

给一个 $m \times n$ 的矩阵，矩阵中的元素不是 0 就是 1，请你统计并输出其中完全由 1 组成的正方形子矩阵的个数。

```
#23n2300017735(夏天明BrightSummer)
m, n = map(int, input().split())
mat = [[int(k) for k in input()] for i in range(m)]
dp = [[0 for j in range(n+1)] for i in range(m+1)]
for i in range(m):
    for j in range(n):
        if mat[i][j]:
            dp[i+1][j+1] = min(dp[i][j], dp[i][j+1], dp[i+1][j])+1
print(sum(dp[i][j] for j in range(n+1) for i in range(m+1)))
```

最长公共子序列:

我们称一个字符的数组 S 为一个序列。对于另外一个字符数组 Z ,如果满足以下条件，则称 Z 是 S 的一个子序列：（1） Z 中的每个元素都是 S 中的元素（2） Z 中元素的顺序与在 S 中的顺序一致。例如：当 $S = (E, R, C, D, F, A, K)$ 时， (E, C, F) 和 (E, R) 等等都是它的子序列。而 (R, E) 则不是。现在我们给定两个序列，求它们最长的公共子序列的长度。

```
def longest_common_subsequence(s1, s2):
    dp = [[0 for _ in range(len(s2)+1)] for _ in range(len(s1)+1)]
    for i in range(len(s1)):
        for j in range(len(s2)):
            if s1[i] == s2[j]:
                dp[i+1][j+1] = dp[i][j] + 1
            else:
                dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1])
    return dp[len(s1)][len(s2)]

s1 = input()
s2 = input()
print(longest_common_subsequence(s1, s2))
```

土豪购物:

给一个整数组成的数列，其中每个数字代表商品价值(可能为负)土豪买东西的方法是 "从第 n 个到第 k 个商品我全要了!!!" ($n \leq k$)，换句话说土豪一定会买下连续的几个商品买完以后土豪会看心情最多放回去其中一个商品(可以不放回)但土豪不能空手而归，他至少要带回去一个商品请问聪明的(?)土豪可以买到最大价值总和为多少的商品?

样例:

商品价值:1,-5,0,3 输出:4 最大价值总和是买[1,-5,0,3]，并放回-5 后的总和

商品价值:-2,-2,-2 输出:-2 最大价值总和是买[-2]，不放回的总和(至少要带回去一个商品)

```

a = list(map(int, input().split(',')))
dp1 = [0] * len(a);
dp2 = [0] * len(a)
dp1[0] = a[0];
dp2[0] = a[0]
for i in range(1, len(a)):
    dp1[i] = max(dp1[i - 1] + a[i], a[i])
    dp2[i] = max(dp1[i - 1], dp2[i - 1] + a[i], a[i])
print(max(dp2))

```

把 M 个同样的苹果放在 N 个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用 K 表示）5, 1, 1 和 1, 5, 1 是同一种分法。

```

def count_ways(m, n):
    dp = [[0 for _ in range(n+1)] for _ in range(m+1)]
    for i in range(m+1):
        dp[i][1] = 1
    for i in range(1, m+1):
        for j in range(2, min(i, n)+1):
            dp[i][j] = dp[i-j][j] + dp[i][j-1]
    return dp[m][n]

# Processing input
try:
    while True:
        m, n = map(int, input().split())
        print(count_ways(m, n))
except EOFError:
    pass

```

21515: 电话线路

<http://cs101.openjudge.cn/practice/21515/>

有 N 座通信基站， P 条双向电缆，第 i 条电缆连接基站 A_i 和 B_i 。特别地，1号基站是通信公司的总站， N 号基站位于一座农场中。现在，农场主希望对通信线路进行升级，其中升级第 i 条电缆需要花费 L_i 。

电话公司正在举行优惠活动。农场主可以指定一条从1号基站到 N 号基站的路径，然后，农场主可以指定路径上不超过 K 条电缆，先由电话公司免费提供升级服务。农场主只需要支付在该路径上剩余的电缆中，升级价格最贵的那条电缆的花费即可。支付完成后，其余电缆也将由电话公司免费升级。求至少用多少钱能完成升级。

输入

第一行三个整数， N ， P ， K 。接下来 P 行，每行三个整数 A_i ， B_i ， L_i 。

输出

若不存在从1到 N 的路径，输出-1。否则输出所需最小费用。

样例输入

```
5 7 1
1 2 5
3 1 4
2 4 8
3 2 3
5 2 9
3 4 7
4 5 6
```



样例输出

```
4
```



```

from heapq import *
n,p,k = map(int,input().split())
graph = {i:{} for i in range(1,n+1)}
h = 0
for _ in range(p):
    a,b,l = map(int,input().split())
    graph[a][b] = graph[b][a] = 1
    h = max(h,l)
l = 0

def search(lim):
    heap = [(-1,-k)]
    heapify(heap)
    vis = {}
    while heap:
        idx,free = heappop(heap)
        idx,free = -idx,-free
        if idx == n:
            return 1
        if idx not in vis or vis[idx] < free:
            vis[idx] = free
        else:
            continue
        for t,length in graph[idx].items():
            new_free = free
            if length > lim:
                if new_free > 0:
                    new_free -= 1
            else:
                continue
            if t in vis and vis[t] > new_free:
                continue
            heappush(heap,(-t,-new_free))
    return 0
while l < h:
    if l + 1 == h:
        ans_l,ans_h = search(l),search(h)
        if ans_l == ans_h == 0:
            print(-1)
        else:
            print(l if ans_l else h)
            exit()
    mid = (l+h)//2
    if search(mid):
        h = mid
    else:
        l = mid

```

22636: 修仙之路

<http://cs101.openjudge.cn/dsapre/22636/>

修仙之路长漫漫，逆水行舟，不进则退！你过五关斩六将，终于来到了仙界。仙界是一个 r 行 c 列的二维格子空间，每个单元格是一个“境界”，每个境界都有等级。你需要任意选择其中一个境界作为起点，从一个境界可以前往上下左右相邻四个境界之一，当且仅当新到达的境界等级增加。你苦苦行走，直到所在的境界等级比相邻四个境界的等级都要高为止，一览众山小。请问包括起始境界在内最长修仙路径需要经过的境界数是多少？

输入

第一行为两个正整数，分别为 r 和 c ($1 \leq r, c \leq 100$)。接下来有 r 行，每行有 c 个0到100000000之间的整数，代表各境界的等级。

输出

输出一行，为最长修仙路径需要经过的境界数（包括起始境界）。

样例输入

```
5 5
1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

样例输出

```
25
```

```
def dfs(i,j):
    if dp[i][j]>0:
        return dp[i][j]
    else:
        for k in range(4):
            if 0<=i+d[k][0]<r and 0<=j+d[k][1]<c and maze[i][j]>maze[i+d[k][0]][j+d[k][1]]:
                dp[i][j]=max(dp[i][j],dfs(i+d[k][0],j+d[k][1])+1)
        return dp[i][j]

r,c=map(int,input().split())
maze=[]
for i in range(r):
    l=list(map(int,input().split()))
    maze.append(l)
dp=[[0]*c for _ in range(r)]
d=[[-1,0],[1,0],[0,1],[0,-1]]
ans=0
for i in range(r):
    for j in range(c):
        ans=max(ans,dfs(i,j))
print(ans+1)
```

24687: 封锁管控

<http://cs101.openjudge.cn/dsapre/24687/>

为减少人员流动，降低疫情传播风险，某城市决定在内部施加封锁管控措施。

为方便讨论，假设城市为一条线段，从左至右排布了 n 个居民区，第 i 个居民区中住有 a_i 个人。现在要建设 $m(m < n)$ 个“管控点”（可视作墙），每个管控点设在相邻两个居民区之间，使得居民的活动不能跨越该管控点。

定义“人口流动指数”为每个居民（从其原住区）能到达的居民区个数的总和。求在建设 m 个管控点后，人口流动指数最小为多少？

例如，5 个居民区被 1 个管控点隔开（数字表示居民区的人数）：

10 50 | 20 30 40

则此时的人口流动指数为 $(10 + 50) * 2 + (20 + 30 + 40) * 3 = 390$ 。

输入

输入有两行。第一行为两个正整数 n, m ($n \leq 100$)；第二行有 n 个数，表示每个居民区的人数 a_i ($a_i \leq 1000$)，用空格隔开。

输出

输出只有一行。一个正整数表示人口流动指数的最小值。

样例输入

```
5 1
10 50 20 30 40
```

样例输出

```
380
```

提示

对样例的解释：在第三个和第四个居民区间设管控点，此时人口流动指数为 $(10+50+20)*3+(30+40)*2=380$ 。

为了找到最小的人口流动指数，我们需要确定在哪里建立管控点才能最大限度地减少人口流动。一个朴素的方法是考虑所有可能的管控点设置，然后选择人口流动指数最小的设置。但是，这样做的时间复杂度是非常高的，特别是当居民区数量较多时。

我们可以使用动态规划来解决这个问题。我们可以定义一个动态规划数组 $dp[i][j]$ 表示前 i 个居民区建立 j 个管控点后的最小人口流动指数。

状态转移方程如下：

$$dp[i][j] = \min(dp[k][j-1] + \text{sum}[k+1 \text{ to } i] * (i-k)) \text{ 对于所有 } k < i$$

其中 $\text{sum}[k+1 \text{ to } i]$ 表示从居民区 $k+1$ 到居民区 i 的人口数总和。

这样，最终答案将是 $dp[n][m]$ 。


```

def min_population_flow(n, m, populations):
    # Initialize the prefix sum array for fast range sum computation
    prefix_sum = [0] * (n + 1)
    for i in range(1, n + 1):
        prefix_sum[i] = prefix_sum[i - 1] + populations[i - 1]

    # Initialize the DP table
    dp = [[float('inf')] * (m + 1) for _ in range(n + 1)]

    # Base case: with 0 control points, the flow index is just the sum of all populations times their district co
    for i in range(1, n + 1):
        dp[i][0] = prefix_sum[i] * i

    # Fill the DP table
    for i in range(1, n + 1):
        for j in range(1, min(i, m) + 1):
            for k in range(j - 1, i):
                dp[i][j] = min(dp[i][j], dp[k][j - 1] + (prefix_sum[i] - prefix_sum[k]) * (i - k))

    # The answer is the minimum flow index after setting up m control points
    return dp[n][m]

# Input
n, m = map(int, input().split())
populations = list(map(int, input().split()))

# Output
print(min_population_flow(n, m, populations))

```

🔗 25815: 回文字符串

<http://cs101.openjudge.cn/dsapre/25815/>

给定一个字符串 S ，最少需要几次增删改操作可以把 S 变成一个回文字符串？

一次操作可以在任意位置插入一个字符，或者删除任意一个字符，或者把任意一个字符修改成任意其他字符。

输入

字符串 S 。 S 的长度不超过100, 只包含'A'-'Z'。

输出

最少的修改次数。

样例输入

```
ABAD
```

样例输出

```
1
```

来源: hihoCoder

```
# 2300011335
S = list(input())
n = len(S)
dp = [[0 for _ in range(n)] for _ in range(n)]
for length in range(1,n):
    for i in range(n-length):
        j = i+length
        if S[i] == S[j]:
            dp[i][j] = dp[i+1][j-1]
        else:
            dp[i][j] = min(dp[i+1][j], dp[i][j-1], dp[i+1][j-1])+1
print(dp[0][-1])
```