room*/
        /* Find the best object */
        maxi = -1;
        for (i = 0; i < n; ++i)
            if ((used[i] == 0) &&
                ((maxi == -1) || ((float)v[i]/c[i] > (float)v[maxi]/c[maxi])))
                maxi = i;

        used[maxi] = 1; /* mark the maxi-th object as used */
        cur_w -= c[maxi]; /* with the object in the bag, I can carry less */
        tot_v += v[maxi];
        if (cur_w >= 0)
            printf("Added object %d ($d$, %dKg) completely in the bag. Space left: %d.\n", maxi + 1, v[maxi], c[maxi], cur_w);
        else {
                num =(int)((1 + (float)cur_w/c[maxi]) * 100), v[maxi], c[maxi], maxi + 1;
            printf("Added %d%% ($d$, %dKg) of object %d in the bag.\n", num);
            tot_v -= v[maxi];
            tot_v += (1 + (float)cur_w/c[maxi]) * v[maxi];
        }
    }

    printf("Filled the bag with objects worth %.2f$.\n", tot_v);
}

int main(int argc, char *argv[]) {
    simple_fill();

    return 0;
```

```
"D:\COLLAGE\Semester 4\_DAA\Program\Gknapsak.exe"
Added object 5 (10$, 4Kg) completely in the bag. Space left: 11.
Added object 2 (2$, 1Kg) completely in the bag. Space left: 10.
Added object 3 (2$, 2Kg) completely in the bag. Space left: 8.
Added object 4 (1$, 1Kg) completely in the bag. Space left: 7.
Added 58% (1$, 1Kg) of object 7 in the bag.
Filled the bag with objects worth 17.33$.

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Topologcal Sorting

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 100

int n;    /*Number of vertices in the graph*/
int adj[MAX][MAX]; /*Adjacency Matrix*/
void create_graph();

int queue[MAX], front = -1,rear = -1;
void insert_queue(int v);
int delete_queue();
int isEmpty_queue();

int indegree(int v);

int main()
{
        int i,v,count,topo_order[MAX],indeg[MAX];
        create_graph();
        /*Find the indegree of each vertex*/
        for(i=0;i<n;i++)
        {
                indeg[i] = indegree(i);
                if( indeg[i] == 0 )
                        insert_queue(i);
        }
        count = 0;

        while(  !isEmpty_queue( ) && count < n )
        {
                v = delete_queue();
        topo_order[++count] = v; /*Add vertex v to topo_order array*/
                /*Delete all edges going from vertex v */
                for(i=0; i<n; i++)
                {
                        if(adj[v][i] == 1)
                        {
                                adj[v][i] = 0;
                                indeg[i] = indeg[i]-1;
                                if(indeg[i] == 0)
                                        insert_queue(i);
                        }
                }
        }
        }

        if( count < n )
        {
                printf("\nNo topological ordering possible, graph contains cycle\n");
                exit(1);
        }
        printf("\nVertices in topological order are :\n");
        for(i=1; i<=count; i++)
                printf( "%d ",topo_order[i] );
        printf("\n");

        return 0;
}/*End of main()*/

void insert_queue(int vertex)
{
        if (rear == MAX-1)
                printf("\nQueue Overflow\n");
        else
        {
                if (front == -1)  /*If queue is initially empty */
                        front = 0;
                rear = rear+1;
                queue[rear] = vertex ;
        }
}/*End of insert_queue()*/

int isEmpty_queue()
{
        if(front == -1 || front > rear )
                return 1;
        else
                return 0;
}/*End of isEmpty_queue()*/

int delete_queue()
{
        int del_item;
        if (front == -1 || front > rear)
        {
                printf("\nQueue Underflow\n");
                exit(1);
        }
```

```
                exit(1);
        }
        else
        {
                del_item = queue[front];
                front = front+1;
                return del_item;
        }
}/*End of delete_queue() */

int indegree(int v)
{
        int i,in_deg = 0;
        for(i=0; i<n; i++)
                if(adj[i][v] == 1)
                        in_deg++;
        return in_deg;
}/*End of indegree() */

void create_graph()
{
        int i,max_edges,origin,destin;

        printf("\nEnter number of vertices : ");
        scanf("%d",&n);
        max_edges = n*(n-1);

        for(i=1; i<=max_edges; i++)
        {
                printf("\nEnter edge %d(-1 -1 to quit): ",i);
                scanf("%d %d",&origin,&destin);

                if((origin == -1) && (destin == -1))
                        break;

                if( origin >= n || destin >= n || origin<0 || destin<0)
                {
                        printf("\nInvalid edge!\n");
                        i--;
                }
                else
                        adj[origin][destin] = 1;
        }
}
```

```
"D:\COLLAGE\Semester 4\_DAA\Program\webTopo.exe"

Enter number of vertices : 3

Enter edge 1(-1 -1 to quit): 0 1

Enter edge 2(-1 -1 to quit): 0 2

Enter edge 3(-1 -1 to quit): 2 1

Enter edge 4(-1 -1 to quit): -1 -1

Vertices in topological order are :
0 2 1

Process returned 0 (0x0)   execution time : 7.992 s
Press any key to continue.
```

Job sequencing with Deadlines

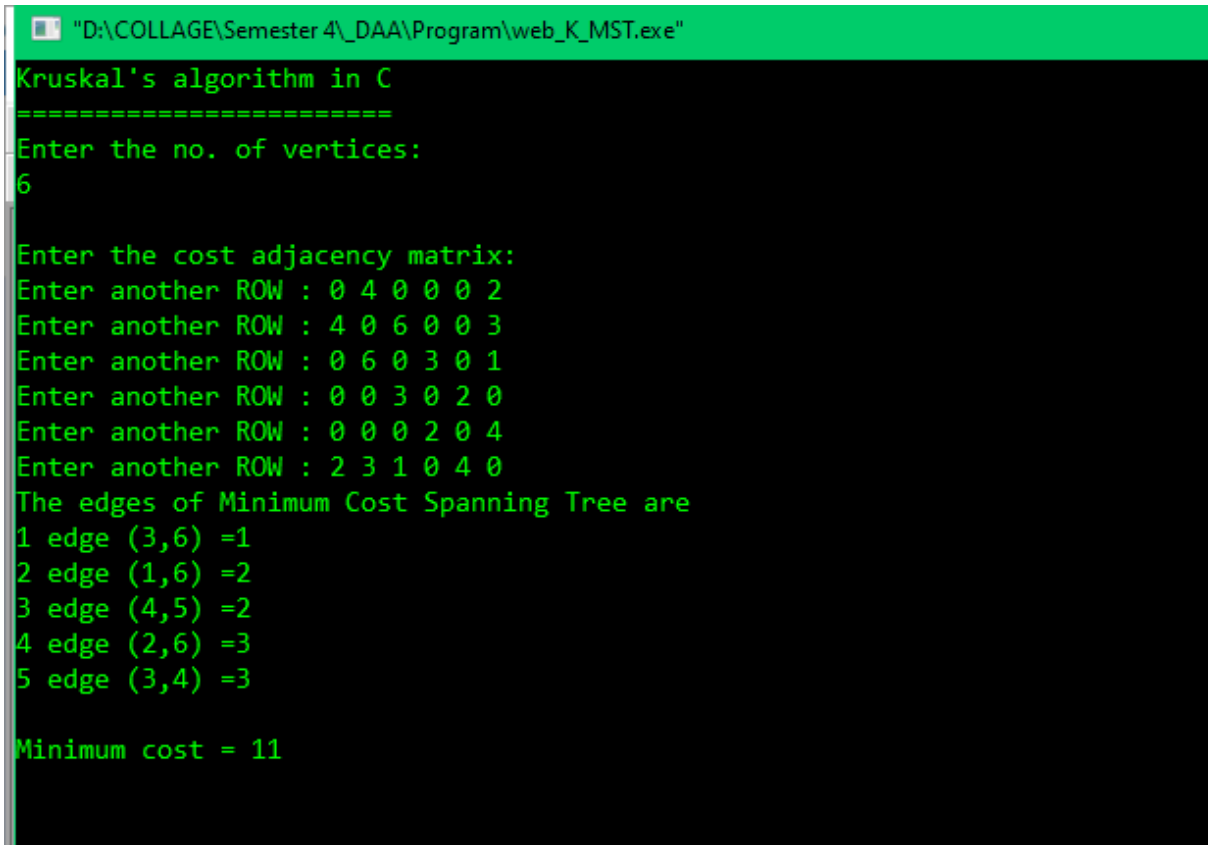
```c
#include <stdio.h>
#define MAX 100

typedef struct Job {
  char id[5];
  int deadline;
  int profit;
} Job;

void jobSequencingWithDeadline(Job jobs[], int n);

int minValue(int x, int y) {
  if(x < y) return x;
  return y;
}

int main(void) {
  //variables
  int i, j;

  //jobs with deadline and profit
  Job jobs[5] = {
    {"j1", 2,  60},
    {"j2", 1, 100},
    {"j3", 3,  20},
    {"j4", 2,  40},
    {"j5", 1,  20},
  };

  //temp
  Job temp;

  //number of jobs
  int n = 5;

  //sort the jobs profit wise in descending order
  for(i = 1; i < n; i++) {
    for(j = 0; j < n - i; j++) {
      if(jobs[j+1].profit > jobs[j].profit) {
        temp = jobs[j+1];
        jobs[j+1] = jobs[j];
        jobs[j] = temp;
      }
    }
  }
  printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");
      }
    }
  }
  printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");
  for(i = 0; i < n; i++) {
    printf("%10s %10i %10i\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
  }

  jobSequencingWithDeadline(jobs, n);
  return 0;
}

void jobSequencingWithDeadline(Job jobs[], int n) {
  //variables
  int i, j, k, maxprofit;
  //free time slots
  int timeslot[MAX];
  //filled time slots
  int filledTimeSlot = 0;
  //find max deadline value
  int dmax = 0;
  for(i = 0; i < n; i++) {
    if(jobs[i].deadline > dmax) {
      dmax = jobs[i].deadline;
    }
  }

  //free time slots initially set to -1 [-1 denotes EMPTY]
  for(i = 1; i <= dmax; i++) {
    timeslot[i] = -1;
  }

  printf("dmax: %d\n", dmax);

  for(i = 1; i <= n; i++) {
    k = minValue(dmax, jobs[i - 1].deadline);
    while(k >= 1) {
      if(timeslot[k] == -1) {
        timeslot[k] = i-1;
        filledTimeSlot++;
        break;
      }
      k--;
    }
    //if all time slots are filled then stop
```

```
69
70      //free time slots initially set to -1 [-1 denotes EMPTY]
71      for(i = 1; i <= dmax; i++) {
72        timeslot[i] = -1;
73      }
74
75      printf("dmax: %d\n", dmax);
76
77      for(i = 1; i <= n; i++) {
78        k = minValue(dmax, jobs[i - 1].deadline);
79        while(k >= 1) {
80          if(timeslot[k] == -1) {
81            timeslot[k] = i-1;
82            filledTimeSlot++;
83            break;
84          }
85          k--;
86        }
87        //if all time slots are filled then stop
88        if(filledTimeSlot == dmax) {
89          break;
90        }
91      }
92
93      //required jobs
94      printf("\nRequired Jobs: ");
95      for(i = 1; i <= dmax; i++) {
96        printf("%s", jobs[timeslot[i]].id);
97
98        if(i < dmax) {
99          printf(" --> ");
100       }
101     }
102
103     //required profit
104     maxprofit = 0;
105     for(i = 1; i <= dmax; i++) {
106       maxprofit += jobs[timeslot[i]].profit;
107     }
108     printf("\nMax Profit: %d\n", maxprofit);
109   }
110
```

```
Select "D:\COLLAGE\Semester 4\_DAA\Program\JobSequencing.exe"
       Job   Deadline     Profit
        j2          1        100
        j1          2         60
        j4          2         40
        j3          3         20
        j5          1         20
dmax: 3

Required Jobs: j2 --> j1 --> j3
Max Profit: 180

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

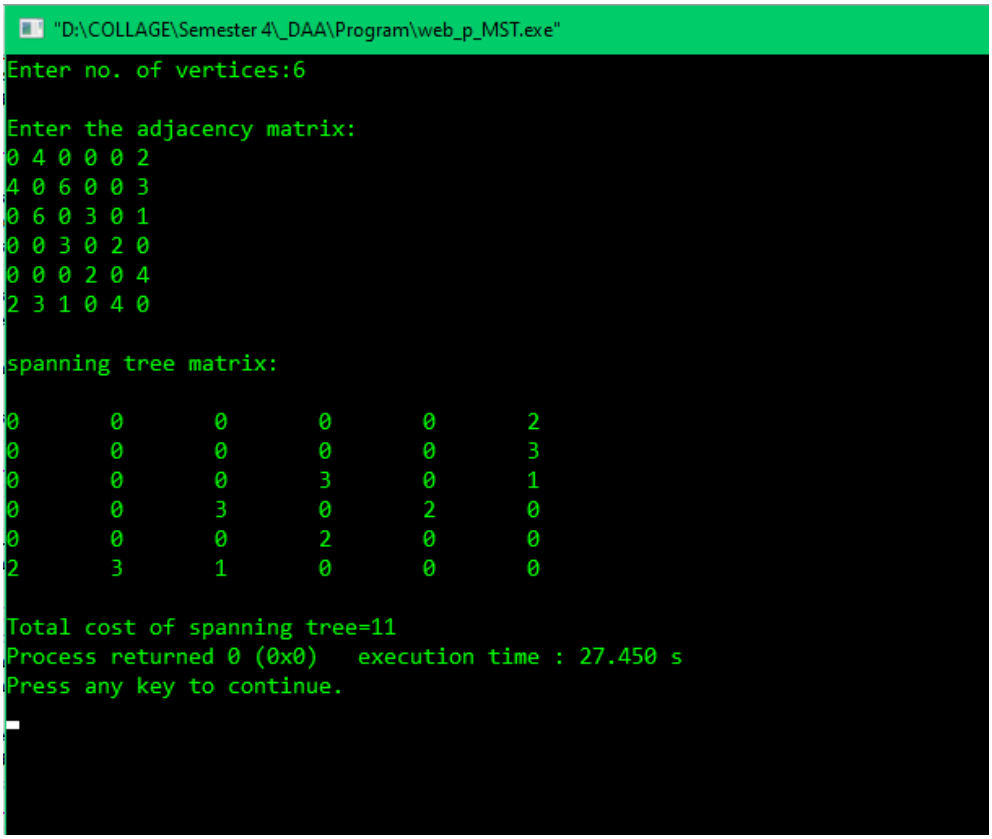
Kruskal's - Minimum Spanning Tree

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

int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[9][9], parent[9];

int find(int);
int uni(int, int);

void main()
{
  printf("Kruskal's algorithm in C\n");
  printf("========================\n");

  printf("Enter the no. of vertices:\n");
  scanf("%d", &n);

  printf("\nEnter the cost adjacency matrix:\n");
  for (i = 1; i <= n; i++)
  {
      printf("Enter another ROW : ");
    for (j = 1; j <= n; j++)
    {
      scanf("%d", &cost[i][j]);

          if (cost[i][j] == 0)
            cost[i][j] = 999;
    }
  }

  printf("The edges of Minimum Cost Spanning Tree are\n");
  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;
        }
      }
    }

    u = find(u);
    v = find(v);

    if (uni(u, v))
    {
      printf("%d edge (%d,%d) =%d\n", ne++, a, b, min);
      mincost += min;
    }

    cost[a][b] = cost[b][a] = 999;
  }

  printf("\nMinimum cost = %d\n", mincost);
  getch();
}



int find(int i)
{
  while (parent[i])
    i = parent[i];
  return i;
}

int uni(int i, int j)
{
  if (i != j)
  {
    parent[j] = i;
    return 1;
  }

  return 0;
}
```


"D:\COLLAGE\Semester 4_DAA\Program\web_K_MST.exe"
Kruskal's algorithm in C
==========
Enter the no. of vertices:
6
Enter the cost adjacency matrix:
Enter another ROW : 0 4 0 0 0 2
Enter another ROW : 4 0 6 0 0 3
Enter another ROW : 0 6 0 3 0 1
Enter another ROW : 0 0 3 0 2 0
Enter another ROW : 0 0 0 2 0 4
Enter another ROW : 2 3 1 0 4 0
The edges of Minimum Cost Spanning Tree are
1 edge (3,6) =1
2 edge (1,6) =2
3 edge (4,5) =2
4 edge (2,6) =3
5 edge (3,4) =3
Minimum cost = 11

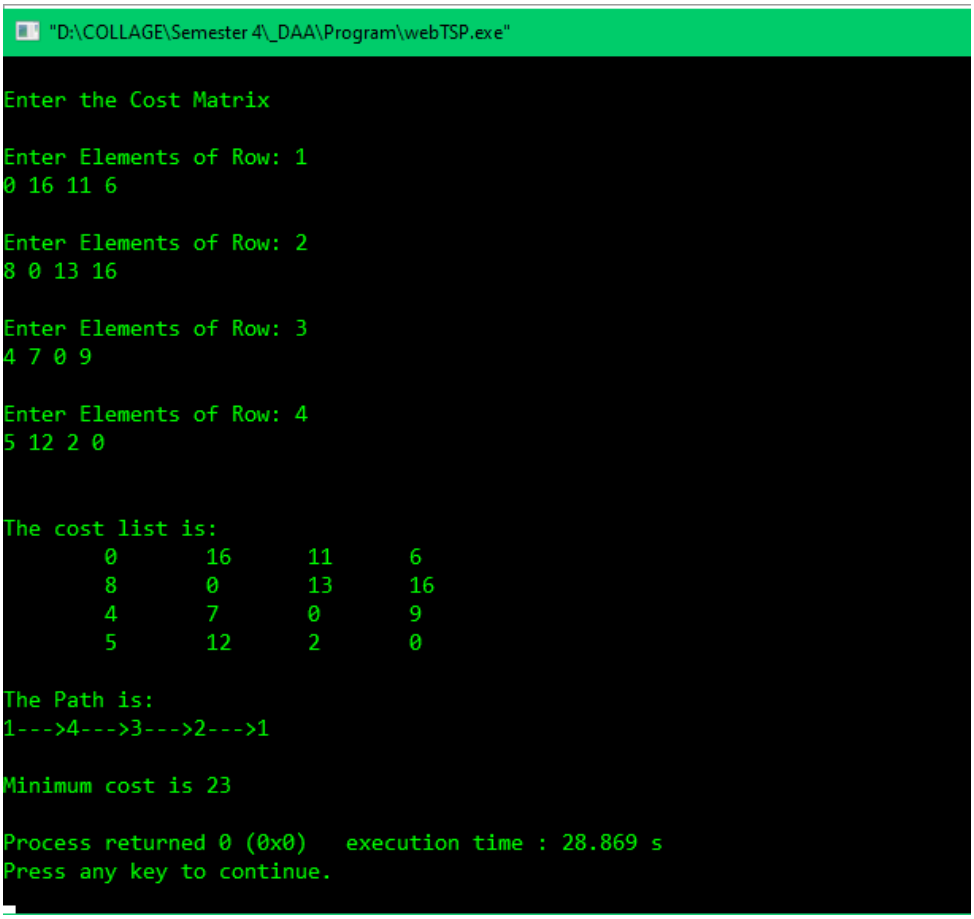
Prim's – Minimum Spanning Tree

```c
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;
int prims();

int main()
{
    int i,j,total_cost;
    printf("Enter no. of vertices:");
    scanf("%d",&n);

    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    scanf("%d",&G[i][j]);
    total_cost=prims();

    printf("\nspanning tree matrix:\n");
    for(i=0;i<n;i++)
    {
        printf("\n");
        for(j=0;j<n;j++)
        printf("%d\t",spanning[i][j]);
    }
    printf("\n\nTotal cost of spanning tree=%d",total_cost);
    return 0;
}

int prims()
{
    int cost[MAX][MAX];
    int u,v,min_distance,distance[MAX],from[MAX];
    int visited[MAX],no_of_edges,i,min_cost,j;

    //create cost[][] matrix,spanning[][]
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            if(G[i][j]==0)
            cost[i][j]=infinity;
            else
            cost[i][j]=G[i][j];
            spanning[i][j]=0;
            cost[i][j]=G[i][j];
            spanning[i][j]=0;
        }
    //initialize visited[],distance[] and from[]
    distance[0]=0;
    visited[0]=1;

    for(i=1;i<n;i++)
    {
        distance[i]=cost[0][i];
        from[i]=0;
        visited[i]=0;
    }
    min_cost=0; //cost of spanning tree
    no_of_edges=n-1; //no. of edges to be added

    while(no_of_edges>0){
    //find the vertex at minimum distance from the tree
    min_distance=infinity;

    for(i=1;i<n;i++)
        if(visited[i]==0&&distance[i]<min_distance)
        {
        v=i;
        min_distance=distance[i];
        }
    u=from[v];
    //insert the edge in spanning tree
    spanning[u][v]=distance[v];
    spanning[v][u]=distance[v];

    no_of_edges--;
    visited[v]=1;
    //updated the distance[] array
    for(i=1;i<n;i++)
        if(visited[i]==0&&cost[i][v]<distance[i])
        {
            distance[i]=cost[i][v];
            from[i]=v;
        }
    min_cost=min_cost+cost[u][v];
    }
    return(min_cost);
}
```


"D:\COLLAGE\Semester 4_DAA\Program\web_p_MST.exe"
Enter no. of vertices:6
Enter the adjacency matrix:
0 4 0 0 0 2
4 0 6 0 0 3
0 6 0 3 0 1
0 0 3 0 2 0
0 0 0 2 0 4
2 3 1 0 4 0
spanning tree matrix:
0 0 0 0 0 2
0 0 0 0 0 3
0 0 0 3 0 1
0 0 3 0 2 0
0 0 0 2 0 0
2 3 1 0 0 0
Total cost of spanning tree=11
Process returned 0 (0x0) execution time : 27.450 s
Press any key to continue.

Travelling Salesman Problem (TSP)

```c
#include<stdio.h>

int ary[10][10],completed[10],n,cost=0;

void takeInput()
{
    int i,j;

    printf("Enter the number of villages: ");
    scanf("%d",&n);

    printf("\nEnter the Cost Matrix\n");

    for(i=0;i < n;i++)
    {
        printf("\nEnter Elements of Row: %d\n",i+1);

        for( j=0;j < n;j++)
            scanf("%d",&ary[i][j]);

        completed[i]=0;
    }

    printf("\n\nThe cost list is:");

    for( i=0;i < n;i++)
    {
        printf("\n");

        for(j=0;j < n;j++)
        printf("\t%d",ary[i][j]);
    }
}

void mincost(int city)
{
    int i,ncity;

    completed[city]=1;

    printf("%d--->",city+1);
    ncity=least(city);

    if(ncity==999)
    {
        ncity=0;
        printf("%d",ncity+1);
        cost+=ary[city][ncity];

        return;
    }

    mincost(ncity);
}

int least(int c)
{
    int i,nc=999;
    int min=999,kmin;

    for(i=0;i < n;i++)
    {
        if((ary[c][i]!=0)&&(completed[i]==0))
            if(ary[c][i]+ary[i][c] < min)
            {
                min=ary[i][0]+ary[c][i];
                kmin=ary[c][i];
                nc=i;
            }
    }

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

    return nc;
}

int main()
{
    takeInput();

    printf("\n\nThe Path is:\n");
    mincost(0); //passing 0 because starting vertex

    printf("\n\nMinimum cost is %d\n ",cost);

    return 0;
}
```


"D:\COLLAGE\Semester 4_DAA\Program\webTSP.exe"
Enter the Cost Matrix
Enter Elements of Row: 1
0 16 11 6
Enter Elements of Row: 2
8 0 13 16
Enter Elements of Row: 3
4 7 0 9
Enter Elements of Row: 4
5 12 2 0
The cost list is:
0 16 11 6
8 0 13 16
4 7 0 9
5 12 2 0
The Path is:
1--->4--->3--->2--->1
Minimum cost is 23
Process returned 0 (0x0) execution time : 28.869 s
Press any key to continue.