2021

Shivaji Rao Kadam Institute of technology and management, Technical Campus, Indore

Transnational Knowledge Society's
Shivajirao Kadam Institute
of Technology & Management
Skill. Innnovation.Transformation

Department of Computer Science & Engineering

Submitted To:
Prof. Surbhi Kushwaha
(Asst. Prof. CSE)

Analysis and Design of Algorithm

Submitted By:

Name of Student: Kartik Kulshreshtha

Enrollment No. :0875CS191048

Class/Year/Sem : CS-A/2nd/4th

[LAB ASSIGNMENT FOR CS-402]

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-01
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	Iterative and Recursive binary search				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
22/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
3. Shown capability to solve the problems		
Average (out of 10)		

Title: Iterative and Recursive binary search

Objective: To know the how to apply binary search method iteratively and recursively

Theory :

Binary Search is a search algorithm that is used to find the position of an element (target value) in a sorted array. The array should be sorted prior to applying a binary search.

Binary search is also known by these names, logarithmic search, binary chop, half interval search.

Working:

The binary search algorithm works by comparing the element to be searched by the middle element of the array and based on this comparison follows the required procedure.

Case 1 − element = middle, the element is found return the index.

Case 2 − element > middle, search for the element in the sub-array starting from middle+1 index to n.

Case 3 − element < middle, search for element in the sub-array starting from 0 index to middle -1

Algorithm of Binary Search:

Step-1: Find the middle element in the array using:

middle = initial value + end value/2;

Step-2: If middle = element, then return "element found" and index.

Step-3: If middle > element, call the function with end value = middle-1.

Step-4: If middle < element, call the function with end value = middle+1.

Step-5: exit.

"Program to implement binary search using Iterative call":

binary.c

```c
#include <stdio.h>
int iterativeBinarySearch(int array[], int start_index, int end_index, int element){
    while (start_index <= end_index){
        int middle = start_index + (end_index- start_index )/2;
        if (array[middle] == element)
            return middle;
        if (array[middle] < element)
            start_index = middle + 1;
        else
            end_index = middle - 1;
    }
    return -1;
}
int main(void){
    int array[] = {1, 4, 7, 9, 16, 56, 70};
    int n = 7;
    int element = 16;
    int found_index = iterativeBinarySearch(array, 0, n-1, element);
    if(found_index == -1 ) {
        printf("Element not found in the array ");
    }
    else {
        printf("Element found at index : %d",found_index);
    }
    return 0;
}
```

“Program to implement binary search using Iterative call”:

binary.c

```c
#include <stdio.h>
int recursiveBinarySearch(int array[], int start_index, int end_index, int element){
    if (end_index >= start_index){
        int middle = start_index + (end_index - start_index )/2;
        if (array[middle] == element)
            return middle;
        if (array[middle] > element)
            return recursiveBinarySearch(array, start_index, middle-1, element);
        return recursiveBinarySearch(array, middle+1, end_index, element);
    }
    return -1;
}
int main(void){
    int array[] = {1, 4, 7, 9, 16, 56, 70};
    int n = 7;
    int element = 9;
    int found_index = recursiveBinarySearch(array, 0, n-1, element);
    if(found_index == -1 ) {
        printf("Element not found in the array ");
    }
    else {
        printf("Element found at index : %d",found_index);
    }
    return 0;
}
```

Output for Binary search using Recursive calls:

```
if ($?) { gcc binary.c -o binary } ; if ($?) { .\binary }
Element found at index : 3
PS C:\Users\kartik kulshreshtha\OneDrive\Data Structures\Searching>
```

Output for Binary search using Iterative calls:

```
if ($?) { gcc binary.c -o binary } ; if ($?) { .\binary }
Element found at index : 4
PS C:\Users\kartik kulshreshtha\OneDrive\Data Structures\Searching>
```

Shivajirao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-02
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	Divide and conquer technique method that is Merge Sort.			**CO No.**	
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
24/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
3. Shown capability to solve the problems		
Average (out of 10)		

Title: Divide and conquer technique method that is Merge Sort.

Objective: To Know the concept of Merge Sort.

Merge Sort:

Merge sort is the algorithm which follows divide and conquer approach. Consider and array algorithm processes the elements in 3 steps.

1. If A Contains 0 or 1 elements then it is already sorted, otherwise, Divide A into two sub-arrays of equal number of elements.
2. Conquer means sort the two sub-arrays recursively using the merge sort.
3. Combine the sub-arrays to form a single final sorted array maintaining the ordering of the array.

Algorithm for Merge Sort:

Step 1- If it is only one element in the list it is already sorted, return.

Step 2- Divide the list recursively into two halves until it can no more be divided.

Step 3- Merge the smaller lists into new list in sorted order.

Program of merge sort in C language:

MergeSort.c

```c
#include<stdio.h>
void mergeSort(int[],int,int);
void merge(int[],int,int,int);
void main ()
{
    int a[10]= {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    int i;
    mergeSort(a,0,9);
    printf("The sorted elements are : \n");
    for(i=0;i<10;i++)
    {
        printf("%d ",a[i]);
    }

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

MergeSort.c

```c
}
void merge(int a[], int beg, int mid, int end)
{
    int i=beg,j=mid+1,k,index = beg;
    int temp[10];
    while(i<=mid && j<=end)
    {
        if(a[i]<a[j])
        {
            temp[index] = a[i];
            i = i+1;
        }
        else
        {
            temp[index] = a[j];
            j = j+1;
        }
        index++;
    }
    if(i>mid)
    {
        while(j<=end)
        {
            temp[index] = a[j];
            index++;
            j++;
```

```
        }
    }
    else
    {
        while(i<=mid)
        {
            temp[index] = a[i];
            index++;
            i++;
        }
    }
    k = beg;
    while(k<index)
    {
        a[k]=temp[k];
        k++;
    }
}
```

The Output is:

```
The sorted elements are :
7 9 10 12 23 23 34 44 78 101
PS C:\Users\kartik kulshreshtha\OneDrive\Data Structures\Searching>
```

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-03
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	**Divide and conquer technique method that is Quick Sort**				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
22/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
3. Shown capability to solve the problems		
Average (out of 10)		

Title: Divide and conquer technique method that is Quick Sort.

Objective: To Know the concept of Quick Sort.

Theory: Like Merge sort, Quicksort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quicksort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quicksort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller

elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Algorithm for Quicksort:

Step-1: Choose the highest index value has pivot.

Step-2: Take two variables to point left and right of the list excluding pivot.

Step-3: left points to the high index.

Step-4: right points to the high.

Step-5: while value at left is less than pivot move right.

Step-6: while value at right is greater than pivot move left.

Step-7: if both step 5 and step 6 does not match swap left and right.

Step-8: if left >=right, the point where they net is new pivot.

Program for Quick Sort:

QuickSort.cpp

```cpp
#include <stdio.h>

void printArray(int *A, int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", A[i]);
    }
    printf("\n");
}

int partition(int A[], int lb, int ub)
{
    int temp;
    int pivot = A[lb];
    int start = lb;
    int end = ub;
    while(start<end){
        while(A[start]<= pivot){
            start++;
        }
        while(A[end]>pivot){
            end--;
        }
        if(start<end){
            temp = A[start];
```

```
25          if(start<end){
26              temp = A[start];
27              A[start] = A[end];
28              A[end] = temp;
29          }
30
31      }
32      temp = A[lb];
33      A[lb] = A[end];
34      A[end] = temp;
35      return end;
36  }
37
38  void quickSort(int A[], int low, int high)
39  {
40      int Loc;
41
42      if (low < high)
43      {
44          Loc = partition(A, low, high);
45          quickSort(A, low, Loc - 1);  // sort left subarray
46          quickSort(A, Loc + 1, high); // sort right subarray
47      }
48  }
49
```

```
int main()
{

    int A[] = {9, 4, 4, 8, 7, 5, 6};
    int n;
    n =7;
    printArray(A, n);
    quickSort(A, 0, n - 1);
    printArray(A, n);
    return 0;
}
```

The Output is:

```
Array before sorting : 9 4 4 8 7 5 6
Array after sorting : 4 4 5 6 7 8 9
PS C:\Users\kartik kulshreshtha\OneDrive\Data Structures\Sorting>
```

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-04
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	Divide and conquer technique				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
28/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
3. Shown capability to solve the problems		
Average (out of 10)		

Title: **Divide and conquer technique**

Objective: **To understand Strassen's Matrix Multiplication.**

Theory: Strassen in 1969 which gives an overview that how we can find the multiplication of two 2*2dimension matrix by the brute-force algorithm. For multiplying the two 2 * 2dimension matrices Strassen's used some formulas in which there are seven multiplication and eighteen addition, subtraction, and in brute force algorithm, there is eight multiplication and four addition. The utility of Strassen's formula is shown by its asymptotic superiority when order n of matrix reaches infinity. Let us consider two
matrices A and B, n*n dimension, where n is a power of two. It can be observed that we can

contain four n/2*n/2 submatrices from A, B and their product C. C is the resultant matrix of A and B.

Procedure of Strassen matrix multiplication

There are some procedures:

1. Divide a matrix of order of 2*2 recursively till we get the matrix of 2*2.
2. Use the previous set of formulas to carry out 2*2 matrix multiplication.
3. In this eight multiplication and four additions, subtraction are performed.
4. Combine the result of two matrixes to find the final product or final matrix.

Algorithm for Strassen's matrix multiplication:

```
begin
    If n = threshold then compute
        C = a * b is a conventional matrix.
    Else
        Partition a into four sub matrices a11, a12, a21, a22.
        Partition b into four sub matrices b11, b12, b21, b22.
        Strassen (n/2, a11 + a22, b11 + b22, d1)
        Strassen (n/2, a21 + a22, b11, d2)
        Strassen (n/2, a11, b12 – b22, d3)
        Strassen (n/2, a22, b21 – b11, d4)
        Strassen (n/2, a11 + a12, b22, d5)
        Strassen (n/2, a21 – a11, b11 + b22, d6)
        Strassen (n/2, a12 – a22, b21 – b22, d7)

        C = d1+d4+d5+d7+d3+d5+d2+d4+d1+d3+d2+d6
    end if
    return (C)
end
```

Program for Strassen's Matrix Multiplication:

Strassen.c

```c
#include<stdio.h>
int main(){
    int a[2][2], b[2][2], c[2][2], i, j;
    int m1, m2, m3, m4 , m5, m6, m7;

    printf("Enter the 4 elements of first matrix: ");
    for(i = 0;i < 2; i++)
        for(j = 0;j < 2; j++)
            scanf("%d", &a[i][j]);

    printf("Enter the 4 elements of second matrix: ");
    for(i = 0; i < 2; i++)
        for(j = 0;j < 2; j++)
            scanf("%d", &b[i][j]);

    printf("\nThe first matrix is\n");
    for(i = 0; i < 2; i++){
        printf("\n");
        for(j = 0; j < 2; j++)
            printf("%d\t", a[i][j]);
    }

    printf("\nThe second matrix is\n");
    for(i = 0;i < 2; i++){
        printf("\n");
        for(j = 0;j < 2; j++)
```

Strassen.c

```c
        for(j = 0;j < 2; j++)
            printf("%d\t", b[i][j]);
    }

    m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
    m2= (a[1][0] + a[1][1]) * b[0][0];
    m3= a[0][0] * (b[0][1] - b[1][1]);
    m4= a[1][1] * (b[1][0] - b[0][0]);
    m5= (a[0][0] + a[0][1]) * b[1][1];
    m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
    m7= (a[0][1] - a[1][1]) * (b[1][0]+b[1][1]);

    c[0][0] = m1 + m4- m5 + m7;
    c[0][1] = m3 + m5;
    c[1][0] = m2 + m4;
    c[1][1] = m1 - m2 + m3 + m6;

    printf("\nAfter multiplication using Strassen's algorithm \n");
    for(i = 0; i < 2 ; i++){
        printf("\n");
        for(j = 0;j < 2; j++)
            printf("%d\t", c[i][j]);
    }

    return 0;
}
```

The Output is:

```
Enter the 4 elements of first matrix: 1
2
3
4
Enter the 4 elements of second matrix: 5
6
7
8

The first matrix is

1       2
3       4
The second matrix is

5       6
7       8
After multiplication using Strassen's algorithm

19      22
43      50
PS C:\Basic Programs Practice>
```

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-05
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	**Greedy Strategy**				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
29/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
Average (out of 10)		

Title: Greedy Strategy.

Objective: To learn Huffman coding.

Theory: Huffman coding is lossless data compression algorithm. In this algorithm a variable-length code is assigned to input different characters. The code length is related with how frequently characters are used. Most frequent characters have smallest codes, and longer codes for least frequent characters.

There are mainly two parts. First one to create Huffman tree, and another one to traverse the tree to find codes.

For an example, consider some strings “YYYZXXYYX”, the frequency of character Y is larger than X and the character Z has least frequency. So the length of code for Y is smaller than X, and code for X will be smaller than Z.

- Complexity for assigning code for each character according to their frequency is O(n log n)

Input − A string with different characters, say “ACCEBFFFFAAXXBLKE”
Output − Code for different characters:

Data: K, Frequency: 1, Code:0000

Data: L, Frequency: 1, Code:0001

Data: E, Frequency: 2, Code: 001

Data: F, Frequency: 4, Code: 01

Data: B, Frequency: 2, Code:100

Data: C, Frequency: 2, Code:101

Data: X, Frequency: 2, Code:110

Data: A, Frequency: 3, Code:111

Algorithm:

HuffmanCoding(string)

Input − A string with different characters.

Output − The codes for each individual characters.

Begin

Define a node with character, frequency, left and right child of the node for Huffman tree.

Create a list ‘freq’ to store frequency of each character, initially all are 0

for each character c in the string do

increase the frequency for character ch in freq list.

Done

For all type of character ch do

If the frequency of ch is non zero then add ch and its frequency as a node of priority queue Q

Done

While Q is not empty do

Remove item from Q and assign it to left child of node

Remove item from Q and assign to the right child of node

Traverse the node to find the assigned code

Done

End

Program for Huffman Coding:

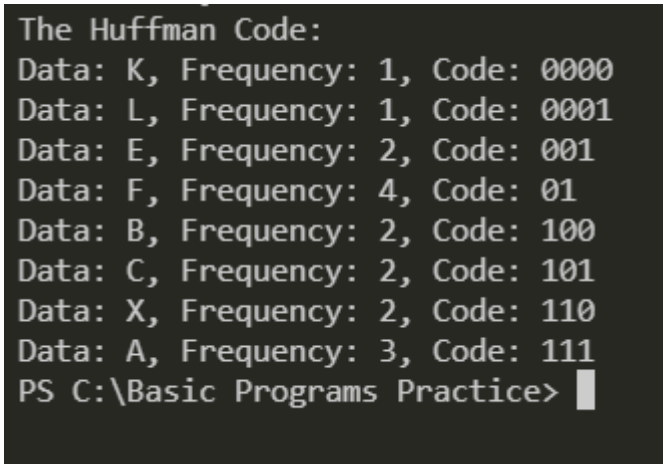
Huffman.cpp

```cpp
#include<iostream>
#include<queue>
#include<string>
using namespace std;
struct node{
    int freq;
    char data;
    const node *child0, *child1;
    node(char d, int f = -1){
        data = d;
        freq = f;
        child0 = NULL;
        child1 = NULL;
    }
    node(const node *c0, const node *c1){
        data = 0;
        freq = c0->freq + c1->freq;
        child0=c0;
        child1=c1;
    }
    bool operator<( const node &a ) const {
        return freq >a.freq;
    }
    void traverse(string code = "")const{
        if(child0!=NULL){
            child0->traverse(code+'0');
```

```
    void traverse(string code = "")const{
        if(child0!=NULL){
            child0->traverse(code+'0');
            child1->traverse(code+'1');
        }else{
            cout << "Data: " << data<< ", Frequency: "<<freq << ", Code: " << code<<endl;
        }
    }
};
void huffmanCoding(string str){
    priority_queue<node> qu;
    int frequency[256];
    for(int i = 0; i<256; i++)
        frequency[i] = 0;
    for(int i = 0; i<str.size(); i++){
        frequency[int(str[i])]++;
    }
    for(int i = 0; i<256; i++){
        if(frequency[i]){
            qu.push(node(i, frequency[i]));
        }
    }
    while(qu.size() >1){
        node *c0 = new node(qu.top());
        qu.pop();
        node *c1 = new node(qu.top());
```

```
46      while(qu.size() >1){
47          node *c0 = new node(qu.top());
48          qu.pop();
49          node *c1 = new node(qu.top());
50          qu.pop();
51          qu.push(node(c0, c1));
52      }
53      cout << "The Huffman Code: "<<endl;
54      qu.top().traverse();
55  }
56  main(){
57      string str = "ACCEBFFFFAAXXBLKE";
58      huffmanCoding(str);
59  }
```

The Output is:



Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-06
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	**Minimum Spanning Tree**				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
29/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
Average (out of 10)		

Title: Minimum Spanning tree.

Objective: To understand the Kruskal's Algorithm

Theory: Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

Algorithm

- **Step 1:** Create a forest in such a way that each graph is a separate tree.
- **Step 2:** Create a priority queue Q that contains all the edges of the graph.
- **Step 3:** Repeat Steps 4 and 5 while Q is NOT EMPTY
- **Step 4:** Remove an edge from Q
- **Step 5:** IF the edge obtained in Step 4 connects two different trees, then Add it to the forest (for combining two trees into one tree).
  ELSE
  Discard the edge
- **Step 6:** END

Program for Kruskal's algorithm:

Kruskal's.cpp

```cpp
#include<iostream>
#include<string.h>
using namespace std;
class Graph
{
  char vertices[10][10];
  int cost[10][10],no;
public:
  Graph();
  void creat_graph();
  void display();
  int Position(char[]);
  void kruskal_algo();
};
/* Initialzing adj matrix with 999 */
/* 999 denotes infinite distance */
Graph::Graph()
{
  no=0;
  for(int i=0;i<10;i++)
  for(int j=0;j<10;j++)
  {
     cost[i][j]=999;
   }
}
```

Kruskal's.cpp

```cpp
/* Taking inputs for creating graph */
void Graph::creat_graph()
{
  char ans,Start[10],End[10];
  int wt,i,j;
  cout<<"Enter the number of vertices: ";
  cin>>no;
  cout<<"\nEnter the vertices: ";
  for(i=0;i<no;i++)
        cin>>vertices[i];
  do
  {
    cout<<"\nEnter Start and End vertex of the edge: ";
    cin>>Start>>End;
    cout<<"Enter weight: ";
    cin>>wt;
    i=Position(Start);
    j=Position(End);
    cost[i][j]=cost[j][i]=wt;
    cout<<"\nDo you want to add more edges (Y=YES/N=NO)? : ";
    cin>>ans;
  }while(ans=='y' || ans=='Y');
}
/* Displaying Cost matrix */
void Graph::display()
```

Kruskal's.cpp

```cpp
void Graph::display()
{
  int i,j;
  cout<<"\n\nCost matrix: ";
  for(i=0;i<no;i++)
  {
     cout<<"\n";
     for(j=0;j<no;j++)
     cout<<"\t"<<cost[i][j];
  }
}
/* Retrieving position of vertices in 'vertices' array */
int Graph::Position(char key[10])
{
  int i;
  for(i=0;i<10;i++)
  if(strcmp(vertices[i],key)==0)
    return i;
return -1;
}
void Graph::kruskal_algo()
{
  int i,j,v[10]={0},x,y,Total_cost=0,min,gr=1,flag=0,temp,d;
  while(flag==0)
  {
    min=999;
```

Kruskal's.cpp

```
74      {
75        min=999;
76         for(i=0;i<no;i++)
77        {
78          for(j=0;j<no;j++)
79             {
80            if(cost[i][j]<min)
81                {
82             min=cost[i][j];
83             x=i;
84             y=j;
85                }
86             }
87         }
88
89        if(v[x]==0 && v[y]==0)
90        {
91          v[x]=v[y]=gr;
92          gr++;
93        }
94        else if(v[x]!=0 && v[y]==0)
95          v[y]=v[x];
96        else if(v[x]==0 && v[y]!=0)
97          v[x]=v[y];
98        else
99        {
```

Kruskal's.cpp

```
99      {
100       if(v[x]!=v[y])
101       {
102         d=v[x];
103         for(i=0;i<no;i++)
104         {
105           if(v[i]==d)
106           v[i]=v[y];
107         }//end for
108       }
109     }
110
111     cost[x][y]=cost[y][x]=999;
112     Total_cost=Total_cost+min;      /* calculating cost of minimum spanning tree */
113     cout<<"\n\t"<<vertices[x]<<"\t\t"<<vertices[y]<<"\t\t"<<min;
114
115       temp=v[0]; flag=1;
116       for(i=0;i<no;i++)
117       {
118         if(temp!=v[i])
119         {
120         flag=0;
121         break;
122         }
123       }
124   }
```

Kruskal's.cpp

Kruskal's.cpp

```
125   cout<<"\nTotal cost of the tree= "<<Total_cost;
126   }
127   int main()
128   {
129     Graph g;
130     g.creat_graph();
131     g.display();
132     cout<<"\n\n\nMinimum Spanning tree using kruskal algo=>";
133     cout<<"\nSource vertex\tDestination vertex\tWeight\n";
134     g.kruskal_algo();
135   return 0;
136   }
```

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-07
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	**Minimum Spanning tree**				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
30/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
Average (out of 10)		

Title: Minimum Spanning Tree.

Objective: To Understand Prim's Algorithm.

Theory: Prim's Algorithm is an approach to determine minimum cost spanning tree. In this case, we start with single edge of graph and we add edges to it and finally we get minimum cost tree. In this case, as well, we have n-1 edges when number of nodes in graph are n.We again and again add edges to tree and tree is extended to create spanning tree, while in case of Kruskal's algorithm there may be more than one tree, which is finally connected through edge to create spanning tree.

Algorithm for Prim's Algorithm:

This algorithm creates spanning tree with minimum weight from a given weighted graph.

1. Begin
2. Create edge list of given graph, with their weights.
3. Draw all nodes to create skeleton for spanning tree.
4. Select an edge with lowest weight and add it to skeleton and delete edge from edge list.
5. Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.
6. Repeat step 5 until n-1 edges are added.
7. Return.

Program for Prim's Algorithm :

Prim's.c

```c
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int minKey(int key[], bool mstSet[])
{
    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;
}
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[i][parent[i]]);
}
void primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    bool mstSet[V];
```

Prim's.c

```c
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}


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 } };


    primMST(graph);

    return 0;
}
```

The Output is:

```
Edge    Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5


...Program finished with exit cod
Press ENTER to exit console.
```

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-08
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	**Greedy Strategy**				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
30/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
Average (out of 10)		

Title: Greedy Strategy**.**

Objective: Single source shortest path algorithm.

Theory: Dijkstra's Algorithm can find the shortest path between nodes in a graph. Particularly, you can find the shortest path from a node (called the "source node") to all other nodes in the graph, producing a shortest-path tree.

Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Basics of Dijkstra's Algorithm

- Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
- The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
- Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.
- The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

Algorithm for Single source shortest path:

1. Initialization of all nodes with distance "infinite"; initialization of the starting node with 0
2. Marking of the distance of the starting node as permanent, all other distances as temporarily.
3. Setting of starting node as active.
4. Calculation of the temporary distances of all neighbour nodes of the active node by summing up its distance with the weights of the edges.
5. If such a calculated distance of a node is smaller as the current one, update the distance and set the current node as antecessor. This step is also called update and is Dijkstra's central idea.
6. Setting of the node with the minimal temporary distance as active. Mark its distance as permanent.
7. Repeating of steps 4 to 7 until there aren't any nodes left with a permanent distance, which neighbors still have temporary distances.

Program:

SingleSourseShortestPath.cpp

```cpp
#include <limits.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

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

    return min_index;
}
int printSolution(int dist[], int n)
{
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d tt %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src)
{
    int dist[V];

    bool sptSet[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
    for (int count = 0; count < V - 1; count++) {
```

SingleSourseShortestPath.cpp

```cpp
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;
        for (int v = 0; v < V; 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];
    }
    printSolution(dist, V);
}
int main()
{
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

The Output is:

```
Vertex Distance from Source
0 tt 0
1 tt 4
2 tt 12
3 tt 19
4 tt 21
5 tt 11
6 tt 9
7 tt 8
8 tt 14
PS C:\Basic Programs Practice>
```

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-09
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	**Dynamic Programming**				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
30/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
Average (out of 10)		

Title: Dynamic Programming.

Objective: To understand Floyd Warshal Algorithm.

Theory: : Floyd–Warshall algorithm (also known as Floyd's algorithm, the Roy–Warshall algorithm, the Roy–Floyd algorithm, or the WFI algorithm) is an algorithm for finding shortest paths in a directed weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of shortest paths between all pairs of vertices. Although it does not return details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm.

Algorithm

1. If k=0, then

dij (0) ={ 0 if i=j ∞ if i≠j

2. Otherwise, for k≥1, dij (k) can be computed from dij (k-1) and the adjacency matrix w.

dij (k) =min{ dij (k-1) , min1≤l≤n { dil (k-1) + wlj }}= min1≤l≤n { dil (k-1) + wlj }

Program:

FloydWarshal.cpp

```cpp
#include <iostream>
using namespace std;

#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

// Implementing floyd warshall algorithm
void floydWarshall(int graph[][nV]) {
  int matrix[nV][nV], i, j, k;

  for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
      matrix[i][j] = graph[i][j];

  // Adding vertices individually
  for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
      for (j = 0; j < nV; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
  printMatrix(matrix);
}
```

```
29
30  void printMatrix(int matrix[][nV]) {
31    for (int i = 0; i < nV; i++) {
32      for (int j = 0; j < nV; j++) {
33        if (matrix[i][j] == INF)
34          printf("%4s", "INF");
35        else
36          printf("%4d", matrix[i][j]);
37      }
38      printf("\n");
39    }
40  }
41
42  int main() {
43    int graph[nV][nV] = {{0, 3, INF, 5},
44                         {2, 0, INF, 4},
45                         {INF, 1, 0, INF},
46                         {INF, INF, 2, 0}};
47    floydWarshall(graph);
48  }
```

The Output is:

```
  0   3   7   5
  2   0   6   4
  3   1   0   5
  5   3   2   0
S C:\Basic Programs Practice>
```

Shivaji Rao Kadam Institute of Technology and Management
Technical Campus, Indore
Computer Science & Engineering Department

Experiment-10
Laboratory Performance Evaluation

Academic Session:	Jan-June 2021	Semester	4th	Batch	B1 & B2
Name of Lab:	Analysis Design of Algorithm	Course Code		CS-402	
Name of Experiment	**Dynamic Programming**				**CO No.**
Name of Student	**Kartik Kulshreshtha**	Enrolment No		**0875CS191048**	

Date of Experiment	**Date of Submission**
30/05/2021	31/05/2021

Grade and remark by the tutor	**Score (0- 10)**	**Remark / Reason**
1. Clarity about the Objective and Outcome of experiment		
2. Submitted the work in desired format		
Average (out of 10)		

Title: Dynamic Programming.

Objective: To understand Travelling salesman problem.

Theory: Traveling salesman problem, an optimization problem in graph theory in which the nodes (cities) of a graph are connected by directed edges (routes), where the weight of an edge indicates the distance between two cities. The problem is to find a path that visits each city once, returns to the starting city, and minimizes the distance traveled. The only known general solution algorithm that guarantees the shortest path requires a solution time that grows exponentially with the problem size (i.e., the number of cities). This is an example of an NP-complete problem (from nonpolynomial), for which no known efficient (i.e., polynomial time) algorithm exists.

Algorithm

1. Start on an arbitrary vertex as current vertex.
2. Find out the shortest edge connecting current vertex and an unvisited vertex V.
3. Set current vertex to V.
4. Mark V as visited.
5. If all the vertices in domain are visited, then terminate.
6. Go to step 2.
7. The sequence of the visited vertices is the output of the algorithm.

Program:

TravellingSalesman.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define vr 4
int TSP(int grph[][vr], int p){
    vector<int> ver; //
    for (int i = 0; i < vr; i++)
        if (i != p)
            ver.push_back(i);
            int m_p = INT_MAX;
    do {
        int cur_pth = 0;
        int k = p;
        for (int i = 0; i < ver.size(); i++) {
            cur_pth += grph[k][ver[i]];
            k = ver[i];
        }
        cur_pth += grph[k][p];
        m_p = min(m_p, cur_pth);
    }
    while (next_permutation(ver.begin(), ver.end()));
    return m_p;
}
```

```
23  int main() {
24      int grph[][vr] = { { 0, 5, 10, 15 },
25          { 5, 0, 20, 30 },
26          { 10, 20, 0, 35 },
27          { 15, 30, 35, 0 }
28      };
29      int p = 0;
30      cout<< "\n The result is: "<< TSP(grph, p) << endl;
31      return 0;
32  }
```

The Output is:

```
 The result is: 75
PS C:\Basic Programs Practice>
```