In [11]:

```
def fibonacci(n):
    if n<=1:
        return n;
    else:
        return fun(n-1)+fun(n-2)
g=fun(4);
print(g)
```

```
[1, 1, 2]
7
```

In [15]:

```
# Fibonacci Series using Dynamic Programming
def fibonacci(n):

    # Taking 1st two fibonacci nubers as 0 and 1
    FibArray = [0, 1]

    while len(FibArray) < n + 1:
        FibArray.append(0)

    if n <= 1:
       return n
    else:
       if FibArray[n - 1] ==  0:
           FibArray[n - 1] = fibonacci(n - 1)

       if FibArray[n - 2] ==  0:
           FibArray[n - 2] = fibonacci(n - 2)

       FibArray[n] = FibArray[n - 2] + FibArray[n - 1]
    return FibArray[n]

print(fibonacci(5))
```

```
5
```

In [31]:

```
def stairs(n):
    res=[];
    res.append(1);
    res.append(2);

    for i in range(2,n):

        res.append(res[i-1]+res[i-2]);
    return res[-1]
print(stairs(47))
```

```
7778742049
```

In [71]:

```
def fun(n,l):
    temp=[]

    r = [0 for x in range(n+1)]

    r[0]=0
    for i in range(1,n+1):
        maxTemp=-1;
        for j in range(i):
            maxTemp = max(maxTemp,l[j]+r[i-j-1]);
        r[i]=maxTemp

    return r
s=fun(5,[1,5,8,9,10])
print(s)
s=fun(8, [1, 5, 8, 9, 10, 17, 17, 20])
print(s)
```

```
[0, 1, 5, 8, 10, 13]
[0, 1, 5, 8, 10, 13, 17, 18, 22]
```

In [59]:

```
def cutRod(price, n):
    val = [0 for x in range(n+1)]
    val[0] = 0

    # Build the table val[] in bottom up manner and return
    # the last entry from the table
    for i in range(1, n+1):
        max_val = -1
        for j in range(i):
             max_val = max(max_val, price[j] + val[i-j-1])
        val[i] = max_val

    return val[n]
```

In [62]:

```
s=cutRod([1,5,8,9,10],5)
print(s)
s=cutRod([1, 5, 8, 9, 10, 17, 17, 20],8)
print(s)
```

```
13
22
```

In [32]:

```
5
4
3
2
1
```

In [10]:

```
#recursion
def house(var,nums):

    if var==0:
        return nums[0];

    max_val = -1
    max_val = max(max_val,house(var-1,nums));
    for j in range(var-2,-1,-1):
         max_val = max(max_val, house(j,nums)+nums[var-1]);
    return max_val;
nums=[1,2,3,4,5];
print(house(5,nums));
```

```
9
```

In [38]:

```
#dynamic
def house(var,nums):
    r=[0 for x in range(var)]
    r[0]=nums[0]
    r[1]=max([nums[0],nums[1]])

    for i in range(2,var):
        r[i]=r[i-1];
        r[i]=max(r[i-1],nums[i]+r[i-2]);

    return r[-1];


nums=[1,2,3,4,5];
print(house(5,nums));
```

```
9
```

In [64]:

```
def stock(nums):
    var=len(nums)
    r=[0 for x in range(var)]

    for i in range(1,var):
        r[i]=nums[i]-min(nums[0:i])
    print(r)
    if max(r)>0:
        return max(r)
    else:
        return 0

nums=[7,1,5,3,6,4];
print(stock(nums));
```

```
[0, -6, 4, 2, 5, 3]
5
```

In [76]:

```
def pair(nums):

    var = len(nums);
    maxVal = 0;
    for i in range(var-1,0,-1):

        for j in range(i-1,-1,-1):
            if nums[i][0]-nums[j][0]>maxVal:
                maxVal = nums[i][0]-nums[j][0];
                length = i-j

    return length;
nums=[[5,24],[15,25],[27,40],[50,60]]
pair(nums)
```

Out[76]:

```
3
```

In [ ]:

```
# A Dynamic Programming based Python Program for 0-1 Knapsack problem
# Returns the maximum value that can be put in a knapsack of capacity W
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]

    # Build table K[][] in bottom up manner
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif 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]
```

In [38]:

```
def gen(x,y):
    var1=len(x);
    var2=len(y);

    c=[[0]*var2]*var1;


    for i in range(1,var1):

        for j in range(1,var2):

            if x[i-1]==y[j-1]:
                c[i][j]=c[i-1][j-1]+1;
            else:
                c[i][j]=max(c[i-1][j],c[i][j-1]);
    return c;
x="abcccdf"
y="abccdf"
print(gen(x,y))
```

```
[[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5], [0, 1, 2,
 3, 4, 5], [0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]]
```

In [82]:

```
def power_set_2(set_):
    subsets = [[]]
    subsetsK=[];
    for element in set_:
        for ind in range(len(subsets)):
            subsets.append(subsets[ind] + [element])


    return subsets
l=[1,2,3,4,5]
s=power_set_2(l)
print(s)
```

```
[[0, 0], [0, 0]]
[0, 0]
```

In [35]:

```
def commonSubstring(x,y):

    var1=len(x);
    var2=len(y);

    #c=[[0]*var2]*var1;
    c=[[0 for k in range(var2+1)] for l in range(var1+1)]
    maxVal=-1

    for i in range(1,var1+1):

        for j in range(1,var2+1):

            if x[i-1]==y[j-1]:
                c[i][j]=c[i-1][j-1]+1;
            else:
                c[i][j]=0
            maxVal=max(maxVal,c[i][j]);
    return c;

x="DEADBEEF"
y="EATBEEF"
print(commonSubstring(x,y))
```

```
[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 1,
1, 0], [0, 0, 2, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
1, 0, 0, 0], [0, 1, 0, 0, 0, 2, 1, 0], [0, 1, 0, 0, 0, 1, 3, 0], [0, 0,
0, 0, 0, 0, 0, 4]]
```

In [ ]:

```
#longest palindromic subsequence
def longestPalindrome(s1):
    var1= len(s1);
    c=[[0 for k in range(var1)] for l in range(var1)];
    for i in range(var1):
        c[i][i]=1;
    for l in range(1,var1+1):
        for i in range(var1-l):
            j=l+i
            if s1[j]==s1[i]:
                c[i][j]=c[i+1][j-1]+2
            else:
                c[i][j]=max(c[i+1][j],c[i][j-1])
    return c[0][-1];

s1='BABCBAB';
print(longestPalindrome(s1))
```

In [40]:

```
def minDist(sRow,sCol):
    varRow=len(sRow);
    varCol=len(sCol);
    c=[[0 for k in range(varRow+1)] for l in range(varCol+1)];
    for i in range(1,varRow+1):
        c[0][i]=i;
    for i in range(1,varCol+1):
        c[i][0]=i;
    for i in range(1,varCol+1):
        for j in range(1,varRow+1):
            if sRow[j-1]==sCol[i-1]:
                c[i][j]=c[i-1][j-1];
            else:
                c[i][j]=min([c[i-1][j-1],c[i-1][j],c[i][j-1]])+1;


    return c[-1][-1];

s1='abcdef';
s2='azced';
print(minDist(s1,s2))
```

```
7
2
6
```

In [23]:

```
def contiguous(arr):
    n=len(arr)
    v = [0 for k in range(len(arr))]
    v[0] = arr[0];
    for i in range(1,n):
        v[i] = max(v[i-1]+a[i],a[i]);
    return max(v)
a=[-2, -3, 4, -1, -2, 1, 5, -3,10]
print(contiguous(a))
```

```
3
```

In [ ]:

```
def minimumSquare(a, b):

    result = 0
    rem = 0

    # swap if a is small size side .
    if (a < b):
        a, b = b, a

    # Iterate until small size side is
    # greater then 0

    while (b > 0):
        print(int(a/b))
        # Update result
        result += int(a / b)
        print(result)
        rem = int(a % b)
        a = b
        b = rem

    return result

print(minimumSquare(4,5))
```

In [4]:

```
#longest palindromic substring
def substring(s,t):
    m= len(s);
    n= len(t);
    d=[[0 for k in range(n)] for l in range(m)];
    max_value = -1;
    for i in range(m):
        for l in range(n):
            if s[i] == t[j]:
                d[i][j] = 1 + d[i-1] + d[j-1];

            else:
                d[i][j]=0;
            max_value = max(max_value,d[i][j])
    return max_value

s1='BABCBAB';
print(substring(s1))
```

```
[[0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
```

In [40]:

```
7
```

In [36]:

```
#longest palindromic substring
def longestPalindromeSubString(s1):
    s2="$"
    print(s1)
    var1= len(s1);
    for i in range(1,2*var1+1):
        if i%2==0:
            s2=s2+"$";
        else:
            s2=s2+s1[i//2];

    print(s2)
    c=[0 for k in range(2*var1+1)]
    temp=0;
    for i in range(1,2*var1):
        counter=1;

        c[i-1]=temp*2+1
        temp=0;

        while (i-counter>=0 and i+counter<(2*var1+1) and temp+1==counter

            if (s2[i-counter]==s2[i+counter]):
                temp+=1

            counter += 1;
    return max(c)//2


s1='abaab';
print(longestPalindromeSubString(s1))
```

```
abaab
$a$b$a$a$b$
4
```

In [37]:

```
#longest palindromic substring
def longestPalindromeSubString(s1):
    var1= len(s1);
    l3=[];
    counter=0;
    ind=0;
    for i in range(var1):

        if s1[ind:i+1]==s1[var1-i-1:var1-ind]:
            l3.append(s1[ind:i+1])
            ind=i+1;
        else:
            counter+=1;
    return len(l3)
s1='abaab'
print(longestPalindromeSubString(s1))
```

```
3
```

In [157]:

```
1
1
4
5
5
```

In [43]:

```
14
```

In [ ]: