U19CS076 DAA ASSIGNMENT-6

KRITHIKHA BALAMURUGAN

• Write programs to solve each of the following problems

1. Longest palindrome subsequence problem

Problem Statement:

Give an efficient algorithm to find the length of longest palindrome that is a subsequence of a given input string.

A palindrome is a nonempty string over some alphabet that reads the same forward and backward. E.g.: civic, racecar, and aibohphobia.

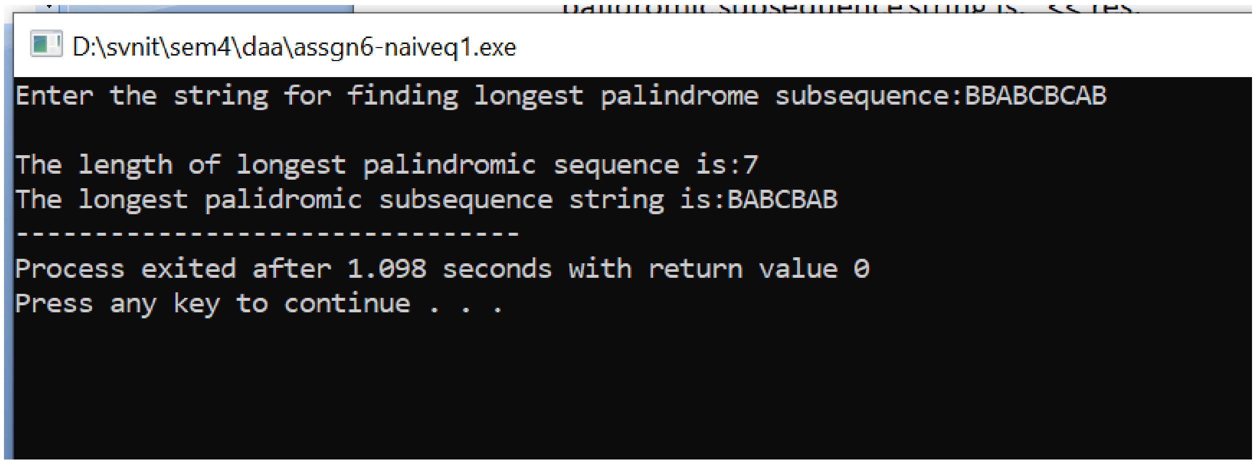
A subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements

If the given sequence is “BBABCBCAB”, then the output should be 7 as “BABCBAB” is the longest palindrome subsequence in it. “BBBBB” and “BBCBB” are also palindrome subsequences of the given sequence, but not the longest ones.

– Naive iterative solution

```
#include <bits/stdc++.h>

using namespace std;

bool ispalindrome(string s) //to check if it is a palindrome or not

{

 int n = s.length();

 for (int i = 0; i < n/2; i++)

 {

        if (s[i]!=s[n-i-1])

        {

                return false;
```

```
            }
  }
  return true;

 }
 int main()
 {
  string s;
  cout<<"Enter the string for finding longest palindrome subsequence:";
  cin >> s;
  int n = s.length();
  int l=0;
  string res;
  for (int i = 0; i < 1<<n; i++)
  {
            string sub = "";
            for (int j = 0; j < n; j++) //to find subsequence
            {
                        if (i & 1<<j)
                        {
                                    sub = sub+s[j];
                        }
            }
            if(ispalindrome(sub) && sub.length()>l)
            {
                        res = sub;
                        l = sub.length();
            }
  }
  cout <<"\nThe length of longest palindromic sequence is:" <<res.length() << "\nThe longest palidromic subsequence string is:" << res;
  return 0;
 }
```



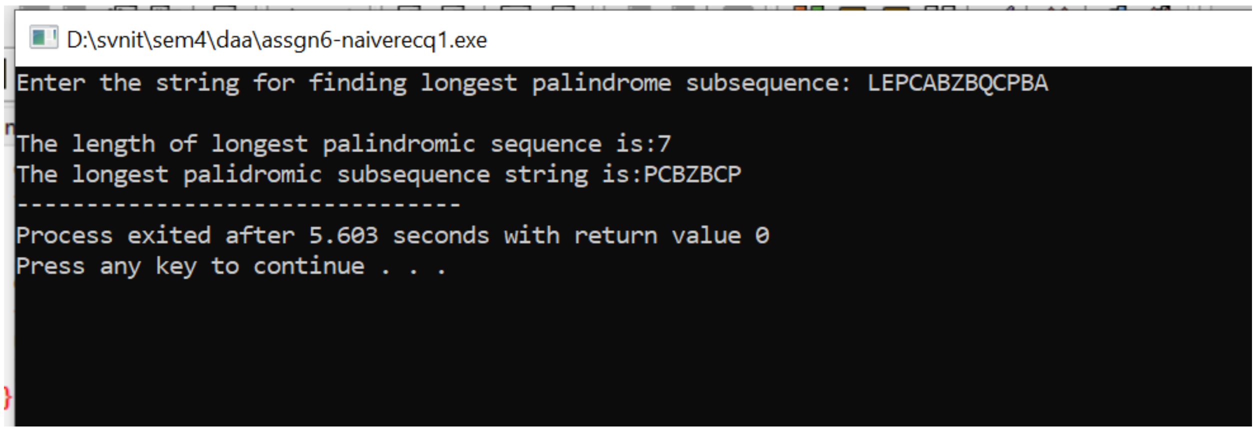
D:\svnit\sem4\daa\assgn6-naiveq1.exe

```
Enter the string for finding longest palindrome subsequence:BBABCBCAB

The length of longest palindromic sequence is:7
The longest palidromic subsequence string is:BABCBAB
--------------------------------
Process exited after 1.098 seconds with return value 0
Press any key to continue . . .
```

Time complexity - $O(2^n)$.

– Naive recursive solution

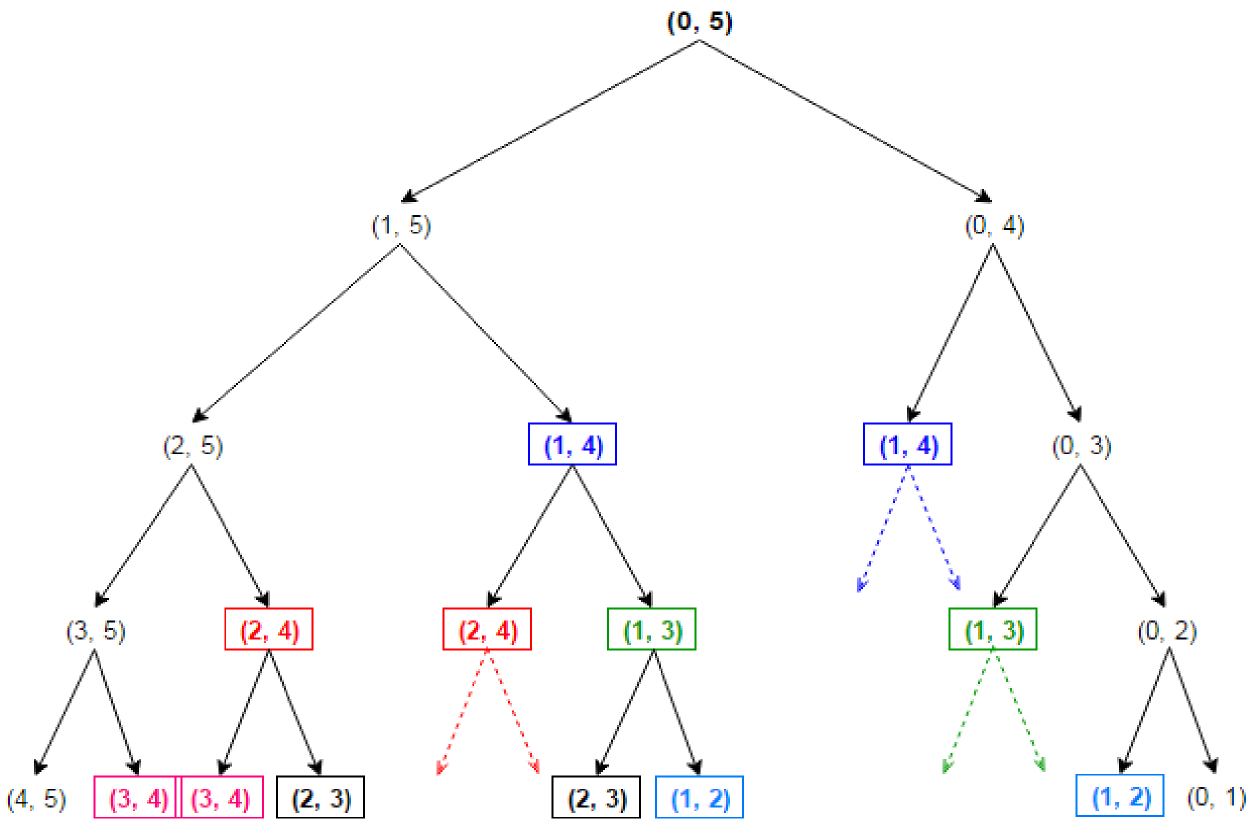
```
#include<bits/stdc++.h>
using namespace std;
string lps(string str,int l,int h)//etr 0 2 12=tr 01=et
{
            if(l>h) //if no charcters return empty string.
            {
                return string("");
            }
            else if(l==h) //if single character return that element only
            {
                string temp="";
                return temp+str[l];
            }
            else if(str[l]==str[h]) //if same elements in string and its reverse
            {
                if(h-l==1) //if only 2 elements
                {
                    string temp="";
                    return temp+str[l]+str[h]; //return both the characters
                }
                else
                {
                    return (str[l]+lps(str,l+1,h-1)+str[h]); //return start character + recurse +
last character
                }
            }
            else if(lps(str,l+1,h).length()>=lps(str,l,h-1).length()) //else return minimum .
            {
                return lps(str,l+1,h);
            }
            else
            {
                        return lps(str,l,h-1);
            }
}


int main()
{
 int n;
 string s;
 cout<<"Enter the string for finding longest palindrome subsequence:";
 cin >> s;
 n = s.length();
 string res=lps(s,0,n-1);
cout <<"\nThe length of longest palindromic sequence is:" <<res.length() << "\nThe longest palidromic subsequence string is:" << res;
 return 0;
}
```



D:\svnit\sem4\daa\assgn6-naiverecq1.exe

```
Enter the string for finding longest palindrome subsequence: LEPCABZBQCPBA

The length of longest palindromic sequence is:7
The longest palidromic subsequence string is:PCBZBCP
--------------------------------
Process exited after 5.603 seconds with return value 0
Press any key to continue . . .
```

Time complexity - O(2^n).



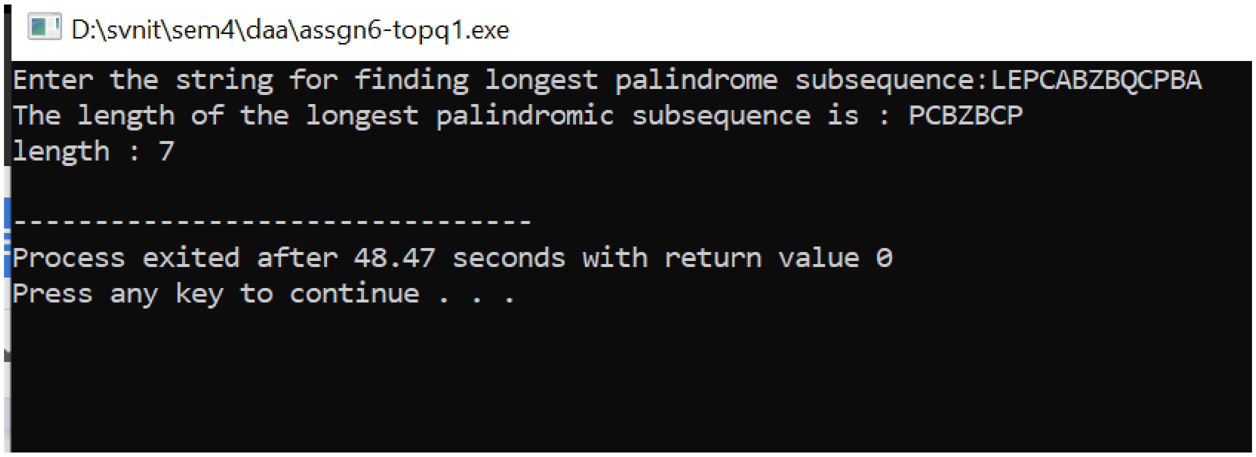
– Top-down dynamic programming solution

Algorithm 1: Solve function that finds the longest palindromic subsequence

```
Data: dp: 2D array that stores the answer to states
      sequence: The sequence to calculate the answer for
      L: The left index of current range
      R: The right index of the current range
Result: Returns the longest palindromic subsequence
if $L > R$ then
    return 0;
else if $L = R$ then
    return 1;
else if $dp[L][R] \neq -1$ then
    return dp[L][R];
else if sequence[L] = sequence[R] then
    $dp[L][R] \leftarrow 2 + solve(dp, sequence, L + 1, R - 1)$;
else
    $moveL \leftarrow solve(dp, sequence, L + 1, R)$;
    $moveR \leftarrow solve(dp, sequence, L, R - 1)$;
    $dp[L][R] \leftarrow maximum(moveL, moveR)$;
return dp[L][R];
```

```
#include<bits/stdc++.h>

using namespace std;

int i,j;

const int maxn=1000;

int dp[1000][1000];

string lps(string str1,string str2,int l,int r)

{

 if(l==0||r==0) //return empty string if size 0

 {

		return string("");

 }

 else if(str1[l-1]==str2[r-1]) //if the elements of string and its reverse are same

 {

		return lps(str1,str2,l-1,r-1)+str1[l-1]; //then use recursive of size -1 for both

 } //plus the character which are equal

 else if(dp[l-1][r]>dp[l][r-1]) //else recurse for minimum of them.

 {

		return lps(str1,str2,l-1,r);

 }

 return lps(str1,str2,l,r-1);

}

int lcs(string str1,string str2,int l,int r)

{

 if(l==0||r==0) //size is 0 then return 0 and assign 0
 {
            dp[l][r]=0;
            return dp[l][r];
 }
 if(dp[l][r]!=-1) //if value in the buffer then return it directly
 {
            return dp[l][r];
 }
 if(str1[l-1]==str2[r-1]) //if elements are same then assign value to dp[][]
 {
            dp[l][r]=lcs(str1,str2,l-1,r-1)+1;
            return dp[l][r];
 }
 else
 {
            dp[l][r]=max(lcs(str1,str2,l-1,r),lcs(str1,str2,l,r-1)); //dp would be minimum of both then
return
            return dp[l][r];
 }
}
int main()
{
 int n;
 string s;
```

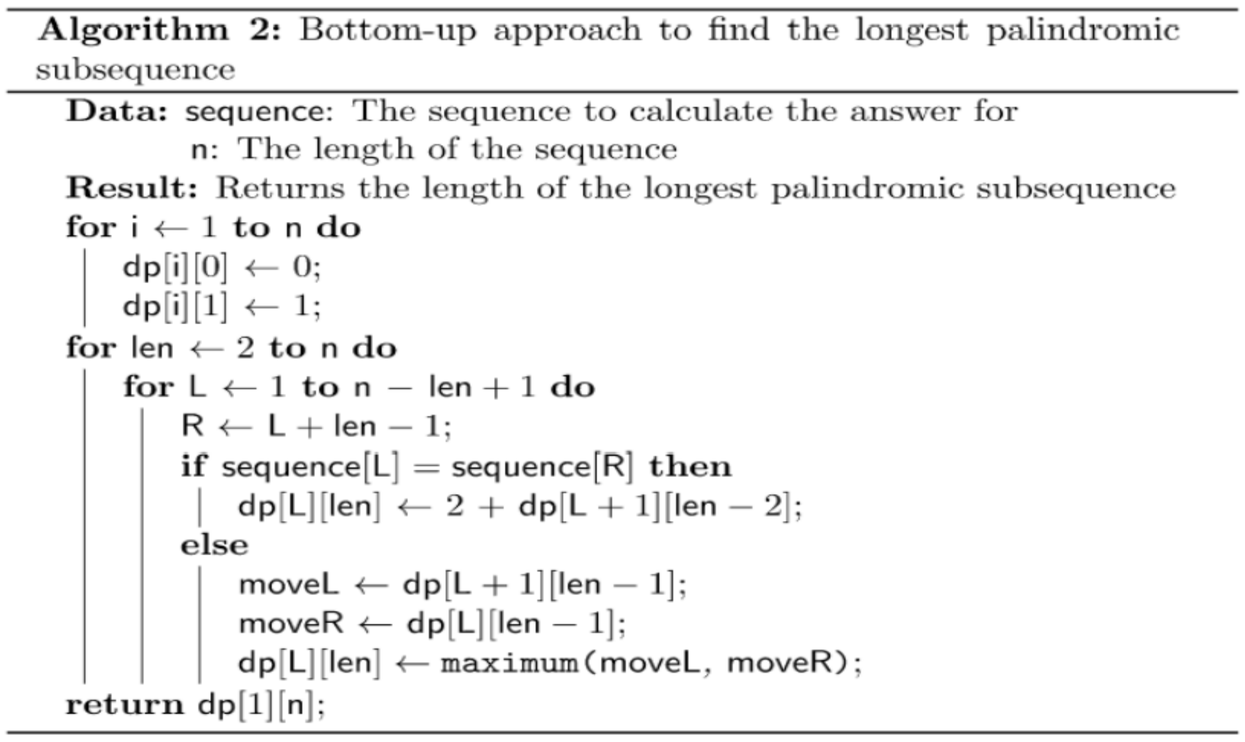
```
cout<<"Enter the string for finding longest palindrome subsequence:";
cin >> s;
n = s.length();
for(i=0;i<=n;i++)
{
        for(j=0;j<=n;j++)
        {
                dp[i][j]=-1;
        }
}
string rstr=s;
reverse(rstr.begin(),rstr.end());
int len=lcs(s,rstr,n,n);
cout<<"The length of the longest palindromic subsequence is : "<<lps(s,rstr,n,n)<<"\nlength : "<<len<<endl;
return 0;
}
```



```
D:\svnit\sem4\daa\assgn6-topq1.exe
Enter the string for finding longest palindrome subsequence:LEPCABZBQCPBA
The length of the longest palindromic subsequence is : PCBZBCP
length : 7

--------------------------------
Process exited after 48.47 seconds with return value 0
Press any key to continue . . .
```

Time complexity- O(n^2)

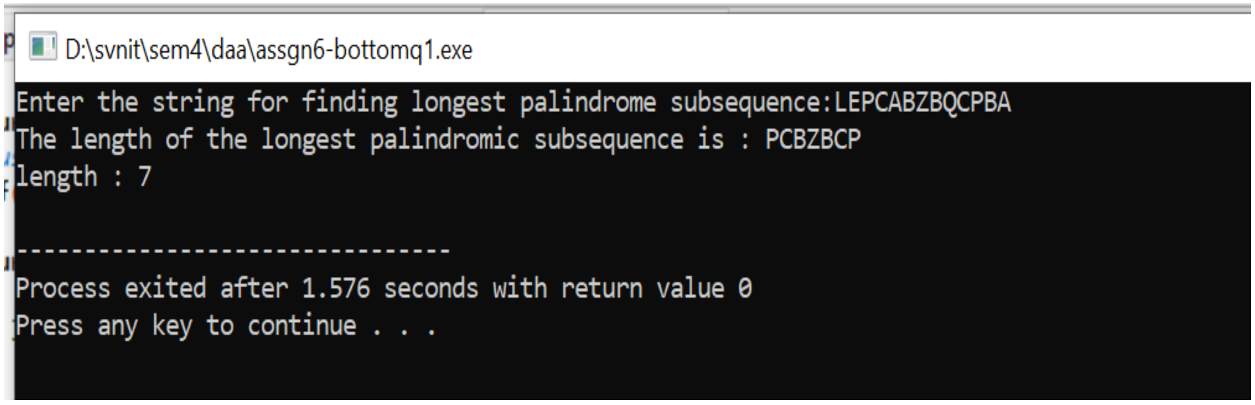
– Bottom-up dynamic programming solution



Algorithm 2: Bottom-up approach to find the longest palindromic subsequence

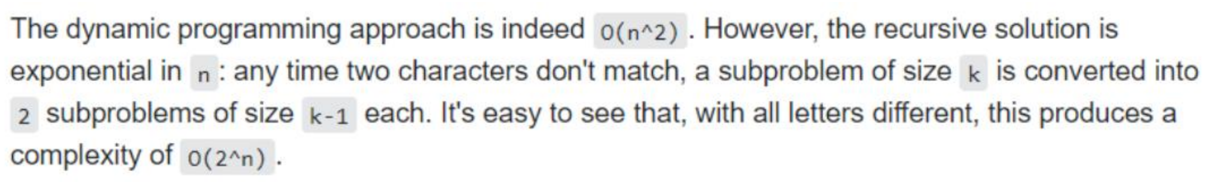
```
Data: sequence: The sequence to calculate the answer for
      n: The length of the sequence
Result: Returns the length of the longest palindromic subsequence
for i ← 1 to n do
    dp[i][0] ← 0;
    dp[i][1] ← 1;
for len ← 2 to n do
    for L ← 1 to n − len + 1 do
        R ← L + len − 1;
        if sequence[L] = sequence[R] then
            dp[L][len] ← 2 + dp[L + 1][len − 2];
        else
            moveL ← dp[L + 1][len − 1];
            moveR ← dp[L][len − 1];
            dp[L][len] ← maximum(moveL, moveR);
return dp[1][n];
```

```
#include<bits/stdc++.h>

using namespace std;

int dp[1000][1000]={0},i,j;

string lps(string str1,string str2,int l,int r)

{

 if(l==0||r==0) //if size is 0 then return empty string

 {

        return string("");

 }

 else if(str1[l-1]==str2[r-1]) //if the elements of string and its reverse are same

 {

        return lps(str1,str2,l-1,r-1)+str1[l-1]; //then use recursive of size -1 for both
 } //plus the character which are equal.

 else if(dp[l-1][r]>dp[l][r-1])

 {

            return lps(str1,str2,l-1,r); //else recurse for minimum of them.

 }

 return lps(str1,str2,l,r-1);

}

int lcs(string str1,string str2,int n) //lcs of string with its reverse gives length of lps

{

 for(i=1;i<=n;i++)

 {

            for(j=1;j<=n;j++)

            {

                        if(str1[i-1]==str2[j-1])

                        {

                                    dp[i][j]=dp[i-1][j-1]+1;

                        }

                        else

                        {

                                    dp[i][j]=max(dp[i-1][j],dp[i][j-1]); //checking min between str1[0-i]str2[0-
(j-1)] and str1[0-(i-1)]str2[0-j]

                        }

   }

 }
 return dp[n][n];
}
int main()
{
 int n;
 string s;
 cout<<"Enter the string for finding longest palindrome subsequence:";
 cin >> s;
 n = s.length();
 string rstr=s;
 reverse(rstr.begin(),rstr.end());
 int len=lcs(s,rstr,n);
 cout<<"The length of the longest palindromic subsequence is : "<<lps(s,rstr,n,n)<<"\nlength :
"<<len<<endl;
 return 0;
}
```



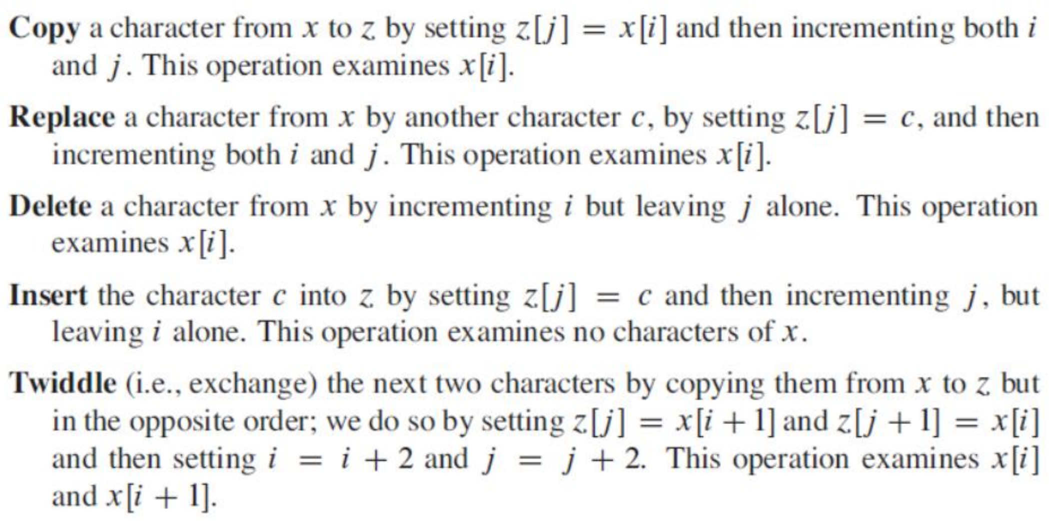
```
D:\svnit\sem4\daa\assgn6-bottomq1.exe
Enter the string for finding longest palindrome subsequence:LEPCABZBQCPBA
The length of the longest palindromic subsequence is : PCBZBCP
length : 7

--------------------------------
Process exited after 1.576 seconds with return value 0
Press any key to continue . . .
```

Time complexity- O(n^2)


The dynamic programming approach is indeed `O(n^2)`. However, the recursive solution is exponential in `n`: any time two characters don't match, a subproblem of size `k` is converted into `2` subproblems of size `k-1` each. It's easy to see that, with all letters different, this produces a complexity of `O(2^n)`.

2. Edit distance problem

Convert String1 to String2 Using Any Set of 5 Operations:


Copy a character from x to z by setting $z[j] = x[i]$ and then incrementing both i and j. This operation examines $x[i]$.

Replace a character from x by another character c, by setting $z[j] = c$, and then incrementing both i and j. This operation examines $x[i]$.

Delete a character from x by incrementing i but leaving j alone. This operation examines $x[i]$.

Insert the character c into z by setting $z[j] = c$ and then incrementing j, but leaving i alone. This operation examines no characters of x.

Twiddle (i.e., exchange) the next two characters by copying them from x to z but in the opposite order; we do so by setting $z[j] = x[i+1]$ and $z[j+1] = x[i]$ and then setting $i = i + 2$ and $j = j + 2$. This operation examines $x[i]$ and $x[i+1]$.

– Naive iterative solution

```
#include<bits/stdc++.h>

using namespace std;

int cost[5]={1,1,1,1,1};

bool check(string s1, string s2, int n, int m, string tmp)

{

        int ptr1 = 0, ptr2 = 0;

        char ch;

        for (int i = 0; i < tmp.length() && ptr1 < n && ptr2 < m; i++)

        {
```

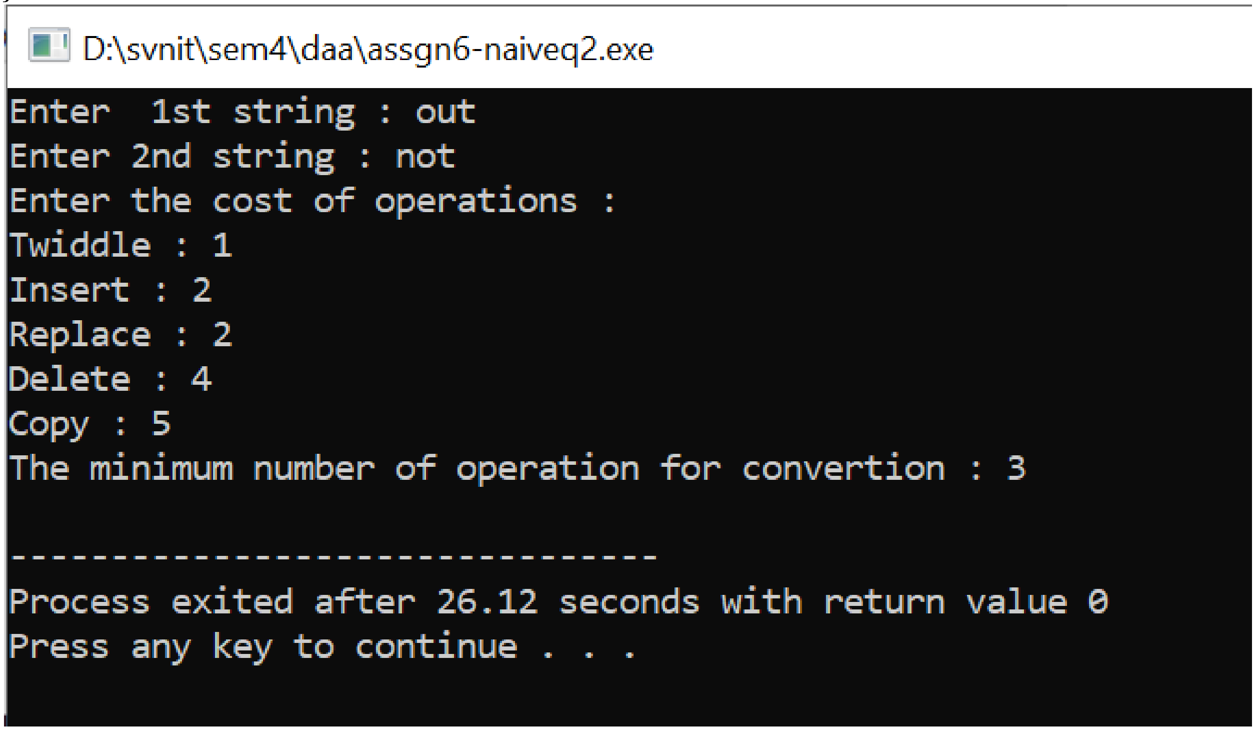
```
ch = tmp[i];

if (ch == 'C')

{

        if (s1[ptr1] == s2[ptr2])

        {

                ptr1++;

                ptr2++;

        }

}

else if (ch == 'R')

{

        ptr1++;

        ptr2++;

}

else if (ch == 'D')

{

        ptr1++;

}

else if (ch == 'I')

{

        ptr2++;

}

else if (ch == 'T')
```

```
{
    if (ptr1 <= n - 2 && ptr2 <= m - 2)
    {
        if (s1[ptr1] == s2[ptr2 + 1] && s1[ptr2] == s2[ptr1 + 1])
        {
            ptr1 += 2;
            ptr2 += 2;
        }
    }
}
else
{
    if (ptr2 == m)
    {
        ptr1 = n;
        break;
    }
    // if (ptr1 == n)
    // {
    // ptr2 = m;
    // break;
    // }
}
```

```
        }
        if (ptr1 == n && ptr2 == m)
                return true;
        else
                return false;
}
// Checks all Possible Operations
int editd(string s1, string s2)
{
int n = s1.length(), m = s2.length();
// Since there are 6 Operations
long int lmt = pow(7, m + n) - 1;
int res = INT_MAX;
// Finding 7-ary Equivalent to Decimal Number
for (int i = 0; i <= lmt; i++)
{
        string tmp;
        int nums = i;
        for (int j = 0; j < max(m,n); j++)
        {
                tmp = to_string(nums % 7) + tmp;
                nums /= 7;
        }
```

```
// cout << tmp << "\n";
bool fl = false;
for (int k = 0; k < tmp.length(); k++)
{
        if (tmp[k] == '0')
        {
                fl = true;
                break;
        }
}
// Skip Number with 0 -> No Operation
if (fl != false)
        continue;
string tmp2;

for (int k = 0; k < tmp.length(); k++)
{
        if (tmp[k] == '1')
                tmp2 += 'C';
        else if (tmp[k] == '2')
                tmp2 += 'R';
        else if (tmp[k] == '3')
                tmp2 += 'D';
        else if (tmp[k] == '4')
                tmp2 += 'I';
        else if (tmp[k] == '5')
                tmp2 += 'T';
        else
                tmp2 += 'K';
}
bool is_possible = check(s1, s2, n, m, tmp2);
if (is_possible)
{
        int cost1 = 0;
        for (int k = 0; k < tmp2.length(); k++)
        {
                if (tmp2[k] == 'C')
                cost1 += cost[4];
                else if (tmp2[k] == 'R')
                cost1 += cost[2];
                else if (tmp2[k] == 'I')
                cost1 += cost[1];
                else if (tmp2[k] == 'T')
                cost1 += cost[0];
                else if (tmp2[k] == 'D')
                cost1 += cost[3];
                }
                if (cost1 < res)
                {
                        // cout << tmp2 << endl;
                        res = cost1;
                }
        }
        }
        return res;
}
int main()
{
 int n,m,i,j;
 string s1,s2;
 cout<<"Enter  1st string : ";
 cin>>s1;
 cout<<"Enter 2nd string : ";
 cin>>s2;
 cout<<"Enter the cost of operations : \n";
 cout<<"Twiddle : ";
 cin>>cost[0];
 cout<<"Insert : ";
 cin>>cost[1];
```

```
cout<<"Replace : ";

cin>>cost[2];

cout<<"Delete : ";

cin>>cost[3];

cout<<"Copy : ";

cin>>cost[4];

cout<<"The minimum number of operation for convertion : "<<editd(s1,s2)<<endl;

return 0;

}
```



```
D:\svnit\sem4\daa\assgn6-naiveq2.exe
Enter  1st string : out
Enter 2nd string : not
Enter the cost of operations :
Twiddle : 1
Insert : 2
Replace : 2
Delete : 4
Copy : 5
The minimum number of operation for convertion : 3

--------------------------------
Process exited after 26.12 seconds with return value 0
Press any key to continue . . .
```

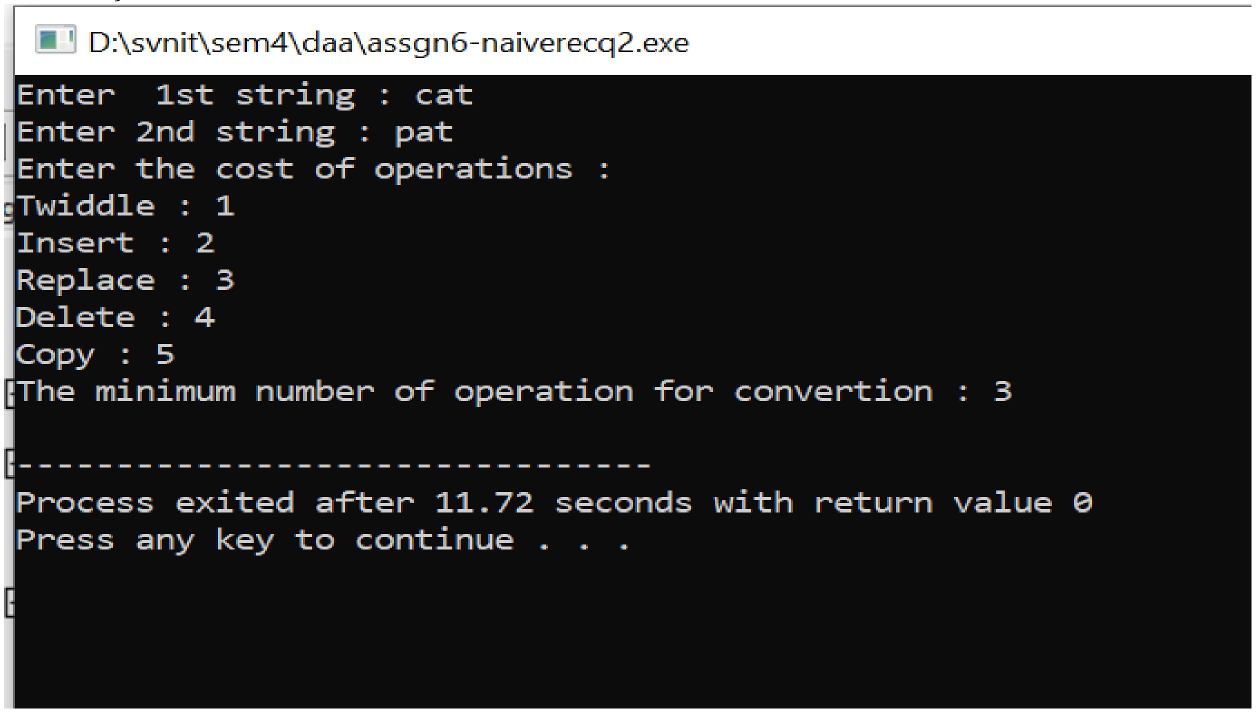
Time Complexity O($5^{(m+n)}$)

– Naive recursive solution

```
#include<bits/stdc++.h>

using namespace std;
```

```
int cost[5]={1,1,1,1,1};
int editd(string str1,string str2,int n,int m)
{
 if(n==0)
 {
        return m;
 }
 if(m==0)
 {
        return n;
 }
 if(str1[n-1]==str2[m-1])
 {
        return editd(str1,str2,n-1,m-1);
 }
 int o1,o2,o3,o4,o0=INT_MAX;
 o1=cost[1]+editd(str1,str2,n,m-1); //insert
 o2=cost[2]+editd(str1,str2,n-1,m-1);//replace
 o3=cost[3]+editd(str1,str2,n-1,m);//delete
 o4=cost[4]+editd(str1,str2,n-1,m-1);//copy
 if(str1[n-2]==str2[m-1]&&str1[n-1]==str2[m-2]&&n>=2&&m>=2)
 {
        o0= cost[0]+editd(str1,str2,n-2,m-2);//twiddle
 }

 return min(o0,min(o1,min(o2,min(o3,o4))));

}

int main()

{

  string s1,s2;

 cout<<"Enter  1st string : ";

 cin>>s1;

 cout<<"Enter 2nd string : ";

 cin>>s2;

 int n=s1.length();

 int m=s2.length();

 cout<<"Enter the cost of operations : \n";

 cout<<"Twiddle : ";

  cin>>cost[0];

 cout<<"Insert : ";

 cin>>cost[1];

 cout<<"Replace : ";

 cin>>cost[2];

 cout<<"Delete : ";

 cin>>cost[3];

 cout<<"Copy : ";

 cin>>cost[4];
```

```
        cout<<"The minimum number of operation for convertion : "<<editd(s1,s2,n,m)<<endl;

        return 0;

       }
```



D:\svnit\sem4\daa\assgn6-naiverecq2.exe

```
Enter  1st string : cat
Enter 2nd string : pat
Enter the cost of operations :
Twiddle : 1
Insert : 2
Replace : 3
Delete : 4
Copy : 5
The minimum number of operation for convertion : 3

--------------------------------
Process exited after 11.72 seconds with return value 0
Press any key to continue . . .
```

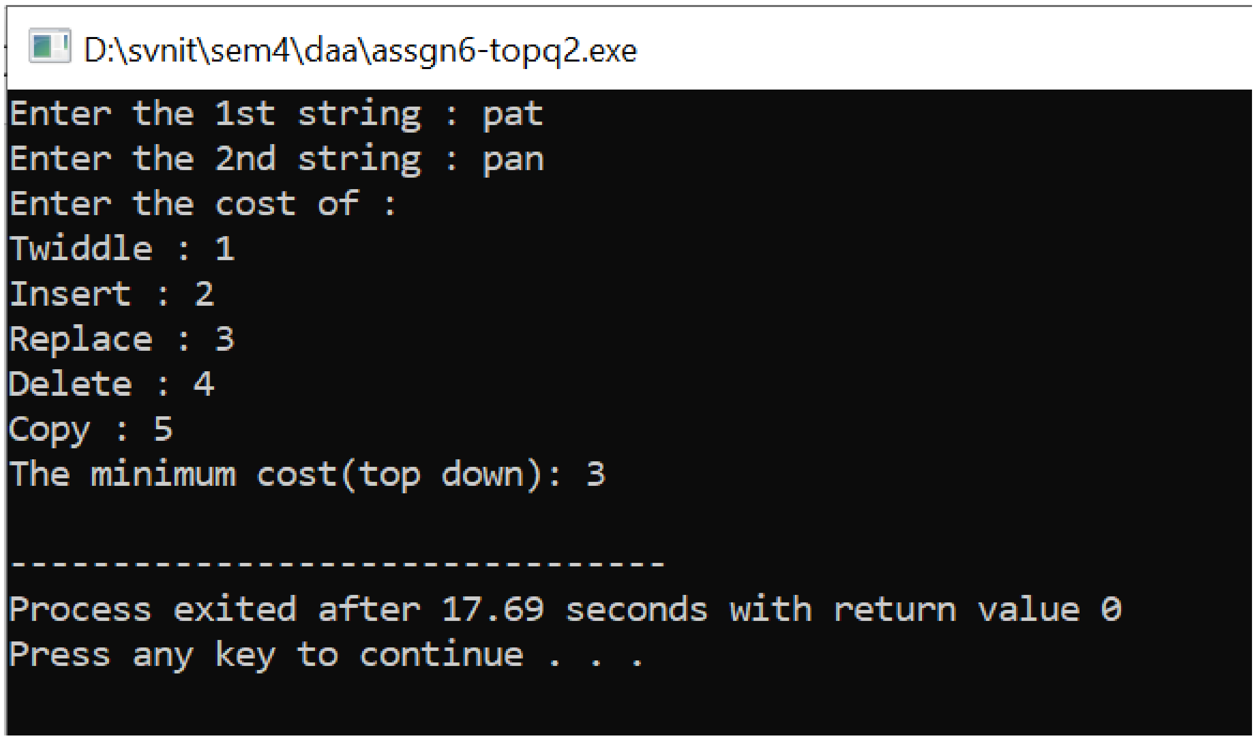
HERE- Replacing c with p thus cost of replace is output

Time Complexity - O(5 $^{(m+n)}$)

– Top-down dynamic programming solution

```
#include<bits/stdc++.h>

using namespace std;

#define N 1000

int dp[N][N], cost[5] = {1,1,1,1,1};

int editdist(string str1, string str2, int n, int m) {


    if (dp[n][m] != -1) {

        return dp[n][m];
    }

    if (n == 0) {

        dp[n][m] = m * cost[1];

    } else if (m == 0) {

        dp[n][m] = n * cost[3];

    } else if (str1[n - 1] == str2[m - 1]) {

        dp[n][m] = editdist(str1, str2, n - 1, m - 1);

    } else {

        int p0 = INT_MAX, p1, p2, p3, p4;

        if (str1[n - 2] == str2[m - 1] && str1[n - 1] == str2[m - 2] && n >= 2 && m >= 2) {

                p0 = cost[0] + editdist(str1, str2, n - 2, m - 2); //twiddle

        }

        p1 = cost[1] + editdist(str1, str2, n, m - 1); //insert

        p2 = cost[2] + editdist(str1, str2, n - 1, m - 1); //replace

        p3 = cost[3] + editdist(str1, str2, n - 1, m); //delete

        p4 = cost[4] + editdist(str1, str2, n - 1, m - 1); //copy

        dp[n][m] = min(p0, min(p1, min(p2, min(p3, p4))));

    }

    return dp[n][m];

}

int main() {

    string s1, s2;

    cout << "Enter the 1st string : ";
```

```
    cin >> s1;

    cout << "Enter the 2nd string : ";

    cin >> s2;

    for (int i = 0; i <= s1.length(); i++) {

        for (int j = 0; j <= s2.length(); j++) {

            dp[i][j] = -1;

        }

    }

    cout << "Enter the cost of : \n";

    cout << "Twiddle : ";

    cin >> cost[0];

    cout << "Insert : ";

    cin >> cost[1];

    cout << "Replace : ";

    cin >> cost[2];

    cout << "Delete : ";

    cin >> cost[3];

    cout << "Copy : ";

    cin >> cost[4];

    cout << "The minimum cost(top down): " << editdist(s1, s2, s1.length(), s2.length()) << endl;

    return 0;

}
```



D:\svnit\sem4\daa\assgn6-topq2.exe

```
Enter the 1st string : pat
Enter the 2nd string : pan
Enter the cost of :
Twiddle : 1
Insert : 2
Replace : 3
Delete : 4
Copy : 5
The minimum cost(top down): 3

--------------------------------
Process exited after 17.69 seconds with return value 0
Press any key to continue . . .
```

Time Complexity- O(n^2)

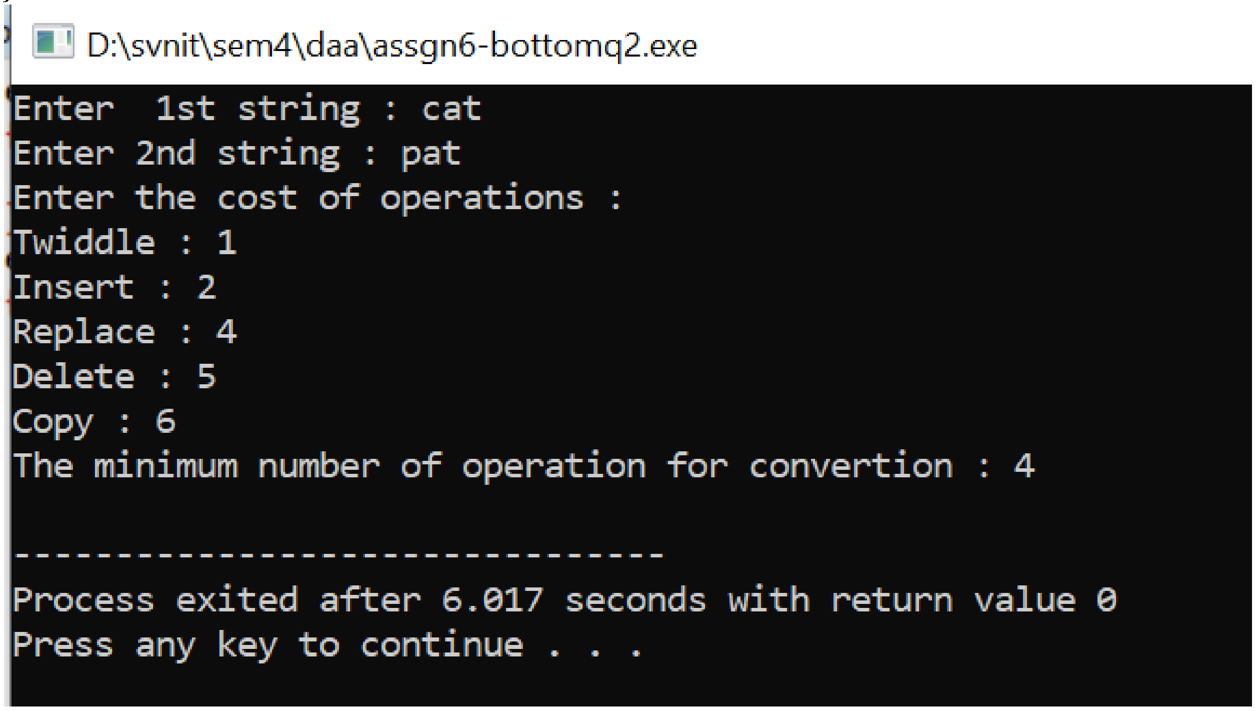
– Bottom-up dynamic programming solution

```
#include<bits/stdc++.h>

using namespace std;

int cost[5]={1,1,1,1,1};


int editd(string str1,string str2,int n,int m)

{

 int i,j;

 int dp[n+1][m+1];


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

{

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

```
{
if(i==0)
{
        dp[i][j]=j*cost[1]; //if the iterator of original string
 //becomes 0 then number remaining elements
 // of target string * cost of insertion
}
else if(j==0)
{
        dp[i][j]=i*cost[3]; //if the iterator of target string
 //becomes 0 then number remaining elements
 // of original string * cost of deletion
}
else if(str1[i-1]==str2[j-1])
{
        dp[i][j]=dp[i-1][j-1];
}
else
{
       int o0=INT_MAX,o1,o2,o3,o4;
       if(str1[i-2]==str2[j-1]&&str1[i-1]==str2[j-2]&&i>=2&&j>=2)
       {
              o0=dp[i-2][j-2]+cost[0];//twiddle
```

```
                    }
                     o1=cost[1]+dp[i][j-1];//insert
                     o2=cost[2]+dp[i-1][j-1];//replace
                     o3=cost[3]+dp[i-1][j];//delete
                     o4=cost[4]+dp[i-1][j-1];//copy
                     dp[i][j]=min(o0,min(o1,min(o2,min(o3,o4))));
                }
            }
  }
 return dp[n][m];
}
int main()
{
  string s1,s2;
 cout<<"Enter  1st string : ";
 cin>>s1;
 cout<<"Enter 2nd string : ";
 cin>>s2;
 int n=s1.length();
 int m=s2.length();
 cout<<"Enter the cost of operations : \n";
 cout<<"Twiddle : ";
 cin>>cost[0];
```

```
cout<<"Insert : ";

cin>>cost[1];

cout<<"Replace : ";

cin>>cost[2];

cout<<"Delete : ";

cin>>cost[3];

cout<<"Copy : ";

cin>>cost[4];

cout<<"The minimum number of operation for convertion : "<<editd(s1,s2,n,m)<<endl;

return 0;

}
```



```
D:\svnit\sem4\daa\assgn6-bottomq2.exe
Enter  1st string : cat
Enter 2nd string : pat
Enter the cost of operations :
Twiddle : 1
Insert : 2
Replace : 4
Delete : 5
Copy : 6
The minimum number of operation for convertion : 4

--------------------------------
Process exited after 6.017 seconds with return value 0
Press any key to continue . . .
```

Time Complexity- O(n^2)