



Estd: 2001

An Institute with a difference

RNS INSTITUTE OF TECHNOLOGY

(AICTE Approved, VTU Affiliated & NAAC Accredited with 'A' Grade)

Dr. Vishnuvardhan Road, Channasandra, Rajarajeshwari Nagar Post, Bengaluru - 560 098

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

LABORATORY EXPERIMENTS

SUBJECT TITLE	Data Structures with Algorithms Laboratory		
SUBJECT CODE	22MCAL16		
ACADEMIC YEAR	2023 – 2024	BATCH	2022-2024
CIE Marks	50	SEE Marks	50
SEMESTER	I	Credits	1.5
FACULTY NAME and DESIGNATION	Vishwanath V Murthy, Asst Prof, MCA		

CO - Course objectives:

1. Implements the applications of stack like converting infix expression to postfix, evaluation of postfix expression and recursion.
2. Demonstrate the working of various data structures viz. Queues and Linked Lists.
3. Implement searching techniques like linear and binary search.
4. Implement the various sorting techniques like bubble sort and selection sort.
5. Implement the applications of graph data structure for Minimum Cost Spanning Tree and the shortest paths to other vertices

Q. No.	Experiments	Bloom's LL	Cos
1	Implement a Program in C for converting an Infix Expression to Postfix Expression.	L1	CO1
2	Design, develop, and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are + (add), - (subtract), * (multiply) and / (divide).	L1	CO1
3	Design, develop, and execute a program in C to simulate the working of a queue of integers using an array. Provide the following operations: a. Insert b. Delete c. Display.	L2	CO2
4	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.	L2	CO2
5	Write a C program to Implement the following searching techniques a. Linear Search b. Binary Search	L2	CO3

6	Write a C program to implement the following sorting algorithms using user defined functions: a. Bubble sort (Ascending order) b. Selection sort (Descending order)	L2	CO4
7	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm (C programming)	L2	CO5
8	From a given vertex in a weighted connected graph, find shortest paths to other vertices Using Dijkstra's algorithm (C programming)	L2	CO5

Demonstration Experiments (For CIE) if any

1	Using circular representation for a polynomial, design, develop, and execute a program in C to accept two polynomials, add them, and then print the resulting polynomial.	L2	CO2
2	Design, develop, and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are + (add), - (subtract), * (multiply) and / (divide)	L2	CO1
3	Write a C Program implement STACK with the following operations. a) PUSH b) POP c) PEEK d) Display	L2	CO1

Assessment Details (both CIE and SEE)

- The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%.
- The minimum passing mark for the CIE is 50% of the maximum marks.
- A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course.
- The student has to secure not less than 40% of maximum marks in the semester-end examination(SEE).
- In total of CIE and SEE student has to secure 50% maximum marks of the course.

Continuous Internal Evaluation (CIE):

CIE marks for the practical course is 50 Marks.

The split-up of CIE marks for record/ journal and test are in the ratio 60:40.

- Each experiment to be evaluated for conduction with observation sheet and record write up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments designed by the faculty who is handling the laboratory session and is made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to 30 marks (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct 02 tests for 100 marks, the first test shall be conducted after the 8th week of the semester and the second test shall be conducted after the 14th week of the semester.
- In each test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The average of 02 tests is scaled down to 20 marks (40% of the maximum marks).
- The Sum of scaled-down marks scored in the report write-up/journal and average marks of two tests is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- **SEE marks for the practical course is 50 Marks.**
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University.
- All laboratory experiments are to be included for practical examination. (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the internal /external examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
- **General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in - 60%, Viva-voce 20% of maximum marks.**
- **SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)**
- Change of experiment is allowed only once and 10% Marks allotted to the procedure part to be made zero.
- The duration of SEE is 03 hours

1. Implement a Program in C for converting an Infix Expression to Postfix Expression.Algorithm for converting an Infix Expression to Postfix Expression.**• Algorithm**

1. Set operator stack to empty.
2. **Scan the Symbol** in infix expression from left to right one character at a time.
 - **If (Symbol = operand)**
 - **Add** symbol to postfix
 - **If (Symbol = open brace)**
 - **Push** symbol to stack
 - **If (Symbol = closing brace)**
 - while (StackTop != open brace)
 Pop stack and add to postfix
 - **Remove** open brace from stack and discard
 - **If (Symbol = operator)**
 - while (precedence(Symbol) <= precedence(StackTop))
 Pop top operator and add it to postfix string
 - **Push** symbol to stack. // as precedence of Symbol is High
3. Repeat step-2 till no more symbols in string. (till end of string)
4. Pop all elements from stack till stack becomes empty and add to the Postfix Expression.

Program in C for converting an Infix Expression to Postfix Expression.

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

#define SIZE 10                /* Size of Stack */
char stack [SIZE];            /* Global declarations of stack and its top*/
int top ;

void push (char symbol)
{
    stack[++top] = symbol;
}

char pop ()
{
    return (stack[top--]);
}
```

```
int priority (char op)           /* Function for precedence */
{
    switch (op)
    {
        case '#': return 0;
        case '(': return 1;
        case '+':
        case '-': return 2;
        case '*':
        case '/': return 3;
    }
}

void infixToPostfix (char *infix, char* postfix)
{
    char symbol, brace;
    int i = 0, k = 0;
    push ('#');                  /* Mark bottom of stack */
    while ( ( symbol = infix[i++] ) != '\0')
    {
        if ( isalnum (symbol) )
            postfix [k++] = symbol;
        else if (symbol == '(')
            push (symbol);
        else if (symbol == ')')
        {
            while (stack[top] != '(')
                postfix [k++] = pop ();
            brace = pop ();        /* Remove open brace '(' */
        }
        else                    /* Operator */
        {
            while (priority(stack[top]) >= priority(symbol))
                postfix [k++] = pop ();
            push(symbol);
        }
    }
    while (stack[top] != '#')    /* Pop() from stack till empty */
        postfix [k++] = pop ();
    postfix[k] = '\0';
}
```

```
void main()
{
    char infix [50], postfix [50];
    top = -1;                                /* Initialize stack to empty */
    printf ("\n Input the Infix Expression: ");
    scanf ("%s", infix);                    /* Read infix expression */

    infixToPostfix (infix, postfix);        /* Convert the expression */

    printf ("\n Postfix Expression: %s", postfix); /* Display postfix expression */
}
```

Run1**Input:** A+B*C**Output:****Run2****Input:****Output:**

2. Design, develop, and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are: + (add), - (subtract), * (multiply) and / (divide)

Algorithm for Evaluating a Postfix Expression.

- **Algorithm**

1. Set Operand Stack to empty
2. **Scan the Symbol** in infix expression from left to right one character at a time.
 - a). **If (symbol = operand)**
 - Push symbol to stack.
 - b). **If (symbol = operator)**
 - Pop first **operand2** and then **operand1**
 - Perform result = **operand1 operator operand2**.
 - Push the result back to stack.
3. Repeat step-2 till end of input string.
4. Pop the final result and Return

C Program to evaluate a Postfix Expression.

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

#define MAX 20          /* Stack size */

float stack [MAX];
int top;

void push (float result)
{
    stack [++top] = result;
}

float pop ()
{
    return (stack[top--]) ;
}
```

```
float getResult (float op1, float op2, char symbol)
{
    /* returns result of operation */
    float result;

    switch(symbol)
    {
        /* perform operation */
        case '+': result = op1 + op2;
                break;

        case '-': result = op1 - op2;
                break;

        case '*': result = op1 * op2;
                break;

        case '/': if(op2)
                    result = op1 / op2;
                else
                { printf("Exception: Divide by zero");
                  exit(0);
                }
                break;
    }
    return (result);
}

float evalPost (char *postfix) /* function to Evaluate the postfix expression*/
{
    int i = 0;
    char symbol;
    float op1, op2, result;
    top = -1; /* Initialize stack to empty */

    while ( (symbol = postfix[i++]) != '\0')
    {
        if ( isdigit(symbol) ) /* Built-in function in ctype.h */
        {
            result = symbol - '0';
            push(result);
        }
        else
        {
            op2 = pop();
            op1 = pop();
            result = getResult (op1, op2, symbol); /* Apply the operator */
            push(result);
        }
    }
    return pop();
}
```



```
int main()
{
    char postfix [MAX];
    float result;

    printf ("\n Enter a valid postfix expression: ");
    scanf ("%s", postfix);          /* Read postfix expression */

    result = evalPost (postfix);    /* Evaluate postfix expression */

    printf ("\n The result is: %f", result);    /* Display evaluated result */
}
```

Run1**Input: 45+3*****Output:****Run2****Input:****Output:****Run3****Input:****Output:**

- 3. Design, develop, and execute a program in C to simulate the working of a queue of integers using an array. Provide the following operations:**
- a. Insert b. Delete c. Display**

C Program to demonstrate queue operations.

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

# define MAX 5

int Q [max];
int front, rear ;

void enqueue (int item)
{
    if ( rear == MAX-1 )
    {
        printf ("\n ** Can't insert as it leads Overflow ** \n");
        return;
    }
    if ( front == -1 )
        front ++;
    Q [ ++rear ] = item;
    if ( rear == MAX-1 )
        printf ("\n ** Full **\n");
}

int dequeue ( )
{
    int item;
    if ( front == -1 )
    {
        printf (" ** Can't remove as it leads 'Underflow **\n");
        return;
    }
    item = Q [ front ];
    if ( front == rear )
    {
        front = rear = -1;          /* set queue to empty */
        printf (" ** Empty **\n");
    }
    else
        front ++;
    printf("\n Deleted element is: %d \n", item) ;
}
```

```
void display ( )
{
    int temp;
    if ( front == -1 )
    {
        printf ("\n **Empty..\n");
        return;
    }
    for ( temp = front ; temp <= rear ; temp++ )
        printf ( " %d ", Q [temp] );
}

void main()
{
    int choice, item;
    front = rear = -1;    /* Initialize queue to empty */
    while(1)
    {
        printf("\n Queue Implementation");
        printf("\n    1. Insert.");
        printf("\n    2. Delete.");
        printf("\n    3. Display.");
        printf("\n    4. Exit.");

        printf ("\n Enter your choice: ");
        scanf ("%d", &choice);

        switch(choice)
        {
            case 1:
                printf("\n Enter the element to be inserted: ");
                scanf("%d", &item);
                enqueue(item);          /* Insert an item in queue */
                break;

            case 2:
                dequeue();              /* Remove an item from queue */
                break;

            case 3:
                printf("\n The Contents of the Queue are: \n");
                display();
                break;

            default:
                exit (0);
        }
    }
}
```

Output

- 4. 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

C Program to demonstrate Singly linked list.

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

typedef struct List
{
    int data;
    struct List *link;
} NODE;

NODE *start = NULL;

NODE* getnode()
{
    NODE *temp;
    temp = (NODE *) malloc(sizeof(NODE));
    if(temp == NULL)
    {
        printf("\n Memory Not Allocated !");
        exit(0);
    }
    temp->link = NULL;
    return temp;
}

void insertFront ()
{
    NODE *p;
    p = getnode();
    printf(" Enter Node value = ");
    scanf("%d", &p->data);

    p->link = start;
    start = p;

    printf("\n Node at Beginning Inserted Successfully..\n");
    getch();
}
```

```
void removeFront ()
{
    NODE *temp;
    if(start == NULL)
    {
        printf("\n List is Empty !! \n");
        getch();
        return;
    }

    temp = start;
    start = start->link;    /* start moves to next node */

    printf("\n ..Start Node removed is %d ..\n", temp->data);
    getch();

    free(temp);    /* Destroy the node */
}

void removeEnd ()
{
    NODE *prev, *last;
    if (start == NULL)
    {
        printf ("..List is Empty..\n");
        getch();
        return ;
    }

    prev = last = start;

    if ( start->link == NULL)    // List has only one Node
        start = NULL;

    while (last->link != NULL)    /* Search last node */
    {
        prev = last;
        last = last->link ;
    }
    prev->link = NULL;

    printf("\n Last Node removed is %d..\n", last->data);

    free (last);
}
```

```
void DisplayList ()
{
    NODE *temp;
    if (start == NULL)
    {
        printf("\n -- List is Empty --\n\n");
        return;
    }
    printf("\n  Linked List Elements are: \n");
    printf(" *****\n");

    temp = start;
    while(temp != NULL)
    {
        printf(" %d \n", temp->data);
        temp = temp->link;
    }
}

void main()
{
    int choice;
    while(1)
    {
        printf("\n ***** LINKED LIST OPERATIONS *****\n");
        printf("\n 1. Insert at Front. \n 2. Remove Front.");
        printf("\n 3. Remove End. \n 4. Display List .");
        printf("\n 5. Exit.");

        printf("\n\n Enter your choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1: start = insertFront (start);
                    break;

            case 2: start = removeFront(start);
                    break;

            case 3: start = removeEnd(start);
                    break;

            case 4: DisplayList(start);
                    break;

            default:
                    return;
        }
    }
}
```

5. Write a C program to Implement the following searching techniques.

- a. Linear Search b. Binary Search**

C Program to demonstrate Searching techniques.

```
#include <stdio.h>

int linearSearch (int*, int, int);
int binarySearch (int*, int, int, int);

int main()
{
    int choice, arr [10], n, key, index;

    printf ("Enter the size of the array : ");
    scanf ("%d", &n);

    printf ("Selct from Menu : \n 1. Linear Search. \n 2. Binary Search. ");
    printf ("Enter Your Choice : ");
    scanf ("%d", &choice);

    if (choice == 1)
        printf ("Enter the array Elements: \n");
    else
        printf (" Enter sorted elements for Binary Search **)", n);

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

    printf ("Enter the Key Element: ");
    scanf ("%d", &key);

    switch (choice)
    {
        case 1:  index = linearSearch(arr, n, key);
                 break;
        case 2:  index = binarySearch (arr, 0, n-1, key);
                 break;
        default :    printf ("***** Wrong Choice Entered *****: \n");
                     break;
    }
    if (index == -1)
        printf ("\n Element Not found. " );
    else
        printf ("\n Element found at location %d \n", index);
}
```



```
int linearSearch (int *a, int n, int key)
{
    int i = 0;
    while( i<n )
    {
        if (a[i]==key)
            return i+1;
        i++;
    }
    return -1;
}

int binarySearch (int a[], int left, int right, int key)
{
    if (left > right)
        return -1;

    int mid = ( left + right ) / 2;

    if (a[mid] == key)
        return mid + 1;
    else if (a[mid] < key)
        binarySearch (a, left, mid-1, key);
    else
        binarySearch (a, mid+1, right, key);
}
```

Run1

6. Write a C program to Implement the following searching techniques.

- a. Bubble sort b. Selection sort

C Program to demonstrate Searching techniques.

```
#include <stdio.h>

void bubbleSort (int*, int);
void selectionSort (int*, int);
void swap (int*, int*);

int main()
{
    int choice, arr[10], n, index, i;

    printf ("Enter the size of the array : \n");
    scanf ("%d", &n);

    printf ("Enter array Elements : \n");
    for (i=0; i<n; i++)
        scanf ("%d", &arr[i]);

    printf ("Selct from Menu : \n 1. Bubble Sort. \n 2. Selection Sort. \n");
    printf ("Enter Your Choice : ");
    scanf ("%d", &choice);

    switch (choice)
    {
        case 1: bubbleSort (arr, n);
                break;

        case 2:
                selectionSort (arr, n);
                break;

        default :   printf ("***** Wrong Key Entered *****: \n");
                    break;
    }

    printf ("\nArray Elements after Sorting: \n");
    for (i=0; i<n; i++)
        printf ("%d ", arr[i]);
}
```

```
void bubbleSort (int *arr, int n)
{
    int i, j;
    for ( i = 0 ; i < n-1 ; i++ )        // Runs n-1 passes
    {
        for (j = 0 ; j < (n - 1) - i ; j++ )    // bubble smallest element to left
            if (arr [j+1] < arr [j] )
                swap ( &arr [j] , &arr [j+1] );
    }
    printf ("END bubble Sort");
}

void selectionSort (int *arr, int n)
{
    int i, j, min;
    for (i = 0 ; i < n-1 ; i++ )    // Runs n-1 passes
    {
        min = i;
        for (j = i + 1 ; j < n ; j++ )    // finds min element
        {
            if ( arr [j] < arr [min] )
                min = j;
        }
        swap(&arr [i] , &arr [min] );    // swap min with leftmost element
    }
}

void swap (int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

Run1

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

In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached.

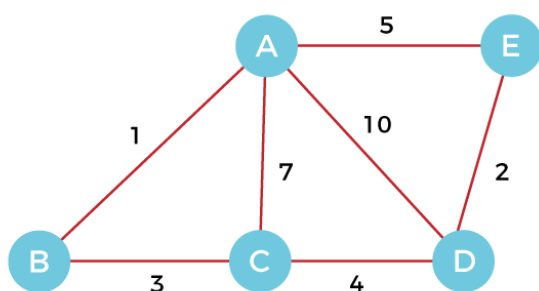
The steps to implement Kruskal's algorithm are listed as follows -

- First, sort all the edges from low weight to high.
- Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
- Continue to add the edges until we reach all vertices, and a minimum spanning tree is created.

The applications of Kruskal's algorithm are -

- Kruskal's algorithm can be used to layout electrical wiring among cities.
- It can be used to lay down LAN connections.

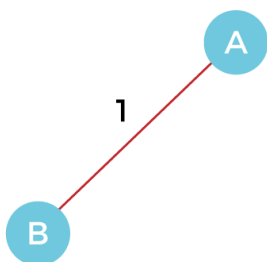
Example of Kruskal's algorithm



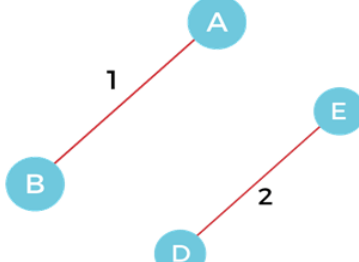
- Sort the edges given above in the ascending order of their weights.

Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

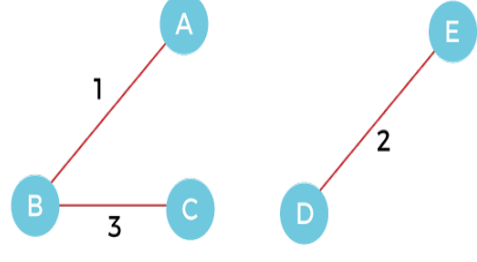
Step 1 - First, add edge **AB** with Min weight **1** to the MST.



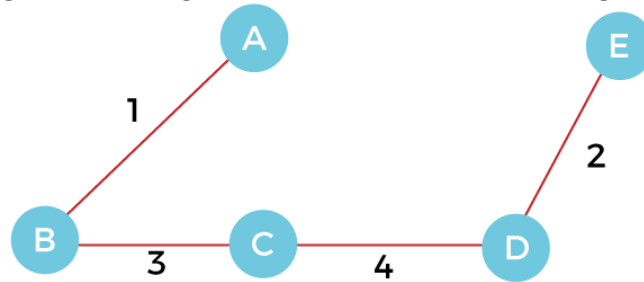
Step 2 - Add the edge **DE** with weight **2** to the MST as it is not creating the cycle.



Step 3 - Add the edge **BC** with weight **3** to the MST, as it is not creating any cycle or loop.



Step 4 - Now, pick the edge **CD** with weight **4** to the MST, as it is not forming the cycle.

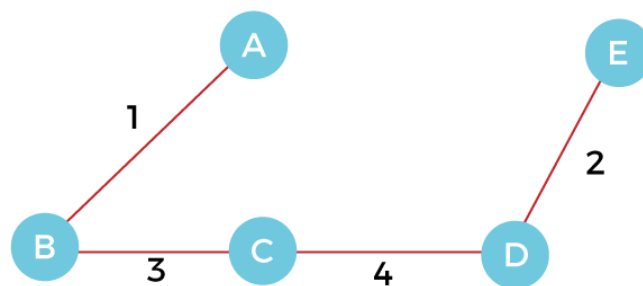


Step 5 - After that, pick the edge **AE** with weight **5**. Including this edge will create the cycle, so discard it.

Step 6 - Pick the edge **AC** with weight **7**. Including this edge will create the cycle, so discard it.

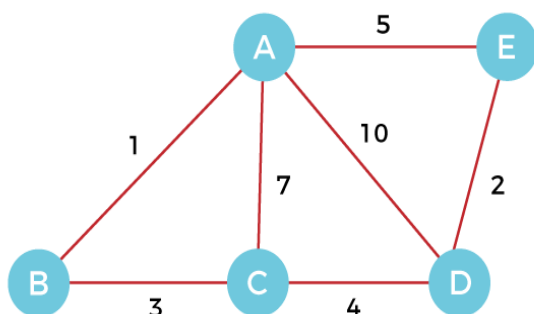
Step 7 - Pick the edge **AD** with weight **10**. Including this edge will also create the cycle, so discard it.

So, the final minimum spanning tree obtained from the given weighted graph by using Kruskal's algorithm is –



The cost of the MST is = $AB + DE + BC + CD = 1 + 2 + 3 + 4 = 10$.

Consider the Graph:



Cost Matrix:

	A	B	C	D	E
A	0	1	7	10	5
B	1	0	3	0	0
C	7	3	0	4	0
D	10	0	4	0	2
E	5	0	0	2	0

C Program to demonstrate MSP.

```
#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 readCostMatrix()
{
    printf("\n Enter the number of nodes:");
    scanf("%d", &n);
    printf("\n Enter the cost 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] = 999;
        }
}

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;
}
```

```
int main()
{
    printf ("\n Kruskal's algorithm to find Minimum Cost spanning Tree ");

    readCostMatrix();          // Function to read the cost matrix

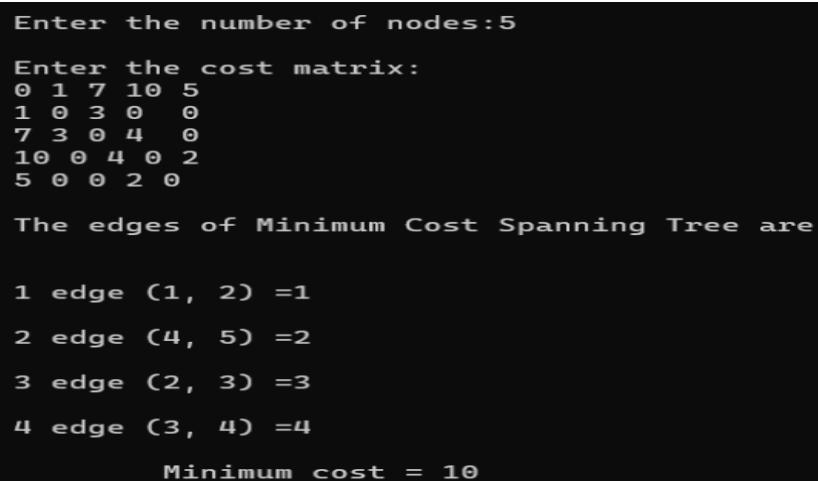
    printf ("\n The edges of Minimum Cost Spanning Tree are\n\n");

    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 ("\n %d edge (%d, %d) =%d\n", ne++, a, b, min);
            mincost += min;
        }
        cost [a][b] = cost[b][a] = 999;
    }
    printf ("\n\t Minimum cost = %d\n", mincost);
}
```

Run1



```
Enter the number of nodes:5

Enter the cost matrix:
0 1 7 10 5
1 0 3 0 0
7 3 0 4 0
10 0 4 0 2
5 0 0 2 0

The edges of Minimum Cost Spanning Tree are

1 edge (1, 2) =1
2 edge (4, 5) =2
3 edge (2, 3) =3
4 edge (3, 4) =4

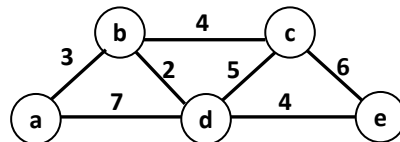
Minimum cost = 10
```

8. From a given vertex in a weighted connected graph, find shortest paths to other vertices Using Dijkstra's algorithm (C programming)

• Single-source shortest-paths problem:

- “For a given vertex called the source in a weighted connected graph, find shortest paths to all its other vertices”. Some paths may, have edges in common.
- The best-known algorithm for the single-source shortest-paths problem is a Dijkstra’s algorithm

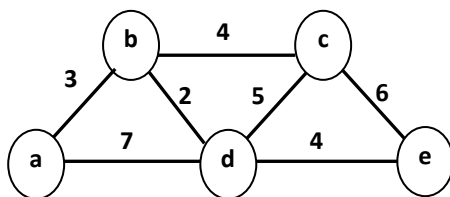
• Example:



Tree vertices	Remaining vertices	Illustration
a (, 0)	b (a,3) c(, ∞) d(, 7) e(, ∞)	
b (a, 3)	c(b, 3+4) d (b, 3+2) e(, ∞)	
d (b, 5)	c(b, 7) e(d, 5+4)	
c (b, 7)	e (d, 5+4)	
e(d, 9)		

The next closest vertex is shown in bold

Consider a graph



Cost Matrix:

	A	B	C	D	E
A	0	3	0	7	0
B	3	0	4	2	0
C	0	4	0	5	6
D	7	2	5	0	4
E	0	0	6	4	0

// C program for Dijkstra's single source shortest path

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

int nodes, graph[10][10];

void readGraph()
{
    int i, j;

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

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

// A utility function to find the vertex with minimum distance value,
// from the set of vertices not yet included in shortest path tree

int minDistance (int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index, v;

    for (v = 0; v < nodes; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance array
void printSolution(int dist[])
{
    int i;
    printf("Vertex \t\t Distance from Source\n");
    for (i = 0; i < nodes; i++)
        printf("%d \t\t\t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's single source shortest path
```

```
void dijkstra(int src)
{
    int dist[nodes], i, count, v;    // The output array. dist[i] will hold the shortest distance from src to i

    bool sptSet[nodes];    // sptSet[i] will be true if vertex i is included in shortest distance from src

    // Initialize all distances as INFINITE and sptSet[] as false

    for (i = 0; i < nodes; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;    // Distance of source vertex from itself is always 0

    // Find shortest path for all vertices
    for (count = 0; count < nodes - 1; count++)
    {
        // Pick the minimum distance vertex from the set of vertices not yet processed.
        // u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (v = 0; v < nodes; v++)    // Update dist[v] only if it is not in sptSet and there is an edge from u
            // to v, and total weight of (src to v) through u is smaller than current value of dist[v]

            if (!sptSet[v] && graph[u][v]
                && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

int main()
{
    int graph[10][10];

    /* Let us create the example graph discussed above */
    readGraph(graph);

    // Function call
    dijkstra(0);

    return 0;
}
```

OUTPUT

```
Enter the no of vertices:
5
Enter the adjacency matrix:
0 3 0 7 0
3 0 4 2 0
0 4 0 5 6
7 2 5 0 4
0 0 6 4 0
Vertex          Distance from Source
0                0
1                3
2                7
3                5
4                9
-----
```