大富翁游戏，玩家根据骰子的点数决定走的步数，即骰子点数为 1 时可以走一步，点数为 2 时可以走两步，点数为 n 时可以走 n 步。求玩家走到第 n 步（n<=骰子最大点数且是方法的唯一入参）时，总共有多少种投骰子的方法

归纳：f(n) = f(n-1)+f(n-2)+f(n-3)+....+f(1)+1，f(1)=1,f(2)=2.则 f(n)=2^(n-1).

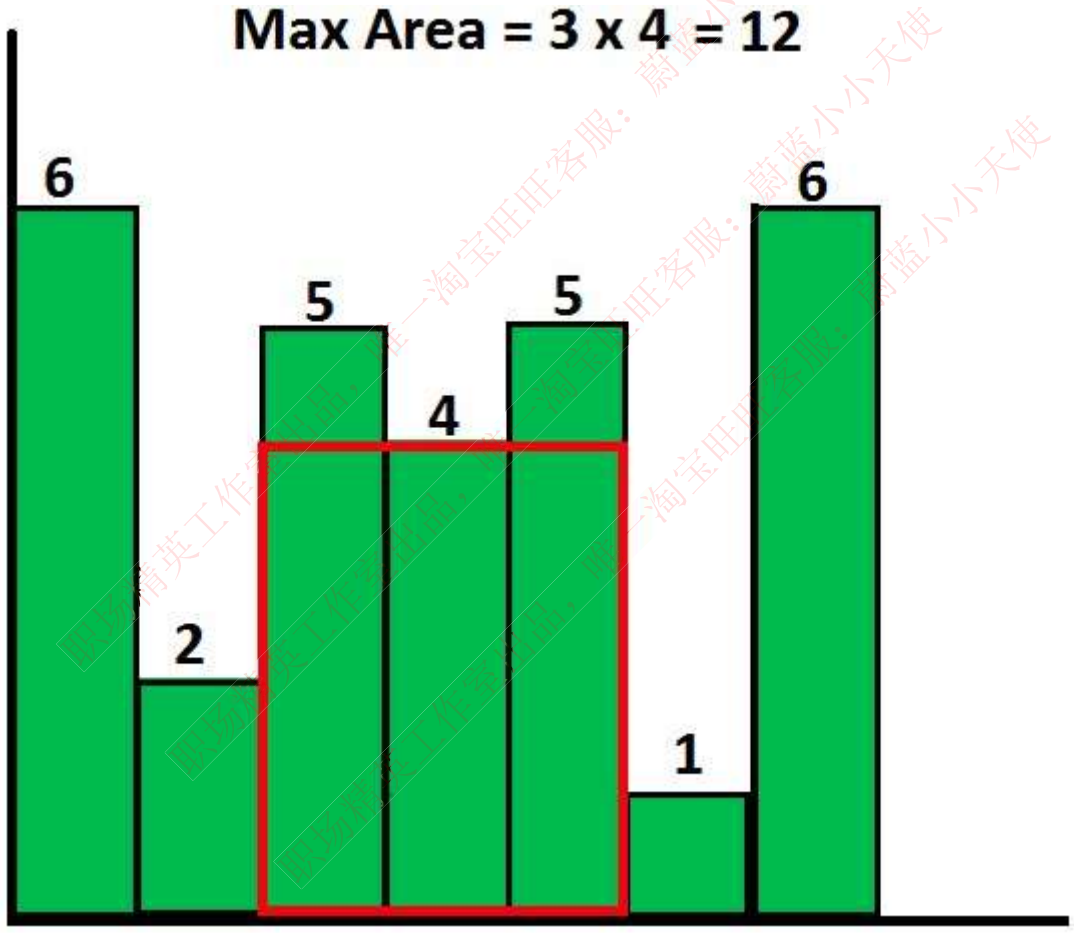
```
#include<iostream>

#include<algorithm>

using namespace std;

int main(){

int n;

while(cin>>n){

cout<<pow(2,n-1)<<endl;

}

return0;

}
```

给你六种面额 1、5、10、20、50、100 元的纸币，假设每种币值的数量都足够多，编写程序求组成 N 元（N 为 0~10000 的非负整数）的不同组合的个数。

```
//原来的答案 算法复杂度太大 空间复杂度也大
//参考讨论区的 更新 新方法
import java.util.Scanner;
import java.util.Arrays;
public class Main{
    public static long count(int n){
        if(n <= 0)return 0;
        int[] coins = new int[]{1,5,10,20,50,100};
        long[] dp = new long[n+1];
        dp[0] = 1;
        for(int i = 0; i < coins.length; i++) {
        for(int j = coins[i]; j <= n; j++) {
        dp[j] = dp[j] + dp[j - coins[i]];//类似斐波那契 后者的种类数基于前者
        }
    }
    return dp[n];
    }
```

```
        public static void main(String args[]){
            Scanner sc=new Scanner(System.in);
            while(sc.hasNext()){
                int n=sc.nextInt();
                long res=count(n);
                System.out.println(res);
            }
        }
}
```

给定一组非负整数组成的数组 h，代表一组柱状图的高度，其中每个柱子的宽度都为 1。 在这组柱状图中找到能组成的最大矩形的面积（如图所示）。 入参 h 为一个整型数组，代表每个柱子的高度，返回面积的值。



'''

分治法：最大矩形面积只可能有三种情况：

1. 取决于高度最小的柱子，此时面积等于高度乘总长度；
2. 最大面积出现在高度最小的柱子左边；
3. 最大面积出现在高度最小的柱子右边；

'''

```
n = int(raw_input())
h = [int(x) for x in raw_input().split()]
```

```
def largestarea(a):
    l = len(a)
    idx = a.index(min(a))

    value1 = a[idx] * l

    if idx != 0:
        value2 = largestarea(a[0:idx])
    else:
        value2 = 0
    if idx != l-1:
        value3 = largestarea(a[idx+1:l])
    else:
        value3 = 0
    return max(value1, value2, value3)

print largestarea(h)
```

给出两个字符串（可能包含空格）,找出其中最长的公共连续子串,输出其长度。

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
using namespace std;
int main(){
    string s1, s2;
    while (getline(cin, s1), getline(cin, s2)){
        int l1 = s1.size();
        int l2 = s2.size();
        vector<vector<int>> dp(l1 + 1, vector<int>(l2 + 1, 0));
        int result = 0;
        for (int i = 1; i <= l1; i++){
            for (int j = 1; j <= l2; j++){
                if (s1[i - 1] == s2[j - 1]){
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                    result = max(dp[i][j], result);
                }
                else{
                    dp[i][j] = 0;
                }
            }
        }
        cout << result << endl;
```

```
        }
        return 0;
    }
```