WISH

Ds lab manual MCA

01.Write a C program to Implement the following searching technique
a. Linear Search, b. Binary Search
Index-Array searching technique.
Program-

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

void main()
{
    int i, n, ch;
    printf("IMPLEMENTATION OF SEARCHING TECHNIQUES\n");
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int *arr=(int*)malloc(sizeof(int)*n);
    printf("Enter the array elements:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    int searchKey;
    printf("Enter the search Element: ");
    scanf("%d", &searchKey);
    system("cls");
    printf("IMPLEMENTATION OF SEARCHING TECHNIQUES\n");
    printf("\n1.\tLinearSearch\n2.\tBinarySearch\n3.\tExit\n");
    printf("Choose the searching technique: ");
    scanf("%d", &ch);
    switch(ch)
    {
        case 1: {
            for(i=0;i<n;i++)
            {
                if(searchKey==arr[i])
                {
                    printf("\nElement found at %d position",i+1);
                    break;
                }
            }
            if(i>=n)
            {
                printf("Element not found in the given array!\n");
            }
            getch();
            exit(1);
        }
        case 2: {
            int first = 0, last = n-1, middle=(first+last)/2;
            while(first<=last)
            {
                if(arr[middle]<searchKey)
                {
                    first=middle+1;
                }
                else if(arr[middle]==searchKey)
                {
                    printf("Element found at %d position",middle+1);
                    getch();
                    break;
                }
                else
                    last=middle-1;
                middle=(first+last)/2;
                if(first>last)
                {
                    printf("Element not found!");
                    getch();
                }
            }
        }
        case 3: exit(1);
        default: printf("\nInvalid Choice");break;
    }
}
```

Output-

```
IMPLEMENTATION OF SEARCHING TECHNIQUES
Enter the number of elements: 5
Enter the array elements:
2
9
6
5
4
Enter the search element:6
IMPLEMENTATION OF SEARCHING TECHNIQUES

1.    LinearSearch
2.    BinarySearch
3.    Exit
Choose the searching technique: 1
Element found at 3 position
```

02.Write a C program to implement the following sorting algorithms using user defined functions: a. Bubble sort (Ascending order) b. Selection sort (Descending order).

Index – Array sorting algorithms

Program –

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

WISH

```c
int inputArr[25], size, i, j;

void bubbleSort(int a[], int n)
{
    int temp;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    printf("\nArray sorted in Ascending Order using Bubble Sort: \n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",a[i]);
    }
}

void selectionSort(int d[], int n)
{
    int least,temp;
    for(i=0;i<n;i++)
    {
        least=i;
        for(j=i+1;j<n;j++)
        {
            if(d[j]<d[least])
            {
                least=j;
            }
        }
        temp=d[least];
        d[i]=d[least];
        d[least]=temp;
    }
    printf("\nArray sorted in Descending Order using Selection Sort: \n");
    for(i=n-1;i>=0;i--)
    {
        printf("%d\t",d[i]);
    }
}

void main()
{
    system("cls");
    printf("Enter the number of elements: ");
    scanf("%d",&size);
    printf("Enter %d elements: \n",size);
    for(i=0;i<size;i++)
    {
        scanf("%d",&inputArr[i]);
    }
    bubbleSort(inputArr,size);
    selectionSort(inputArr,size);
}
```

Output-

```
Enter the number of elements: 5
Enter 5 elements:
15
45
35
75
25

Array sorted in Ascending Order using Bubble Sort:
15    25    35    45    75
Array sorted in Descending Order using Selection Sort:
75    45    35    25    15
```

03.Write a C Program implement STACK with the following operations a. Push an Element on to Stack b. Pop an Element from Stack

Index – Stack operations

Program –

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

void main()
{
    int i,n,ch;
    printf("IMPLEMENTATION OF STACK OPERATIONS\n");
    printf("Enter the number of elements in the stack: ");
    scanf("%d", &n);
    int *stk=(int*)malloc(sizeof(int)*n);
    int top=-1;
    do
    {
        printf("1.\tPUSH\n2.\tPOP\n3.\tDISPLAY\n4.\tEXIT");
        printf("\nChoice: ");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            {
                if(top==n-1)
```

WISH

```
        {
          printf("Stack is full!Overflow!\n");
          getch();
          break;
        }
        else
        {
          printf("Enter the element: ");
          top++;
          scanf("%d",&stk[top]);
        }
        break;
      }
    case 2:
      {
        if(top==-1)
        {
          printf("Stack is empty!Underflow!\n");
          getch();
          break;
        }
        else
        {
          printf("Element Popped: %d",stk[top]);
          top--;
        }
        break;
      }
    case 3:
      {
        if(top==-1)
        {
          printf("Empty Stack!\n");
          getch();
          break;
        }
        else
        {
          for(i=0;i<=top;i++)
          {
            printf("%d\t",stk[i]);
          }
          getch();
        }
        break;
      }
      case 4: exit(1);
      default: printf("Invalid Choice\n");
    }
  }while(ch!=4);
}
```

WISH

Output-

```
IMPLEMENTATION OF STACK OPERATIONS
Enter the number of elements in the
1.     PUSH
2.     POP
3.     DISPLAY
4.     EXIT
Choice: 1
Enter the element: 10
1.     PUSH
2.     POP
3.     DISPLAY
4.     EXIT
Choice: 1
Enter the element: 20
1.     PUSH
2.     POP
3.     DISPLAY
4.     EXIT
Choice: 1
Enter the element: 30
1.     PUSH
2.     POP
3.     DISPLAY
4.     EXIT
Choice: 1
Enter the element: 40
1.     PUSH
2.     POP
3.     DISPLAY
4.     EXIT
Choice: 2
Element Popped: 401.    PUSH
2.     POP
3.     DISPLAY
4.     EXIT
Choice: 3
10     20     30     1.     PUSH
2.     POP
3.     DISPLAY
4.     EXIT
Choice: 4
```

04.Implement a Program in C for converting an Infix Expression to Postfix Expression.

Index –Stack infix to post fix

Program-

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

WISH

```
struct StackI
{
    int top;
    unsigned capacity;
    int* arr;
};

struct StackI* createStack( unsigned capacity )
{
    struct StackI* stackV = (struct StackI*)
        malloc(sizeof(struct StackI));

    if (!stackV)
        return NULL;

    stackV->top = -1;
    stackV->capacity = capacity;

    stackV->arr = (int*) malloc(stackV->capacity *
                                sizeof(int));
    return stackV;
}

int isEmpty(struct StackI* stackV)
{
    return stackV->top == -1 ;
}

char peek(struct StackI* stackv)
{
    return stackv->arr[stackv->top];
}

char pop(struct StackI* stackv)
{
    if (!isEmpty(stackv))
        return stackv->arr[stackv->top--] ;
    return '$';
}

void push(struct StackI* stackV, char op)
{
    stackV->arr[++stackV->top] = op;
}

int isOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') ||
        (ch >= 'A' && ch <= 'Z');
}

int precedence(char ch)
```

WISH

```c
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
        return 2;

    case '^':
        return 3;
    }
    return -1;
}

int infixToPostfix(char* exp)
{
    int i, k;

    struct StackI* stackV = createStack(strlen(exp));
    if(!stackV)
        return -1 ;

    for (i = 0, k = -1; exp[i]; ++i)
    {
        if (isOperand(exp[i]))
            exp[++k] = exp[i];

        else if (exp[i] == '(')
            push(stackV, exp[i]);

        else if (exp[i] == ')')
        {
            while (!isEmpty(stackV) && peek(stackV) != '(')
                exp[++k] = pop(stackV);
            if (!isEmpty(stackV) && peek(stackV) != '(')
                return -1;
            else
                pop(stackV);
        }
        else
        {
            while (!isEmpty(stackV) &&
                precedence(exp[i]) <= precedence(peek(stackV)))
                exp[++k] = pop(stackV);
            push(stackV, exp[i]);
        }

    }
    while (!isEmpty(stackV))
        exp[++k] = pop(stackV);

    exp[++k] = '\0';
    printf( "%s", exp );
}

void main()
{
    char exp[50];
    printf("Enter the INFIX NOTATION: \n");
    scanf("%s",&exp);
    printf("\nPOSTFIX NOTATION:\n");
    infixToPostfix(exp);
}
```

Output-

```
Enter the INFIX NOTATION:
3*4-5
POSTFIX NOTATION:
*3-45
Enter the INFIX NOTATION:
a+b-c
POSTFIX NOTATION:
ab+c-
```

05.Implement a Program in C for evaluating a Postfix Expression

Index -Stack postfix evaluation

Program –

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

struct StackI
{
    int top;
    unsigned capacity;
    int* arr;
};

struct StackI* createStack( unsigned capacity )
{
    struct StackI* stackV = (struct StackI*) malloc(sizeof(struct StackI));
    if (!stackV)
        return NULL;
    stackV->top = -1;
    stackV->capacity = capacity;
    stackV->arr = (int*) malloc(stackV->capacity * sizeof(int));

    if (!stackV->arr)
```

WISH

```c
        return NULL;

    return stackV;
}

int isEmpty(struct StackI* stackV)
{
    return stackV->top == -1 ;
}

char peek(struct StackI* stackV)
{
    return stackV->arr[stackV->top];
}

char pop(struct StackI* stackV)
{
    if (!isEmpty(stackV))
        return stackV->arr[stackV->top--] ;
    return '$';
}

void push(struct StackI* stackV, char op)
{
    stackV->arr[++stackV->top] = op;
}

int evaluatePostfix(char* exp)
{
    struct StackI* stackV = createStack(strlen(exp));
    int i;

    if (!stackV)
        return -1;
    for (i = 0; exp[i]; ++i)
    {
        if (isdigit(exp[i]))
            push(stackV, exp[i] - '0');

        else
        {
            int val1 = pop(stackV);
            int val2 = pop(stackV);
            switch (exp[i])
            {
            case '+': push(stackV, val2 + val1); break;
            case '-': push(stackV, val2 - val1); break;
            case '*': push(stackV, val2 * val1); break;
            case '/': push(stackV, val2/val1); break;
            }
        }
    }
    return pop(stackV);
}

void main()
{
    char exp[30];// = "231*+9-";
    printf("Enter the postfix Expression: \n");
    scanf("%s",&exp);
    printf ("postfix evaluation: %d", evaluatePostfix(exp));
}
```

Output –

```
Enter the postfix expression:
5678+*-
Postfix evaluation: -85
```

06.Write a C program to simulate the working of a singly linked list providing the following operations: a. Display & Insert b. Delete from the beginning/end c. Delete a given element.

Index – linked list working with singly linked list

Program –

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

struct llist
{
    int data;
    struct llist *next;
};

typedef struct llist node;
node *head, *ptr, *new1;
int choice, count=0;

void countNodes()
{
    count = 0;
    ptr = head;
    while (ptr!=NULL)
    {
        ptr = ptr->next;
        count++;
    }
}

void displayMenu()
{
    system("cls");
```

WISH

```
    printf("LINKED LIST OPERATIONS\n\n");
    printf("1.\tDisplay and Insert\n2.\tDelete from beginning/end\n3.\tDelete a given
element\n4.\tExit\n");
    printf("\nChoice: ");
}

void displayLL()
{
    printf("\nLinkedList elements:\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\t",ptr->data);
        ptr = ptr->next;
    }
    getch();
}

void insertNodeAtEnd()
{
    ptr = head;
    while(ptr != NULL)
    {
        ptr = ptr->next;
    }
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter data: ");
    scanf("%d",&ptr->data);
    ptr->next = new1;
    ptr = new1;
}

void insertNodeInBegin()
{
    new1 = (node*)malloc(sizeof(node));
    ptr = new1;
    printf("\nEnter data: ");
    scanf("%d",&ptr->data);
    ptr->next = head;
    head = new1;
}

void insertNodeAtN()
{
    int pos,i;
    node *new2,*loc;
    printf("Enter position: ");
    scanf("%d",&pos);
    countNodes();
    if(pos>count)
    {
        printf("\nInvalid position!");
```

WISH

```
        return;
    }
    else
    {
        new2=(node*)malloc(sizeof(node));
        printf("\nEnter data: ");
        scanf("%d",&new2->data);
        new2->next=NULL;
        if(pos==1)
        {
            new2->next=head;
            head=new2;
        }
        else
        {
            loc=head;
            for(i=2;i<pos;i++)
            {
                loc=loc->next;
            }
            new2->next=loc->next;
            loc->next=new2;
        }
    }
}

void insertNodeMenu()
{
    do
    {
        printf("\n\n1.\tInsert at beginning\n2.\tInsert at end\n3.\tInsert at Nth 
Position\n4.\tReturn to main menu\n");
        printf("Choice: ");
        scanf("%d",&choice);
        switch (choice)
        {
            case 1: {
                insertNodeInBegin();break;
            }
            case 2: {
                insertNodeAtEnd();break;
            }
            case 3: {
                insertNodeAtN();break;
            }
            case 4 : {
                break;
            }
            default: printf("WrongInput!");break;
        }
    }while(choice != 4);
}
```

WISH

```c
void deleteNodeBegin()
{
    ptr = head;
    if (ptr!= NULL)
    {
        new1 = head;
        head = head->next;
        printf("\nDeleted element: %d", ptr->data);
        getch();
        free(ptr);
    }
    else
    {
        printf("\nLinked List is empty\n");
        getch();
    }
}

void deleteNodeEnd()
{
    ptr = head;
    if(ptr!= NULL)
    {
        while(ptr->next!=NULL)
        {
            new1=ptr;
            ptr = ptr->next;
        }
        printf("\nDeleted element: %d", ptr->data);
        new1->next=NULL;
        getch();
        free(ptr);
    }
    else
    {
        printf("Linked List is empty!\n");
        getch();
    }
}
void deleteNodeMenu()
{
    do
    {
        system("cls");
        printf("1.\tDelete from beginning\n2.\tDelete from end\n3.\tReturn to main 
menu\n");
        printf("Choice: ");
        scanf("%d",&choice);
        switch (choice)
        {
            case 1: {
```

WISH

```
                deleteNodeBegin();break;
            }
            case 2: {
                deleteNodeEnd();break;
            }
            case 3: {
                break;
            }
            default: printf("WrongInput!");break;
        }
    }while(choice != 3);
}

void deleteElement()
{
    int temp;
    printf("\nEnter the element to delete: ");
    scanf("%d", &temp);
    ptr = head;
    new1 = NULL;
    while (ptr!= NULL)
    {
        if(ptr->data == temp)
        {
            break;
        }
        new1 = ptr;
        ptr = ptr->next;
    }

        if(ptr == NULL)
        {
            printf("\nElement not found !\n");
            getch();
        }
        else if (new1 == NULL)
        {
            printf("\nDeleting element from firstNode: %d\n", ptr->data);
            head = ptr->next;
            getch();
        }
        else
        {
            printf("\nDeleting element: %d\n", ptr->data);
            getch();
            new1->next = ptr->next;
        }
        free(ptr);
}

void main()
{
```

WISH

```
    char ch;
    int op;
    head = (node*)malloc(sizeof(node));
    ptr = head;
    printf("Input Initial Linked List:\n");
    printf("\nEnter data: ");
    scanf("%d",&ptr->data);
    do
    {
        printf("\nContinue LinkedList? y or n");
        ch = getch();
        switch(ch)
        {
            case 'y': {
                new1 = (node*)malloc(sizeof(node));
                ptr->next = new1;
                ptr = new1;
                printf("\nEnter data: ");
                scanf("%d",&ptr->data);
                break;
            }
            case 'n': {
                ptr->next = NULL;
                break;
            }
            default: printf("\nPlease only input 'y' or 'n'\n");break;
        }
    }while (ch!='n');
    do
    {
        displayMenu();
        scanf("%d",&op);
        switch(op)
        {
            case 1 : displayLL();insertNodeMenu();break;
            case 2 : deleteNodeMenu();break;
            case 3 : deleteElement();break;
            case 4 : exit(1);
            default : system("cls");printf("Wrong Input!");
        }
    }while(op!=4);
}
```

Output –

```
Input initial linked list:
Enter data:2
Continue linked list? y or n
Enter data:3
Continue linked list? y or n
Enter data:4
LINKED LIST OPERATIONS
```

```
WISH
1.Display and insert
2.Delete from beginning/end
3.Delete a given element
4.exit
Choice:1
Linked list elements:
2         3         4
```

07.Obtain the Topological ordering of vertices in a given graph with the help of a c programming.

Index – Graph topological sorting

Program –

```c
#include <stdio.h>

void main()
{
    int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

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

    printf("Enter the adjacency matrix:\n");
    for(i=0;i<n;i++){
        printf("Enter row %d\n",i+1);
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    }

    for(i=0;i<n;i++){
        indeg[i]=0;
        flag[i]=0;
    }

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            indeg[i]=indeg[i]+a[j][i];

    printf("\nThe topological order is: \n");

    while(count<n){
        for(k=0;k<n;k++){
            if((indeg[k]==0) && (flag[k]==0)){
                printf("%d ",(k+1));
                flag [k]=1;
            }

            for(i=0;i<n;i++){
                if(a[i][k]==1)
                    indeg[k]--;
            }
        }
        count++;
    }
}
```

Output –

```
Enter the no of vertices:
6
Enter the adjacency matrix:
Enter row 1
0 0 0 0 0 0
Enter row 2
0 0 0 0 0 0
Enter row 3
0 0 0 1 0 0
Enter row 4
0 1 0 0 0 0
Enter row 5
1 1 0 0 0 0
Enter row 6
1 0 1 0 0 0

The topological order is:
5 6 1 2 3 4
```

08Check whether a given graph is connected or not using DFS method using C programming

Index – Graph is connected or not using DFS method

Program –

```
#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
    int i;
    reach[v]=1;
    for(i=1;i<=n;i++)
    if(a[v][i] && !reach[i])
    {
        printf("\n %d->%d",v,i);
        dfs(i);
    }
}

void main()
{
    //clrscr();
    int i,j,count=0;
    printf("\n Enter number of vertices: ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        reach[i]=0;
        for(j=1;j<=n;j++)
            a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix: \n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");
    for(i=1;i<=n;i++)
    {
        if(reach[i])
        count++;
    }
    if(count==n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
    getch();
}
```

Output –

```
Enter the number of vertices:4
Enter the adjacency matrix:
0 1
0 2
3 1
2 0
1 2
3 0
1 0
2 0
1->2
2->3
1->4
Graph is connected
```

09.From a given vertex in a weighted connected graph, find shortest paths to other vertices

Using Dijkstra's algorithm (C programming)

Index – Find shortest path using Dijkstra’s algorithm

Program –

```
/* Dijkstra's Algorithm in C */
#include<stdio.h>
#include<conio.h>
//#include<process.h>
#include<string.h>
#include<math.h>
#define IN 99
```

WISH

```c
#define N 6
int dijkstra(int cost[][N], int source, int target);
int main()
{
    int cost[N][N],i,j,w,ch,co;
    int source, target,x,y;
    printf("\t The Shortest Path Algorithm ( DIJKSTRA'S ALGORITHM in C \n\n");
    for(i=1;i< N;i++)
    for(j=1;j< N;j++)
    cost[i][j] = IN;
    for(x=1;x< N;x++)
    {
        for(y=x+1;y< N;y++)
        {
            printf("Enter the weight of the path between nodes %d and %d: ",x,y);
            scanf("%d",&w);
            cost [x][y] = cost[y][x] = w;
        }
        printf("\n");
    }
    printf("\nEnter the source:");
    scanf("%d", &source);
    printf("\nEnter the target");
    scanf("%d", &target);
    co = dijsktra(cost,source,target);
    printf("\nThe Shortest Path: %d",co);
}
int dijsktra(int cost[][N],int source,int target)
{
    int dist[N],prev[N],selected[N]={0},i,m,min,start,d,j;
    char path[N];
    for(i=1;i< N;i++)
    {
        dist[i] = IN;
        prev[i] = -1;
    }
    start = source;
    selected[start]=1;
    dist[start] = 0;
    while(selected[target] ==0)
    {
        min = IN;
        m = 0;
        for(i=1;i< N;i++)
        {
            d = dist[start] +cost[start][i];
            if(d< dist[i]&&selected[i]==0)
            {
                dist[i] = d;
                prev[i] = start;
            }
            if(min>dist[i] && selected[i]==0)
```

WISH

```
            {
                min = dist[i];
                m = i;
            }
        }
        start = m;
        selected[start] = 1;
    }
    start = target;
    j = 0;
    while(start != -1)
    {
        path[j++] = start+65;
        start = prev[start];
    }
    path[j]='\0';
    strrev(path);
    printf("%s", path);
    return dist[target];
}
```

Output –

```
 THE SHORTEST PATH ALGORITHM IN (DIJKSTRA'S ALGORITHM) IN C
Enter the weight of the path between nodes 1 and 2: 10
Enter the weight of the path between nodes 1 and 3: 20
Enter the weight of the path between nodes 1 and 4: 30
Enter the weight of the path between nodes 1 and 5: 40
Enter the weight of the path between nodes 2 and 3: 50
Enter the weight of the path between nodes 2 and 4: 60
Enter the weight of the path between nodes 2 and 5: 70
Enter the weight of the path between nodes 3 and 4: 80
Enter the weight of the path between nodes 3 and 5: 90
Enter the weight of the path between nodes 4 and 5: 100

Enter the source:2
Enter the target:4
CBE
The shortest path: 40
```

10.Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm (C programming)

Index – minimum cost finding using Kruskal’s algorithm

Program –

```
#include<stdio.h>
#include<stdlib.h>
#define VAL 999
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
// union - find
int find(int i)
{
```

WISH

```
    while(parent[i])
    i=parent[i];
    return i;
}
int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
        return 1;
    }
    return 0;
}
int main()
{
    printf("Implementation of Kruskal's algorithm\n");
    printf("Enter the no. of vertices:");
    scanf("%d",&n);
    printf("Enter the cost adjacency matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=VAL;
        }
    }
    printf("The edges of Minimum Cost Spanning Tree are\n");
    while(ne < n)
    {
        for(i=1,min=VAL;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))
        {
            // printing edges
            printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
            mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    // minimum cost
    printf("\n\tMinimum cost = %d\n",mincost);
    return 0;
}
```

Output –

```
Implementation of Kruskal's algorithm
Enter the no of vertices:6
Enter the cost adjacency of matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
The edges of Minimum cost spanning tree are
1 edge <1,3> =1
2 edge <4,6> =2
3 edge <1,2> =3
4 edge <2,5> =3
5 edge <3,6> =4
Minimum cost = 13
```