



Vidya Vikas Education Trust's

Universal College of Engineering, Kaman Road, Vasai-401212

Accredited by B+ Grade by NAAC

## Department of Computer Engineering

### ANALYSIS OF ALGORITHMS (CSL 401)

#### Lab Manual

**S.E.(Comp) / Sem IV/ SE-A/B**

Name of the student:

**VIVEK. SHIVAKUMAR. HOTTI**

Roll No:

**31**

**2020-21**

# Department of Computer Engineering

**Vision:**

To be recognized globally as a department provides quality technical education that eventually caters to helping and serving the community

**Mission:**

To develop human resources with sound knowledge in theory and practice of computer science and engineering

To motivate the students to solve real-world problems to help the society grow

To provide a learning ambience to enhance innovations, team spirit and leadership qualities for students

Course Name	Lab Name	Credit
CSL401	Analysis of Algorithms Lab	1

<b>Prerequisite:</b> Basic knowledge of programming and data structure	
<b>Lab Objectives:</b>	
1	To introduce the methods of designing and analyzing algorithms
2	Design and implement efficient algorithms for a specified application
3	Strengthen the ability to identify and apply the suitable algorithm for the given real-world problem.
4	Analyze worst-case running time of algorithms and understand fundamental algorithmic problems.
<b>Lab Outcomes:</b> At the end of the course, the students will be able to	
1	Implement the algorithms using different approaches.
2	Analyze the complexities of various algorithms.
3	Compare the complexity of the algorithms for specific problem.

Lab Plan		
Name of the Course		Analysis of Algorithms (AoA)
Year/Sem/Class		S.E.(Comp) / Sem IV/ A/B
Sr. No.	Module	List of Practical Experiments
1	Introduction to Analysis of Algorithm	Comparative analysis on the basis of Algorithm required to sort list is expected for large values of n using Selection Sort and Insertion Sort
2	Divide and Conquer	a. Merge Sort b. Quick Sort
3	Greedy Method	Minimum Cost Spanning Tree a. Kruskal Algorithm b. Prim’s Algorithm
4	Dynamic Programming	a. 0/1 Knapsack b. Longest common subsequence algorithm
5	Backtracking and Branch-and-bound	a. 8 queen problem( N-queen problem) b. Graph Coloring
6	String Matching Algorithm	a. Naive String Matching b. Rabin Karp
7	Other than the syllabus	a. Binary Search Tree

# INDEX

**Name:** Vivek S Hotti    **Roll:** 31    **Class:** SE-Sem4    **Div.:** A

Sr. No.	Module	List of Practical Experiments	From PG	To PG
1	Introduction to Analysis of Algorithm	Comparative analysis on the basis of Algorithm required to sort list is expected for large values of n using Selection Sort and Insertion Sort	5	9
2	Divide and Conquer	a) Merge Sort b) Quick Sort	10	15
3	Greedy Method	Minimum Cost Spanning Tree a) Kruskal Algorithm b) Prim's Algorithm	16	22
4	Dynamic Programming	a) 0/1 Knapsack b) Longest common subsequence algorithm	23	29
5	Backtracking and Branch-and-bound	a) 8 queen problem(N-queen problem) b) Graph Coloring	30	35
6	String Matching Algorithm	a) Naive String Matching b) Rabin Karp	36	41
7	Other than the syllabus	a) Binary Search Tree	42	46

## ➡ EXPERIMENT NO. 1 ⬅

**AIM:**

## Program to implement Selection sort & Insertion sort.

## THEORY:

## - Selection Sort

A selection sort is one in which successive elements are selected in order and placed in their proper sorted positions. The elements of the input array may have to be preprocessed to make the ordered selection possible. This algorithm is called the general selection sort.

Straight Selection Sort or push down sort implements the descending priority queue as an unordered array. Therefore, the straight selection sort consists entirely of a selection phase in which the largest of the remaining elements large is repeatedly placed in its proper position i.e. the end of the array. To do so, large is interchanged with the element  $x[i]$ .

**Algorithm:**

1. Establish the array  $a[0 \dots n-1]$  of  $n$  elements.
2. Set variables  $large$  to  $x[0]$  and  $index$  to 0.
3. Compare  $large$  with remaining elements of the array  $x$  if any  $x[i]$  is found to be greater the  
 $large$   
then 3.1 change  $large$  to  $x[i]$   
3.2 change  $index$  to  $i$
4. Swap the values of  $x[0]$  with  $x[index]$
5. Repeat steps 1 to 4 for different values of  $large$  ranging from  $x[i+1]$  to  $x[n-1]$
6. Return the sorted array.

### Analysis of Selection Sort:

Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array. Selecting the lowest element requires scanning all

elements (this takes  $n-1$  comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining  $n-1$  elements and so on, for

$(n-1) + (n-2) + \dots + 2 + 1 = n(n-1) / 2 \in \Theta(n^2)$  comparisons. Each of these scans requires one swap for  $n-1$  elements (the final element is already in place).

## - Insertion Sort:

An insertion sort is one that sorts a set of records by inserting records into an existing sorted file. The simplest way to insert next element into the sorted part is to sift it down, until it occupies correct position. Initially the element stays right after the sorted part. At each step algorithm compares the element with one before it and, if they stay in reversed order, swap them

Algorithm:

```
function insertionSort(array A)
  for i from 1 to length[A]-1 do
    y = A[i]
    j = i-1
    while j >= 0 and A[j] > y do
      A[j+1] = A[j]
    j = j-1
  done
  A[j+1] = y
Done
```

## Analysis of Insertion Sort:

The best case input is an array that is already sorted. In this case insertion sort has a linear running time (i.e.,  $\Theta(n)$ ). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the array.

The simplest worst case input is an array sorted in reverse order. The set of all worst case inputs consists of all arrays where each element is the smallest or second-smallest of the elements before it. In these cases every iteration of the inner loop will scan and shift the entire sorted

subsection of the array before inserting the next element. This gives insertion sort a quadratic running time (i.e.,  $O(n^2)$ ).

The average case is also quadratic, which makes insertion sort impractical for sorting large arrays.

## **CODE for Insertion Sort: -**

```
#include <stdio.h>
#include <math.h>
void insertionSort(int arr[], int n)
{
    int i, key, j, temp;
    for (i = 1; i < n; i++)
    {
        j = i;
        key = arr[j];
        while (j > 0 && arr[j-1] > key)
        {
            temp=arr[j];
            arr[j]=arr[j-1];
            arr[j-1]=temp;
            j = j-1;
        }
    }
}
void main()
{
    int arr[100],i,n;
    printf("ENTER THE NO. OF ELEMENTS: \n");
    scanf("%d", &n);
    printf("ENTER THE ELEMENTS: \n");
    for (i=0; i < n; i++)
    {
        scanf("%d",&arr[i]);
    }
    insertionSort(arr, n);
    printf("THE SORTED ARRAY IS: \n");
    for (i=0; i < n; i++)
    {
        printf("%d \n", arr[i]);
    }
}
```

### *Output obtained from the above code for Insertion Sort:*

```
C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
ENTER THE NO. OF ELEMENTS:
5
ENTER THE ELEMENTS:
24
233
10
492
9
THE SORTED ARRAY IS:
9
10
24
233
492

C:\Users\Vivek hotti\Desktop>
```

### **CODE for Selection Sort: -**

```
#include <stdio.h>
void main()
{
    int array[100], n, i, j, position, swap;
    printf("Enter number of elements\n");
    scanf("%d", &n);
    printf("Enter %d integers\n", n);
    for ( i = 0 ; i < n ; i++ )
        scanf("%d", &array[i]);
    for ( i = 0 ; i < ( n - 1 ) ; i++ )
    {
        position = i;
        for ( j = i + 1 ; j < n ; j++ )
        {
            if ( array[position] > array[j] )
                position = j;
        }
        if ( position != i )
        {
            swap = array[i];
            array[i] = array[position];
            array[position] = swap;
        }
    }
    printf("Sorted list in ascending order:\n");
    for ( i = 0 ; i < n ; i++ )
        printf("%d\n", array[i]);
}
```



### *Output obtained from the above code for Selection Sort:*

```
C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
Enter number of elements
5
Enter 5 integers
1492
231
587
10
484
Sorted list in ascending order:
10
231
484
587
1492

C:\Users\Vivek hotti\Desktop>
```

### **CONCLUSION:**

Insertion sort is very similar in that after the  $k$ th iteration, the first  $k$  elements in the array are in sorted order. Insertion sort's advantage is that it only scans as many elements as it needs in order to place the  $k+1$ st element, while selection sort must scan all remaining elements to find the  $k+1$ st element.

Insertion sort is one of the fastest algorithms for sorting very small arrays. Insertion sort typically makes fewer comparisons than selection sort.

## ➡ EXPERIMENT NO. 2 ⬅

### AIM:

Program to implement Merge sort analysis and Quick Sort Analysis.

### THEORY:

#### - Merge Sort:

Merge sort is based on the divide-and-conquer paradigm.

It consists of three steps as given below:

1. Divide Step
2. Conquer Step
3. Combine Step

#### Algorithm: Merge Sort

To sort entire sequence  $A[1 \dots n]$ , make the initial call to the procedure MERGE-SORT ( $A, 1, n$ ).

#### Analyzing Merge Sort

For simplicity, assume that  $n$  is a power of 2 so that each divide step yields two subproblems, both of size exactly  $n/2$ . The base case occurs when  $n = 1$ .

Time complexity of Merge sort is  $O(n \log n)$

#### - Quick Sort

Quicksort also known as "partition-exchange sort" is a comparison sort. Let  $x$  be an array and  $n$  the number of elements in the array to be sorted. Choose an element  $a$  from a specific position within the array. (e.g.  $a$  can be chosen as the first element, so that  $a = x[0]$ ). Suppose the elements of  $x$  are partitioned so that  $a$  is placed in position  $j$  and the following conditions hold:

1. Each of the elements in positions 0 through  $j - 1$  is less than or equal to  $a$ .
2. Each of the elements in positions  $j + 1$  through  $n - 1$  is greater than or equal to  $a$ .

If these two conditions hold for a particular  $a$  and  $j$ th smallest element of  $x$ , so that  $a$  remains in position  $j$  when the array is completely sorted. If the above process is repeated with the sub arrays  $x[0]$  through  $x[j-1]$  and  $x[j+1]$  through  $x[n-1]$  and any sub arrays created by the process in successive iterations, the final result is a sorted file.

### **Algorithm: Quick Sort**

Step 1 – Make the right-most index value pivot  
 Step 2 – partition the array using pivot value  
 Step 3 – quicksort left partition recursively  
 Step 4 – quicksort right partition recursively

### **Complexity of Quicksort**

#### **Worst-case: $O(N^2)$**

This happens when the pivot is the smallest (or the largest) element.

Then one of the partitions is empty, and we repeat recursively the procedure for  $N-1$  elements.

#### **Best-case $O(N \log N)$**

The best case is when the pivot is the median of the array, and then the left and the right part will have same size.

There are  $\log N$  partitions, and to obtain each partitions we do  $N$  comparisons (and not more than  $N/2$  swaps). Hence the complexity is  $O(N \log N)$

#### **Average-case- $O(N \log N)$**

When  $n \geq 2$ , time for merge sort steps:

- **Divide:** Just compute  $q$  as the average of  $p$  and  $r$ , which takes constant time i.e.  $\Theta(1)$ .
- **Conquer:** Recursively solve 2 subproblems, each of size  $n/2$ , which is  $2T(n/2)$ .
- **Combine:** MERGE on an  $n$ -element subarray takes  $\Theta(n)$  time.

Summed together they give a function that is linear in  $n$ , which is  $\Theta(n)$ . Therefore, the recurrence for merge sort running time is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

So solving these using Master's theorem best and average time complexity is  **$O(n \log n)$**

## CODE for Merge Sort: -

```
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[100], R[100];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
```

```

        int m = l+(r-l)/2;

        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}

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

void main()
{
    int arr[100],i,a,n;
    printf("ENTER THE NO. OF ELEMENTS: \n");
    scanf("%d",&n);
    printf("ENTER THE ELEMENTS: \n");
    for(i=0;i<n;i++)
        scanf("%d", &arr[i]);
    printf("Given array is \n");
    printArray(arr,n);
    mergeSort(arr, 0, n - 1);
    printf("\nSorted array is \n");
    printArray(arr, n);
}

```

***Output obtained from the above code for Merge Sort:***

```

C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
ENTER THE NO. OF ELEMENTS:
5
ENTER THE ELEMENTS:
24
332
12
3
765
Given array is
24 332 12 3 765

Sorted array is
3 12 24 332 765

```

## CODE for Quick Sort: -

```
#include <conio.h>
#include <stdio.h>
void quick_sort(int[],int,int);
int partition(int[],int,int);
void main()
{
    int a[50],n,i;
    printf("How many elements?\n");
    scanf("%d",&n);
    printf("Enter array elements:\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    quick_sort(a,0,n-1);
    printf("Array after sorting using Quick Sort is:\n");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
    getch();
}

void quick_sort(int a[],int l,int u)
{
    int j;
    if(l<u)
    {
        j=partition(a,l,u);
        quick_sort(a,l,j-1);
        quick_sort(a,j+1,u);
    }
}

int partition(int a[],int l,int u)
{
    int v,i,j,temp;
    v=a[l];
    i=l;
    j=u+1;
    do
    {
        do
            i++;
        while(a[i]<v&i<=u);
        do
            j--;
        while(v<a[j]);
        if(i<j){
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
    }while(i<j);
    a[l]=a[j];
    a[j]=v;
    return(j);
}
```

### *Output obtained from the above code for Quick Sort:*

```
C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
How many elements?
5
Enter array elements:
147
23
156
1560
01
Array after sorting using Quick Sort is:
1 23 147 156 1560
```

## CONCLUSION:

- One of the fastest algorithms on average.
- Does not need additional memory (the sorting takes place in the array - this is called **in-place** processing). Compare with merge sort: merge sort needs additional memory for merging. In sorting  $n$  objects, merge sort has an average and worst-case performance of  $O(n \log n)$ .

## ➡ EXPERIMENT NO. 3 ⬅

### AIM:

- a) To implement 0/1 Knapsack.

### THEORY:

**Problem Statement:** A thief robbing a store knapsack. There are  $n$  items and  $i^{th}$  item weigh take? and can carry a maximal weight of  $W$  into their  $w_i$  and is worth  $v_i$  dollars. What items should thief 0-1 knapsack problem

- Exhibit No greedy choice property.
- No greedy algorithm exists.
- Exhibit optimal substructure property.
- Only dynamic programming algorithm exists.

Given weights and values of  $n$  items, put these items in a knapsack of capacity  $W$  to get the maximum total value in the knapsack. In other words, given two integer arrays  $val[0..n-1]$  and  $wt[0..n-1]$  which represent values and weights associated with  $n$  items respectively. Also given an integer  $W$  which represents knapsack capacity, find out the maximum value subset of  $val[]$  such that sum of the weights of this subset is smaller than or equal to  $W$ . You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

### Algorithm:

```
for w = 0 to W
  B[0,w] = 0
for i = 1 to n
  B[i,0] = 0
for i = 1 to n
  for w = 0 to W
    if  $w_i \leq w$  // item i can be part of the solution
      if  $b_i + B[i-1, w-w_i] > B[i-1, w]$ 
         $B[i, w] = b_i + B[i-1, w-w_i]$ 
      else  $B[i, w] = B[i-1, w]$ 
    else  $B[i, w] = B[i-1, w]$  //  $w_i > w$ 
```



**Example:** Selection of n=4 items, capacity of knapsack M=8

Item i	Value $v_i$	Weight $w_i$
1	15	1
2	10	5
3	9	3
4	5	4

$$f(0,g) = 0, f(k,0) = 0$$

**Recursion formula:**

$$f(k,g) = \begin{cases} f(k-1,g) & \text{if } w_k > g \\ \max \{v_k + f(k-1,g-w_k), f(k-1,g)\} & \text{if } w_k \leq g \text{ and } k > 0 \end{cases}$$

Solution tabulated:

		Capacity remaining								
		g=0	g=1	g=2	g=3	g=4	g=5	g=6	g=7	g=8
k=0	$f(0,g) =$	0	0	0	0	0	0	0	0	0
k=1	$f(1,g) =$	0	15	15	15	15	15	15	15	15
k=2	$f(2,g) =$	0	15	15	15	15	15	25	25	25
k=3	$f(3,g) =$	0	15	15	15	24	24	25	25	25
k=4	$f(4,g) =$	0	15	15	15	24	24	25	25	<u>29</u>

Last value:  $k=n, g=M, f = f(n,M) = f(4,8) = 29$

## CODE for Knapsack Problem: -

```
#include<stdio.h>

int max(int a, int b) { return (a > b)? a : b; }

int knapSack(int W, int wt[], int val[], int n)
{
    int i, w;
    int K[n+1][W+1];

    for (i = 0; i <= n; i++)
    {
        for (w = 0; w <= W; w++)
        {
            if (i==0 || w==0)
                K[i][w] = 0;
            else if (wt[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
            else
                K[i][w] = K[i-1][w];
        }
    }
    return K[n][W];
}
```

```

        K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
    else
        K[i][w] = K[i-1][w];
    }
}

return K[n][W];
}

int main()
{
    int i, n, val[20], wt[20], W;

    printf("Enter number of items: \n");
    scanf("%d", &n);

    printf("Enter value and weight of items:\n");
    for(i = 0; i < n; ++i){
        scanf("%d%d", &val[i], &wt[i]);
    }

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

    printf("Answer is: %d", knapSack(W, wt, val, n));
    return 0;
}

```

***Output obtained from the above code for Knapsack Problem:***

```

C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
Enter number of items:
3
Enter value and weight of items:
100 20
50 10
150 30
Enter size of knapsack:
50
Answer is: 250
C:\Users\Vivek hotti\Desktop>

```

## **CONCLUSION:**

Time complexity  $O(nW)$  where  $n$  is the number of items and  $W$  is the capacity of knapsack.

## AIM:

- b) To implement LCS problem using dynamic programming approach.

## THEORY:

**Definition 1:** Given a sequence  $X=x_1x_2\dots x_m$ , another sequence  $Z=z_1z_2\dots z_k$  is a subsequence of  $X$ , if there exists a strictly increasing sequence  $i_1i_2\dots i_k$  of indices of  $X$  such that for all  $j=1,2,\dots,k$ , we have  $x_{i_j}=z_j$ .

**Example 1:**

If  $X=abcdefg$ ,  $Z=abdg$  is a subsequence of  $X$ .

$X=abcdefg$ ,

$Z=ab\ d\ g$

**Definition 2:**

Given two sequences  $X$  and  $Y$ . A sequence  $Z$  is a common subsequence of  $X$  and  $Y$  if  $Z$  is a subsequence of both  $X$  and  $Y$ .

**Example 2:**  $X=abcdefg$  and  $Y=aaadgfd$ .  $Z=adf$  is a common subsequence of  $X$  and  $Y$ .

$X=abc\ defg$

$Y=aaaadgfd$

$Z=a\ d\ f$

**Definition 3:**

A longest common subsequence of  $X$  and  $Y$  is a common subsequence of  $X$  and  $Y$  with the longest length. (The length of a sequence is the number of letters in the sequence.)

Longest common subsequence may not be unique.

### **Theorem (Optimal substructure of an LCS)**

Let  $X=x_1x_2\dots x_m$ , and  $Y=y_1y_2\dots y_n$  be two sequences, and  $Z=z_1z_2\dots z_k$  be any LCS of  $X$  and  $Y$ .

1. If  $x_m=y_n$ , then  $z_k=x_m=y_n$  and  $Z[1..k-1]$  is an LCS of  $X[1..m-1]$  and  $Y[1..n-1]$ .
2. If  $x_m\neq y_n$ , then  $z_k\neq x_m$  implies that  $Z$  is an LCS of  $X[1..m-1]$  and  $Y$ .
3. If  $x_m\neq y_n$ , then  $z_k\neq y_n$  implies that  $Z$  is an LCS of  $X$  and  $Y[1..n-1]$ .

### **The recursive equation**

Let  $c[i,j]$  be the length of an LCS of  $X[1\dots i]$  and  $X[1\dots j]$ .

$c[i,j]$  can be computed as follows:

$$\begin{array}{ll} 0 & \text{if } i=0 \text{ or } j=0, \\ c[i,j] = c[i-1,j-1] + 1 & \text{if } i,j > 0 \text{ and } x_i = y_j, \\ \max\{c[i,j-1], c[i-1,j]\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j. \end{array}$$

### Computing the length of an LCS

- There are  $n \times m$   $c[i,j]$ 's. So we can compute them in a specific order.

### The algorithm to compute an LCS

```
1. for i=1 to m do
2.   c[i,0]=0;
3. for j=0 to n do
4.   c[0,j]=0;
5. for i=1 to m do
6.   for j=1 to n do
7.     {
8.       if xi == yj then
9.         c[i,j]=c[i-1,j-1]+1;
10.        b[i,j]=1;
11.      else if c[i-1,j]>=c[i,j-1] then
12.        c[i,j]=c[i-1,j]
13.        b[i,j]=2;
14.      else c[i,j]=c[i,j-1]
15.        b[i,j]=3;
16.    }
```

### Constructing an LCS (back-tracking)

- We can find an LCS using  $b[i,j]$ 's.
- We start with  $b[n,m]$  and track back to some cell  $b[0,i]$  or  $b[i,0]$ .

### The algorithm to construct an LCS

```
1. i=m
2. j=n;
3. if i==0 or j==0 then exit;
4. if b[i,j]=1 then
  {
    i=i-1;
    j=j-1;
    print "xi";
  }
5. if b[i,j]==2 i=i-1
6. if b[i,j]==3 j=j-1
7. Goto Step 3.
```

## **CODE for LCS Problem using DP: -**

```
#include<stdio.h>
#include<string.h>
int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];

void print(int i,int j)
{
if(i==0 || j==0)
return;
if(b[i][j]=='c')
{
print(i-1,j-1);
printf("%c",x[i-1]);}
else if(b[i][j]=='u')
print(i-1,j);
else
print(i,j-1);
}

void lcs()
{
m=strlen(x);
n=strlen(y);
for(i=0;i<=m;i++)
c[i][0]=0;
for(i=0;i<=n;i++)
c[0][i]=0;
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)
{
if(x[i-1]==y[j-1])
{
c[i][j]=c[i-1][j-1]+1;
b[i][j]='c';
}
else if(c[i-1][j]>=c[i][j-1])
{
c[i][j]=c[i-1][j];
b[i][j]='u';
}
else
{
c[i][j]=c[i][j-1];
b[i][j]='l';
} }
}

void main()
{
printf("Enter 1st sequence: \n");
scanf("%s",x);
printf("Enter 2nd sequence: \n");
scanf("%s",y);
printf("\nThe Longest Common Subsequence is: \n ");
lcs();
print(m,n);
}
```

*Output obtained from the above code for LCS Problem using Dynamic Programming:*

```
C:\Users\Vivek hotti\Desktop>a
Enter 1st sequence:
ACFGHD
Enter 2nd sequence:
ABFHD

The Longest Common Subsequence is:
AFHD
```

### **CONCLUSION:**

Time Complexity of the above implementation is  $O(mn)$ .

## ➡ EXPERIMENT NO. 4 ⬅

### AIM:

To implement Minimum Spanning Tree Algorithm Prim's & Kruskal's algorithm.

### THEORY:

#### Minimum Spanning Tree:

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

A minimum spanning tree has  $(V - 1)$  edges where  $V$  is the number of vertices in the given graph.

#### Prim's Algorithm:

Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

A group of edges that connects two set of vertices in a graph is called cut in graph theory. So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

**How does Prim's Algorithm Work?** The idea behind Prim's algorithm is simple; a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning Tree*. And they must be connected with the minimum weight edge to make it a *Minimum Spanning Tree*.

### ***Algorithm***

1) Create a set *mstSet* that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE.

Assign key value as 0 for the first vertex so that it is picked first.

3) While *mstSet* doesn't include all vertices

....a) Pick a vertex *u* which is not there in *mstSet* and has minimum key value.

....b) Include *u* to *mstSet*.

....c) Update key value of all adjacent vertices of *u*. To update the key values, iterate through all

adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v*

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

### **Analysis**

The algorithm spends most of its time in finding the smallest edge. So, time of the algorithm basically depends on how do we search this edge.

Straightforward method. Just find the smallest edge by searching the adjacency list of the vertices in *V*. In this case, each iteration costs  $O(m)$  time, yielding a total running time of  $O(mn)$ .

### **Kruskal's Algorithm:**

**Kruskal's algorithm** is a greedy algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each connected component).

Below are the steps for finding MST using Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.



3.Repeat step#2 until there are (V-1) edges in the spanning tree.

### Algorithm

Start with an empty set A, and select at every stage the shortest edge that has not been chosen or

rejected, regardless of where this edge is situated in the graph.

KRUSKAL(V, E, w)

$A \leftarrow \{ \}$   $\triangleright$  Set A will ultimately contains the edges of the

MST **foreach** vertex  $v$  in V

**do** MAKE-SET( $v$ )

    sort E into nondecreasing order by weight  $w$

**for each** ( $u,v$ ) taken from the sorted list

**do if** FIND-SET( $u$ ) = FIND-SET( $v$ )

**then**  $A \leftarrow A \cup \{(u,v)\}$

        UNION( $u,v$ )

**return** A

### Analysis

The edge weight can be compared in constant time. Initialization of priority queue takes  $O(E)$  time by repeated insertion. At each iteration of while-loop, minimum edge can be removed in  $O(\log E)$  time, which is  $O(\log V)$ , since graph is simple. The total running time is  $O((V + E) \log V)$ , which is  $O(E \lg V)$  since graph is simple and connected.

## CODE for PRIMS Algorithm: -

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0},min,mincost=0,cost[10][10];
void main()
{
    printf("\n Enter the number of nodes:");
    scanf("%d",&n);
    printf("\n Enter the 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]=999;
        }
    visited[1]=1;
```

```

printf("\n");
while(ne<n)
{
    for (i=1,min=999;i<=n;i++)
        for (j=1;j<=n;j++)
            if(cost[i][j]<min)
                if(visited[i]!=0)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
    if(visited[u]==0 || visited[v]==0)
    {
        printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
        mincost+=min;
        visited[b]=1;
    }
    cost[a][b]=cost[b][a]=999;
}
printf("\n Minimun cost=%d",mincost);
getch();
}

```

***Output obtained from the above code for PRIMS Algorithm:***

```
C:\Users\Vivek hotti\Desktop>gcc exp1.c
```

```
C:\Users\Vivek hotti\Desktop>a
```

```
Enter the number of nodes:7
```

```
Enter the adjacency matrix:
```

```

0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24
10 0 0 0 25 0 0
0 14 0 18 24 0 0

```

```
Edge 1:(1 6) cost:10
```

```
Edge 2:(6 5) cost:25
```

```
Edge 3:(5 4) cost:22
```

```
Edge 4:(4 3) cost:12
```

```
Edge 5:(3 2) cost:16
```

```
Edge 6:(2 7) cost:14
```

```
Minimun cost=99
```

## CODE for Kruskal's Algorithm: -

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
    printf("Implementation of Kruskal's algorithm \n");
    printf("Enter the no. of vertices: \n");
    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]=999;
        }
    }
    printf("The edges of Minimum Cost Spanning Tree are: \n");
    while(ne<n)
    {
        for(i=1,min=999;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                if(cost[i][j]<min)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }
            }
        }
        u=find(u);
        v=find(v);
        if(uni(u,v))
        {
            printf("%d edge (%d,%d) =%d \n",ne++,a,b,min);
            mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("tMinimum cost = %d \n",mincost);
    getch();
}
int find(int i)
```

```

{
    while (parent[i])
        i=parent[i];
    return i;
}
int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
        return 1;
    }
    return 0;
}

```

***Output obtained from the above code for Kruskal's Algorithm:***

```

C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
Implementation of Kruskal's algorithm
Enter the no. of vertices:
7
Enter the cost adjacency matrix:
0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24
10 0 0 0 25 0 0
0 14 0 18 24 0 0
The edges of Minimum Cost Spanning Tree are:
1 edge (1,6) =10
2 edge (3,4) =12
3 edge (2,7) =14
4 edge (2,3) =16
5 edge (4,5) =22
6 edge (5,6) =25
tMinimum cost = 99

```

## **CONCLUSION:**

- Prim's algorithm initializes with a node, whereas Kruskal's algorithm initiates with an edge.
- Prim's algorithms span from one node to another while Kruskal's algorithm select the edges in a way that the position of the edge is not based on the last step.
- In prim's algorithm, graph must be a connected graph while the Kruskal's can function on disconnected graphs too.
- Prim's algorithm has a time complexity of  $O(V^2)$ , and Kruskal's time complexity is  $O(\log V)$ .

## ➡ EXPERIMENT NO. 5 ⬅

### AIM:

- a) To implement n queen problem

### THEORY:

The **eight queens puzzle** is the problem of placing eight chess queens on an  $8 \times 8$  chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general ***n*-queens problem** of placing  $n$  queens on an  $n \times n$  chessboard, where solutions exist for all natural numbers  $n$  with the exception of  $n=2$  and  $n=3$

A way to place all  $n$  queens on the board such that no queens are attacking another queen. Using backtracking, this algorithm prints all possible placements of  $n$  queens on an  $n \times n$  chessboard so that they are non attacking. Place ( $k, i$ ) – Returns true if a queen can be placed in  $k$ th row and  $i$ th column. Otherwise it returns false.  $X []$  is a Global array whose first  $(k - 1)$  values have been set. Abs( $r$ ) returns the absolute value of  $r$ .

### **Algorithm:**

Algorithm NQueens ( $k, n$ ) //Prints all Solution to the  $n$ -queens problem

```
{
for i := 1 to n do
{
if Place ( $k, i$ ) then{
 $x[k] := i$ ;
if ( $k = n$ ) then write (  $x [1 : n]$ 
else NQueens( $k+1, n$ )
}}}
```

Algorithm Place ( $k, i$ )

```
{
for j := 1 to  $k-1$  do

if ((  $x[j] = i$  // in the same column

or (Abs(  $x[j] - i$ ) = Abs (  $j - k$  ))) // or in the same diagonal

then return false;

return true;
}
```

## CODE for 'N' Queen Problem: -

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<process.h>
int board[20];
int count;
void main(){
int n,i,j;
void Queen(int row,int n);
printf("\n\t Program for n-Queen's Using Backtracking");
printf("\nEnter Number of Queen's");
scanf("%d",&n);
Queen(1,n);//trace using backtrack
getch();
}
/* This function is for printing the solution
   Second Year Computer Engineering AOA
51
to n-queen's problem
*/
void print_board(int n)
{
int i,j;
printf("\n\nSolution %d : \n\n",++count);
//number of solution
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++)// for board
{
if(board[i]==j)
printf("\tQ");//Queen at i,j position
else
printf("\t-");// empty slot
}
}
printf("\n Press any key to continue...");
getch();
}
int place(int row,int column)
{
int i;
for(i=1;i<=row-1;i++)
{ //checking for column and diagonal conflicts
if(board[i] == column)
return 0;
else
if(abs(board[i] - column) == abs(i - row))
return 0;
```

```

}
//no conflicts hence Queen can be placed
return 1;
}
void Queen(int row,int n)
{
int column;

for(column=1;column<=n;column++)
{
if(place(row,column))
{
board[row] = column;//no conflict so place queen
if(row==n)//dead end
print_board(n);
//printing the board configuration
else //try next queen with next position
Queen(row+1,n);
}
}
}
}

```

***Output obtained from the above code for 'N' Queen Problem:***

```

C:\Users\Vivek hotti\Desktop>a

          Program for n-Queen's Using Backtracking
Enter Number of Queen's5

Solution 1 :

          1      2      3      4      5
1      Q      -      -      -      -
2      -      -      Q      -      -
3      -      -      -      -      Q
4      -      Q      -      -      -
5      -      -      -      Q      -
Press any key to continue...

```



## **CONCLUSION:**

- Backtracking provides the hope to solve some problem instances of nontrivial sizes by pruning non-promising branches of the state-space tree.
- The success of backtracking varies from problem to problem and from instance to instance.
- Backtracking possibly generates all possible candidates in an exponentially growing state-space tree.

## **AIM:**

b) To implement Graph Coloring

## **THEORY:**

Given an undirected graph and a number  $m$ , determine if the graph can be colored with at most  $m$  colors such that no two adjacent vertices of the graph are colored with same color. Here coloring of a graph means assignment of colors to all vertices.

For the graph-coloring problem we are interested in assigning colors to the vertices of an undirected graph with the restriction that no two adjacent vertices are assigned the same color. The optimization version calls for coloring a graph using the minimum number of colors. The decision version, known as  $k$ -coloring, asks whether a graph is colorable using at most  $k$  colors. The 3-coloring problem is a special case of the  $k$ -coloring problem where  $k=3$  (i.e., we are allowed to use at most 3 colors).

*Input:*

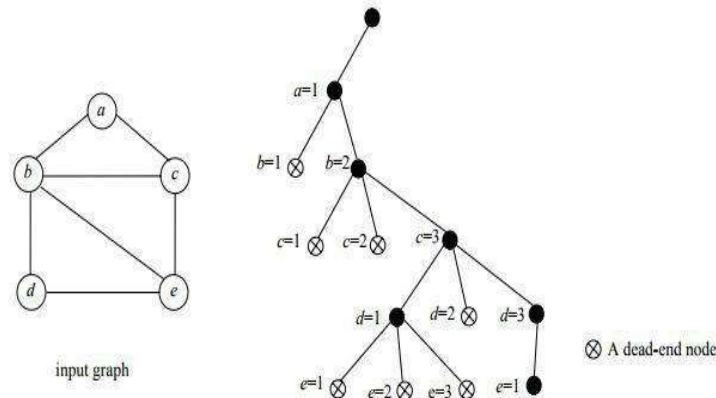
1) A 2D array  $\text{graph}[V][V]$  where  $V$  is the number of vertices in graph and  $\text{graph}[V][V]$  is adjacency matrix representation of the graph. A value  $\text{graph}[i][j]$  is 1 if there is a direct edge from  $i$  to  $j$ , otherwise  $\text{graph}[i][j]$  is 0.

- 2) An integer  $m$  which is maximum number of colors that can be used.

*Output:*

An array `color[V]` that should have numbers from 1 to  $m$ . `color[i]` should represent the color assigned to the  $i$ th vertex. The code should also return false if the graph cannot be colored with  $m$  colors.

### Example



### Algorithm:

If all colors are assigned,

print vertex assigned colors

Else a. Trying all possible colors, assign a color to the vertex

b. If color assignment is possible, recursively assign colors to next vertices

c. If color assignment is not possible, de-assign color, return False

## CODE to implement GRAPH COLORING: -

```
#include<stdio.h>
int G[50][50],x[50]; //G:adjacency matrix,x:colors
void next_color(int k){
    int i,j;
    x[k]=1; //coloring vertex with color1
    for(i=0;i<k;i++){ //checking all k-1 vertices-backtracking
        if(G[i][k]!=0 && x[k]==x[i]) //if connected and has same color
            x[k]=x[i]+1; //assign higher color than x[i]
    }
}
int main(){
```

```

int n,e,i,j,k,l;
printf("Enter no. of vertices : ");
scanf("%d",&n); //total vertices
printf("Enter no. of edges : ");
scanf("%d",&e); //total edges

for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        G[i][j]=0; //assign 0 to all index of adjacency matrix

printf("Enter indexes where value is 1-->\n");
for(i=0;i<e;i++){
    scanf("%d %d",&k,&l);
    G[k][l]=1;
    G[l][k]=1;
}

for(i=0;i<n;i++)
    next_color(i); //coloring each vertex

printf("Colors of vertices -->\n");
for(i=0;i<n;i++) //displaying color of each vertex
    printf("Vertex[%d] : %d\n",i+1,x[i]);

return 0;
}

```

***Output obtained from the above code for Graph Coloring:***

```

C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
Enter no. of vertices : 4
Enter no. of edges : 5
Enter indexes where value is 1-->
0 1
1 2
1 3
2 3
3 0
Colors of vertices -->
Vertex[1] : 1
Vertex[2] : 2
Vertex[3] : 1
Vertex[4] : 3

C:\Users\Vivek hotti\Desktop>

```

## ➡ EXPERIMENT NO. 6 ⬅

### AIM:

a) To implement Naive String Matching

### THEORY:

Given a text `txt[0..n-1]` and a pattern `pat[0..m-1]`, write a function `search(char pat[], char txt[])` that prints all occurrences of `pat[]` in `txt[]`. You may assume that  $n > m$ .

Input : `txt[] = "THIS IS A TEST TEXT"`

`pat[] =  
"TEST"`

Output : Pattern found at index  
10

Input : `txt[] = "AABAACAADAABAABA"`

`pat[] = "AABA"`

Output : Pattern found at index  
0

Pattern found at index  
9

Pattern found at index  
12

Pattern searching is an important problem in Computer Science. When we do search for a string in notepad/word file or browser or database, pattern searching algorithms are used to show the search results.

#### **Naive Pattern Searching:**

Slide the pattern over text one by one and check for a match. If a match is found, then slides by 1 again to check for subsequent matches.

## PSEUDOCODE:

```
search(txt,
pat) {
    int m =
    pat.length; int n
    = txt.length;

    for( i = 0 to n-m
    ) {
        int j; for( j = 0
        to M )
            if (txt[i + j] != pat[j])
                break
            ;

        if (j == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1] printf("Pattern found at index %d n",
        i); } } /* Driver program to test above function */ int main() {
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA"; search(txt, pat);
    return 0; } What is the best case?
```

The best case occurs when the first character of the pattern is not present in text at all.

```
txt[] = "AABCCAADDEE"; pat[] = "FAA";
The number of comparisons in best case is O(n).
```

### What is the worst case ?

The worst case of Naive Pattern Searching occurs in following scenarios.

1) When all characters of the text and pattern are same.

```
txt[] = "AAAAAAAAAAAAAAAAAAAAA"; pat[] = "AAAAA"; 2)
Worst case also occurs when only the last character is different.
```

txt[] = "AAAAAAAAAAAAAAAAAAB"; pat[] = "AAAAB"; Number of comparisons in worst case is  $O(m \cdot (n-m+1))$ . Although strings which have repeated characters are not likely to appear in English text, they may well occur in other applications (for example, in binary texts). The KMP matching algorithm improves the worst case to  $O(n)$ . We will be covering KMP in the next

post. Also, we will be writing more posts to cover all pattern searching algorithms and data structures.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

## **CODE for Naïve String Machine: -**

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

int main()
{
    int str_length, pattern_length, j, i, count = 0;
    char str[30], pattern[30] ;
    printf("\nEnter a String:\t");
    scanf("%s", str);
    printf("\nEnter a Pattern to Match:\t");
    scanf("%s", pattern);
    str_length = strlen(str);
    pattern_length = strlen(pattern);
    printf("\nPattern Matched at Position:\t");
    for(i = 0; i < str_length - pattern_length; i++)
    {
        for(j = 0; j < pattern_length; j++)
        {
            if(str[i + j] == pattern[j]) count++;
        }
        if(count == pattern_length)
        {
            printf("%d\t", i);
        }
        count = 0;
    }
    printf("\n");
    return 0;
}
```

***Output obtained from the above code for Naive String Machine:***

```
C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a

Enter a String: VIVEKHOTTI

Enter a Pattern to Match:      EKH

Pattern Matched at Position:   3
```

## AIM:

b) To implement Rabin Karp

## THEORY:

Given a text  $txt[0..n-1]$  and a pattern  $pat[0..m-1]$ , write a function `search(char pat[], char txt[])` that prints all occurrences of  $pat[]$  in  $txt[]$ . You may assume that  $n > m$ .

Input : `txt[] = "THIS IS A TEST TEXT"`

`pat[] =  
"TEST"`

Output : Pattern found at index  
10

Input : `txt[] = "AABAACAADAABAABA"`

`pat[] = "AABA"`

Output : Pattern found at index  
0

Pattern found at index  
9

Pattern found at index  
12

Like the Naive Algorithm, Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, Rabin Karp algorithm matches the hash value of the pattern with the hash value of current substring of text, and if the hash values match then only it starts matching

individual characters. So Rabin Karp algorithm needs to calculate hash values for following strings.

1) Pattern itself.

2) All the substrings of text of length  $m$ .

Since we need to efficiently calculate hash values for all the substrings of size  $m$  of text, we must have a hash function which has following property.

Hash at the next shift must be efficiently computable from the current hash value and next character in text or we can say  $hash(txt[s+1 .. s+m])$  must be efficiently computable from  $hash(txt[s .. s+m-1])$  and  $txt[s+m]$  i.e.,  $hash(txt[s+1 .. s+m]) = rehash(txt[s+m], hash(txt[s .. s+m-1]))$  and rehash must be  $O(1)$  operation.

The hash function suggested by Rabin and Karp calculates an integer value. The integer value for a string is numeric value of a string. For example, if all possible characters are from 1 to 10, the numeric value of "122" will be 122. The number of possible characters is higher than 10 (256 in general) and pattern length can be large. So the numeric values cannot be practically stored as an integer. Therefore, the numeric value is calculated using modular arithmetic to make sure that the hash values can be stored in an integer variable (can fit in memory words). To do rehashing, we need to take off the most significant digit and add the new least significant digit for in hash value. Rehashing is done using the following formula.

$$hash(txt[s+1 .. s+m]) = (d (hash(txt[s .. s+m-1]) - txt[s]*h) + txt[s+m]) \bmod q$$

$hash(txt[s .. s+m-1])$  : Hash value at shift  $s$ .

$hash(txt[s+1 .. s+m])$  : Hash value at next shift (or shift  $s+1$ )

$d$ : Number of characters in the alphabet

$q$ : A prime number

$h: d^{(m-1)}$

## **CODE for Rabin Karp: -**

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <math.h>
#define d 10
void RabinKarpStringMatch(char*, char*, int);
void main()
{
    char *Text, *Pattern;
    int Number = 11; //Prime Number
    printf("Enter Text String : \n");
    gets(Text);
    printf("Enter Pattern String : \n");
    gets(Pattern);

    RabinKarpStringMatch(Text, Pattern, Number);
    getch();
}

void RabinKarpStringMatch(char* Text, char* Pattern, int Number)
{
    int M, N, h, P = 0, T = 0, TempT, TempP;
    int i, j;
```



```

M = strlen(Pattern);
N = strlen(Text);
h = (int)pow(d, M - 1) % Number;
for (i = 0; i < M; i++) {
    P = ((d * P) + ((int)Pattern[i])) % Number;
    TempT = ((d * T) + ((int)Text[i]));
    T = TempT % Number;
}
for (i = 0; i <= N - M; i++) {
    if (P == T) {
        for (j = 0; j < M; j++)
            if (Text[i + j] != Pattern[j])
                break;
        if (j == M)
            printf("\nPattern Found at Position: %d", i + 1);
    }
    TempT = ((d * (T - Text[i] * h)) + ((int)Text[i + M]));
    T = TempT % Number;
    if (T < 0)
        T = T + Number;
}
}

```

*Output obtained from the above code for Rabin Karp:*

```

Enter Text String : farheen khan
Enter Pattern String : kh
Pattern Found at Position: 9_

```

## **CONCLUSION:**

The average and best case running time of the Rabin-Karp algorithm is  $O(n+m)$ , but its worst-case time is  $O(nm)$ . Worst case of Rabin-Karp algorithm occurs when all characters of pattern and text are same as the hash values of all the substrings of `txt[]` match with hash value of `pat[]`. For example `pat[] = "AAA"` and `txt[] = "AAAAAAA"`.

## ➡ EXPERIMENT NO. 7 ⬅

### AIM:

To implement Binary Search Tree

### THEORY:

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

The left sub-tree of a node has a key less than or equal to its parent node's key.

The right sub-tree of a node has a key greater than to its parent node's key.

Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as –

$\text{left\_subtree}(\text{keys}) \leq \text{node}(\text{key}) \leq \text{right\_subtree}(\text{keys})$

Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST –

Binary Search Tree

We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

Basic Operations

Following are the basic operations of a tree –

Search – Searches an element in a tree.

Insert – Inserts an element in a tree.

Pre-order Traversal – Traverses a tree in a pre-order manner.

In-order Traversal – Traverses a tree in an in-order manner.

Post-order Traversal – Traverses a tree in a post-order manner.

Search Operation

Whenever an element is to be searched, start searching from the root node. Then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree. Follow the same algorithm for each node.

#### Algorithm

```
struct node* search(int data){
    struct node *current = root;
    printf("Visiting elements: ");

    while(current->data != data){

        if(current != NULL) {
            printf("%d ",current->data);

            //go to left tree
            if(current->data > data){
                current = current->leftChild;
            } //else go to right tree
            else {
                current = current->rightChild;
            }

            //not found
            if(current == NULL){
                return NULL;
            }
        }
    }

    return current;
}
```

#### Insert Operation

Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

#### Algorithm

```
void insert(int data) {
    struct node *tempNode = (struct node*) malloc(sizeof(struct node));
    struct node *current;
    struct node *parent;

    tempNode->data = data;
    tempNode->leftChild = NULL;
```

```

tempNode->rightChild = NULL;

//if tree is empty
if(root == NULL) {
    root = tempNode;
} else {
    current = root;
    parent = NULL;

    while(1) {
        parent = current;

        //go to left of the tree
        if(data < parent->data) {
            current = current->leftChild;
            //insert to the left

            if(current == NULL) {
                parent->leftChild = tempNode;
                return;
            }
        } //go to right of the tree
        else {
            current = current->rightChild;

            //insert to the right
            if(current == NULL) {
                parent->rightChild = tempNode;
                return;
            }
        }
    }
}
}

```

## CODE to implement Binary Search Tree: -

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

struct node
{
    int key;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to do inorder traversal of BST
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d \n", root->key);
        inorder(root->right);
    }
}

/* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}

// Driver Program to test above functions
int main()
{
    /* Let us create following BST
    50
    30
    20
    40
    10
    60
    70
    80
    90
    100
    110
    120
    130
    140
    150
    160
    170
    180
    190
    200
    210
    220
    230
    240
    250
    260
    270
    280
    290
    300
    310
    320
    330
    340
    350
    360
    370
    380
    390
    400
    410
    420
    430
    440
    450
    460
    470
    480
    490
    500
    510
    520
    530
    540
    550
    560
    570
    580
    590
    600
    610
    620
    630
    640
    650
    660
    670
    680
    690
    700
    710
    720
    730
    740
    750
    760
    770
    780
    790
    800
    810
    820
    830
    840
    850
    860
    870
    880
    890
    900
    910
    920
    930
    940
    950
    960
    970
    980
    990
    1000
    1010
    1020
    1030
    1040
    1050
    1060
    1070
    1080
    1090
    1100
    1110
    1120
    1130
    1140
    1150
    1160
    1170
    1180
    1190
    1200
    1210
    1220
    1230
    1240
    1250
    1260
    1270
    1280
    1290
    1300
    1310
    1320
    1330
    1340
    1350
    1360
    1370
    1380
    1390
    1400
    1410
    1420
    1430
    1440
    1450
    1460
    1470
    1480
    1490
    1500
    1510
    1520
    1530
    1540
    1550
    1560
    1570
    1580
    1590
    1600
    1610
    1620
    1630
    1640
    1650
    1660
    1670
    1680
    1690
    1700
    1710
    1720
    1730
    1740
    1750
    1760
    1770
    1780
    1790
    1800
    1810
    1820
    1830
    1840
    1850
    1860
    1870
    1880
    1890
    1900
    1910
    1920
    1930
    1940
    1950
    1960
    1970
    1980
    1990
    2000
    2010
    2020
    2030
    2040
    2050
    2060
    2070
    2080
    2090
    2100
    2110
    2120
    2130
    2140
    2150
    2160
    2170
    2180
    2190
    2200
    2210
    2220
    2230
    2240
    2250
    2260
    2270
    2280
    2290
    2300
    2310
    2320
    2330
    2340
    2350
    2360
    2370
    2380
    2390
    2400
    2410
    2420
    2430
    2440
    2450
    2460
    2470
    2480
    2490
    2500
    2510
    2520
    2530
    2540
    2550
    2560
    2570
    2580
    2590
    2600
    2610
    2620
    2630
    2640
    2650
    2660
    2670
    2680
    2690
    2700
    2710
    2720
    2730
    2740
    2750
    2760
    2770
    2780
    2790
    2800
    2810
    2820
    2830
    2840
    2850
    2860
    2870
    2880
    2890
    2900
    2910
    2920
    2930
    2940
    2950
    2960
    2970
    2980
    2990
    3000
    3010
    3020
    3030
    3040
    3050
    3060
    3070
    3080
    3090
    3100
    3110
    3120
    3130
    3140
    3150
    3160
    3170
    3180
    3190
    3200
    3210
    3220
    3230
    3240
    3250
    3260
    3270
    3280
    3290
    3300
    3310
    3320
    3330
    3340
    3350
    3360
    3370
    3380
    3390
    3400
    3410
    3420
    3430
    3440
    3450
    3460
    3470
    3480
    3490
    3500
    3510
    3520
    3530
    3540
    3550
    3560
    3570
    3580
    3590
    3600
    3610
    3620
    3630
    3640
    3650
    3660
    3670
    3680
    3690
    3700
    3710
    3720
    3730
    3740
    3750
    3760
    3770
    3780
    3790
    3800
    3810
    3820
    3830
    3840
    3850
    3860
    3870
    3880
    3890
    3900
    3910
    3920
    3930
    3940
    3950
    3960
    3970
    3980
    3990
    4000
    4010
    4020
    4030
    4040
    4050
    4060
    4070
    4080
    4090
    4100
    4110
    4120
    4130
    4140
    4150
    4160
    4170
    4180
    4190
    4200
    4210
    4220
    4230
    4240
    4250
    4260
    4270
    4280
    4290
    4300
    4310
    4320
    4330
    4340
    4350
    4360
    4370
    4380
    4390
    4400
    4410
    4420
    4430
    4440
    4450
    4460
    4470
    4480
    4490
    4500
    4510
    4520
    4530
    4540
    4550
    4560
    4570
    4580
    4590
    4600
    4610
    4620
    4630
    4640
    4650
    4660
    4670
    4680
    4690
    4700
    4710
    4720
    4730
    4740
    4750
    4760
    4770
    4780
    4790
    4800
    4810
    4820
    4830
    4840
    4850
    4860
    4870
    4880
    4890
    4900
    4910
    4920
    4930
    4940
    4950
    4960
    4970
    4980
    4990
    5000
    5010
    5020
    5030
    5040
    5050
    5060
    5070
    5080
    5090
    5100
    5110
    5120
    5130
    5140
    5150
    5160
    5170
    5180
    5190
    5200
    5210
    5220
    5230
    5240
    5250
    5260
    5270
    5280
    5290
    5300
    5310
    5320
    5330
    5340
    5350
    5360
    5370
    5380
    5390
    5400
    5410
    5420
    5430
    5440
    5450
    5460
    5470
    5480
    5490
    5500
    5510
    5520
    5530
    5540
    5550
    5560
    5570
    5580
    5590
    5600
    5610
    5620
    5630
    5640
    5650
    5660
    5670
    5680
    5690
    5700
    5710
    5720
    5730
    5740
    5750
    5760
    5770
    5780
    5790
    5800
    5810
    5820
    5830
    5840
    5850
    5860
    5870
    5880
    5890
    5900
    5910
    5920
    5930
    5940
    5950
    5960
    5970
    5980
    5990
    6000
    6010
    6020
    6030
    6040
    6050
    6060
    6070
    6080
    6090
    6100
    6110
    6120
    6130
    6140
    6150
    6160
    6170
    6180
    6190
    6200
    6210
    6220
    6230
    6240
    6250
    6260
    6270
    6280
    6290
    6300
    6310
    6320
    6330
    6340
    6350
    6360
    6370
    6380
    6390
    6400
    6410
    6420
    6430
    6440
    6450
    6460
    6470
    6480
    6490
    6500
    6510
    6520
    6530
    6540
    6550
    6560
    6570
    6580
    6590
    6600
    6610
    6620
    6630
    6640
    6650
    6660
    6670
    6680
    6690
    6700
    6710
    6720
    6730
    6740
    6750
    6760
    6770
    6780
    6790
    6800
    6810
    6820
    6830
    6840
    6850
    6860
    6870
    6880
    6890
    6900
    6910
    6920
    6930
    6940
    6950
    6960
    6970
    6980
    6990
    7000
    7010
    7020
    7030
    7040
    7050
    7060
    7070
    7080
    7090
    7100
    7110
    7120
    7130
    7140
    7150
    7160
    7170
    7180
    7190
    7200
    7210
    7220
    7230
    7240
    7250
    7260
    7270
    7280
    7290
    7300
    7310
    7320
    7330
    7340
    7350
    7360
    7370
    7380
    7390
    7400
    7410
    7420
    7430
    7440
    7450
    7460
    7470
    7480
    7490
    7500
    7510
    7520
    7530
    7540
    7550
    7560
    7570
    7580
    7590
    7600
    7610
    7620
    7630
    7640
    7650
    7660
    7670
    7680
    7690
    7700
    7710
    7720
    7730
    7740
    7750
    7760
    7770
    7780
    7790
    7800
    7810
    7820
    7830
    7840
    7850
    7860
    7870
    7880
    7890
    7900
    7910
    7920
    7930
    7940
    7950
    7960
    7970
    7980
    7990
    8000
    8010
    8020
    8030
    8040
    8050
    8060
    8070
    8080
    8090
    8100
    8110
    8120
    8130
    8140
    8150
    8160
    8170
    8180
    8190
    8200
    8210
    8220
    8230
    8240
    8250
    8260
    8270
    8280
    8290
    8300
    8310
    8320
    8330
    8340
    8350
    8360
    8370
    8380
    8390
    8400
    8410
    8420
    8430
    8440
    8450
    8460
    8470
    8480
    8490
    8500
    8510
    8520
    8530
    8540
    8550
    8560
    8570
    8580
    8590
    8600
    8610
    8620
    8630
    8640
    8650
    8660
    8670
    8680
    8690
    8700
    8710
    8720
    8730
    8740
    8750
    8760
    8770
    8780
    8790
    8800
    8810
    8820
    8830
    8840
    8850
    8860
    8870
    8880
    8890
    8900
    8910
    8920
    8930
    8940
    8950
    8960
    8970
    8980
    8990
    9000
    9010
    9020
    9030
    9040
    9050
    9060
    9070
    9080
    9090
    9100
    9110
    9120
    9130
    9140
    9150
    9160
    9170
    9180
    9190
    9200
    9210
    9220
    9230
    9240
    9250
    9260
    9270
    9280
    9290
    9300
    9310
    9320
    9330
    9340
    9350
    9360
    9370
    9380
    9390
    9400
    9410
    9420
    9430
    9440
    9450
    9460
    9470
    9480
    9490
    9500
    9510
    9520
    9530
    9540
    9550
    9560
    9570
    9580
    9590
    9600
    9610
    9620
    9630
    9640
    9650
    9660
    9670
    9680
    9690
    9700
    9710
    9720
    9730
    9740
    9750
    9760
    9770
    9780
    9790
    9800
    9810
    9820
    9830
    9840
    9850
    9860
    9870
    9880
    9890
    9900
    9910
    9920
    9930
    9940
    9950
    9960
    9970
    9980
    9990
    10000
    10010
    10020
    10030
    10040
    10050
    10060
    10070
    10080
    10090
    10100
    10110
    10120
    10130
    10140
    10150
    10160
    10170
    10180
    10190
    10200
    10210
    10220
    10230
    10240
    10250
    10260
    10270
    10280
    10290
    10300
    10310
    10320
    10330
    10340
    10350
    10360
    10370
    10380
    10390
    10400
    10410
    10420
    10430
    10440
    10450
    10460
    10470
    10480
    10490
    10500
    10510
    10520
    10530
    10540
    10550
    10560
    10570
    10580
    10590
    10600
    10610
    10620
    10630
    10640
    10650
    10660
    10670
    10680
    10690
    10700
    10710
    10720
    10730
    10740
    10750
    10760
    10770
    10780
    10790
    10800
    10810
    10820
    10830
    10840
    10850
    10860
    10870
    10880
    10890
    10900
    10910
    10920
    10930
    10940
    10950
    10960
    10970
    10980
    10990
    11000
    11010
    11020
    11030
    11040
    11050
    11060
    11070
    11080
    11090
    11100
    11110
    11120
    11130
    11140
    11150
    11160
    11170
    11180
    11190
    11200
    11210
    11220
    11230
    11240
    11250
    11260
    11270
    11280
    11290
    11300
    11310
    11320
    11330
    11340
    11350
    11360
    11370
    11380
    11390
    11400
    11410
    11420
    11430
    11440
    11450
    11460
    11470
    11480
    11490
    11500
    11510
    11520
    11530
    11540
    11550
    11560
    11570
    11580
    11590
    11600
    11610
    11620
    11630
    11640
    11650
    11660
    11670
    11680
    11690
    11700
    11710
    11720
    11730
    11740
    11750
    11760
    11770
    11780
    11790
    11800
    11810
    11820
    11830
    11840
    11850
    11860
    11870
    11880
    11890
    11900
    11910
    11920
    11930
    11940
    11950
    11960
    11970
    11980
    11990
    12000
    12010
    12020
    12030
    12040
    12050
    12060
    12070
    12080
    12090
    12100
    12110
    12120
    12130
    12140
    12150
    12160
    12170
    12180
    12190
    12200
    12210
    12220
    12230
    12240
    12250
    12260
    12270
    12280
    12290
    12300
    12310
    12320
    12330
    12340
    12350
    12360
    12370
    12380
    12390
    12400
    12410
    12420
    12430
    12440
    12450
    12460
    12470
    12480
    12490
    12500
    12510
    12520
    12530
    12540
    12550
    12560
    12570
    12580
    12590
    12600
    12610
    12620
    12630
    12640
    12650
    12660
    12670
    12680
    12690
    12700
    12710
    12720
    12730
    12740
    12750
    12760
    12770
    12780
    12790
    12800
    12810
    12820
    12830
    12840
    12850
    12860
    12870
    12880
    12890
    12900
    12910
    12920
    12930
    12940
    12950
    12960
    12970
    12980
    12990
    13000
    13010
    13020
    13030
    13040
    13050
    13060
    13070
    13080
    13090
    13100
    13110
    13120
    13130
    13140
    13150
    13160
    13170
    13180
    13190
    13200
    13210
    13220
    13230
    13240
    13250
    13260
    13270
    13280
    13290
    13300
    13310
    13320
    13330
    13340
    13350
    13360
    13370
    13380
    13390
    13400
    13410
    13420
    13430
    13440
    13450
    13460
    13470
    13480
    13490
    13500
    13510
    13520
    13530
    13540
    13550
    13560
    13570
    13580
    13590
    13600
    13610
    13620
    13630
    13640
    13650
    13660
    13670
    13680
    13690
    13700
    13710
    13720
    13730
    13740
    13750
    13760
    13770
    13780
    13790
    13800
    13810
    13820
    13830
    13840
    13850
    13860
    13870
    13880
    13890
    13900
    13910
    13920
    13930
    13940
    13950
    13960
    13970
    13980
    13990
    14000
    14010
    14020
    14030
    14040
    14050
    14060
    14070
    14080
    14090
    14100
    14110
    14120
    14130
    14140
    14150
    14160
    14170
    14180
    14190
    14200
    14210
    14220
    14230
    14240
    14250
    14260
    14270
    14280
    14290
    14300
    14310
    14320
    14330
    14340
    14350
    14360
    14370
    14380
    14390
    14400
    14410
    14420
    14430
    14440
    14450
    14460
    14470
    14480
    14490
    14500
    14510
    14520
    14530
    14540
    14550
    14560
    14570
    14580
    14590
    14600
    14610
    14620
    14630
    14640
    14650
    14660
    14670
    14680
    14690
    14700
    14710
    14720
    14730
    14740
    14750
    14760
    14770
    14780
    14790
    14800
    14810
    14820
    14830
    14840
    14850
    14860
    14870
    14880
    14890
    14900
    14910
    14920
    14930
    14940
    14950
    14960
    14970
    14980
    14990
    15000
    15010
    15020
    15030
    15040
    15050
    15060
    15070
    15080
    15090
    15100
    15110
    15120
    15130
    15140
    15150
    15160
    15170
    15180
    15190
    15200
    15210
    15220
    15230
    15240
    15250
    15260
    15270
    15280
    15290
    15300
    15310
    15320
    15330
    15340
    15350
    15360
    15370
    15380
    15390
    15400
    15410
    15420
    15430
    15440
    15450
    15460
    15470
    15480
    15490
    15500
    15510
    15520
    15530
    15540
    15550
    15560
    15570
    15580
    15590
    15600
    15610
    15620
    15630
    15640
    15650
    15660
    15670
    15680
    15690
    15700
    15710
    15720
    15730
    15740
    15750
    15760
    15770
    15780
    15790
    15800
    15810
    15820
    15830
    15840
    15850
    15860
    15870
    15880
    15890
    15900
    15910
    15920
    15930
    15940
    15950
    15960
    15970
    15980
    15990
    16000
    16010
    16020
    16030
    16040
    16050
    16060
    16070
    16080
    16090
    16100
    16110
    16120
    16130
    16140
    16150
    16160
    16170
    16180
    16190
    16200
    16210
    16220
    16230
    16240
    16250
    16260
    16270
    16280
    16290
    16300
    16310
    16320
    16330
    1634
```

```

      /      \
     30      70
    /  \    /  \
   20  40 60  80 */
struct node *root = NULL;
root = insert(root, 50);
insert(root, 30);
insert(root, 20);
insert(root, 40);
insert(root, 70);
insert(root, 60);
insert(root, 80);

// print inoder traversal of the BST
inorder(root);

return 0;
}
```

***Output obtained from the above code for Binary Search Tree:***

```
C:\Users\Vivek hotti\Desktop>gcc exp1.c

C:\Users\Vivek hotti\Desktop>a
20
30
40
50
60
70
80
```

## **CONCLUSION:**

Hence we have implemented a C program to implement a Binary Search Tree.

((((( ( ) )))))