Change String to Make LCS Longer

Given two strings of length n and m respectively, design an algorithm using dynamic programming technique and implement using C++ to compute the

(i) Length of the Longest Common Subsequence of X and Y

(ii) Identify the position of the rightmost character in Y that has to be changed in order to make the length of the LCS(X, Y) increase by 1.

For example, If X=ABBCD, Y=BBEDF, then the LCS is BBD and the length of the LCS is 3. If we change 'E' in Y to letter 'C', then LCS (X,Y) is BBCD and length of the LCS is increased by 1.

Your code should return the the index of the character 'E' that has to be changed (i.e.) 3. For some sequences like : X=ABBCD, Y=BBDEF, LCS is BBD. Here, we cannot increase the length of the LCS by changing any character of Y. In that case, your code should return -1

```
#include<iostream>
using namespace std;
void lcs_length(string x,string y,int c[100][100])
{
   int m=x.length(),n=y.length();
   for(int i=1;i<=m;i++)
   {
      for(int j=1;j<=n;j++)
      {
         if(x[i-1]==y[j-1])
         {
            c[i][j]=c[i-1][j-1]+1;
         }
         else if(c[i-1][j]>=c[i][j-1])
         {
            c[i][j]=c[i-1][j];
         }
         else
         {
            c[i][j]=c[i][j-1];
         }
      }
   }
}
```

```
int rightmost_change(string x,string y,int c[100][100])
{
    int m=x.length(),n=y.length(),lcs_len=c[m][n],idx=-1;
    for(int i=n-1;i>=0;i--)
    {
        char curr=y[i];
        for(int j=0;j<=26;j++)
        {
            y[i]=65+j;
            lcs_length(x,y,c);
            int new_len=c[m][n];
            if(new_len==lcs_len+1)
            {
                idx=i;
                return idx+1;
            }
        }
        y[i]=curr;
    }
    return idx;
}
int main()
{
    string x,y;
    cin>>x>>y;
    int c[100][100];
    for(int i=0;i<=x.length();i++)
    {
        for(int j=0;j<=y.length();j++)
        {
            c[i][j]=0;
        }
    }
    lcs_length(x,y,c);
    int lcs_len=c[x.length()][y.length()];
    cout<<lcs_len<<endl;
    cout<<rightmost_change(x,y,c)<<endl;
}
```

MCM with Paired Column and Row matrices inside Square matrices

Matrix Chain Multiplication (MCM) Problem is one in which given a chain of matrices < A1,A2, ...,An > of n matrices, task is to fully parenthesize the product A1 ∗A2∗, ... ∗ An in such a way that the number of scalar

multiplications (the number of multiplications between the entries of the given matrices) performed in computing the product A1 * A2*, ... * An is minimum. Here is a special kind of problem in which we do MCM in a chain of matrices in which the first and last matrices are square matrices of some dimension 'm' and has some 'k' number of paired column and row matrices of dimension 'm'. A column matrix of dimension 'm' is one in which number of columns is 1 and number of rows is 'm'. A row matrix of dimension 'm' is one in which number of rows is 1 and number of columns is 'm'.

Given the number of paired column and row matrices, k and dimension 'm', develop an algorithm and C++ code to find the mimimum number of multiplications to be done to get the resultant matrix. For example, if the input is 1, 2 then there is a chain of 4 matrices of the dimensions 2x2, 2x1, 1x2, 2x2. If the input is 3, 2 then there is a chain of 8 matrices 2x2, 1x2, 2x1, 1x2, 2x1, 1x2, 2x1, 2x2

Input Format

First line contains the value of k

Next line contains the dimension 'm'

Output Format

Print minimum number of multiplications to be done

```
#include<iostream>
#include<limits.h>
using namespace std;
void mcm(int p[100],int s[100][100],int n)
{
   for(int l=2;l<=n;l++)
   {
      for (int i=1;i<=n-l+1;i++)
      {
         int j=i+l-1;
         s[i-1][j-1]=INT_MAX;
         for(int k=i;k<=j-1;k++)
         {
            int q=s[i-1][k-1]+s[k][j-1]+p[i-1]*p[k]*p[j];
            if(q<s[i-1][j-1])
```

```
            {
                s[i-1][j-1]=q;
            }
        }
    }
  }
}
int main()
{
    int k,m,p[100],s[100][100],i=0;
    cin>>k>>m;
    int n=2*(k+1)+1;
    while(i!=n)
    {
        if(i==0)
        {
            p[i]=m;
            i++;
            p[i]=m;
        }
        else if(i==n-1)
        {
            p[i]=m;
        }
        else
        {
            p[i]=m;
            i++;
            p[i]=1;
        }
        i++;
    }
    mcm(p,s,n);
    cout<<"\n"<<s[0][n-2]<<endl;
}
```