# 二分查找

## 1、河中跳房子

**描述**

每年奶牛们都要举办各种特殊版本的跳房子比赛，包括在河里从一个岩石跳到另一个岩石。这项激动人心的活动在一条长长的笔直河道中进行，在起点和离起点L远 (1 ≤ *L*≤ 1,000,000,000) 的终点处均有一个岩石。在起点和终点之间，有*N* (0 ≤ *N* ≤ 50,000) 个岩石，每个岩石与起点的距离分别为*Di (0 < *Di* < *L*)。*

在比赛过程中，奶牛轮流从起点出发，尝试到达终点，每一步只能从一个岩石跳到另一个岩石。当然，实力不济的奶牛是没有办法完成目标的。

农夫约翰为他的奶牛们感到自豪并且年年都观看了这项比赛。但随着时间的推移，看着其他农夫的胆小奶牛们在相距很近的岩石之间缓慢前行，他感到非常厌烦。他计划移走一些岩石，使得从起点到终点的过程中，最短的跳跃距离最长。他可以移走除起点和终点外的至多*M* (0 ≤ *M* ≤ *N*) 个岩石。

请帮助约翰确定移走这些岩石后，最长可能的最短跳跃距离是多少?

**输入**

第一行包含三个整数L, N, M，相邻两个整数之间用单个空格隔开。
接下来N行，每行一个整数，表示每个岩石与起点的距离。岩石按与起点距离从近到远给出，且不会有两个岩石出现在同一个位置。

**输出**

一个整数，最长可能的最短跳跃距离。

样例输入

```
25 5 2
2
11
14
17
21
```

样例输出

```
4
```

**代码**

```
l,n,m=map(int,input().split())
rock=[0]
for _ in range(n):
    rock.append(int(input()))
rock.append(l)
left,right=0,l
while left<right:
    mid=(left+right)//2
    cnt=0
    now=0
```

```
    for i in range(1,n+2):
        if rock[i]-now<mid:
            cnt+=1
        else:
            now=rock[i]
    if cnt>m:
        right=mid
    else:
        ans=mid
        left=mid+1
print(ans)
```

# 动态规划（DP）

## 1、0-1背包问题

### 1.1 小偷背包

**描述**

这是《算法图解》[1]书中第9章动态规划的例子：一个小贼正在一家店里偷商品。

假设一种情况如下：

一个小偷背着一个可装4磅东西的背包。商场有三件物品分别为：
价值3000美元重4磅的音响，价值2000美元重3磅的笔记本，价值1500美元重1磅的吉他。

问小偷应该怎样选择商品，才能使得偷取的价值最高？

**输入**

第一行是两个整数N和B，空格分隔。N表示物品件数，B表示背包最大承重。
第二行是N个整数，空格分隔。表示各个物品价格。
第三行是N个整数，空格分隔。表示各个物品重量（是与第二行物品对齐的）。

**输出**

输出一个整数。保证在满足背包容量的情况下，偷的价值最高。

样例输入

```
3 4
3000 2000 1500
4 3 1
```

样例输出

```
3500
```

### 基本思路

```
n,b=map(int,input().split())
p=list(map(int,input().split()))
w=list(map(int,input().split()))
staff=[(0,0)]+list(zip(p,w))
```

```
mx=[[0]*b for _ in range(n+1)]
for i in range(1,n+1):
    for j in range(b):
        if j+1<staff[i][1]:
            mx[i][j]=mx[i-1][j]
        elif j+1==staff[i][1]:
            mx[i][j]=max(mx[i-1][j],staff[i][0])
        else:
            mx[i][j]=max(mx[i-1][j],staff[i][0]+mx[i-1][j-staff[i][1]])
print(mx[-1][-1])
```

### 优化空间复杂度

```
# 压缩矩阵/滚动数组 方法
N,B = map(int, input().split())
*p, = map(int, input().split())
*w, = map(int, input().split())
dp=[0]*(B+1)
for i in range(N):
    for j in range(B, w[i] - 1, -1):
        dp[j] = max(dp[j], dp[j-w[i]]+p[i])
print(dp[-1])
```

## 1.2 最佳凑单

**描述**

消费者为了享受商家发布的满减优惠，常常需要面临凑单问题。

假设有n件商品，每件的价格分别为p1,p2,...,pn，每件商品最多只能买1件。为了享受优惠，需要凑单价为t。那么我们要找到一种凑单方式，使得凑单价格不小于t（从而享受优惠），同时尽量接近t。被称为“最佳凑单”

如果不存在任何一种凑单方式，使得凑单价格不小于t（即无法享受优惠），那么最佳凑单不存在。

比如当前还差10元享受满减，可选的小件商品有5件，价格分别为3元、5元、8元、8元和9元，每件商品最多只能买1件。那么当前的最佳凑单就是11元（3元+8元）。

**输入**

第一行输入商品数n（n <= 100），和需要凑单价t。
第二行输入每件商品的价格。

**输出**

如果可以凑单成功，则输出最佳凑单的价格 。
如果无法凑单成功，则输出0。

样例输入

```
Sample1 Input:
5 10
3 5 8 8 9

Sample1 Output:
11
```

样例输出

```
Sample2 Input:
10 89
90 10 10 10 10 5 5 3 4 1

Sample2 Output:
90
```

**代码**

```
n,t=map(int,input().split())
p=list(map(int,input().split()))
if sum(p)<t:
    print(0)
    exit()
dp=[[0]*(sum(p)+1) for _ in range(n+1)]
for i in range(1,n+1):
    for j in range(1,sum(p)+1):
        if j<p[i-1]:
            dp[i][j]=dp[i-1][j]
        else:
            dp[i][j]=max(dp[i-1][j],dp[i-1][j-p[i-1]]+p[i-1])
for j in range(t,sum(p)+1):
    if dp[n][j]==j:
        print(j)
        break
```

# 2、“恰好”背包问题

## 2.1 健身房

**描述**

小嘤是大不列颠及北爱尔兰联合王国大力士，为了完成增肌计划，他需要选择一些训练组进行训练：有n个训练组，每天做第i个训练需要耗费ti分钟，每天坚持做第i个训练一个月后预计可增肌wi千克。因为会导致效果变差，小嘤一天不会做相同的训练组多次。由于小嘤是强迫症，他希望每天用于健身的时间**恰好**为**T** 分钟，他希望在一个月后获得最多的增肌量，请帮助小嘤计算：他训练一个月后最大增肌量是多少呢?

**输入**

第一行两个整数 T,n。

第 2 行到第 n+1 行，每行两个整数 ti,wi。

保证 0 < ti ≤ T ≤ 103, 0 < n ≤ 103, 0 < wi < 20。

**输出**

如果不存在满足条件的训练计划，输出-1。

如果存在满足条件的训练计划，输出一个整数，表示训练一个月后的最大增肌量。

样例输入

```
sample1 in
6 4
2 1
4 7
3 5
3 5

sample1 out
10
```

样例输出

```
sample2 in
700 4
450 5
340 1
690 10
9 2

sample2 out
-1
```

样例2解释：无法找出一种方案满足训练时间恰好等于T.

**代码**

```
t,n=map(int,input().split())
dp=[[0]*(t+1) for _ in range(n)]
t1,w1=map(int,input().split())
dp[0][t1]=w1
for i in range(1,n):
    ti,wi=map(int,input().split())
    for j in range(1,t+1):
        if j<ti:
            dp[i][j]=dp[i-1][j]
        elif j==ti:
            dp[i][j]=max(dp[i-1][j],wi)
        else:
            if dp[i-1][j-ti]==0:
                dp[i][j]=dp[i-1][j]
            else:
                dp[i][j]=max(dp[i-1][j],dp[i-1][j-ti]+wi)
if dp[-1][-1]==0:
    print(-1)
else:
    print(dp[-1][-1])
```

# 3、完全背包问题

## 3.1 Cut Ribbon

brute force/dp, 1300, https://codeforces.com/problemset/problem/189/A

**代码**

```
n,a,b,c=map(int,input().split())
dp=[0]+[-4000]*4000
for i in range(1,n+1):
    dp[i]=max(dp[i-a],dp[i-b],dp[i-c])+1
print(dp[n])
```

## 3.2 Sumsets

描述

Farmer John commanded his cows to search for different sets of numbers that sum to a given number. The cows use only numbers that are an integer power of 2. Here are the possible sets of numbers that sum to 7:

1. 1+1+1+1+1+1+1
2. 1+1+1+1+1+2
3. 1+1+1+2+2
4. 1+1+1+4
5. 1+2+2+2
6. 1+2+4

Help FJ count all possible representations for a given integer N (1 <= N <= 1,000,000).

输入

A single line with a single integer, N.

输出

The number of ways to represent N as the indicated sum. Due to the potential huge size of this number, print only last 9 digits (in base 10 representation).

样例输入

```
7
```

样例输出

```
6
```

**代码**

```
n=int(input())
dp=[1]*(n+1)
for i in range(1,n+1):
    if i%2==0:
        dp[i]=(dp[i-1]+dp[i//2])%1e9
    else:
        dp[i]=dp[i-1]%1e9
print(int(dp[n]))
```

# 4、最大连续子序列和

## 4.1 最大连续子序列和

现有一个整数序列a1,…，an,求连续子序列ai+…+aj的最大值。

**输入**

第一行一个正整数n(1≤n≤10^4)，表示序列长度；

第二行为用空格隔开的n个整数a_i(−10^5≤a_i≤10^5)，表示序列元素。

**输出**

输出一个整数，表示最大连续子序列和。

样例1

输入

```
6
-2 11 -4 13 -5 -2
```

输出

```
20
```

**代码**

```
n=int(input())
s=list(map(int,input().split()))
dp=[0]*n
dp[0]=s[0]
for i in range(1,n):
    dp[i]=max(dp[i-1]+s[i],s[i])
print(max(dp))
```

## 4.2 最大子矩阵

**描述**

已知矩阵的大小定义为矩阵中所有元素的和。给定一个矩阵，你的任务是找到最大的非空(大小至少是1 * 1)子矩阵。

比如，如下4 * 4的矩阵

0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2

的最大子矩阵是

9 2
-4 1
-1 8

这个子矩阵的大小是15。

**输入**

输入是一个N * N的矩阵。输入的第一行给出N (0 < N <= 100)。再后面的若干行中，依次（首先从左到右给出第一行的N个整数，再从左到右给出第二行的N个整数……）给出矩阵中的N2个整数，整数之间由空白字符分隔（空格或者空行）。已知矩阵中整数的范围都在[-127, 127]。

**输出**

输出最大子矩阵的大小。

样例输入

```
4
0 -2 -7 0 9 2 -6 2
-4 1 -4  1 -1

8  0 -2
```

样例输出

```
15
```

**代码：数据处理+Kadane算法**

```
n=int(input())
number=[]
while len(number)<n**2:
    number.extend(input().split())
mx=[list(map(int,number[i*n:(i+1)*n])) for i in range(n)]
def kadane(arr):
    dp=[0]*len(arr)
    dp[0]=arr[0]
    for i in range(1,len(arr)):
        dp[i]=max(dp[i-1]+arr[i],arr[i])
    return max(dp)
t=-float('inf')
for l in range(n):
    tmp=[0]*n
    for r in range(l,n):
        for i in range(n):
            tmp[i]+=mx[i][r]
        t=max(t,kadane(tmp))
print(t)
```

# 5、最长上升子序列

## 5.1 最长上升子序列

**描述**

一个数的序列*bi*，当*b1* < *b2* < ... < *bS*的时候，我们称这个序列是上升的。对于给定的一个序列(*a1*, *a2*, ..., *aN*)，我们可以得到一些上升的子序列(*ai1*, *ai2*, ..., *aiK*)，这里1 <= *i1* < *i2* < ... < *iK* <= N。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中最长的长度是4，比如子序列(1, 3, 5, 8).

你的任务，就是对于给定的序列，求出最长上升子序列的长度。

**输入**

输入的第一行是序列的长度N (1 <= N <= 1000)。第二行给出序列中的N个整数，这些整数的取值范围都在0到10000。

**输出**

最长上升子序列的长度。

样例输入

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

样例输出

```
4
```

## dp

```
n=int(input())
s=list(map(int,input().split()))
dp=[1]*n
for i in range(1,n):
    for j in range(i):
        if s[i]>s[j]:
            dp[i]=max(dp[i],dp[j]+1)
print(max(dp))
```

## bisect

```
import bisect
n=int(input())
l=list(map(int, input().split()))
dp=[1e9]*n
for i in l:
    dp[bisect.bisect_left(dp, i)] = i
print(bisect.bisect_left(dp, 1e8))
```

## 5.2 最大上升子序列和

**描述**

一个数的序列bi，当b1 < b2 < … < bS的时候，我们称这个序列是上升的。对于给定的一个序列(a1, a2, …,aN)，我们可以得到一些上升的子序列(ai1, ai2, …, aiK)，这里1 <= i1 < i2 < … < iK <= N。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中序列和最大为18，为子序列(1, 3, 5, 9)的和.

你的任务，就是对于给定的序列，求出最大上升子序列和。注意，最长的上升子序列的和不一定是最大的，比如序列(100, 1, 2, 3)的最大上升子序列和为100，而最长上升子序列为(1, 2, 3)

**输入**

输入的第一行是序列的长度N (1 <= N <= 1000)。第二行给出序列中的N个整数，这些整数的取值范围都在0到10000（可能重复）。

**输出**

最大上升子序列和

样例输入

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

样例输出

```
18
```

**代码**

```
n=int(input())
s=list(map(int,input().split()))
dp=list(s)
for i in range(1,n):
    for j in range(i):
        if s[i]>s[j]:
            dp[i]=max(dp[i],dp[j]+s[i])
print(max(dp))
```

## 5.3 充实的寒假生活

**描述**

寒假马上就要到了，龙傲天同学获得了从第0天开始到第60天结束为期61天超长寒假，他想要尽可能丰富自己的寒假生活。
现提供若干个活动的起止时间，请计算龙同学这个寒假至多可以参加多少个活动？注意所参加的活动不能有任何时间上的重叠，在第x天结束的活动和在第x天开始的活动不可同时选择。

**输入**

第一行为整数n，代表接下来输入的活动个数(n < 10000)
紧接着的n行，每一行都有两个整数，第一个整数代表活动的开始时间，第二个整数代表全结束时间

**输出**

输出至多参加的活动个数

样例输入

```
5
0 0
1 1
2 2
3 3
4 4
```

样例输出

```
5
```

## greedy

```
n=int(input())
activity=[]
for _ in range(n):
    activity.append(list(map(int,input().split())))
activity.sort(key=lambda x:x[1])
ans=1
tmp=activity[0][1]
for i in range(1,n):
    if activity[i][0]>tmp:
        ans+=1
        tmp=activity[i][1]
print(ans)
```

## dp

```
n=int(input())
act=[62]*61
for _ in range(n):
    s,e=map(int,input().split())
    act[s]=min(act[s],e)
dp=[1]*61
for i in range(1,61):
    if act[i]<62:
        for j in range(i):
            if act[j]<i:
                dp[i]=max(dp[i],dp[j]+1)
print(max(dp))
```

# 6、最长公共子串/子序列

## 6.1 最长公共子串（连续）

```
if word_a[i] == word_b[j]:  ←---------两个字母相同
  cell[i][j] = cell[i-1][j-1] + 1
else:  ←------------------------------两个字母不同
  cell[i][j] = 0
```

## 6.2 最长公共子序列（不一定连续）

**描述**

我们称序列Z = < z1, z2, ..., zk >是序列X = < x1, x2, ..., xm >的子序列当且仅当存在 **严格上升** 的序列< i1, i2, ..., ik >，使得对j = 1, 2, ... ,k, 有xij = zj。比如Z = < a, b, f, c > 是X = < a, b, c, f, b, c >的子序列。

现在给出两个序列X和Y，你的任务是找到X和Y的最大公共子序列，也就是说要找到一个最长的序列Z，使得Z既是X的子序列也是Y的子序列。

**输入**

输入包括多组测试数据。每组数据包括一行，给出两个长度不超过200的字符串，表示两个序列。两个字符串之间由若干个空格隔开。

**输出**

对每组输入数据，输出一行，给出两个序列的最大公共子序列的长度。

样例输入

```
abcfbc         abfcab
programming    contest
abcd           mnp
```

样例输出

```
4
2
0
```

**代码**

```
while True:
    try:
        x,y=input().split()
    except:
        break
    lx=len(x)
    ly=len(y)
    dp=[[0]*(ly+1) for _ in range(lx+1)]
    for i in range(1,lx+1):
        for j in range(1,ly+1):
            if x[i-1]==y[j-1]:
                dp[i][j]=dp[i-1][j-1]+1
            else:
```

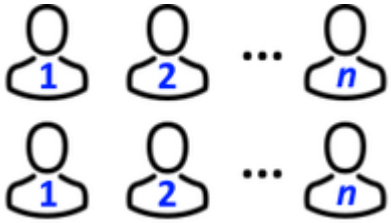
```
                dp[i][j]=max(dp[i-1][j],dp[i][j-1])
    print(dp[-1][-1])
```

# 7、定义多个dp数组

## 7.1 basketball

dp, 1400, https://codeforces.com/problemset/problem/1195/C

Finally, a basketball court has been opened in SIS, so Demid has decided to hold a basketball exercise session. $2 \cdot n$ students have come to Demid's exercise session, and he lined up them into two rows of the same size (there are exactly $n$ people in each row). Students are numbered from 11 to $n$ in each row in order from left to right.




Now Demid wants to choose a team to play basketball. He will choose players from left to right, and the index of each chosen player (excluding the first one **taken**) will be strictly greater than the index of the previously chosen player. To avoid giving preference to one of the rows, Demid chooses students in such a way that no consecutive chosen students belong to the same row. The first student can be chosen among all $2n$ students (there are no additional constraints), and a team can consist of any number of students.

Demid thinks, that in order to compose a perfect team, he should choose students in such a way, that the total height of all chosen students is maximum possible. Help Demid to find the maximum possible total height of players in a team he can choose.

Input

The first line of the input contains a single integer $n$ (1≤$n$≤10^5^) — the number of students in each row.

The second line of the input contains $n$ integers $$h_{1,1}, h_{1,2}, \ldots, h_{1,n}$$ (1≤$h_{1,i}$≤10^9)$, where $$h_{1,i}$$is the height of the $i$-th student in the first row.

The third line of the input contains $n$ integers $$h_{2,1}, h_{2,2}, \ldots, h_{2,n}$$ (1≤$h_{2,i}$≤10^9)$, where $$h_{2,i}$$ is the height of the $i$-th student in the second row.

Output

Print a single integer — the maximum possible total height of players in a team Demid can choose.

Examples

input

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

output

```
29
```

input

```
3
1 2 9
10 1 1
```

output

```
19
```

input

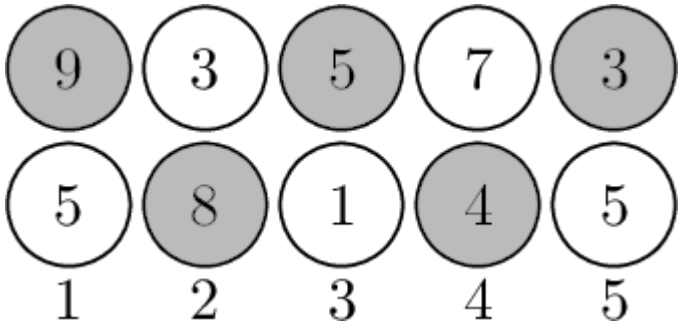
```
1
7
4
```

output

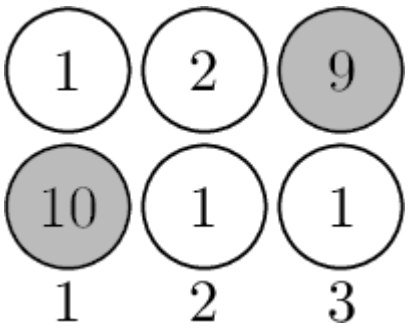
```
7
```

Note

In the first example Demid can choose the following team as follows:




In the second example Demid can choose the following team as follows:



```
n=int(input())
r1=list(map(int,input().split()))
r2=list(map(int,input().split()))
dp1=[0]*n
dp1[0]=r1[0]
dp2=[0]*n
dp2[0]=r2[0]
for i in range(1,n):
    dp1[i]=max(dp2[i-1]+r1[i],dp1[i-1])
    dp2[i]=max(dp1[i-1]+r2[i],dp2[i-1])
print(max(dp1[-1],dp2[-1]))
```

## 7.2 红蓝玫瑰

dp, greedy, http://cs101.openjudge.cn/practice/25573/

**描述**

“玫瑰的红，容易受伤的梦，握在手中却流失于指缝，又落空”

有n (n<500000)支玫瑰从左到右排成一排，它们的颜色是红色或蓝色，红色玫瑰用R表示，蓝色玫瑰用B表示

作为魔法女巫的你，掌握两种魔法：

魔法1：对一支玫瑰施加颜色反转咒语

魔法2：对从左数前k支玫瑰同时施加颜色反转咒语（每次施法时的k值可以不同）

颜色反转咒语将使红玫瑰变成蓝玫瑰，蓝玫瑰变成红玫瑰

请你求出，最少使用多少次魔法，能使得这一排玫瑰全都变为红玫瑰

**输入**

一个字符串，由R和B组成

**输出**

一个整数，最少使用多少次魔法

样例输入

```
Sample Input1:
RRRRRBR

Sample Output1:
1
```

样例输出

```
Sample Input2:
RRRBBBRRRBBB

Sample Output2:
4

解释：先使用魔法2令k=12，得到BBBRRRBBBRRR，然后使用魔法2令k=9，得到RRRBBBRRRRRR，
然后使用魔法2令k=6，得到BBBRRRRRRRRR，然后使用魔法2令k=3，得到RRRRRRRRRRRR。
共使用了4次魔法
```

**代码**

```python
s=input()
n=len(s)
red=[0]*n
blue=[0]*n
if s[0]=="R":
    red[0]=0
    blue[0]=1
else:
    red[0]=1
    blue[0]=0
for i in range(1,n):
    if s[i]=="R":
        red[i]=min(red[i-1],blue[i-1]+1)
        blue[i]=min(red[i-1],blue[i-1])+1
    else:
        red[i]=min(red[i-1],blue[i-1])+1
        blue[i]=min(red[i-1]+1,blue[i-1])
print(red[n-1])
```

# 其他

## 幸福的寒假生活：双层遍历

**描述**

p大计划延长2021-2022学年度的寒假——从1月7日到2月20日共45天。Casper和Emily得知这个消息后非常激动，迫不及待地开始规划起寒假日程，希望能从中获得满满的幸福。

现在，可供Casper选择的有n个浪漫计划，包括开始日期和结束日期，以及通过参加这个计划能得到的幸福值，请计算Casper在这个假期能够获得的最大幸福值。

注意所有参加的计划不能有任何时间上的重叠，在第x天结束的计划和在第x天开始的计划不可同时选择；此外，由于Emily必须在2月21日返校，所以不能参加在2月20日之后结束的计划。

**输入**

第一行包含一个整数n，表示有n个浪漫计划（n<200）

紧接着的n行，每一行由3个部分组成，首先是第 i 个计划开始的日期 si，然后是结束日期 ei，最后是Casper和Emily在这个计划中能获得的幸福值 vi，以空格分隔。输入保证每个计划的时长都在1天至10天的范围内，且计划的开始日期都在寒假范围内。

**输出**

输出他们所能获得的最大幸福值

样例输入

```
Sample Input1:
3
1.8 1.27 100
1.23 1.27 20
2.4 2.8 81

Sample Output1:
181
```

样例输出

```
Sample Input2:
5
1.8 1.27 100
1.10 1.19 90
1.23 1.27 20
2.4 2.8 81
2.10 2.23 100

Sample Output2:
191
```

**代码**

```
def reflect(x):
    month,date=map(int,x.split("."))
    return (month-1)*31+date-7
n=int(input())
plan=[]
for _ in range(n):
    s,e,h=input().split()
    s,e,h=reflect(s),reflect(e),int(h)
    if e<=44:
        plan.append((s,e,h))
dp=[0]*45
for i in range(45):
    dp[i]=dp[i-1]
    for j in plan:
        if j[1]==i:
            dp[i]=max(dp[i],dp[j[0]-1]+j[2])
print(dp[-1])
```

# 核电站

**描述**

一个核电站有 N 个放核物质的坑，坑排列在一条直线上。如果连续 M 个坑中放入核物质，则会发生爆炸，于是，在某些坑中可能不放核物质。

任务：对于给定的 N 和 M ，求不发生爆炸的放置核物质的方案总数。

**输入**

只有一行，两个正整数 N，M (1 < N < 50, 2 <= M <= 5)。

**输出**

一个正整数 S，表示方案总数。

样例输入

```
4 3
```

样例输出

```
13
```

**代码**

```
n,m=map(int,input().split())
dp=[0]*(n+1)
dp[0]=1
for i in range(1,n+1):
    if i<m:
        dp[i]=dp[i-1]*2
    elif i==m:
        dp[i]=dp[i-1]*2-1
    else:
        dp[i]=dp[i-1]*2-dp[i-m-1]
print(dp[n])
```

# 图搜索

## 1、DFS（深度优先搜索）

### 1.1 迷宫可行路径数

https://sunnywhy.com/sfbj/8/1/313

现有一个 n*m 大小的迷宫，其中`1`表示不可通过的墙壁，`0`表示平地。每次移动只能向上下左右移动一格（不允许移动到曾经经过的位置），且只能移动到平地上。求从迷宫左上角到右下角的所有可行路径的条数。

**输入**

第一行两个整数n、m（2<=n<=5,2<=m<=5），分别表示迷宫的行数和列数；

接下来 n 行，每行 m 个整数（值为`0`或`1`），表示迷宫。

**输出**

一个整数，表示可行路径的条数。

样例1

输入

```
3 3
0 0 0
0 1 0
0 0 0
```

输出

```
2
```

解释

假设左上角坐标是(1,1)，行数增加的方向为增长的方向，列数增加的方向为增长的方向。

可以得到从左上角到右下角有两条路径：

1. (1,1)=>(1,2)=>(1,3)=>(2,3)=>(3,3)
2. (1,1)=>(2,1)=>(3,1)=>(3,2)=>(3,3)

## 加保护圈，原地修改

```
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def dfs(maze,x,y):
    global cnt
    for i in range(4):
        nx=x+dx[i]
        ny=y+dy[i]
        if maze[nx][ny]==1:
            continue
        if maze[nx][ny]=="e":
            cnt+=1
            continue
        maze[x][y]=1
        dfs(maze,nx,ny)
        maze[x][y]=0
    return
n,m=map(int,input().split())
maze=[[1]*(m+2)]
for _ in range(n):
    maze.append([1]+list(map(int,input().split()))+[1])
maze.append([1]*(m+2))
maze[1][1]="s"
maze[n][m]="e"
cnt=0
dfs(maze,1,1)
print(cnt)
```

### 辅助visited空间

```
dx=[-1,0,1,0]
dy=[0,1,0,-1]
def is_valid(x,y):
    return 0<=x<n and 0<=y<m and maze[x][y]==0 and (not visited[x][y])
def dfs(x,y):
    global cnt
    if x==n-1 and y==m-1:
        cnt+=1
        return
    visited[x][y]=True
    for i in range(4):
        nx=x+dx[i]
        ny=y+dy[i]
        if is_valid(nx,ny):
            dfs(nx,ny)
    visited[x][y]=False
n,m=map(int,input().split())
maze=[]
for _ in range(n):
    maze.append(list(map(int,input().split())))
visited=[[False]*m for _ in range(n)]
cnt=0
dfs(0,0)
print(cnt)
```

## 1.2 指定步数的迷宫问题

https://sunnywhy.com/sfbj/8/1/314

现有一个 n*m 大小的迷宫，其中`1`表示不可通过的墙壁，`0`表示平地。每次移动只能向上下左右移动一格（不允许移动到曾经经过的位置），且只能移动到平地上。现从迷宫左上角出发，问能否在恰好第步时到达右下角。

**输入**

第一行三个整数n、m、k（2<=n<=5,2<=m<=5,2<=k<=n*m），分别表示迷宫的行数、列数、移动的步数；

接下来行，每行个整数（值为`0`或`1`），表示迷宫。

**输出**

如果可行，那么输出`Yes`，否则输出`No`。

样例1

输入

```
3 3 4
0 1 0
0 0 0
0 1 0
```

输出

```
Yes
```

解释

假设左上角坐标是(1,1)，行数增加的方向为增长的方向，列数增加的方向为增长的方向。

可以得到从左上角到右下角的步数为 `4` 的路径为：(1,1)=>(2,1)=>(2,2)=>(2,3)=>(3,3)。

样例2

输入

```
3 3 6
0 1 0
0 0 0
0 1 0
```

输出

```
No
```

解释

由于不能移动到曾经经过的位置，因此无法在恰好第 `6` 步时到达右下角。

## 加保护圈，原地修改

```
dx=[-1,1,0,0]
dy=[0,0,-1,1]
canreach=False
def dfs(maze,x,y,step):
    global canreach
    for i in range(4):
        if canreach:
            return
        nx=x+dx[i]
        ny=y+dy[i]
        if maze[nx][ny]=="e":
            if step==k-1:
                canreach=True
                return
        elif maze[nx][ny]==0:
            if step<k:
                maze[x][y]=1
                dfs(maze,nx,ny,step+1)
                maze[x][y]=0
    return
n,m,k=map(int,input().split())
maze=[[1]*(m+2)]
for _ in range(n):
    maze.append([1]+list(map(int,input().split()))+[1] )
maze.append([1]*(m+2))
maze[n][m]="e"
```

```
dfs(maze,1,1,0)
if canreach:
    print("Yes")
else:
    print("No")
```

### 辅助visited空间

```
# gpt translated version of the C++ code
MAXN = 5
n, m, k = map(int, input().split())
maze = []
for _ in range(n):
    row = list(map(int, input().split()))
    maze.append(row)

visited = [[False for _ in range(m)] for _ in range(n)]
canReach = False

MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def is_valid(x, y):
    return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and not visited[x][y]

def DFS(x, y, step):
    global canReach
    if canReach:
        return
    if x == n - 1 and y == m - 1:
        if step == k:
            canReach = True
        return
    visited[x][y] = True
    for i in range(MAXD):
        nextX = x + dx[i]
        nextY = y + dy[i]
        if step < k and is_valid(nextX, nextY):
            DFS(nextX, nextY, step + 1)
    visited[x][y] = False

DFS(0, 0, 0)
print("Yes" if canReach else "No")
```

## 1.3 矩阵最大权值

https://sunnywhy.com/sfbj/8/1/315

现有一个 n*m 大小的矩阵，矩阵中的每个元素表示该位置的权值。现需要从矩阵左上角出发到达右下角，每次移动只能向上下左右移动一格（不允许移动到曾经经过的位置）。求最后到达右下角时路径上所有位置的权值之和的最大值。

**输入**

第一行两个整数 n、m（2<=n<=5,2<=m<=5），分别表示矩阵的行数和列数；

接下来 n 行，每行 m 个整数（-100<=m<=100），表示矩阵每个位置的权值。

**输出**

一个整数，表示权值之和的最大值。

样例1

输入

```
2 2
1 2
3 4
```

输出

```
8
```

解释

从左上角到右下角的最大权值之和为。

## 加保护圈，原地修改

```
dx=[-1,1,0,0]
dy=[0,0,-1,1]
maximum=-2500
def dfs(maze,x,y,weight):
    global maximum
    for i in range(4):
        nx=x+dx[i]
        ny=y+dy[i]
        if maze[nx][ny]=="e":
            maximum=max(maximum,weight+e)
        elif maze[nx][ny]==-200:
            continue
        else:
            m=maze[x][y]
            maze[x][y]=-200
            dfs(maze, nx, ny, weight+maze[nx][ny])
            maze[x][y]=m
    return
n,m=map(int,input().split())
maze=[[-200]*(m+2)]
for _ in range(n):
    maze.append([-200]+list(map(int,input().split()))+[-200] )
maze.append([-200]*(m+2))
s=maze[1][1]
e=maze[n][m]
maze[n][m]="e"
dfs(maze,1,1,s)
```

```
print(maximum)
```

### 辅助visited空间

```
# gpt translated version of the C++ code
MAXN = 5
INF = float('inf')
n, m = map(int, input().split())
maze = []
for _ in range(n):
    row = list(map(int, input().split()))
    maze.append(row)

visited = [[False for _ in range(m)] for _ in range(n)]
maxValue = -INF

MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def is_valid(x, y):
    return 0 <= x < n and 0 <= y < m and not visited[x][y]

def DFS(x, y, nowValue):
    global maxValue
    if x == n - 1 and y == m - 1:
        if nowValue > maxValue:
            maxValue = nowValue
        return
    visited[x][y] = True
    for i in range(MAXD):
        nextX = x + dx[i]
        nextY = y + dy[i]
        if is_valid(nextX, nextY):
            nextValue = nowValue + maze[nextX][nextY]
            DFS(nextX, nextY, nextValue)
    visited[x][y] = False

DFS(0, 0, maze[0][0])
print(maxValue)
```

## 1.4 矩阵最大权值路径

https://sunnywhy.com/sfbj/8/1/316

现有一个 n*m 大小的矩阵，矩阵中的每个元素表示该位置的权值。现需要从矩阵左上角出发到达右下角，每次移动只能向上下左右移动一格（不允许移动到曾经经过的位置）。假设左上角坐标是(1,1)，行数增加的方向为增长的方向，列数增加的方向为增长的方向。求最后到达右下角时路径上所有位置的权值之和最大的路径。

**输入**

第一行两个整数 n、m（2<=n<=5,2<=m<=5），分别表示矩阵的行数和列数；

接下来 n 行，每行 m 个整数（-100<=m<=100），表示矩阵每个位置的权值。

**输出**

从左上角的坐标开始，输出若干行（每行两个整数，表示一个坐标），直到右下角的坐标。

数据保证权值之和最大的路径存在且唯一。

样例1

输入

```
2 2
1 2
3 4
```

输出

```
1 1
2 1
2 2
```

解释

显然当路径是(1,1)=>(2,1)=>(2,2)时，权值之和最大，即 1+3+4 = 8。

## 加保护圈，原地修改

```
dx=[-1,1,0,0]
dy=[0,0,-1,1]
maximum=-2500
def dfs(maze,x,y,weight):
    global maximum,path,route
    for i in range(4):
        nx=x+dx[i]
        ny=y+dy[i]
        if maze[nx][ny]=="e":
            if weight+e>maximum:
                maximum=weight+e
                route=list(path)
        elif maze[nx][ny]==-200:
            continue
        else:
            path.append([nx,ny])
            tmp=maze[x][y]
            maze[x][y]=-200
            dfs(maze, nx, ny, weight+maze[nx][ny])
            path.pop()
            maze[x][y]=tmp
    return
n,m=map(int,input().split())
maze=[[-200]*(m+2)]
for _ in range(n):
```

```
    maze.append([-200]+list(map(int,input().split()))+[-200] )
maze.append([-200]*(m+2))
s=maze[1][1]
e=maze[n][m]
maze[n][m]="e"
route=[]
path=[[1,1]]
dfs(maze,1,1,s)
route.append([n,m])
for i in route:
    print(" ".join(map(str,i)))
```

## 辅助visited空间

```
# gpt translated version of the C++ code
MAXN = 5
INF = float('inf')
n, m = map(int, input().split())
maze = []
for _ in range(n):
    row = list(map(int, input().split()))
    maze.append(row)

visited = [[False for _ in range(m)] for _ in range(n)]
maxValue = -INF
tempPath, optPath = [], []

MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def is_valid(x, y):
    return 0 <= x < n and 0 <= y < m and not visited[x][y]

def DFS(x, y, nowValue):
    global maxValue, tempPath, optPath
    if x == n - 1 and y == m - 1:
        if nowValue > maxValue:
            maxValue = nowValue
            optPath = list(tempPath)
        return
    visited[x][y] = True
    for i in range(MAXD):
        nextX = x + dx[i]
        nextY = y + dy[i]
        if is_valid(nextX, nextY):
            nextValue = nowValue + maze[nextX][nextY]
            tempPath.append((nextX, nextY))
            DFS(nextX, nextY, nextValue)
            tempPath.pop()
    visited[x][y] = False

tempPath.append((0, 0))
```

```
DFS(0, 0, maze[0][0])
for pos in optPath:
    print(pos[0] + 1, pos[1] + 1)
```

## 1.5 迷宫最大权值

https://sunnywhy.com/sfbj/8/1/317

题目描述

现有一个大小的迷宫，其中`1`表示不可通过的墙壁，`0`表示平地。现需要从迷宫左上角出发到达右下角，每次移动只能向上下左右移动一格（不允许移动到曾经经过的位置），且只能移动到平地上。假设迷宫中每个位置都有权值，求最后到达右下角时路径上所有位置的权值之和的最大值。

**输入**

第一行两个整数$n、m \hspace{1em} (2 \le n \le5, 2 \le m \le 5)$，分别表示矩阵的行数和列数；

接下来 n 行，每行个 m 整数（值为`0`或`1`），表示迷宫。

再接下来行，每行个整数（$-100 \le 整数 \le 100$），表示迷宫每个位置的权值。

**输出**

一个整数，表示权值之和的最大值。

样例1

输入

```
3 3
0 0 0
0 1 0
0 0 0
1 2 3
4 5 6
7 8 9
```

输出

```
29
```

解释：从左上角到右下角的最大权值之和为 1+4+7+8+9 = 29。

### 加保护圈，原地修改

```
dx=[-1,1,0,0]
dy=[0,0,-1,1]
maximum=-2500
def dfs(maze,x,y,weight):
    global maximum
    for i in range(4):
        nx=x+dx[i]
        ny=y+dy[i]
        if maze[nx][ny]=="e":
```

```
            maximum=max(maximum,weight+mx[n][m])
        elif maze[nx][ny]==0:
            maze[x][y]=1
            dfs(maze, nx, ny, weight+mx[nx][ny])
            maze[x][y]=0
    return
n,m=map(int,input().split())
maze=[[1]*(m+2)]
for _ in range(n):
    maze.append([1]+list(map(int,input().split()))+[1] )
maze.append([1]*(m+2))
mx=[[0]*(m+2)]
for _ in range(n):
    mx.append([0]+list(map(int,input().split()))+[0])
mx.append([0]*(m+2))
maze[n][m]="e"
dfs(maze,1,1,mx[1][1])
print(maximum)
```

## 辅助visited空间

```
# gpt translated version of the C++ code
MAXN = 5
INF = float('inf')
n, m = map(int, input().split())
maze = [list(map(int, input().split())) for _ in range(n)]
w = [list(map(int, input().split())) for _ in range(n)]
visited = [[False] * m for _ in range(n)]
maxValue = -INF

MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def is_valid(x, y):
    return 0 <= x < n and 0 <= y < m and not maze[x][y] and not visited[x][y]

def dfs(x, y, nowValue):
    global maxValue
    if x == n - 1 and y == m - 1:
        if nowValue > maxValue:
            maxValue = nowValue
        return
    visited[x][y] = True
    for i in range(MAXD):
        nextX = x + dx[i]
        nextY = y + dy[i]
        if is_valid(nextX, nextY):
            nextValue = nowValue + w[nextX][nextY]
            dfs(nextX, nextY, nextValue)
    visited[x][y] = False

dfs(0, 0, w[0][0])
```

```
print(maxValue)
```

# 2、BFS（广度优先搜索）

## 2.1 数字操作

https://sunnywhy.com/sfbj/8/2/318

从整数`1`开始，每轮操作可以选择将上轮结果加`1`或乘`2`。问至少需要多少轮操作才能达到指定整数。

输入描述

一个整数 $n \hspace{1em} (2 \le n \le 10^5)$，表示需要达到的整数。

输出描述

输出一个整数，表示至少需要的操作轮数。

样例1

输入

复制

```
7
```

输出

复制

```
4
```

解释

第`1`轮：1 + 1 = 2

第`2`轮：2 + 1 =3

第`3`轮：3 * 2 = 6

第`4`轮：6 + 1 = 7

因此至少需要操作`4`轮。

**代码**

```
from collections import deque
n=int(input())
in_queue=[0]*(n+1)
def bfs(n):
    q=deque()
    q.append(1)
    in_queue[1]=1
    step=0
    while True:
```

```
        cnt=len(q)
        for _ in range(cnt):
            front=q.popleft()
            if front==n:
                return step
            if front*2<=n and not in_queue[front*2]:
                q.append(front*2)
                in_queue[front*2]=1
            if front+1<=n and not in_queue[front+1]:
                q.append(front+1)
                in_queue[front+1]=1
        step+=1
print(bfs(n))
```

## 2.2 矩阵中的块

https://sunnywhy.com/sfbj/8/2/319

题目描述

现有一个 n*m 的矩阵，矩阵中的元素为`0`或`1`。然后进行如下定义：

1. 位置(x,y)与其上下左右四个位置 $(x,y + 1)、(x,y - 1)、(x + 1,y)、(x-1,y)$ 是相邻的；
2. 如果位置 (x1,y1) 与位置 (x2,y2) 相邻，且位置 (x2,y2) 与位置 (x3,y3) 相邻，那么称位置(x1,y1)与位置(x3,y3)也相邻；
3. 称个数尽可能多的相邻的`1`构成一个“块”。

求给定的矩阵中“块”的个数。

**输入**

第一行两个整数 n、m（2<=n<=100,2<=m<=100），分别表示矩阵的行数和列数；

接下来 n 行，每行 m 个`0`或`1`（用空格隔开），表示矩阵中的所有元素。

**输出**

输出一个整数，表示矩阵中“块”的个数。

样例1

输入

```
6 7
0 1 1 1 0 0 1
0 0 1 0 0 0 0
0 0 0 0 1 0 0
0 0 0 1 1 1 0
1 1 1 0 1 0 0
1 1 1 1 0 0 0
```

输出

```
4
```

### in_queue列表

```
from collections import deque
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def bfs(mx,x,y):
    q=deque()
    q.append([x,y])
    while q:
        front=q.popleft()
        for i in range(4):
            nx=front[0]+dx[i]
            ny=front[1]+dy[i]
            if mx[nx][ny]==1 and not in_queue[nx][ny]:
                q.append([nx,ny])
                in_queue[nx][ny]=1
    return
n,m=map(int,input().split())
mx=[[0]*(m+2)]
for _ in range(n):
    mx.append([0]+list(map(int,input().split()))+[0])
mx.append([0]*(m+2))
in_queue=[[0]*(m+2) for _ in range(n+2)]
cnt=0
for i in range(1,n+1):
    for j in range(1,m+1):
        if mx[i][j]==1 and not in_queue[i][j]:
            bfs(mx,i,j)
            cnt+=1
print(cnt)
```

### 直接修改矩阵（bfs矩阵可直接修改矩阵）

```
from collections import deque
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def bfs(maze,x,y):
    q=deque()
    q.append((x,y))
    while q:
        front=q.popleft()
        for i in range(4):
            nx=front[0]+dx[i]
            ny=front[1]+dy[i]
            if maze[nx][ny]==1:
                maze[nx][ny]=0
                q.append((nx,ny))
    return
n,m=map(int,input().split())
maze=[[0]*(m+2)]
for _ in range(n):
    maze.append([0]+list(map(int,input().split()))+[0])
maze.append([0]*(m+2))
cnt=0
```

```
for i in range(1,n+1):
    for j in range(1,m+1):
        if maze[i][j]==1:
            cnt+=1
            bfs(maze,i,j)
print(cnt)
```

## 2.3 迷宫问题

https://sunnywhy.com/sfbj/8/2/320

现有一个 n*m 大小的迷宫，其中`1`表示不可通过的墙壁，`0`表示平地。每次移动只能向上下左右移动一格，且只能移动到平地上。求从迷宫左上角到右下角的最小步数。

**输入**

第一行两个整数 n、m(2<=n<=100,2<=m<=100)，分别表示迷宫的行数和列数；

接下来 n 行，每行 m 个整数（值为`0`或`1`），表示迷宫。

**输出**

输出一个整数，表示最小步数。如果无法到达，那么输出`-1`。

样例1

输入

```
3 3
0 1 0
0 0 0
0 1 0
```

输出

```
4
```

解释: 假设左上角坐标是(1,1)，行数增加的方向为增长的方向，列数增加的方向为增长的方向。

可以得到从左上角到右下角的前进路线：(1,1)=>(2,1)=>(2,2)=>(2,3)=>(3,3)。

因此最少需要`4`步。

样例2

输入

```
3 3
0 1 0
0 1 0
0 1 0
```

输出

```
-1
```

解释: 显然从左上角无法到达右下角。

**代码**

```
from collections import deque
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def bfs(mx,x,y):
    q=deque()
    q.append([x,y])
    in_queue[x][y]=1
    step=0
    while q:
        cnt=len(q)
        for _ in range(cnt):
            front=q.popleft()
            for i in range(4):
                nx=front[0]+dx[i]
                ny=front[1]+dy[i]
                if mx[nx][ny]=="e":
                    return step+1
                if mx[nx][ny]==0 and not in_queue[nx][ny]:
                    q.append([nx, ny])
                    in_queue[nx][ny]=1
        step+=1
    return -1
n,m=map(int,input().split())
mx=[[1]*(m+2)]
for _ in range(n):
    mx.append([1]+list(map(int,input().split()))+[1])
mx.append([1]*(m+2))
mx[n][m]="e"
in_queue=[[0]*(m+2) for _ in range(n+2)]
print(bfs(mx,1,1))
```

## 2.4 迷宫最短路径

https://sunnywhy.com/sfbj/8/2/321

现有一个 n*m 大小的迷宫，其中`1`表示不可通过的墙壁，`0`表示平地。每次移动只能向上下左右移动一格，且只能移动到平地上。假设左上角坐标是(1,1)，行数增加的方向为增长的方向，列数增加的方向为增长的方向，求从迷宫左上角到右下角的最少步数的路径。

**输入**

第一行两个整数 n、m(2<=n<=100,2<=m<=100)，分别表示迷宫的行数和列数；

接下来 n 行，每行 m 个整数（值为`0`或`1`），表示迷宫。

**输出**

从左上角的坐标开始，输出若干行（每行两个整数，表示一个坐标），直到右下角的坐标。

数据保证最少步数的路径存在且唯一。

样例1

输入

```
3 3
0 1 0
0 0 0
0 1 0
```

输出

```
1 1
2 1
2 2
2 3
3 3
```

解释

假设左上角坐标是(1,)，行数增加的方向为增长的方向，列数增加的方向为增长的方向。

可以得到从左上角到右下角的最少步数的路径为：(1,1)=>(2,1)=>(2,2)=>(2,3)=>(3,3)。

**代码**

```
from collections import deque
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def bfs(maze,x,y):
    q=deque()
    q.append((x,y))
    in_queue=[[0]*(m+2) for _ in range(n+2)]
    while q:
        front=q.popleft()
        for i in range(4):
            nx=front[0]+dx[i]
            ny=front[1]+dy[i]
            if nx==n and ny==m:
                pre[nx][ny]=(front[0],front[1])
                return
            if maze[nx][ny]==0 and not in_queue[nx][ny]:
                pre[nx][ny]=(front[0],front[1])
                q.append((nx, ny))
                in_queue[nx][ny]=1
def printpath(x,y):
    if x==1 and y==1:
        print(1,1)
        return
    prex,prey=pre[x][y][0],pre[x][y][1]
    printpath(prex,prey)
    print(x,y)
    return
n,m=map(int,input().split())
maze=[[1]*(m+2)]
for _ in range(n):
    maze.append([1]+list(map(int,input().split()))+[1])
maze.append([1]*(m+2))
pre=[[0]*(m+2) for _ in range(n+2)]
bfs(maze,1,1)
```

```
printpath(n,m)
```

## 2.5 走山路

**描述**

某同学在一处山地里，地面起伏很大，他想从一个地方走到另一个地方，并且希望能尽量走平路。
现有一个m*n的地形图，图上是数字代表该位置的高度，"#"代表该位置不可以经过。
该同学每一次只能向上下左右移动，每次移动消耗的体力为移动前后该同学所处高度的差的绝对值。现在给出该同学出发的地点和目的地，需要你求出他最少要消耗多少体力。

**输入**

第一行是m,n,p，m是行数，n是列数，p是测试数据组数
接下来m行是地形图
再接下来n行每行前两个数是出发点坐标（前面是行，后面是列），后面两个数是目的地坐标（前面是行，后面是列）（出发点、目的地可以是任何地方，出发点和目的地如果有一个或两个在"#"处，则将被认为是无法达到目的地）

**输出**

n行，每一行为对应的所需最小体力，若无法达到，则输出"NO"

样例输入

```
4 5 3
0 0 0 0 0
0 1 1 2 3
# 1 0 0 0
0 # 0 0 0
0 0 3 4
1 0 1 4
3 4 3 0
```

样例输出

```
2
3
NO

解释：
第一组：从左上角到右下角，要上1再下来，所需体力为2
第二组：一直往右走，高度从0变为1，再变为2，再变为3，消耗体力为3
第三组：左下角周围都是"#"，不可以经过，因此到不了
```

### dfs:易超时

```
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def dfs(mx,x1,y1,x2,y2,st):
    global ans
    if x1==x2 and y1==y2:
        ans.append(0)
        return
    if mx[x1][y1]=="#" or mx[x2][y2]=="#":
```

```
        return
    for i in range(4):
        nx=x1+dx[i]
        ny=y1+dy[i]
        if nx==x2 and ny==y2:
            ans.append(st+abs(int(mx[nx][ny])-int(mx[x1][y1])))
            continue
        if mx[nx][ny]=="#":
            continue
        dst=abs(int(mx[nx][ny])-int(mx[x1][y1]))
        tmp=mx[x1][y1]
        mx[x1][y1]="#"
        dfs(mx,nx,ny,x2,y2,st+dst)
        mx[x1][y1]=tmp
    return
m,n,p=map(int,input().split())
mx=[["#"]*(n+2)]
for _ in range(m):
    mx.append(["#"]+input().split()+["#"])
mx.append(["#"]*(n+2))
for _ in range(p):
    x1,y1,x2,y2=map(int,input().split())
    ans=[]
    dfs(mx,x1+1,y1+1,x2+1,y2+1,0)
    if ans:
        print(min(ans))
    else:
        print("NO")
```

## bfs+deque

```
from collections import deque
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def bfs(mx,x1,y1,x2,y2):
    if mx[x1][y1]=="#" or mx[x2][y2]=="#":
        return "NO"
    q=deque()
    q.append((x1,y1))
    ans={(x1,y1):0}
    while q:
        front=q.popleft()
        for i in range(4):
            nx=front[0]+dx[i]
            ny=front[1]+dy[i]
            if mx[nx][ny]=="#":
                continue
            tmp=ans[(front[0], front[1])]+abs(int(mx[nx][ny])-int(mx[front[0]]
[front[1]]))
            if not (nx, ny) in ans.keys() or tmp<ans[(nx,ny)]:
                ans[(nx, ny)]=tmp
                q.append((nx,ny))
    if (x2, y2) in ans.keys():
        return ans[(x2, y2)]
    return "NO"
```

```
m,n,p=map(int,input().split())
mx=[["#"]*(n+2)]
for _ in range(m):
    mx.append(["#"]+input().split()+["#"])
mx.append(["#"]*(n+2))
for _ in range(p):
    x1,y1,x2,y2=map(int,input().split())
    print(bfs(mx,x1+1,y1+1,x2+1,y2+1))
```

## bfs+heap

```
from heapq import heappop,heappush
dx=[-1,1,0,0]
dy=[0,0,-1,1]
def bfs(mx,x1,y1,x2,y2):
    if mx[x1][y1]=="#" or mx[x2][y2]=="#":
        return "NO"
    q=[(0,x1,y1)]
    visited=set()
    while q:
        t,x,y=heappop(q)
        if x==x2 and y==y2:
            return t
        visited.add((x,y))
        for i in range(4):
            nx=x+dx[i]
            ny=y+dy[i]
            if mx[nx][ny]=="#":
                continue
            tmp=t+abs(int(mx[nx][ny])-int(mx[x][y]))
            if not (nx, ny) in visited:
                heappush(q,(tmp,nx,ny))
    return "NO"
m,n,p=map(int,input().split())
mx=[["#"]*(n+2)]
for _ in range(m):
    mx.append(["#"]+input().split()+["#"])
mx.append(["#"]*(n+2))
for _ in range(p):
    x1,y1,x2,y2=map(int,input().split())
    print(bfs(mx,x1+1,y1+1,x2+1,y2+1))
```