EXPERIMENT NO. 1

Aim :- Write a Program in C to implement Selection Sort.

```
#include <stdio.h>


void selection(int arr[], int n)
{
 int i, j, small;


 for (i = 0; i < n-1; i++) // One by one move boundary of unsorted subarray
 {
 small = i; //minimum element in unsorted array


 for (j = i+1; j < n; j++)
 if (arr[j] < arr[small])
 small = j;
// Swap the minimum element with the first element
 int temp = arr[small];
 arr[small] = arr[i];
 arr[i] = temp;
 }
}


void printArr(int a[], int n) /* function to print the array */
{
 int i;
 for (i = 0; i < n; i++)
```

```
 printf("%d ", a[i]);
}

int main()
{
 int a[] = { 12, 31, 25, 8, 32, 17 };
 int n = sizeof(a) / sizeof(a[0]);
 printf("Before sorting array elements are - \n");
 printArr(a, n);
 selection(a, n);
 printf("\nAfter sorting array elements are - \n");
 printArr(a, n);
 return 0;
}
```

Output:-

```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
```

EXPERIMENT NO. 2

Aim :- Write a Program in C to implement insertion sort.

```
#include <stdio.h>

void insert(int a[], int n) /* function to sort an array with insertion sort */
{
 int i, j, temp;
 for (i = 1; i < n; i++) {
 temp = a[i];
 j = i - 1;

 while(j>=0 && temp <= a[j]) /* Move the elements greater than temp to one
position ahead from their current position*/
 {
 a[j+1] = a[j];
 j = j-1;
 }
 a[j+1] = temp;
 }
}

void printArr(int a[], int n) /* function to print the array */
{
 int i;
 for (i = 0; i < n; i++)
```

```
 printf("%d ", a[i]);

}


int main()

{

 int a[] = { 31,12,4,20,43 };

 int n = sizeof(a) / sizeof(a[0]);

 printf("Before sorting array elements are - \n");

 printArr(a, n);

 insert(a, n);

 printf("\nAfter sorting array elements are - \n");

 printArr(a, n);


 return 0;

}
```

Output:-

```
Before sorting array elements are -
31 12 4 20 43
After sorting array elements are -
4 12 20 31 43
```

EXPERIMENT NO. 3

Aim :- Write a Program in c to implement Merge Sort.

```
#include <stdio.h>

/* Function to merge the subarrays of a[] */
void merge(int a[], int beg, int mid, int end)
{
 int i, j, k;
 int n1 = mid - beg + 1;
 int n2 = end - mid;

 int LeftArray[n1], RightArray[n2]; //temporary arrays

 /* copy data to temp arrays */
 for (int i = 0; i < n1; i++)
 LeftArray[i] = a[beg + i];
 for (int j = 0; j < n2; j++)
 RightArray[j] = a[mid + 1 + j];

 i = 0; /* initial index of first sub-array */
 j = 0; /* initial index of second sub-array */
 k = beg; /* initial index of merged sub-array */

 while (i < n1 && j < n2)
 {
 if(LeftArray[i] <= RightArray[j])
```

```
{
a[k] = LeftArray[i];
i++;
}
else
{
a[k] = RightArray[j];
j++;
}
k++;
}
while (i<n1)
{
a[k] = LeftArray[i];
i++;
k++;
}

while (j<n2)
{
a[k] = RightArray[j];
j++;
k++;
}
}
```

```
void mergeSort(int a[], int beg, int end)
{
 if (beg < end)
 {
 int mid = (beg + end) / 2;
 mergeSort(a, beg, mid);
 mergeSort(a, mid + 1, end);
 merge(a, beg, mid, end);
 }
}

/* Function to print the array */
void printArray(int a[], int n)
{
 int i;
 for (i = 0; i < n; i++)
 printf("%d ", a[i]);
 printf("\n");
}

int main()
{
 int a[] = { 12, 31, 25, 8, 32, 17, 40, 42 };
 int n = sizeof(a) / sizeof(a[0]);
 printf("Before sorting array elements are - \n");
 printArray(a, n);
```

```
 mergeSort(a, 0, n - 1);
 printf("After sorting array elements are - \n");
 printArray(a, n);
 return 0;
}
```

Output:-

```
Before sorting array elements are -
12 31 25 8 32 17 40 42
After sorting array elements are -
8 12 17 25 31 32 40 42
```

EXPERIMENT NO. 4

Aim :- Write a program in c to implement Quick sort

```
#include <stdio.h>

/* function that consider last element as pivot,

place the pivot at its exact position, and place

smaller elements to left of pivot and greater

elements to right of pivot. */

int partition (int a[], int start, int end)

{

int pivot = a[end]; // pivot element

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

// If current element is smaller than the pivot

if (a[j] < pivot)

{

i++; // increment index of smaller element

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

int t = a[i+1];

a[i+1] = a[end];

a[end] = t;

return (i + 1);
```

```
}
/* function to implement quick sort */
void quick(int a[], int start, int end) /* a[] = array to be sorted, start =
Starting index, end = Ending index */
{
if (start < end)
{
int p = partition(a, start, end); //p is the partitioning index
quick(a, start, p - 1);
quick(a, p + 1, end);
}
}
/* function to print an array */
void printArr(int a[], int n)
{
int i;
for (i = 0; i < n; i++)
printf("%d ", a[i]);
}
int main()
{
int a[] = { 24, 9, 29, 14, 19, 27 };
int n = sizeof(a) / sizeof(a[0]);
printf("Before sorting array elements are - \n");
printArr(a, n);
quick(a, 0, n - 1);
```

```
printf("\nAfter sorting array elements are - \n");

printArr(a, n);

return 0;

}
```

Output:-

```
Before sorting array elements are -
24 9 29 14 19 27
After sorting array elements are -
9 14 19 24 27 29
```

EXPERIMENT NO. 5

Aim :- Write a program in c to implement Binary Search.

```
#include <stdio.h>

int binarySearch(int a[], int beg, int end, int val)

{

int mid;

if(end >= beg)

{ mid = (beg + end)/2;

/* if the item to be searched is present at middle */

if(a[mid] == val)

{

return mid+1;

}

/* if the item to be searched is smaller than middle, then it

can only be in left subarray */

else if(a[mid] < val)

{

return binarySearch(a, mid+1, end, val);

}

/* if the item to be searched is greater than middle, then it

can only be in right subarray */

else

{

return binarySearch(a, beg, mid-1, val);

}

}
```

```
return -1;

}

int main() {

int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array

int val = 40; // value to be searched

int n = sizeof(a) / sizeof(a[0]); // size of array

int res = binarySearch(a, 0, n-1, val); // Store result

printf("The elements of the array are - ");

for (int i = 0; i < n; i++)

printf("%d ", a[i]);

printf("\nElement to be searched is - %d", val);

if (res == -1)

printf("\nElement is not present in the array");

else

printf("\nElement is present at %d position of array", res);

return 0;

}
```

Output:-

```
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 40
Element is present at 5 position of array
```

EXPERIMENT NO. 6

Aim :- Write a program in c to implement Dijkstra Algorithm.

```
// Implementation of Dijkstra's Algorithm in C

// importing the standard I/O header file

#include <stdio.h>

// defining some constants

#define INF 9999

#define MAX 10

// prototyping of the function

void DijkstraAlgorithm(int Graph[MAX][MAX], int size, int start);

// defining the function for Dijkstra's Algorithm

void DijkstraAlgorithm(int Graph[MAX][MAX], int size, int start) {

int cost[MAX][MAX], distance[MAX], previous[MAX];

int visited_nodes[MAX], counter, minimum_distance, next_node, i, j;

// creating cost matrix

for (i = 0; i < size; i++)

for (j = 0; j < size; j++)

if (Graph[i][j] == 0)

cost[i][j] = INF;

else

cost[i][j] = Graph[i][j];

for (i = 0; i < size; i++) {

distance[i] = cost[start][i];

previous[i] = start;

visited_nodes[i] = 0;

}
```

```
distance[start] = 0;
visited_nodes[start] = 1;
counter = 1;
while (counter < size - 1) {
minimum_distance = INF;
for (i = 0; i < size; i++)
if (distance[i] < minimum_distance && !visited_nodes[i]) {
minimum_distance = distance[i];
next_node = i;
}
visited_nodes[next_node] = 1;
for (i = 0; i < size; i++)
if (!visited_nodes[i])
if (minimum_distance + cost[next_node][i] < distance[i]) {
distance[i] = minimum_distance + cost[next_node][i];
previous[i] = next_node;
}
counter++;
}
// printing the distance
for (i = 0; i < size; i++)
if (i != start) {
printf("\nDistance from the Source Node to %d: %d", i,
distance[i]);
}
}
```

```
// main function
int main() {
// defining variables
int Graph[MAX][MAX], i, j, size, source;
// declaring the size of the matrix
size = 7;
// declaring the nodes of graph
Graph[0][0] = 0;
Graph[0][1] = 4;
Graph[0][2] = 0;
Graph[0][3] = 0;
Graph[0][4] = 0;
Graph[0][5] = 8;
Graph[0][6] = 0;
Graph[1][0] = 4;
Graph[1][1] = 0;
Graph[1][2] = 8;
Graph[1][3] = 0;
Graph[1][4] = 0;
Graph[1][5] = 11;
Graph[1][6] = 0;
Graph[2][0] = 0;
Graph[2][1] = 8;
Graph[2][2] = 0;
Graph[2][3] = 7;
Graph[2][4] = 0;
```

```
Graph[2][5] = 4;

Graph[2][6] = 0;

Graph[3][0] = 0;

Graph[3][1] = 0;

Graph[3][2] = 7;

Graph[3][3] = 0;

Graph[3][4] = 9;

Graph[3][5] = 14;

Graph[3][6] = 0;

Graph[4][0] = 0;

Graph[4][1] = 0;

Graph[4][2] = 0;

Graph[4][3] = 9;

Graph[4][4] = 0;

Graph[4][5] = 10;

Graph[4][6] = 2;

Graph[5][0] = 0;

Graph[5][1] = 0;

Graph[5][2] = 4;

Graph[5][3] = 14;

Graph[5][4] = 10;

Graph[5][5] = 0;

Graph[5][6] = 2;

Graph[6][0] = 0;

Graph[6][1] = 0;

Graph[6][2] = 0;
```

Graph[6][3] = 0;

Graph[6][4] = 2;

Graph[6][5] = 0;

Graph[6][6] = 1;

source = 0;

// calling the DijkstraAlgorithm() function by passing the Graph, thenumber of nodes and the source node

DijkstraAlgorithm(Graph, size, source);

return 0;

}

Output:-

```
Distance from the Source Node to 1: 4
Distance from the Source Node to 2: 12
Distance from the Source Node to 3: 19
Distance from the Source Node to 4: 12
Distance from the Source Node to 5: 8
Distance from the Source Node to 6: 10
```

EXPERIMENT NO. 8

Aim :- Write a program in c to implement Floyd Worshall Algorithm.

```
#include <stdio.h>

#include <stdlib.h>


void floydWarshall(int **graph, int n)

{

int i, j, k;

for (k = 0; k < n; k++)

{

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

if (graph[i][j] > graph[i][k] + graph[k][j])

graph[i][j] = graph[i][k] + graph[k][j];  }

}

}

}


int main(void)

{

int n, i, j;

printf("Enter the number of vertices: ");

scanf("%d", &n);

int **graph = (int **)malloc((long unsigned) n * sizeof(int *));
```

```
for (i = 0; i < n; i++)
{
graph[i] = (int *)malloc((long unsigned) n * sizeof(int));  }
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
if (i == j)
graph[i][j] = 0;
else
graph[i][j] = 100;
}
}
printf("Enter the edges: \n");
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
{
printf("[%d][%d]: ", i, j);
scanf("%d", &graph[i][j]); }
}
printf("The original graph is:\n");
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
 {
```

```
 printf("%d ", graph[i][j]);  }
 printf("\n");
 }
 floydWarshall(graph, n);
 printf("The shortest path matrix is:\n");
for (i = 0; i < n; i++)
 {
 for (j = 0; j < n; j++)
 {
 printf("%d ", graph[i][j]);  }
 printf("\n");
 }
 return 0;
 }
```

Output:-

```
Enter the number of vertices: 3
Enter the edges:
[0][0]: 3
[0][1]: 3
[0][2]: 3
[1][0]: 1
```

```
[1][1]: 2
[1][2]: 6
[2][0]: 7
[2][1]: 7
[2][2]: 5
The original graph is:
3 3 3
1 2 6
7 7 5
The shortest path matrix is:
3 3 3
1 2 4
7 7 5
```

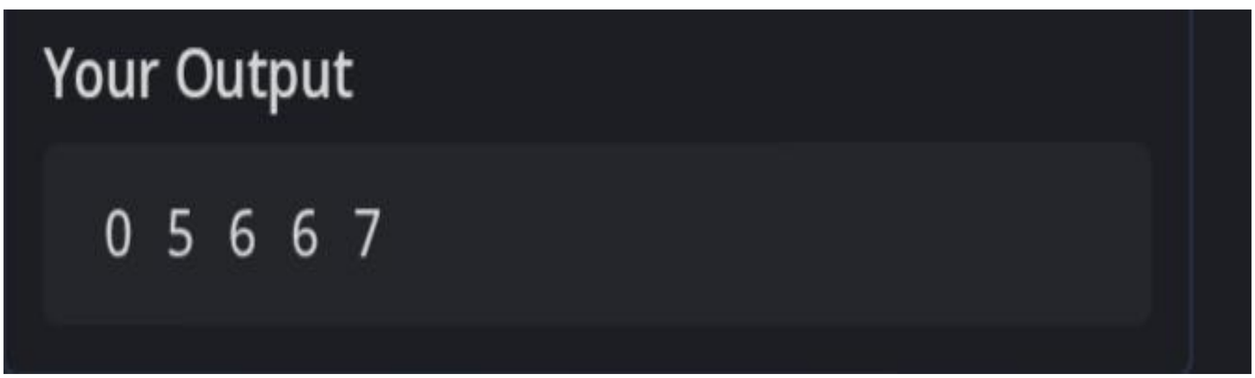
EXPERIMENT NO. 7

Aim :- Write a program in c to implement Belman Ford Algorithm.

```
#include <iostream>

#include <vector>

using namespace std;

vector<int> bellmanFord(int V, vector<vector<int>>& edges, int src) {

// Initially distance from source to all

// other vertices is not known(Infinite).

vector<int> dist(V, 1e8);

dist[src] = 0;

// Relaxation of all the edges V times, not (V - 1) as we

// need one additional relaxation to detect negative cycle

for (int i = 0; i < V; i++) {

for (vector<int> edge : edges) {

int u = edge[0];

int v = edge[1];

int wt = edge[2];

if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {

// If this is the Vth relaxation, then there is

// a negative cycle

if(i == V - 1)

return {-1};

// Update shortest distance to node v

dist[v] = dist[u] + wt;

}

}
```

```
}
return dist;
}
int main() {
int V = 5;
vector<vector<int>> edges = {{1, 3, 2}, {4, 3, -1}, {2, 4, 1}, {1, 2, 1}, {0, 1, 5}};
int src = 0;
vector<int> ans = bellmanFord(V, edges, src);
for (int dist : ans)
cout << dist << " ";
return 0;
}
```

Output :-



EXPERIMENT NO. 9

Aim :- Write a program to implement n queen problem using c.

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


int board[20], count = 0;


// Function prototypes (important!)

void queen(int row, int n);

int place(int row, int column);

void print(int n);


int main() {

   int n;

   printf(" - N Queens Problem Using Backtracking -");

   printf("\n\nEnter number of Queens: ");

   scanf("%d", &n);

   queen(1, n);

   return 0;

}


// Print the solution board

void print(int n) {

   int i, j;

   printf("\n\nSolution %d:\n\n", ++count);
```

```
    for (i = 1; i <= n; ++i)
        printf("\t%d", i);
    for (i = 1; i <= n; ++i) {
        printf("\n\n%d", i);
        for (j = 1; j <= n; ++j) {
            if (board[i] == j)
                printf("\tQ");
            else
                printf("\t-");
        }
    }
    printf("\n");
}
// Check whether queen can be placed
int place(int row, int column) {
    int i;
    for (i = 1; i <= row - 1; ++i) {
        if (board[i] == column || abs(board[i] - column) == abs(i - row))
            return 0;
    }
    return 1;
}
// Backtracking
void queen(int row, int n) {
    int column;
    for (column = 1; column <= n; ++column) {
```

```
        if (place(row, column)) {
          board[row] = column;
          if (row == n)
            print(n);
          else
            queen(row + 1, n);
        }
      }
    }
```

Output :-

```
Output                                          Clear
 - N Queens Problem Using Backtracking -

Enter number of Queens: 4

Solution 1:

      1    2    3    4

1     -    Q    -    -

2     -    -    -    Q

3     Q    -    -    -

4     -    -    Q    -

Solution 2:

      1    2    3    4

1     -    -    Q    -

2     Q    -    -    -

3     -    -    -    Q

4     -    Q    -    -

=== Code Execution Successful ===
```

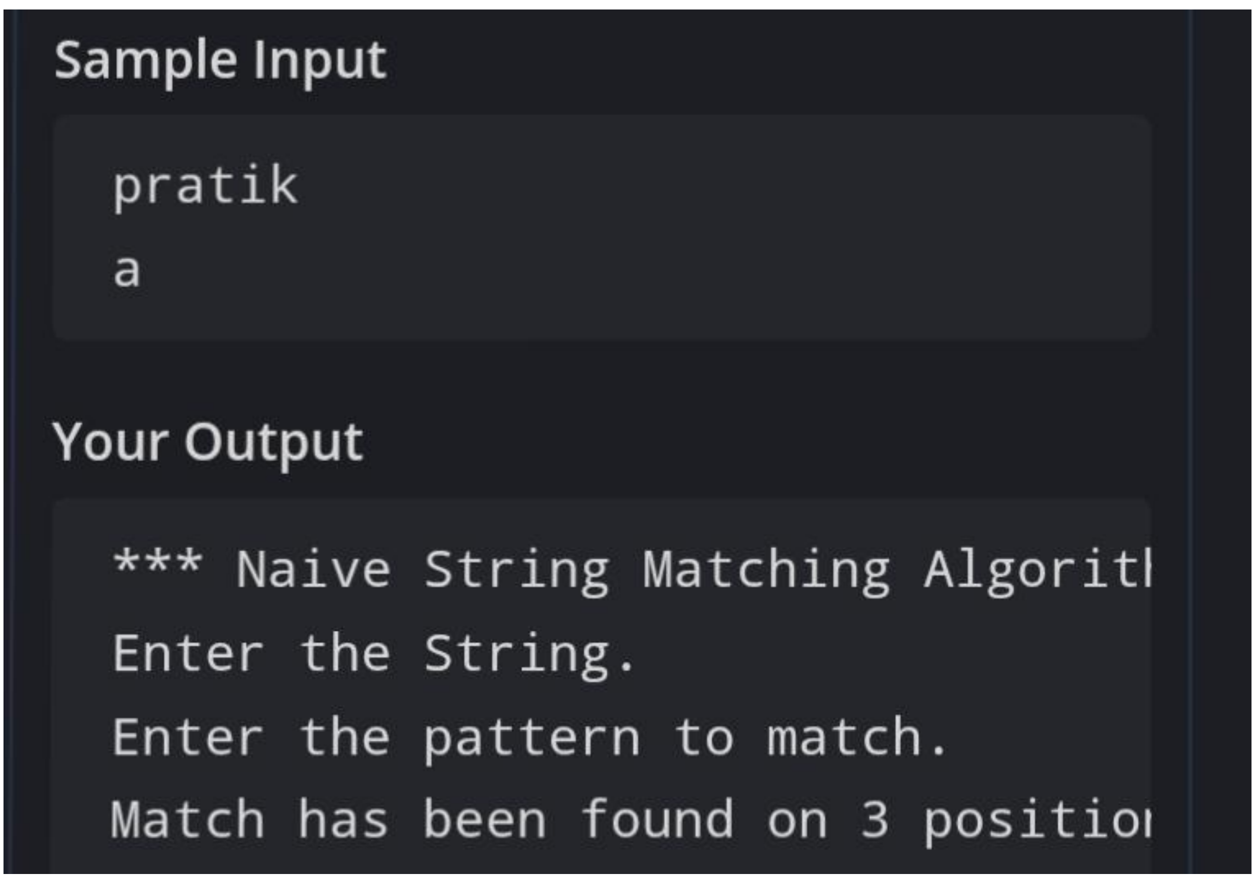
EXPERIMENT NO. 10

Aim :- Implement Naive String Matching Algorithm using C.

```
#include<stdio.h>

#include<string.h>

int match(char st[100], char pat[100]);

int main(int argc, char **argv) {

char st[100], pat[100];

int status;

printf("*** Naive String Matching Algorithm ***\n");

printf("Enter the String.\n");

gets(st);

printf("Enter the pattern to match.\n");

gets(pat);

status = match(st, pat);

if (status == -1)

printf("\nNo match found");

else

printf("Match has been found on %d position.", status); return

0;

}

int match(char st[100], char pat[100]) {

int n, m, i, j, count = 0, temp = 0;

n = strlen(st);

m = strlen(pat);

for (i = 0; i <= n - m; i++) {

temp++;
```

```
for (j = 0; j < m; j++) {

if (st[i + j] == pat[j])

count++;

}

if (count == m)

return temp;

count = 0;

}

return -1;

}
```

Output :-



EXPERIMENT NO. 11

Aim :- Prim’s Algorithm for Minimum Spanning Tree (MST)using c

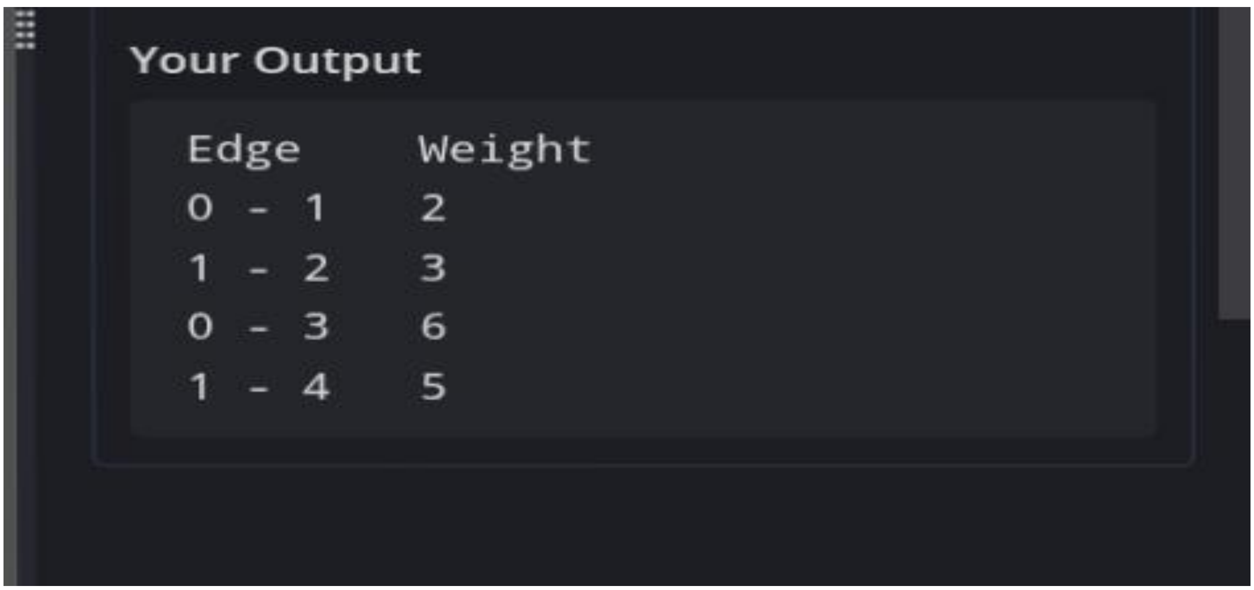
```
#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

// Number of vertices in the graph

#define V 5

// A utility function to find the vertex with

// minimum key value, from the set of vertices

// not yet included in MST

int minKey(int key[], bool mstSet[])

{

// Initialize min value

int min = INT_MAX, min_index;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min)

min = key[v], min_index = v;

return min_index;

}

// A utility function to print the

// constructed MST stored in parent[]

int printMST(int parent[], int graph[V][V])

{

printf("Edge \tWeight\n");

for (int i = 1; i < V; i++)

printf("%d - %d \t%d \n"
```

```
, parent[i], i,
graph[parent[i]][i]);
}
// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
// Array to store constructed MST
int parent[V];
// Key values used to pick minimum weight edge in cut
int key[V];
// To represent set of vertices included in MST
bool mstSet[V];
// Initialize all keys as INFINITE
for (int i = 0; i < V; i++)
key[i] = INT_MAX, mstSet[i] = false;
// Always include first 1st vertex in MST.
// Make key 0 so that this vertex is picked as first
// vertex.
key[0] = 0;
// First node is always root of MST
parent[0] = -1;
// The MST will have V vertices
for (int count = 0; count < V - 1; count++) {
// Pick the minimum key vertex from the
```

```
// set of vertices not yet included in MST
int u = minKey(key, mstSet);
// Add the picked vertex to the MST Set
mstSet[u] = true;
// Update key value and parent index of
// the adjacent vertices of the picked vertex.
// Consider only those vertices which are not
// yet included in MST
for (int v = 0; v < V; v++)
// graph[u][v] is non zero only for adjacent
// vertices of m mstSet[v] is false for vertices
// not yet included in MST Update the key only
// if graph[u][v] is smaller than key[v]
if (graph[u][v] && mstSet[v] == false
&& graph[u][v] < key[v])
parent[v] = u, key[v] = graph[u][v];
}
// print the constructed MST
printMST(parent, graph);
}
// Driver's code
int main()
{
int graph[V][V] = { { 0, 2, 0, 6, 0 },
{ 2, 0, 3, 8, 5 },
{ 0, 3, 0, 0, 7 },
```

```
{ 6, 8, 0, 0, 9 },
{ 0, 5, 7, 9, 0 } };
// Print the solution
primMST(graph);
return 0;
}
```

Output :-



EXPERIMENT NO. 12

Aim :- Implementation of Fractional Knapsack Problem.

```
#include <stdio.h>

int n = 5;

int p[10] = {3, 3, 2, 5, 1};

int w[10] = {10, 15, 10, 12, 8};

int W = 10;


int main() {

  int cur_w;

  float tot_v = 0.0; //  initialize

  int i, maxi;

  int used[10];

  for (i = 0; i < n; ++i)

    used[i] = 0;

  cur_w = W;

  while (cur_w > 0) {

    maxi = -1;

    for (i = 0; i < n; ++i)

      if ((used[i] == 0) &&

        ((maxi == -1) || ((float) w[i] / p[i] > (float) w[maxi] / p[maxi])))

        maxi = i;

    used[maxi] = 1;

    cur_w -= p[maxi];

    tot_v += w[maxi];
```

```
        if (cur_w >= 0)
          printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n",
              maxi + 1, w[maxi], p[maxi], cur_w);
        else {
          printf("Added %d%% (%d, %d) of object %d in the bag.\n",
              (int)((1 + (float)cur_w / p[maxi]) * 100),
              w[maxi], p[maxi], maxi + 1);

          tot_v -= w[maxi];
          tot_v += (1 + (float)cur_w / p[maxi]) * w[maxi];
        }
      }
      printf("Filled the bag with objects worth %.2f.\n", tot_v);
      return 0;
    }
```

Output :-

Output

```
Added object 5 (8, 1) completely in the bag. Space left: 9.
Added object 2 (15, 3) completely in the bag. Space left: 6.
Added object 3 (10, 2) completely in the bag. Space left: 4.
Added object 1 (10, 3) completely in the bag. Space left: 1.
Added 19% (12, 5) of object 4 in the bag.
Filled the bag with objects worth 45.40.
```