Practical-4

Aim: Greedy Approach.

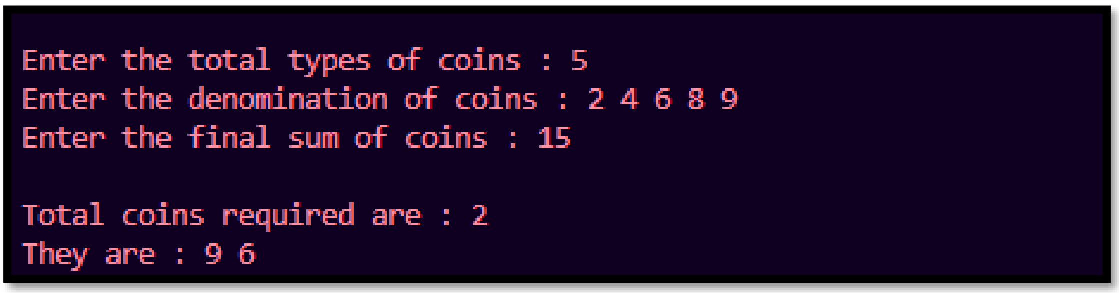
4.1 A cashier at any mall needs to give change of an amount to customers many times in a day. Cashier has multiple number of coins available with different denominations which is described by a set C. Implement the program fora cashier to find the minimum number of coins required to find a change of a particular amount A. Output should be the total number of coins required of given denominations. Check the program for following test cases:

Test Case	Coin denominations C	Amount A
1	₹1, ₹2, ₹3	₹ 5
2	₹18, ₹17, ₹5, ₹1	₹ 22
3	₹100, ₹25, ₹10, ₹5, ₹1	₹ 289

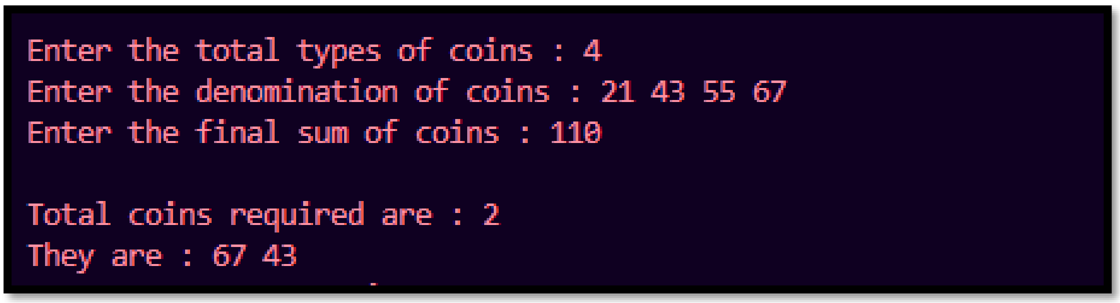
CODE:

```
#include <bits/stdc++.h>
using namespace std;
void minFinder(int sum,int length,int coins[])
{
        int ans[sum],j=0,i=0;

        for(i=length-1;i>=0;i--)
        {
       while(sum>=coins[i])
       {
           sum=sum-coins[i];
               ans[j]=coins[i];
           j++;
       }
     }
        cout<<"\nTotal coins required are : "<<j<<"\nThey are : ";
        for(i=0;i<j;i++)
     cout<<ans[i]<<" ";
     cout<<"\n";
}

int main()
{
        int length,i,sum;
        cout<<"\nEnter the total types of coins : ";
        cin>>length;
        int coins[length];
        cout<<"Enter the denomination of coins : ";
        for(i=0;i<length;i++)
    cin>>coins[i];
        int n = sizeof(coins)/sizeof(coins[0]);
        sort(coins,coins+n);
        cout<<"Enter the final sum of coins : ";
        cin>>sum;
        minFinder(sum,length,coins);

        return 0;
}
```

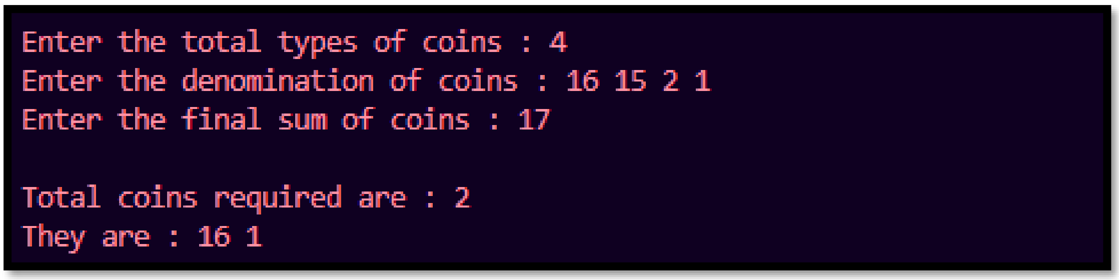
OUTPUT:



```
Enter the total types of coins : 5
Enter the denomination of coins : 2 4 6 8 9
Enter the final sum of coins : 15

Total coins required are : 2
They are : 9 6
```



```
Enter the total types of coins : 4
Enter the denomination of coins : 21 43 55 67
Enter the final sum of coins : 110

Total coins required are : 2
They are : 67 43
```



```
Enter the total types of coins : 4
Enter the denomination of coins : 16 15 2 1
Enter the final sum of coins : 17

Total coins required are : 2
They are : 16 1
```

OBSERVATION:

By performing the above practical it observed that it didn’t gave optimum solution for every test cases.
Analysis of different test cases:

Test Case	Denomination	Amount	No. of coins	Optimum Sol.
1	Rs 1,2,3,5	Rs 9	3	YES
2	Rs 22,20,5,1	Rs 25	4	NO
3	Rs 16,15,2,1	Rs 17	2	YES

From the observation we can say that the solution for test case 2 is not optimal.

Obtained Solution: 22+1+1+1 = 25 (4 coins)
Optimal Solution: 20 + 5 = 25 (2 coins)

4.2 Let S be a collection of objects with profit-weight values. Implement the fractional knapsack problem for S assuming we have a sack that can hold objects with total weight W. Check the program for following test cases:

Test Case	**S**	**profit-weight values**	**W**

1	{A,B,C}	Profit:(1,2,5)
		Weight:(2,3,4)
			5

2	{A,B,C,D,E,F,G}	Profit:(10,5,15,7,6,18,3)
		Weight(2,3,5,7,1,4,1)
			15

3	{A,B,C,D,E,F,G}	A:(12,4), B:(10,6), C:(8,5), D:(11,7), E:(14,3), F:(7,1), G:(9,6)	18

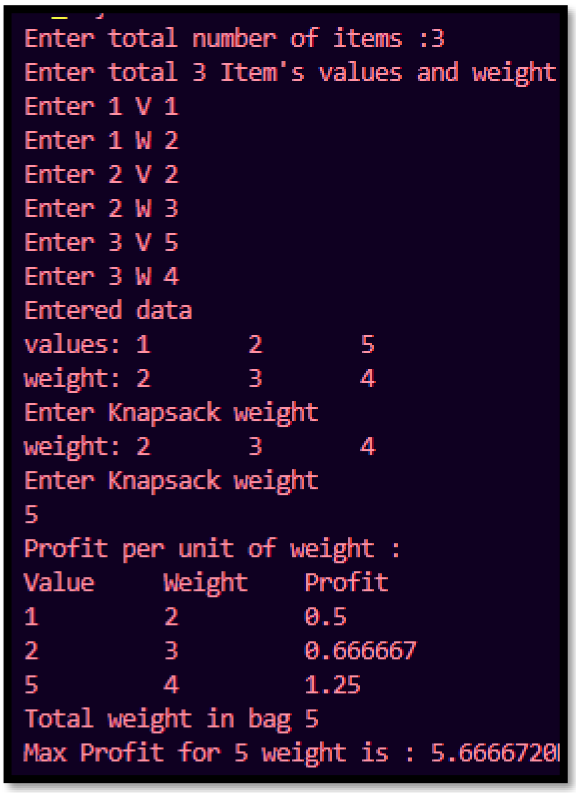
CODE:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
typedef struct {
  double v;
  double w;
} Item;
void input(Item items[],int sizeOfItems) {
  cout << "Enter total "<< sizeOfItems <<" Item's values and weight" <<
  endl;
  for(int i = 0; i < sizeOfItems; i++) {
    cout << "Enter "<< i+1 << " V ";
```

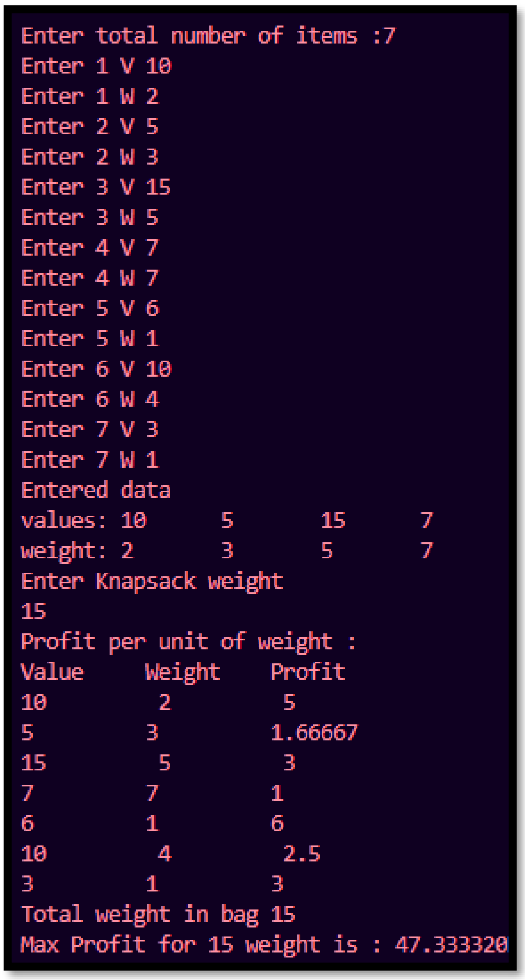
```
      cin >> items[i].v;
      cout << "Enter "<< i+1 << " W ";
      cin >> items[i].w;
   }
}
void display(Item items[], int sizeOfItems) {
   int i;
   cout << "values: ";
   for(i = 0; i < sizeOfItems; i++) {
      cout << items[i].v << "\t";
   }
   cout << endl << "weight: ";
   for (i = 0; i < sizeOfItems; i++) {
      cout << items[i].w << "\t";
   }
   cout << endl;
}
bool compare(Item a, Item b) {
    double r1 = (double)(a.v / a.w);
    double r2 = (double)(b.v / b.w);
    return r1 > r2;
}
double knapsack(Item items[], int sizeOfItems, int W) {
   int i, j;
   double totalValue = 0, totalWeight = 0;

   cout<<"Profit per unit of weight :\n";
   cout<<"Value     Weight    Profit\n";
   for (int i = 0; i < sizeOfItems; i++)
  {
      cout << items[i].v << "        " << items[i].w << "        " << ((double)items[i].v / 
items[i].w) << endl;
  }
  sort(items, items+sizeOfItems, compare);
  for(i=0; i<sizeOfItems; i++) {
    if(totalWeight + items[i].w<= W) {
      totalValue += items[i].v ;
      totalWeight += items[i].w;
    } else {
      int wt = W-totalWeight;
      totalValue += items[i].v*((double)wt / items[i].w);
      totalWeight += wt;
      break;
    }
  }
  cout << "Total weight in bag " << totalWeight<<endl;
  return totalValue;
}
int main() {
  int W,n;
  cout<<"Enter total number of items :";
  cin>>n;
  Item items[n];
  input(items, n);
  cout << "Entered data \n";
  display(items,n);
  cout<< "Enter Knapsack weight \n";
  cin >> W;
  double mxVal = knapsack(items, n, W);
  cout << "Max Profit for "<< W <<" weight is : "<< mxVal;
  return 0;
```

}

OUTPUT:



```
Enter total number of items :3
Enter total 3 Item's values and weight
Enter 1 V 1
Enter 1 W 2
Enter 2 V 2
Enter 2 W 3
Enter 3 V 5
Enter 3 W 4
Entered data
values: 1       2       5
weight: 2       3       4
Enter Knapsack weight
weight: 2       3       4
Enter Knapsack weight
5
Profit per unit of weight :
Value     Weight     Profit
1         2          0.5
2         3          0.666667
5         4          1.25
Total weight in bag 5
Max Profit for 5 weight is : 5.666720
```



```
Enter total number of items :7
Enter 1 V 10
Enter 1 W 2
Enter 2 V 5
Enter 2 W 3
Enter 3 V 15
Enter 3 W 5
Enter 4 V 7
Enter 4 W 7
Enter 5 V 6
Enter 5 W 1
Enter 6 V 10
Enter 6 W 4
Enter 7 V 3
Enter 7 W 1
Entered data
values: 10      5       15      7
weight: 2       3       5       7
Enter Knapsack weight
15
Profit per unit of weight :
Value     Weight    Profit
10         2         5
5         3         1.66667
15         5         3
7         7         1
6         1         6
10         4         2.5
3         1         3
Total weight in bag 15
Max Profit for 15 weight is : 47.333320
```

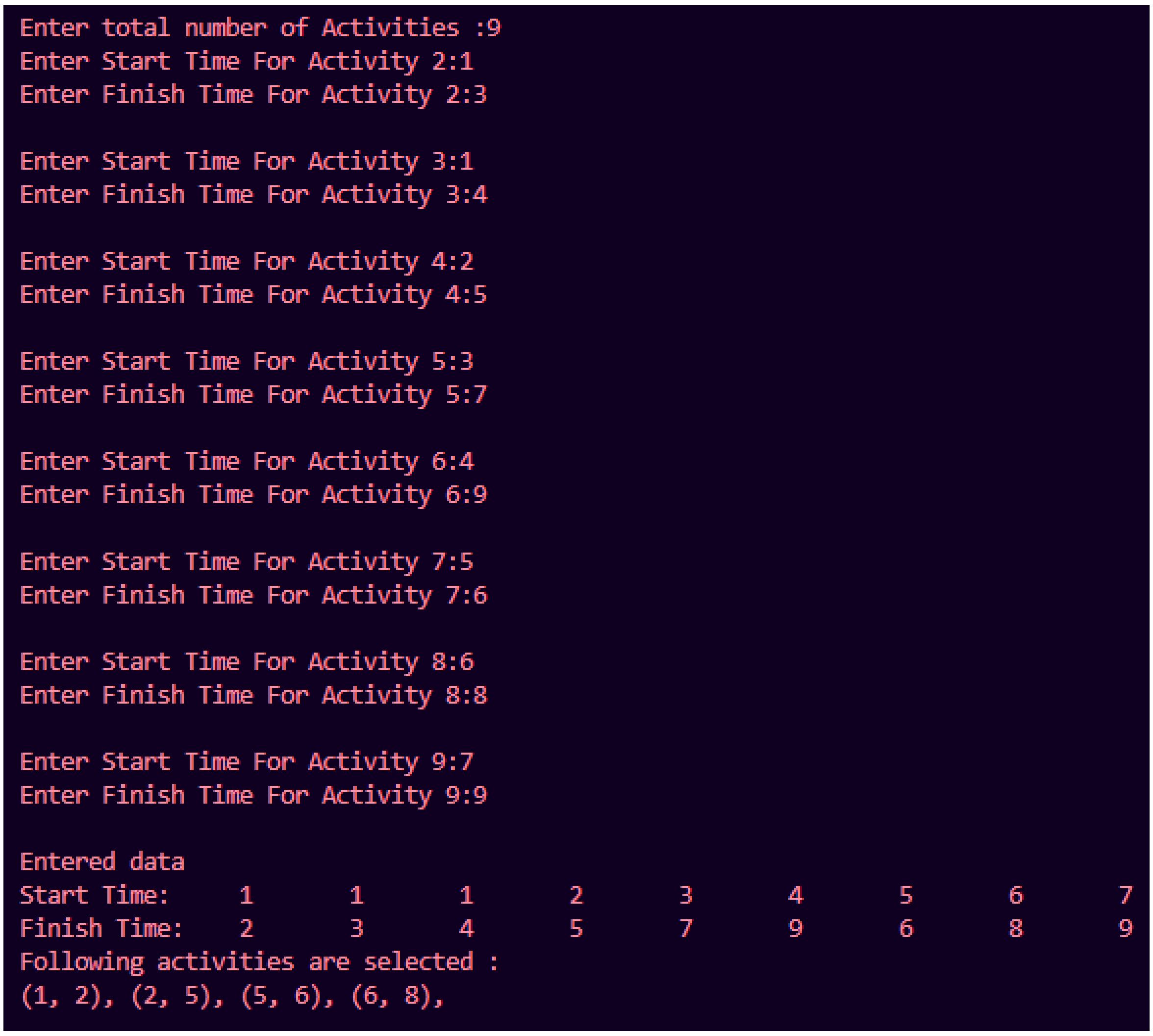
4.3 Suppose you want to schedule N activities in a Seminar Hall. Start time and Finish time of activities are given by pair of (si,fi) for ith activity. Implement the program to maximize the utilization of Seminar Hall. (Maximum activities should be selected.)

Test Case	Number of activities	(si, fi)
1	9	(1,2), 1,3),(1,4),(2,5),(3,7), (4,9), (5,6), (6,8), (7,9)
2	11	(1,4),(3,5),(0,6),(3,8),(5,7), (5,9), (6,10), (8,12),(8,11) (12,14), (2,13)

CODE:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
typedef struct {
  int s;
  int f;
} activ;
void input(activ arr[],int lng) {
  cout << "Enter total "<< lng <<" Item's Start and Finish time :-\n\n";
  for(int i = 0; i < lng; i++) {
    cout << "Enter Start Time For Activity "<< i+1<<":" ;
    cin >> arr[i].s;
    cout << "Enter Finish Time For Activity "<< i+1<<":";
    cin >> arr[i].f;
    cout<<"\n";
  }
}
void display(activ arr[], int lng) {
  int i;
  cout << "Start Time: ";
  for(i = 0; i < lng; i++) {
    cout << "\t"<< arr[i].s ;
  }
  cout << endl << "Finish Time: ";
  for (i = 0; i < lng; i++) {
     cout << "\t"<< arr[i].f ;
  }
  cout << endl;
}
bool compare(activ a, activ b) {
   return a.f < b.f;
}
void MaxAct(activ arr[], int n)
{
   sort(arr, arr+n, compare);

   cout << "Following activities are selected :\n";
   int i = 0;
   cout << "(" << arr[i].s << ", " << arr[i].f << "), ";
   for (int j = 1; j < n; j++)
   {
     if (arr[j].s >= arr[i].f)
     {
         cout << "(" << arr[j].s << ", "
<< arr[j].f << "), ";
         i = j;
     }
   }
}
int main() {
  int n;
  cout<<"Enter total number of Activities :";
  cin>>n;
  activ arr[n];
  input(arr, n);
  cout << "Entered data \n";
  display(arr,n);
  MaxAct(arr, n);
  cout << "Max Profit for "<< W <<" weight is : "<< mxVal;
}
```

OUTPUT:



```
Enter total number of Activities :9
Enter Start Time For Activity 2:1
Enter Finish Time For Activity 2:3

Enter Start Time For Activity 3:1
Enter Finish Time For Activity 3:4

Enter Start Time For Activity 4:2
Enter Finish Time For Activity 4:5

Enter Start Time For Activity 5:3
Enter Finish Time For Activity 5:7

Enter Start Time For Activity 6:4
Enter Finish Time For Activity 6:9

Enter Start Time For Activity 7:5
Enter Finish Time For Activity 7:6

Enter Start Time For Activity 8:6
Enter Finish Time For Activity 8:8

Enter Start Time For Activity 9:7
Enter Finish Time For Activity 9:9

Entered data
Start Time:     1       1       1       2       3       4       5       6       7
Finish Time:    2       3       4       5       7       9       6       8       9
Following activities are selected :
(1, 2), (2, 5), (5, 6), (6, 8),
```

CONCLUSION:

This problem is a variation of 'Coin Change Problem'. Worst case: When the only coin present is Rs. 1 Coin. So in that case, time complexity becomes O(A) where A is the amount to be paid.

In this practical we have tried to code a cpp PROGRAM CODE to solve the fractional knapsack problem. The problem is based on getting maximum profit items in the knapsack having fixed carry capacity and also to utilize maximum (whole, in fractional knapsack) carrying capacity of the knapsack. Here greedy approach is used to solve the problem.

In this practical we have attempted to code a cpp PROGRAM CODE to solve the activity selection problem. The problem is based on getting maximum activities completed such that maximum amount of the schedule can be utilized .In the problem definition we are given to solve this problem for seminar hall. Here greedy approach is used to solve the problem which allows to complete maximum activities so that maximum schedule time of seminar hall can be used.

Practical-5

Aim: Dynamic Programming

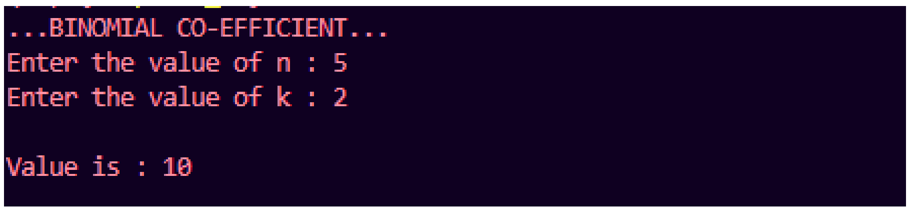
5.1 Implement a program which has BNMCOEF() function that takes two parameters n and k and returns the value of Binomial Coefficient C(n, k). Compare the dynamic programming implementation with recursive implementation of BNMCOEF(). (In output, entire table should be displayed.)

Test Case	n	k
1	5	2
2	11	6
3	12	5

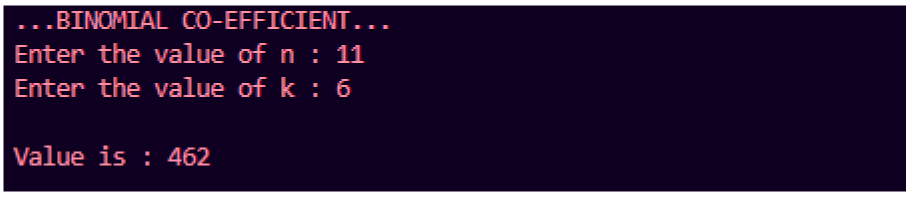
CODE:

```
#include<iostream>

using namespace std;

int binomialCoeff(int n, int k)
{
   if (k == 0 || k == n)
       return 1;

   return binomialCoeff(n - 1, k - 1) +
              binomialCoeff(n - 1, k);
}
int main()
{
   int n,k;
   cout<<"...BINOMIAL CO-EFFICIENT...\n";
   cout<<"Enter the value of n : ";
   cin>>n;
   cout<<"Enter the value of k : ";
   cin>>k;
   cout<<"\nValue is : "<<binomialCoeff(n,k)<<endl;

   return 0;
}
```

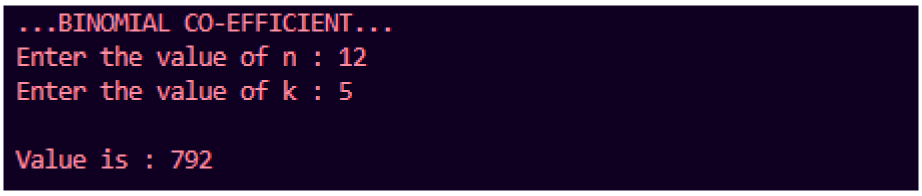
OUTPUT:



```
...BINOMIAL CO-EFFICIENT...
Enter the value of n : 5
Enter the value of k : 2

Value is : 10
```



```
...BINOMIAL CO-EFFICIENT...
Enter the value of n : 11
Enter the value of k : 6

Value is : 462
```



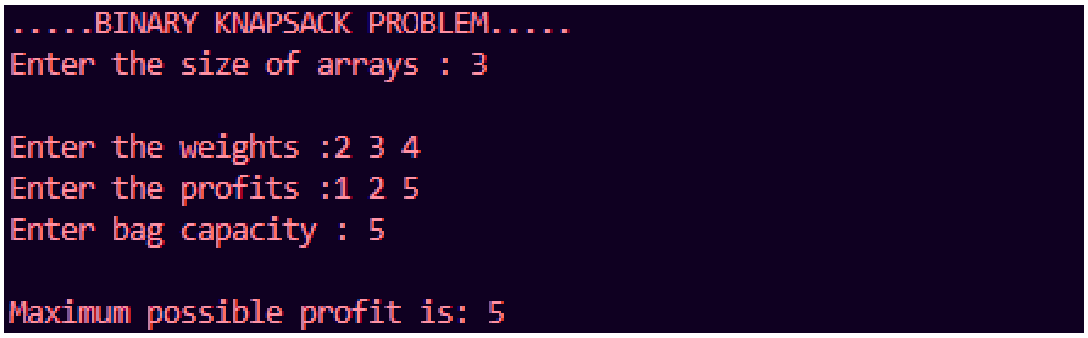
```
...BINOMIAL CO-EFFICIENT...
Enter the value of n : 12
Enter the value of k : 5

Value is : 792
```

5.2 Implement the program 4.2 using Dynamic Programing. Compare Greedy and Dynamic approach.

CODE:

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

int maximum(int x,int y)
{
   if(x>y)
      return x;
   else
      return y;
}

int knapsack(int bag_capacity,int weight[],int profit[],int number)
{
   int matrix[number+1][bag_capacity+1];

   for(int i=0;i<number+1;i++)
      for(int j=0;j<bag_capacity+1;j++)
      {
            if(i==0 || j==0)
               matrix[i][j]=0;
            else if (j>=weight[i-1])
               matrix[i][j]=maximum(matrix[i-1][j],profit[i-1]+matrix[i-1][j-weight[i-1]]);
            else
               matrix[i][j]=matrix[i-1][j];
      }

   return matrix[number][bag_capacity];
}

int main()
{
   int number,bag_capacity;
   cout<<".....BINARY KNAPSACK PROBLEM.....";
   cout<<"\nEnter the size of arrays : ";
   cin>>number;

   int weight[number],profit[number];

   cout<<"\nEnter the weights :";
   for(int i=0;i<number;i++)
      cin>>weight[i];

   cout<<"Enter the profits :";
   for(int i=0;i<number;i++)
      cin>>profit[i];

   cout<<"Enter bag capacity : ";
   cin>>bag_capacity;

   cout<<"\nMaximum possible profit is: " << knapsack(bag_capacity,weight,profit,number)
      <<endl;
   return 0;
}
```

OUTPUT:



```
.....BINARY KNAPSACK PROBLEM.....
Enter the size of arrays : 3

Enter the weights :2 3 4
Enter the profits :1 2 5
Enter bag capacity : 5

Maximum possible profit is: 5
```

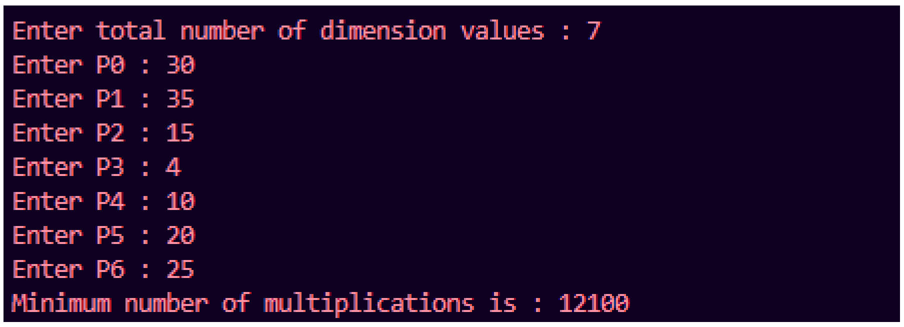
5.3 Given a chain < A1, A2,...,An> of n matrices, where for i=1,2,...,n matrix Ai with dimensions. Implement the program to fully parenthesize the product A1,A2,...,An in a way that minimizes the number of scalar multiplications. Also calculate the number of scalar
multiplications for all possible combinations of matrices.

Test Case	n	Matrices with dimensions
1	3	A1:3*5,A2:5*6,A3:6*4
2	6	A1: 30*35, A2: 35*15, A3: 15*5, A4: 5*10, A5: 10*20, A6: 20*25

CODE:

```
#include<bits/stdc++.h>
using namespace std;

int MatrixMultiplication(int p[], int n)
{
        int m[n][n];
        int i, j, k, L, q;

        for (i = 1; i < n; i++)
                m[i][i] = 0;

        // L is chain length.
        for (L = 2; L < n; L++)
        {
                for (i = 1; i < n - L + 1; i++)
                {
                        j = i + L - 1;
                        m[i][j] = INT_MAX;
                        for (k = i; k <= j - 1; k++)
                        {
                                q = m[i][k] + m[k + 1][j] +
                                        p[i - 1] * p[k] * p[j];
                                if (q < m[i][j])
                                        m[i][j] = q;
                        }
                }
        }
        return m[1][n - 1];
}

int main()
{
   int n;
   cout<<"...MATRIX CHAIN MULTIPLICATION...\n\n";
   cout<<"Enter total number of dimension values : ";
        cin>>n;
        int arr[n];
   for(int i=0;i<n;i++)
       {
          cout<<"Enter P"<<i<<" : ";
          cin>>arr[i];
       }
          int length=sizeof(arr)/sizeof(arr[0]);
          cout << "Minimum number of multiplications is : "<<MatrixMultiplication(arr,length)<<endl;

   return 0;
}
```

OUTPUT:



```
Enter total number of dimension values : 7
Enter P0 : 30
Enter P1 : 35
Enter P2 : 15
Enter P3 : 4
Enter P4 : 10
Enter P5 : 20
Enter P6 : 25
Minimum number of multiplications is : 12100
```

5.4 Implement a PROGRAM CODE to print the longest common subsequence for the following strings:
Check the PROGRAM CODE for following test cases:

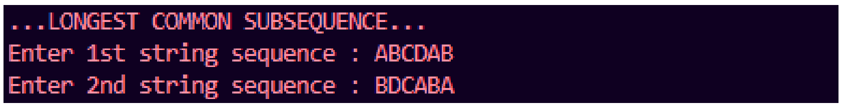
Test Case	String1	String2
1	ABCDAB	BDCABA
2	EXPONENTIAL	POLYNOMIAL
3	LOGARITHM	ALGORITHM

CODE:

```
#include<iostream>
#include<string.h>
using namespace std;
int max(int a, int b);
int lcs( char *X, char *Y, int m, int n )
{       if (m == 0 || n == 0)
                return 0;
        if (X[m-1] == Y[n-1])
                return 1 + lcs(X, Y, m-1, n-1);
        else
                return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));} int max(int a, int b)
{
        return (a > b)? a : b;
}
int main()
{       cout<<"...LONGEST COMMON SUBSEQUENCE..."<<endl;
```

```
        char X[100],Y[100];
        cout<<"Enter 1st string sequence : ";cin>>X;
        cout<<"Enter 2nd string sequence : ";cin>>Y;
        int m = strlen(X);int n = strlen(Y);
        cout<<"Length of LCS is : "<<lcs(X,Y,m,n);

        return 0;
}
```

OUTPUT:



```
...LONGEST COMMON SUBSEQUENCE...
Enter 1st string sequence : ABCDAB
Enter 2nd string sequence : BDCABA
```

CONCLUSION

Time Complexity of the above implementation is O(mn) where m and n is length of string1 and string2.And, it is observed to be much better than the worst-case time complexity of Naive Recursive implementation.

Practical-6

Aim: Graph

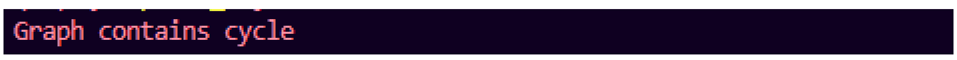
6.1 Write a program to detect cycles in a directed graph.

CODE:

```
#include <iostream>
#include <list>
#include <limits.h>
using namespace std;
class Graph
{
   int V;
   list<int> *adj;
   bool isCyclicUtil(int v, bool visited[], bool *rs);

public:
   Graph(int V);
   void addEdge(int v, int w);
   bool isCyclic();
};
Graph::Graph(int V)
{
   this->V = V;
   adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
   adj[v].push_back(w);
}
bool Graph::isCyclicUtil(int v, bool visited[], bool *recStack)
{
   if (visited[v] == false)
   {
      visited[v] = true;
      recStack[v] = true;
      list<int>::iterator i;
      for (i = adj[v].begin(); i != adj[v].end(); ++i)
      {
         if (!visited[*i] && isCyclicUtil(*i, visited, recStack))
            return true;
         else if (recStack[*i])
            return true;
      }
   }
   recStack[v] = false;
   return false;
}
```

```
bool Graph::isCyclic()
{
    bool *visited = new bool[V];
    bool *recStack = new bool[V];
    for (int i = 0; i < V; i++)
    {
        visited[i] = false;
        recStack[i] = false;
    }
    for (int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;
    return false;
}
int main()
{
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
    if (g.isCyclic())
        cout << "Graph contains cycle";
    else
        cout << "Graph doesn't contain cycle";
    return 0;
}
```

OUTPUT:



```
Graph contains cycle
```

6.2 From a given vertex in a weighted graph, implement a program to find shortest paths to other vertices using Dijkstra's algorithm.

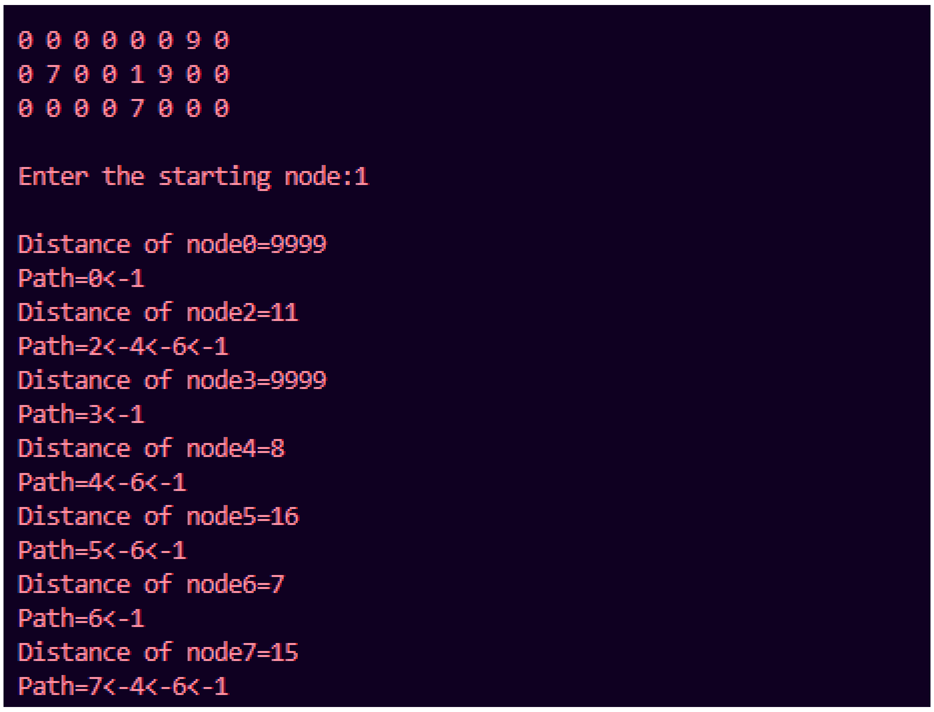
Test Case	Adjacency Matrix of graph	Start Vertex
1	$\begin{array}{c\|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & & & & 2 & & & & \\ 1 & & & & & & & 7 & \\ 2 & & & & & 3 & & & \\ 3 & 2 & & & & & & & \\ 4 & & & 3 & & & & 1 & 7 \\ 5 & & & & & & & 9 & \\ 6 & & 7 & & & 1 & 9 & & \\ 7 & & & & & 7 & & & \end{array}$	1
2	$\begin{array}{c\|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & & & 2 & & & & & \\ 1 & 6 & & & & & & & \\ 2 & 3 & 8 & & & 5 & & & \\ 3 & & 9 & & & & & & \\ 4 & & & & & & & 1 & \\ 5 & & & & 7 & & & & \\ 6 & & & 9 & & 4 & & & 3 \\ 7 & & & & & & 1 & 6 & \end{array}$	3

CODE:

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX], int n, int startnode);

int main()
{
   int G[MAX][MAX], i, j, n, u;
   printf("Enter no. of vertices:");
   scanf("%d", &n);
   printf("\nEnter the adjacency matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         scanf("%d", &G[i][j]);
   printf("\nEnter the starting node:");
   scanf("%d", &u);
   dijkstra(G, n, u);
   return 0;
}
void dijkstra(int G[MAX][MAX], int n, int startnode)
{
   int cost[MAX][MAX], distance[MAX], pred[MAX];
   int visited[MAX], count, mindistance, nextnode, i, j;
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         if (G[i][j] == 0)
```

```
                cost[i][j] = INFINITY;
            else
                cost[i][j] = G[i][j];
    for (i = 0; i < n; i++)
    {
        distance[i] = cost[startnode][i];
        pred[i] = startnode;
        visited[i] = 0;
    }
    distance[startnode] = 0;
    visited[startnode] = 1;
    count = 1;
    while (count < n - 1)
    {
        mindistance = INFINITY;
        for (i = 0; i < n; i++)
            if (distance[i] < mindistance && !visited[i])
            {
                mindistance = distance[i];
                nextnode = i;
            }
        visited[nextnode] = 1;
        for (i = 0; i < n; i++)
            if (!visited[i])
                if (mindistance + cost[nextnode][i] < distance[i])
                {
                    distance[i] = mindistance + cost[nextnode][i];
                    pred[i] = nextnode;
                }
        count++;
    }
    for (i = 0; i < n; i++)
        if (i != startnode)
        {
            printf("\nDistance of node%d=%d", i, distance[i]);
            printf("\nPath=%d", i);
            j = i;
            do
            {
                j = pred[j];
                printf("<-%d", j);
            } while (j != startnode);
        }
}
```

OUTPUT:



```
0 0 0 0 0 0 9 0
0 7 0 0 1 9 0 0
0 0 0 0 7 0 0 0

Enter the starting node:1

Distance of node0=9999
Path=0<-1
Distance of node2=11
Path=2<-4<-6<-1
Distance of node3=9999
Path=3<-1
Distance of node4=8
Path=4<-6<-1
Distance of node5=16
Path=5<-6<-1
Distance of node6=7
Path=6<-1
Distance of node7=15
Path=7<-4<-6<-1
```

CONCLUSION:

Depth First Traversal can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. Time Complexity: O(V + E)

Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.Time Complexity: O(ElogV)

Practical-7

Aim: Backtracking

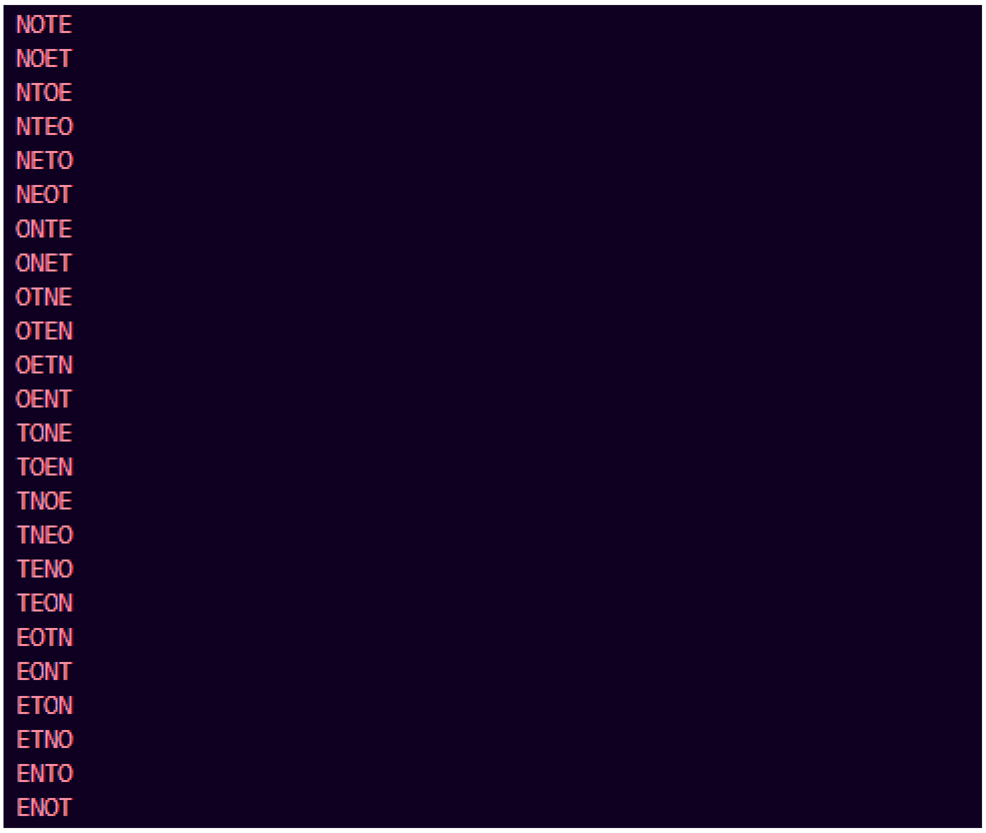
7.1 Implement a program to print all permutations of a given string.

Test Case	String
1	ACT
2	NOTE

CODE:

```
#include <stdio.h>
#include <string.h>
void swap(char *x, char *y)
{
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
void permute(char *a, int l, int r)
{
    int i;
    if (l == r)
        printf("%s\n", a);
    else
    {
        for (i = l; i <= r; i++)
        {
            swap((a + l), (a + i));
            permute(a, l + 1, r);
            swap((a + l), (a + i)); // backtrack
        }
    }
}
int main()
{
    char str[] = "NOTE"; // ACT
    int n = strlen(str);
    permute(str, 0, n - 1);
    return 0;
}
```

OUTPUT:



```
NOTE
NOET
NTOE
NTEO
NETO
NEOT
ONTE
ONET
OTNE
OTEN
OETN
OENT
TONE
TOEN
TNOE
TNEO
TENO
TEON
EOTN
EONT
ETON
ETNO
ENTO
ENOT
```

Practical-8

Aim: String Matching Algorithm

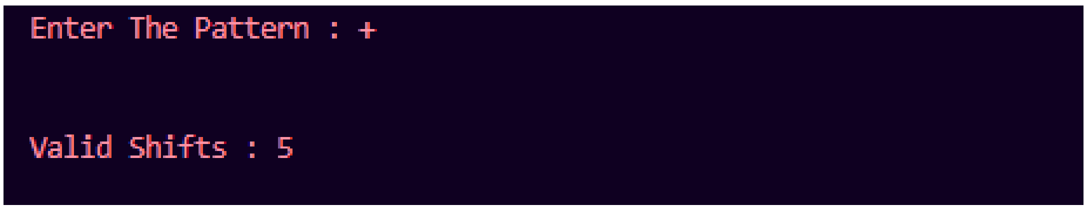
**8.1 Suppose you are given a source string S [0 ..n − 1] of length n, consisting of symbols a and b. Suppose that you are given a pattern string P[0 ..m − 1] of length m < n, consisting of symbols a, b, and *, representing a pattern to be found in string S. The symbol * is a "wild
card" symbol, which matches a single symbol, either a or b. The other symbols must match exactly. The problem is to output a sorted list M of valid "match positions", which are positions j in S such that pattern P matches the substring S [j..j + |P|− 1]. For example, if S = ababbab and P = ab*, then the output M should be [0, 2]. Implement a straightforward, naive algorithm to solve the problem.**

CODE:

```
#include <stdio.h>
#include <string.h>
int main()
{
   char t[100], p[100];
   int tn, pn, shift[20] = {0}, s = 0, i, j = 0, count = 0, m = 0;
   printf("\n Enter The Text : ");
   scanf("%s", t);
   fflush(stdin);
   printf("\n Enter The Pattern : ");
   scanf("%s", p);
   tn = strlen(t);
   pn = strlen(p);
   while (s != (tn - pn + 1))
   {
      j = 0;
      for (i = s; i < pn + s; i++)
      {
         if (p[j] == t[i])
         {
            count++;
            if (count == pn)
            {
               count = 0;
               shift[m] = s;
               m++;
            }
         }
         else
         {
            count = 0;
            break;
         }
```

```
            j++;
        }
        s++;
    }
    if (m > 0)
    {
        printf("\n\n Valid Shifts : ");
        for (i = 0; i < m; i++)
            printf("%d \n", shift[i]);
    }
    else
    {
        printf("\n\n No Valid Shifts.");
    }
}
```

OUTPUT:



```
Enter The Pattern : +


Valid Shifts : 5
```

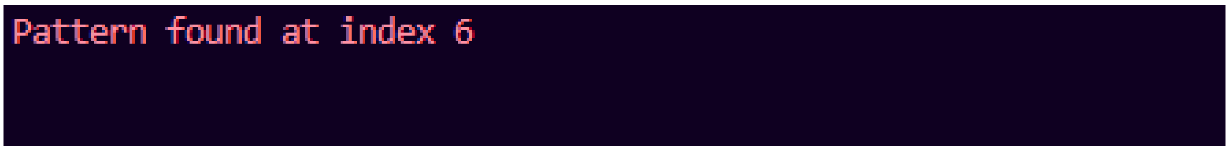
8.2 Implement Rabin karp algorithm and test it on the following test cases:

Test Case	String	Pattern
1	2359023141526739921	31415 q=13
2	ABAAABCDBBABCDDEBCABC	ABC q=101

CODE:

```
#include <stdio.h>
#include <string.h>
#define d 256
using namespace std;

void search(char pat[], char txt[], int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0; // hash value for pattern
    int t = 0; // hash value for txt
    int h = 1;

    for (i = 0; i < M - 1; i++)
        h = (h * d) % q;

    for (i = 0; i < M; i++)
    {
        p = (d * p + pat[i]) % q;
        t = (d * t + txt[i]) % q;
    }
    for (i = 0; i <= N - M; i++)
    {
        if (p == t)
        {
            for (j = 0; j < M; j++)
            {
                if (txt[i + j] != pat[j])
                    break;
            }
            if (j == M)
                printf("Pattern found at index %d \n", i);
        }
        if (i < N - M)
        {
            t = (d * (t - txt[i] * h) + txt[i + M]) % q;

            if (t < 0)
                t = (t + q);
        }
    }
}

int main()
{
    char txt[] = "2359023141526739921"; // ABAAABCDBBABCDDEBCABC
    char pat[] = "31415";             // ABC
    int q = 101;

    search(pat, txt, q);

    return 0;
}
```

OUTPUT:



```
Pattern found at index 6
```

CONCLUSION:

- Naive pattern searching is the simplest method among other pattern searching algorithms.
- It checks for all character of the main string to the pattern.
- This algorithm is helpful for smaller texts. It does not need any pre-processing phases.
- The time complexity is O(m*n). The m is the size of pattern and n is the size of the main string.
- The Rabin–Karp algorithm is a form of rolling hash used in string searching [1] to find any one of a set of pattern strings in a text.
- For text of length n and p patterns of combined length m, its average and best case time complexity is O(n+m) , but its worst-case time complexity is O(nm).