

20241119-Week11 搜索专题

Updated 1408 GMT+8 Nov 19 2024

2024 fall, Complied by Hongfei Yan

Log:

2024/11/13 部分内容取自, https://github.com/GMyhf/2023fall-cs101/blob/main/searching_questions.md

0 Introduction

矩阵、迷宫和树都是图的特例。图搜索包括深度优先搜索（Depth First Search, DFS）和广度优先搜索（Breath First Search, BFS），是通过穷举法的思路来教电脑如何最有效地走迷宫。

在解决如何避免漏算或者陷入无限循环的问题时，DFS是从起点开始走，遇到分岔路进行标记，并沿着第一个分岔路继续前进；遇到死路则返回上一个分岔路口，选择下一个分叉路继续探寻，这样最终能够达到终点。这里使用了递归的概念，通过将“路口”、“分叉路”和“死路”等条件编写成代码，指导电脑走迷宫。这种一路走到底，撞到障碍才回头的方式称为深度优先搜索。BFS则是一种“探路型”走迷宫的策略，首先找到与入口相连的所有分叉路，然后逐一探寻这些分叉路所连接的分叉路，以此类推，直到找到出口位置。找出所有可能的分叉路，就称为广度优先搜索。如果搜索超时，可以考虑进行剪枝，以避免搜索不满足约束条件的路径。

当讨论图论时，矩阵、迷宫和树都是图的特例。

1. 矩阵:

- **说明:** 在图论中，矩阵通常指邻接矩阵或者关联矩阵，它们用于表示图的结构和连接关系。
- **示例:** 例如，以下是一个简单的无向图的邻接矩阵表示：

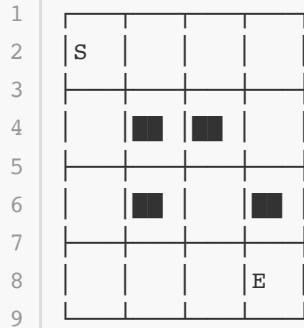
1		1		2		3		4			
<hr/>											
3		1		0		1		1		0	
<hr/>											
5		2		1		0		1		1	
<hr/>											
7		3		1		1		0		1	
<hr/>											
9		4		0		1		1		0	
<hr/>											
10											

在这个邻接矩阵中，行和列代表图中的节点，相应的值表示节点之间的连接关系。

2. 迷宫:

- **说明:** 迷宫可以被视为一个特殊的图，其中包含了节点（代表迷宫的房间或位置）和边（代表可通行的路径）。

- 示例: 以下是一个简单的迷宫示例:



在这个迷宫中, S代表起点, E代表终点, ■代表墙壁, 玩家需要在迷宫中寻找一条从起点到终点的路径。

3. 树:

- 说明: 树是一种无环连通图, 其中任意两个顶点间存在唯一路径。
- 示例: 以下是一个简单的树示例:



这是一个简单的树, 其中每个节点具有唯一的父节点 (除了根节点), 并且形成了一个无环的结构。

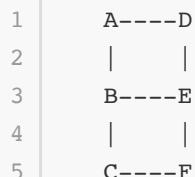
当讨论图论时, 除了矩阵、迷宫和树, 还有一些其他常见的图的特例, 包括:

4. 完全图 (Complete Graph) :

- 完全图是指每一对不同的顶点之间都有一条边相连的图。
- 示例: 如果一个图有 n 个顶点, 并且每一对顶点之间都有一条边相连, 那么这个图就是一个完全图, 通常用 K_n 表示, 其中 n 表示顶点的个数。

5. 二部图 (Bipartite Graph) :

- 二部图是指图的所有顶点可以被分成两个不相交的子集, 使得同一个子集内的顶点不相连。
- 示例: 下图就是一个简单的二部图, 它的顶点可以被分成两个子集 {A, B, C} 和 {D, E, F}, 使得同一个子集内的顶点不相连。



6. 有向无环图 (Directed Acyclic Graph, DAG) :

- 有向无环图是指图中的边都是有方向的, 并且不存在任何环路的图。

- 示例：许多调度和计划问题都可以建模为有向无环图，例如任务的依赖关系图。

《算法笔记》第8章

1 深度优先搜索(DFS)

设想我们现在以第一视角身处一个巨大的迷宫当中，没有上帝视角，没有通信设施，更没有热血动漫里的奇迹，有的只是四周长得一样的墙壁。于是，我们只能自己想办法走出去。如果迷失了内心，随便乱走，那么很可能被四周完全相同的景色绕晕在其中，这时只能放弃所谓的侥幸，而去采取下面这种看上去很盲目但实际上会很有效的方法。

以当前所在位置为起点，沿着一条路向前走，当碰到岔路口时，选择其中一个岔路前进如果选择的这个岔路前方是一条死路，就退回到这个岔路口，选择另一个岔路前进。如果岔路中存在新的岔路口，那么仍然按上面的方法枚举新岔路口的每一条岔路。这样，只要迷宫存在出口，那么这个方法一定能够找到它。可能有读者会问，如果在第一个岔路口处选择了一条没有出路的分支，而这个分支比较深，并且路上多次出现新的岔路口，那么当发现这个分支是个死分支之后，如何退回到最初的这个岔路口？其实方法很简单，只要让右手始终贴着右边的墙壁一路往前走，那么自动会执行上面这个走法，并且最终一定能找到出口。图 8-1 即为使用这个方法走一个简单迷宫的示例。

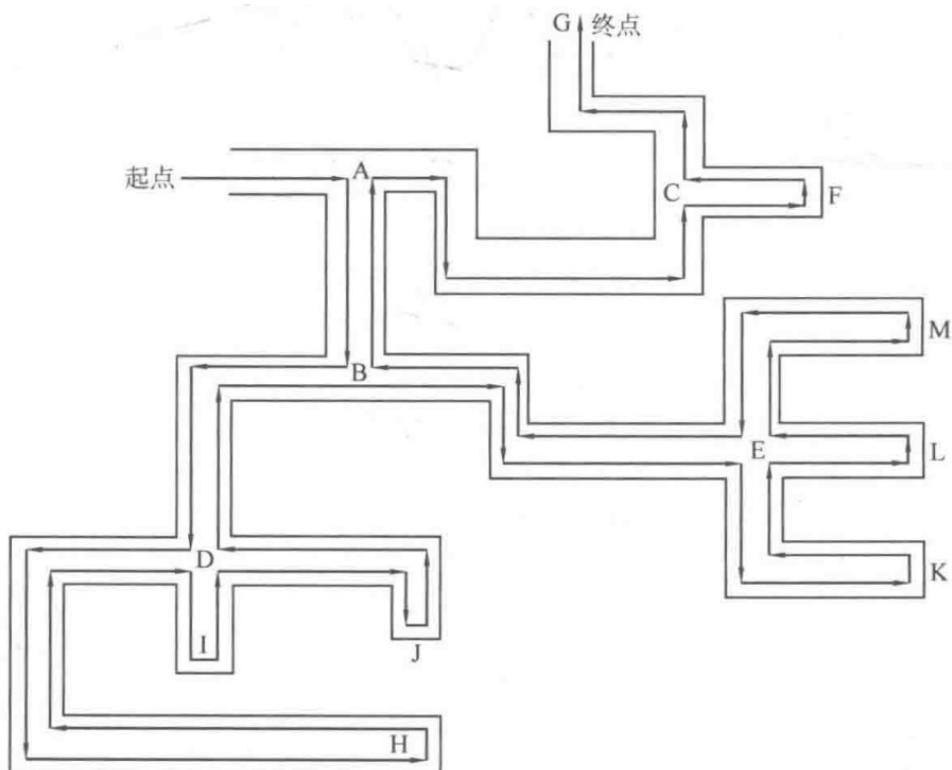


图 8-1 DFS 迷宫示意图

从图 8-1 可知，从起点开始前进，当碰到岔道口时，总是选择其中一条岔路前进（例如图中总是先选择最右手边的岔路），在岔路上如果又遇到新的岔道口，仍然选择新岔道口的其中一条岔路前进，直到碰到死胡同才回退到最近的岔道口选择另一条岔路。也就是说，当碰到岔道口时，总是以“深度”作为前进的关键词，不碰到死胡同就不回头，因此把这种搜索的方式称为深度优先搜索(Depth First Search, **DFS**)。

从迷宫的例子还应该注意到，深度优先搜索会走遍所有路径，并且每次走到死胡同就代表一条完整路径的形成。这就是说，深度优先搜索是一种枚举所有完整路径以遍历所有情况的搜索方法。

深度优先搜索(DFS)可以使用栈来实现。但是实现起来却并不轻松，有没有既容易理解又容易实现的方法呢？有的——递归。现在从 DFS 的角度来看当初求解 Fibonacci 数列的过程。

回顾一下 Fibonacci 数列的定义： $F(0) = 1, F(1) = 1, F(n) = F(n - 1) + F(n - 2)$ ($n \geq 2$)。可以从这个定义中挖掘到，每当将 $F(n)$ 分为两部分 $F(n-1)$ 与 $F(n-2)$ 时，就可以把 $F(n)$ 看作迷宫的岔道口，由它可以到达两个新的关键结点 $F(n-1)$ 与 $F(n-2)$ 。而之后计算 $F(n-1)$ 时，又可以把 $F(n-1)$ 当作在岔道口 $F(n)$ 之下的岔道口。

既然有岔道口，那么一定有死胡同。很容易想象，当访问到 $F(0)$ 和 $F(1)$ 时，就无法再向下递归下去，因此 $F(0)$ 和 $F(1)$ 就是死胡同。这样说来，**递归中的递归式就是岔道口，而递归边界就是死胡同**，这样就可以把如何用递归实现深度优先搜索的过程理解得很清楚。为了使上面的过程更清晰，可以直接来分析递归图(见图 4-3)：可以在递归图中看到，只要 $n > 1$ ， $F(n)$ 就有两个分支，即把 $F(n)$ 当作岔道口；而当 n 为 1 或 0 时， $F(1)$ 与 $F(0)$ 就是迷宫的死胡同，在此处程序就需要返回结果。这样当遍历完所有路径(从顶端的 $F(4)$ 到底层的所有 $F(1)$ 与 $F(0)$) 后，就可以得到 $F(4)$ 的值。

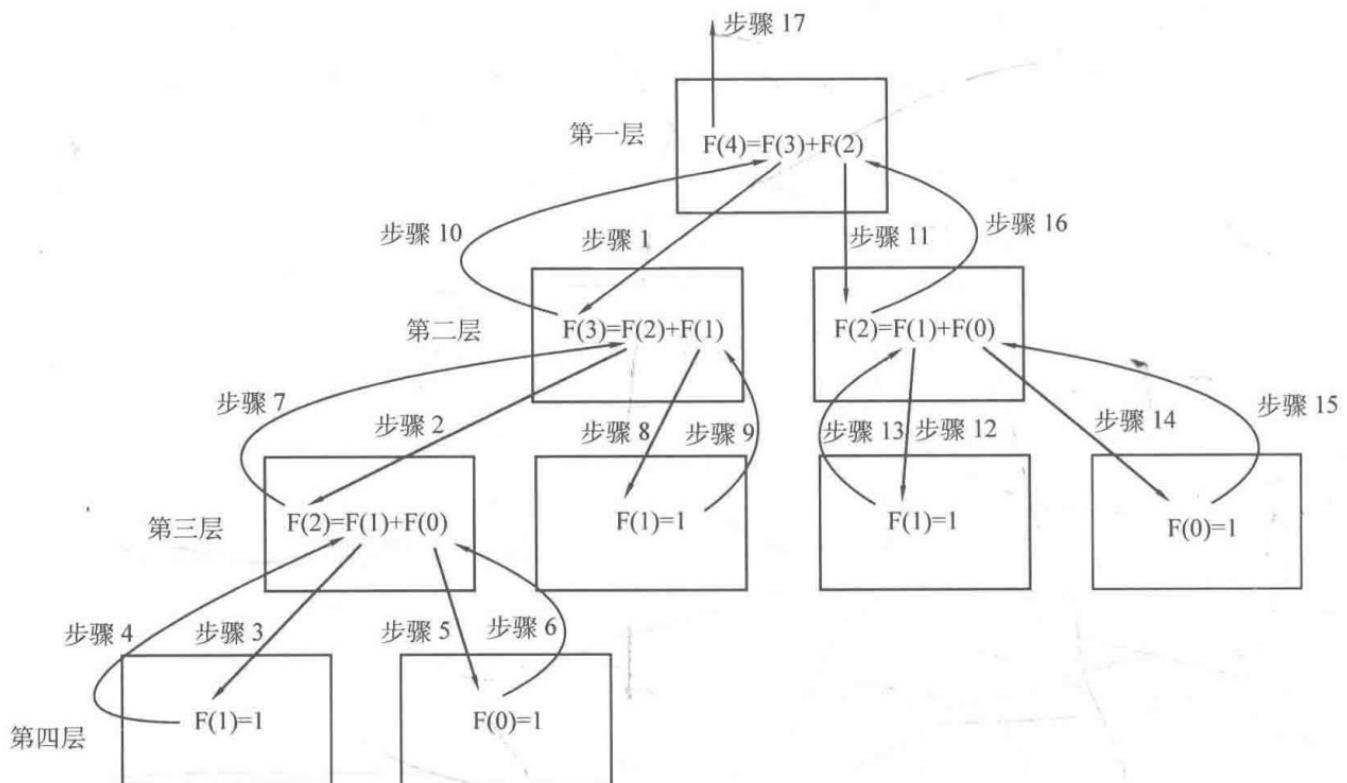


图 4-3 斐波那契数列递归求解示意图

因此，使用递归可以很好地实现深度优先搜索。这个说法并不是说深度优先搜索就是递归，只能说递归是深度优先搜索的一种实现方式，因为使用非递归也是可以实现 DFS 的思想的，但是一般情况下会比递归麻烦。不过，使用递归时，系统会调用一个叫系统栈的东西来存放递归中每一层的状态，因此使用递归来实现 DFS 的本质其实还是栈。

示例：sy313 迷宫可行路径数 简单

<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 ($2 \leq n \leq 5, 2 \leq m \leq 5$)，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

```
1 | 3 3
2 | 0 0 0
3 | 0 1 0
4 | 0 0 0
```

输出

```
1 | 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

```
1 | dx = [-1, 0, 1, 0]
2 | dy = [0, 1, 0, -1]
3 |
4 | def dfs(maze, x, y):
5 |     global cnt
6 |
7 |     for i in range(4):
8 |         nx = x + dx[i]
9 |         ny = y + dy[i]
10 |
11 |         if maze[nx][ny] == 'e':
12 |             cnt += 1
13 |             continue
14 |
15 |         if maze[nx][ny] == 0:
16 |             maze[x][y] = 1
17 |             dfs(maze, nx, ny)
18 |             maze[x][y] = 0
19 |
```

```

20     return
21
22 n, m = map(int, input().split())
23 maze = []
24 maze.append([-1 for x in range(m+2)])
25 for _ in range(n):
26     maze.append([-1 + [int(_) for _ in input().split()] + [-1]])
27 maze.append([-1 for x in range(m+2)])
28
29 maze[1][1] = 's'
30 maze[n][m] = 'e'
31
32 cnt = 0
33 dfs(maze, 1, 1)
34 print(cnt)

```

OJ的pylint是静态检查，有时候报的不对。解决方法有两种，如下：

- 1) 第一行加# pylint: skip-file
- 2) 方法二：如果函数内使用全局变量（变量类型是immutable，如int），则需要在程序最开始声明一下。如果是全局变量是list类型，则不受影响。

辅助visited空间

```

1 MAXN = 5
2 n, m = map(int, input().split())
3 maze = []
4 for _ in range(n):
5     row = list(map(int, input().split()))
6     maze.append(row)
7
8 visited = [[False for _ in range(m)] for _ in range(n)]
9 counter = 0
10
11 MAXD = 4
12 dx = [0, 0, 1, -1]
13 dy = [1, -1, 0, 0]
14
15 def is_valid(x, y):
16     return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and not visited[x][y]
17
18 def DFS(x, y):
19     global counter
20     if x == n - 1 and y == m - 1:
21         counter += 1
22         return
23     visited[x][y] = True
24     for i in range(MAXD):
25         nextX = x + dx[i]

```

```

26     nextY = y + dy[i]
27     if is_valid(nextX, nextY):
28         DFS(nextX, nextY)
29     visited[x][y] = False
30
31 DFS(0, 0)
32 print(counter)
33

```

示例：sy314指定步数的迷宫问题 中等

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

现有一个 $n \times m$ 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 k 步时到达右下角。

输入

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

接下来行，每行 n 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

1	3 3 4
2	0 1 0
3	0 0 0
4	0 1 0

输出

1	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

```
1 | 3 3 6
2 | 0 1 0
3 | 0 0 0
4 | 0 1 0
```

输出

```
1 | No
```

解释

由于不能移动到曾经过的位置，因此无法在恰好第 6 步时到达右下角。

加保护圈，原地修改

```
1 | dx = [-1, 0, 1, 0]
2 | dy = [ 0, 1, 0, -1]
3 |
4 | canReach = False
5 | def dfs(maze, x, y, step):
6 |     global canReach
7 |     if canReach:
8 |         return
9 |
10 |     for i in range(4):
11 |         nx = x + dx[i]
12 |         ny = y + dy[i]
13 |         if maze[nx][ny] == 'e':
14 |             if step==k-1:
15 |                 canReach = True
16 |                 return
17 |
18 |             continue
19 |
20 |         if maze[nx][ny] == 0:
21 |             if step < k:
22 |                 maze[x][y] = -1
23 |                 dfs(maze, nx, ny, step+1)
24 |                 maze[x][y] = 0
25 |
26 |
27 | n, m, k = map(int, input().split())
28 | maze = []
29 | maze.append([-1 for x in range(m+2)])
30 | for _ in range(n):
31 |     maze.append([-1] + [int(_) for _ in input().split()] + [-1])
32 | maze.append([-1 for x in range(m+2)])
33 |
34 | maze[1][1] = 's'
```

```

35 maze[n][m] = 'e'
36
37 dfs(maze, 1, 1, 0)
38 print("Yes" if canReach else "No")

```

辅助visited空间

```

1 MAXN = 5
2 n, m, k = map(int, input().split())
3 maze = []
4 for _ in range(n):
5     row = list(map(int, input().split()))
6     maze.append(row)
7
8 visited = [[False for _ in range(m)] for _ in range(n)]
9 canReach = False
10
11 MAXD = 4
12 dx = [0, 0, 1, -1]
13 dy = [1, -1, 0, 0]
14
15 def is_valid(x, y):
16     return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and not visited[x][y]
17
18 def DFS(x, y, step):
19     global canReach
20     if canReach:
21         return
22     if x == n - 1 and y == m - 1:
23         if step == k:
24             canReach = True
25         return
26     visited[x][y] = True
27     for i in range(MAXD):
28         nextX = x + dx[i]
29         nextY = y + dy[i]
30         if step < k and is_valid(nextX, nextY):
31             DFS(nextX, nextY, step + 1)
32     visited[x][y] = False
33
34 DFS(0, 0, 0)
35 print("Yes" if canReach else "No")
36

```

示例：sy315矩阵最大权值 中等

<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 ($2 \leq n \leq 5, 2 \leq m \leq 5$)，分别表示矩阵的行数和列数；

接下来 n 行，每行 m 个整数 ($-100 \leq$ 整数 ≤ 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

1	2 2
2	1 2
3	3 4

输出

1	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

```
1  dx = [-1, 0, 1, 0]
2  dy = [0, 1, 0, -1]
3
4  maxValue = float("-inf")
5  def dfs(maze, x, y, nowValue):
6      global maxValue
7      if x==n and y==m:
8          if nowValue > maxValue:
9              maxValue = nowValue
10
11     return
12
13    for i in range(4):
14        nx = x + dx[i]
15        ny = y + dy[i]
16
17        if maze[nx][ny] != -9999:
```

```

18         tmp = maze[x][y]
19         maze[x][y] = -9999
20         nextValue = nowValue + maze[nx][ny]
21         dfs(maze, nx, ny, nextValue)
22         maze[x][y] = tmp
23
24
25 n, m = map(int, input().split())
26 maze = []
27 maze.append([-9999 for x in range(m+2)])
28 for _ in range(n):
29     maze.append([-9999] + [int(_) for _ in input().split()] + [-9999])
30 maze.append([-9999 for x in range(m+2)])
31
32
33 dfs(maze, 1, 1, maze[1][1])
34 print(maxValue)

```

辅助visited空间

```

1 MAXN = 5
2 INF = float('inf')
3 n, m = map(int, input().split())
4 maze = []
5 for _ in range(n):
6     row = list(map(int, input().split()))
7     maze.append(row)
8
9 visited = [[False for _ in range(m)] for _ in range(n)]
10 maxValue = -INF
11
12 MAXD = 4
13 dx = [0, 0, 1, -1]
14 dy = [1, -1, 0, 0]
15
16 def is_valid(x, y):
17     return 0 <= x < n and 0 <= y < m and not visited[x][y]
18
19 def DFS(x, y, nowValue):
20     global maxValue
21     if x == n - 1 and y == m - 1:
22         if nowValue > maxValue:
23             maxValue = nowValue
24     return
25     visited[x][y] = True
26     for i in range(MAXD):
27         nextX = x + dx[i]
28         nextY = y + dy[i]
29         if is_valid(nextX, nextY):

```

```

30         nextValue = nowValue + maze[nextX][nextY]
31         DFS(nextX, nextY, nextValue)
32     visited[x][y] = False
33
34 DFS(0, 0, maze[0][0])
35 print(maxValue)
36

```

示例：sy316矩阵最大权值路径 中等

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

现有一个 $n \times m$ 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 ($2 \leq n \leq 5, 2 \leq m \leq 5$)，分别表示矩阵的行数和列数；

接下来 n 行，每行 m 个整数 ($-100 \leq$ 整数 ≤ 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

1	2 2
2	1 2
3	3 4

输出

1	1 1
2	2 1
3	2 2

解释

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

样例2

输入

```
1 4 5
2 59 -62 -71 91 -12
3 -36 42 -32 -36 43
4 -68 -88 -94 -43 -39
5 48 -38 53 31 -92
```

输出

```
1 1 1
2 2 1
3 2 2
4 2 3
5 2 4
6 1 4
7 1 5
8 2 5
9 3 5
10 4 5
```

样例3

输入

```
1 3 4
2 -36 -10 -84 -28
3 12 94 95 22
4 61 -13 26 29
```

输出

```
1 1 1
2 1 2
3 2 2
4 2 1
5 3 1
6 3 2
7 3 3
8 2 3
9 2 4
10 3 4
```

DFS辅助visited空间

```
1 # 读取输入
2 n, m = map(int, input().split())
3 maze = [list(map(int, input().split())) for _ in range(n)]
```

```

4
5 # 定义方向
6 directions = [(0, 1), (1, 0), (0, -1), (-1, 0)] # 右、下、左、上
7 visited = [[False] * m for _ in range(n)] # 标记访问
8 max_path = []
9 max_sum = -float('inf') # 最大权值初始化为负无穷
10
11 # 深度优先搜索
12 def dfs(x, y, current_path, current_sum):
13     global max_path, max_sum
14
15     # 到达终点，更新结果
16     if (x, y) == (n - 1, m - 1):
17         if current_sum > max_sum:
18             max_sum = current_sum
19             max_path = current_path[:]
20     return
21
22     # 遍历四个方向
23     for dx, dy in directions:
24         nx, ny = x + dx, y + dy
25
26         # 检查边界和是否访问过
27         if 0 <= nx < n and 0 <= ny < m and not visited[nx][ny]:
28             # 标记访问
29             visited[nx][ny] = True
30             current_path.append((nx, ny))
31
32             # 递归搜索
33             dfs(nx, ny, current_path, current_sum + maze[nx][ny])
34
35             # 回溯
36             current_path.pop()
37             visited[nx][ny] = False
38
39     # 初始化起点
40     visited[0][0] = True
41     dfs(0, 0, [(0, 0)], maze[0][0])
42
43     # 输出结果
44     for x, y in max_path:
45         print(x + 1, y + 1)
46

```

DFS辅助visited空间

```

1 MAXN = 5
2 INF = float('inf')
3 n, m = map(int, input().split())

```

```

4 maze = []
5 for _ in range(n):
6     row = list(map(int, input().split()))
7     maze.append(row)
8
9 visited = [[False for _ in range(m)] for _ in range(n)]
10 maxValue = -INF
11 tempPath, optPath = [], []
12
13 MAXD = 4
14 dx = [0, 0, 1, -1]
15 dy = [1, -1, 0, 0]
16
17 def is_valid(x, y):
18     return 0 <= x < n and 0 <= y < m and not visited[x][y]
19
20 def DFS(x, y, nowValue):
21     global maxValue, tempPath, optPath
22     if x == n - 1 and y == m - 1:
23         if nowValue > maxValue:
24             maxValue = nowValue
25             optPath = list(tempPath)
26     return
27     visited[x][y] = True
28     for i in range(MAXD):
29         nextX = x + dx[i]
30         nextY = y + dy[i]
31         if is_valid(nextX, nextY):
32             nextValue = nowValue + maze[nextX][nextY]
33             tempPath.append((nextX, nextY))
34             DFS(nextX, nextY, nextValue)
35             tempPath.pop()
36     visited[x][y] = False
37
38 tempPath.append((0, 0))
39 DFS(0, 0, maze[0][0])
40 for pos in optPath:
41     print(pos[0] + 1, pos[1] + 1)

```

示例：sy317迷宫最大权值 中等

<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 ($2 \leq n \leq 5, 2 \leq m \leq 5$)，分别表示矩阵的行数和列数；

接下来 n 行，每行 m 整数（值为 0 或 1），表示迷宫。

再接下来行，每行 m 整数 ($-100 \leq$ 整数 ≤ 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

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

输出

```
1 | 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

```
1 dx = [-1, 0, 1, 0]
2 dy = [0, 1, 0, -1]
3
4 maxValue = float("-inf")
5 def dfs(maze, x, y, nowValue):
6     global maxValue
7     if x==n and y==m:
8         if nowValue > maxValue:
9             maxValue = nowValue
10
11     return
12
13     for i in range(4):
14         nx = x + dx[i]
15         ny = y + dy[i]
16
17         if maze[nx][ny] == 0:
18             maze[nx][ny] = -1
19             tmp = w[x][y]
```

```

20         w[x][y] = -9999
21         nextValue = nowValue + w[nx][ny]
22         dfs(maze, nx, ny, nextValue)
23         maze[nx][ny] = 0
24         w[x][y] = tmp
25
26
27 n, m = map(int, input().split())
28 maze = []
29 maze.append([-1 for x in range(m+2)])
30 for _ in range(n):
31     maze.append([-1] + [int(_) for _ in input().split()] + [-1])
32 maze.append([-1 for x in range(m+2)])
33
34 w = []
35 w.append([-9999 for x in range(m+2)])
36 for _ in range(n):
37     w.append([-9999] + [int(_) for _ in input().split()] + [-9999])
38 w.append([-9999 for x in range(m+2)])
39
40
41 dfs(maze, 1, 1, w[1][1])
42 print(maxValue)

```

辅助visited空间

```

1 # gpt translated version of the C++ code
2 MAXN = 5
3 INF = float('inf')
4 n, m = map(int, input().split())
5 maze = [list(map(int, input().split())) for _ in range(n)]
6 w = [list(map(int, input().split())) for _ in range(n)]
7 visited = [[False] * m for _ in range(n)]
8 maxValue = -INF
9
10 MAXD = 4
11 dx = [0, 0, 1, -1]
12 dy = [1, -1, 0, 0]
13
14 def is_valid(x, y):
15     return 0 <= x < n and 0 <= y < m and not maze[x][y] and not visited[x][y]
16
17 def dfs(x, y, nowValue):
18     global maxValue
19     if x == n - 1 and y == m - 1:
20         if nowValue > maxValue:
21             maxValue = nowValue
22         return
23     visited[x][y] = True

```

```

24     for i in range(MAXD):
25         nextX = x + dx[i]
26         nextY = y + dy[i]
27         if is_valid(nextX, nextY):
28             nextValue = nowValue + w[nextX][nextY]
29             dfs(nextX, nextY, nextValue)
30             visited[x][y] = False
31
32     dfs(0, 0, w[0][0])
33     print(maxValue)
34

```

练习: sy358受到祝福的平方 中等

<https://sunnywhy.com/sfbj/8/3/539>

在小元的世界里，任何人出生后会被世界分配一个随机 `ID`，如果在被切割后，即 `ID` 满足按照从左至右顺序分割，且分割出来的数字都是某一个 `正整数` 的平方，分割时可以包括前导 `0`，那么他就被这个世界祝福，最后获得快乐的数量和质量都比不满足这样的人多的多。

令 `ID` 为 `A`，且 `A` 是一个正整数，取值范围为 $1 \leq A \leq 10^9$ ，问是否是一个被受到祝福的。

比如 `A=8194` 时，它是一个被受到祝福的 `ID`，因为他可以被分割为 $\{81, 9, 4\} = \{9^2, 3^2, 2^2\}$ ；

比如 `A=1001` 时，它是一个被受到祝福的 `ID`，因为他可以被分割为 $\{1, 001\} = \{1^2, 1^2\}$ ，或者 $\{100, 1\} = \{10^2, 1^2\}$ 。注意 $\{1, 00, 1\} = \{1^2, 0^2, 1^2\}$ 不是一个合法切割，因为分割出来的数字必须为正整数的平方；

比如 `A=36` 时，`36` 已经是一个平方数了，所以它同样满足条件；

比如 `A=54`，它不是一个被受到祝福的 `ID`，因为他无法被切割为满足条件的集合。

输入描述

一个正整数 `A`，无前导 `0`。

其中 $1 \leq A \leq 10^9$

输出描述

如果是一个满足题意的数字则输出 `yes`，否则 `no`。

2 广度优先搜索(BFS)

前面介绍了深度优先搜索，可知 DFS 是以深度作为第一关键词的，即当碰到岔道口时总是先选择其中的一条岔路前进，而不管其他岔路，直到碰到死胡同才返回岔道口并选择其他岔路。接下来将介绍的**广度优先搜索 (Breadth First Search, BFS)**则是以广度为第一关键词，当碰到岔道口时，总是先依次访问从该岔道口能直接到达的所有结点，然后再按这些结点被访问的顺序去依次访问它们能直接到达的所有结点，以此类推，直到所有结点都被访问为止。这就跟平静的水面中投入一颗小石子一样，水花总是以石子落水处为中心，并以同心圆的方式向外扩散至整个水面(见图 8-2)，从这点来看和 DFS 那种沿着一条线前进的思路是完全不同的。

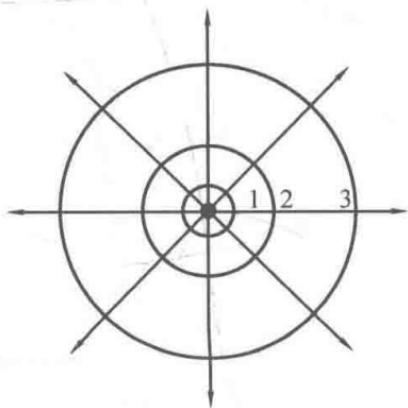


图 8-2 广度优先搜索示意图

广度优先搜索 (BFS)一般由队列实现,且总是按层次的顺序进行遍历, 其基本写法如下(可作模板用):

```

1  from collections import deque
2
3  def bfs(s, e):
4      inq = set()
5      inq.add(s)
6
7      q = deque()
8      q.append((0, s))
9
10     while q:
11         now, top = q.popleft() # 取出队首元素
12         if top == e:
13             return now # 返回需要的结果, 如: 步长、路径等信息
14
15         # 将 top 的下一层结点中未曾入队的结点全部入队q, 并加入集合inq设置为已入队
16

```

下面是对该模板中每一个步骤的说明,请结合代码一起看:

- ① 定义队列 q, 并将起点(0, s)入队, 0表示步长目前是0。
- ② 写一个 while 循环, 循环条件是队列q非空。
- ③ 在 while 循环中, 先取出队首元素 top。
- ④ 将top 的下一层结点中所有未曾入队的结点入队, 并标记它们的层号为 now 的层号加1, 并加入集合inq设置为已入队。
- ⑤ 返回 ② 继续循环。

为了防止走回头路, 一般可以设置一个bool类型数组inq (即in queue的简写) 来记录每个位置是否在BFS中已入过队。再强调一点, 在BFS 中设置的 inq 数组的含义是判断结点是否已入过队, 而不是结点是否已被访问。区别在于: 如果设置成是否已被访问, 有可能在某个结点正在队列中 (但还未访问) 时由于其他结点可以到达它而将这个结点再次入队, 导致很多结点反复入队, 计算量大大增加。因此BFS 中让每个结点只入队一次, 故需要设置 inq 数组的含义为结点是否已入过队而非结点是否已被访问。

示例：sy318数字操作（一维BFS）

<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 ($2 \leq n \leq 10^5$)，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

```
1 | 7
```

输出

```
1 | 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。

数学思维

```

1   ...
2   2023TA-陈威宇, 思路: 是n的二进制表示 里面 1的个数+1的个数+0的个数-2。
3   如果我们将 n 的二进制表示的每一位数从左到右依次编号为 0、1、2、..., 那么:
4
5   1 的个数表示需要进行加 1 的操作次数;
6   0 的个数表示需要进行乘 2 的操作次数;
7   len(l) - 2 表示操作的总次数减去初始状态的操作次数 1, 即剩余的操作次数;
8   sum(l) + len(l) - 2 表示所有操作次数之和。
9   ...
10  n = int(input())
11  s = bin(n)
12  l = [int(i) for i in s[2:]]
13  print(sum(l) + len(l) - 2)

```

计算机思维

Python

```

1  from collections import deque
2
3  def bfs(n):
4
5      inq = set()
6      inq.add(1)
7      q = deque()
8      q.append((1, 0))
9      while q:
10         front, step = q.popleft()
11         if front == n:
12             return step
13
14         if front * 2 <= n and front * 2 not in inq:
15             inq.add(front * 2)
16             q.append((front * 2, step + 1))
17         if front + 1 <= n and front + 1 not in inq:
18             inq.add(front + 1)
19             q.append((front + 1, step + 1))
20
21
22 n = int(input())
23 print(bfs(n))
24

```

```

1  # gpt translated version of the C++ code
2  from collections import deque
3
4  MAXN = 100000

```

```

5  in_queue = [False] * (MAXN + 1)
6
7  def get_step(n):
8      step = 0
9      q = deque()
10     q.append(1)
11     while True:
12         cnt = len(q)
13         for _ in range(cnt):
14             front = q.popleft()
15             if front == n:
16                 return step
17             in_queue[front] = True
18             if front * 2 <= n and not in_queue[front * 2]:
19                 q.append(front * 2)
20             if front + 1 <= n and not in_queue[front + 1]:
21                 q.append(front + 1)
22             step += 1
23
24 if __name__ == "__main__":
25     n = int(input())
26     print(get_step(n))

```

示例：sy319矩阵中的块

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

题目描述

现有一个 $n \times m$ 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 (x_1, y_1) 与位置 (x_2, y_2) 相邻，且位置 (x_2, y_2) 与位置 (x_3, y_3) 相邻，那么称位置 (x_1, y_1) 与位置 (x_3, y_3) 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 ($2 \leq n \leq 100, 2 \leq m \leq 100$)，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

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

输出

```
1 | 4
```

解释

矩阵中的 1 共有 4 块，如下图所示。

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

加保护圈，inq_set集合判断是否入过队

```
1 from collections import deque
2
3 # Constants
4 MAXD = 4
5 dx = [0, 0, 1, -1]
6 dy = [1, -1, 0, 0]
7
8 def bfs(x, y):
9     q = deque([(x, y)])
10    inq_set.add((x,y))
11    while q:
12        front = q.popleft()
13        for i in range(MAXD):
14            next_x = front[0] + dx[i]
15            next_y = front[1] + dy[i]
16            if matrix[next_x][next_y] == 1 and (next_x,next_y) not in inq_set:
17                inq_set.add((next_x, next_y))
18                q.append((next_x, next_y))
19
20 # Input
```

```

21 n, m = map(int, input().split())
22 matrix=[[-1]*(m+2)]+[-1]+list(map(int,input().split()))+[-1] for i in range(n)]+[-1]* (m+2)
23 inq_set = set()
24
25 # Main process
26 counter = 0
27 for i in range(1,n+1):
28     for j in range(1,m+1):
29         if matrix[i][j] == 1 and (i,j) not in inq_set:
30             bfs(i, j)
31             counter += 1
32
33 # Output
34 print(counter)

```

inq 数组，结点是否已入过队

```

1 # gpt translated version of the C++ code
2 from collections import deque
3
4 # Constants
5 MAXN = 100
6 MAXD = 4
7 dx = [0, 0, 1, -1]
8 dy = [1, -1, 0, 0]
9
10 # Functions
11 def can_visit(x, y):
12     return 0 <= x < n and 0 <= y < m and matrix[x][y] == 1 and not in_queue[x][y]
13
14 def bfs(x, y):
15     q = deque([(x, y)])
16     in_queue[x][y] = True
17     while q:
18         front = q.popleft()
19         for i in range(MAXD):
20             next_x = front[0] + dx[i]
21             next_y = front[1] + dy[i]
22             if can_visit(next_x, next_y):
23                 in_queue[next_x][next_y] = True
24                 q.append((next_x, next_y))
25
26 # Input
27 n, m = map(int, input().split())
28 matrix = [list(map(int, input().split())) for _ in range(n)]
29 in_queue = [[False] * MAXN for _ in range(MAXN)]
30
31 # Main process

```

```

32 counter = 0
33 for i in range(n):
34     for j in range(m):
35         if matrix[i][j] == 1 and not in_queue[i][j]:
36             bfs(i, j)
37             counter += 1
38
39 # Output
40 print(counter)
41

```

示例：sy320迷宫问题

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

现有一个 $n \times m$ 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 ($2 \leq n \leq 100, 2 \leq m \leq 100$)，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

1	3 3
2	0 1 0
3	0 0 0
4	0 1 0

输出

1	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

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

输出

```
1 -1
```

解释：显然从左上角无法到达右下角。

加保护圈，inq_set集合判断是否入过队

```
1 from collections import deque  
2  
3 # 声明方向变化的数组，代表上下左右移动  
4 dx = [0, 0, 1, -1]  
5 dy = [1, -1, 0, 0]  
6  
7 def bfs(x, y):  
8     q = deque()  
9     q.append((x, y))  
10    inq_set.add((x, y))  
11    step = 0  
12    while q:  
13        for _ in range(len(q)):  
14            cur_x, cur_y = q.popleft()  
15            if cur_x == n and cur_y == m:  
16                return step  
17            for direction in range(4):  
18                next_x = cur_x + dx[direction]  
19                next_y = cur_y + dy[direction]  
20                if maze[next_x][next_y] == 0 and (next_x, next_y) not in inq_set:  
21                    inq_set.add((next_x, next_y))  
22                    q.append((next_x, next_y))  
23            step += 1  
24    return -1  
25  
26 if __name__ == '__main__':  
27  
28     n, m = map(int, input().split())  
29     maze = [[-1] * (m + 2)] + [[-1] + list(map(int, input().split())) + [-1] for i in  
range(n)] + [[-1] * (m + 2)]  
30     inq_set = set()  
31  
32     step = bfs(1, 1)  
33     print(step)  
34
```

inq 数组，结点是否已入过队

```
1 # gpt translated version of the C++ code
2 from collections import deque
3
4 # 声明方向变化的数组，代表上下左右移动
5 dx = [0, 0, 1, -1]
6 dy = [1, -1, 0, 0]
7
8 # 检查是否可以访问位置 (x, y)
9 def can_visit(x, y):
10     return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and not in_queue[x][y]
11
12 # BFS函数 实现广度优先搜索
13 def bfs(x, y):
14     q = deque()
15     q.append((x, y))
16     in_queue[x][y] = True
17     step = 0
18     while q:
19         for _ in range(len(q)):
20             cur_x, cur_y = q.popleft()
21             if cur_x == n - 1 and cur_y == m - 1:
22                 return step
23             for direction in range(4):
24                 next_x = cur_x + dx[direction]
25                 next_y = cur_y + dy[direction]
26                 if can_visit(next_x, next_y):
27                     in_queue[next_x][next_y] = True
28                     q.append((next_x, next_y))
29             step += 1
30     return -1
31
32 # 主函数
33 if __name__ == '__main__':
34     # 读取 n 和 m
35     n, m = map(int, input().split())
36     maze = []
37     in_queue = [[False] * m for _ in range(n)]
38
39     # 填充迷宫和访问状态数组
40     for i in range(n):
41         maze.append(list(map(int, input().split())))
42
43     # 执行BFS并输出步数
44     step = bfs(0, 0)
45     print(step)
46
```

示例：sy321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 ($2 \leq n \leq 100, 2 \leq m \leq 100$)，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

1	3 3
2	0 1 0
3	0 0 0
4	0 1 0

输出

1	1 1
2	2 1
3	2 2
4	2 3
5	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)。

inq 数组，结点是否已入过队

```
1 # gpt translated version of the C++ code
2 from queue import Queue
3
```

```

4 MAXN = 100
5 MAXD = 4
6 dx = [0, 0, 1, -1]
7 dy = [1, -1, 0, 0]
8
9 def canVisit(x, y):
10    return x >= 0 and x < n and y >= 0 and y < m and maze[x][y] == 0 and not
inQueue[x][y]
11
12 def BFS(x, y):
13    q = Queue()
14    q.put((x, y))
15    inQueue[x][y] = True
16    while not q.empty():
17        front = q.get()
18        if front[0] == n - 1 and front[1] == m - 1:
19            return
20        for i in range(MAXD):
21            nextX = front[0] + dx[i]
22            nextY = front[1] + dy[i]
23            if canVisit(nextX, nextY):
24                pre[nextX][nextY] = (front[0], front[1])
25                inQueue[nextX][nextY] = True
26                q.put((nextX, nextY))
27
28 def printPath(p):
29    prePosition = pre[p[0]][p[1]]
30    if prePosition == (-1, -1):
31        print(p[0] + 1, p[1] + 1)
32        return
33    printPath(prePosition)
34    print(p[0] + 1, p[1] + 1)
35
36 n, m = map(int, input().split())
37 maze = []
38 for _ in range(n):
39    row = list(map(int, input().split()))
40    maze.append(row)
41
42 inQueue = [[False] * m for _ in range(n)]
43 pre = [[(-1, -1)] * m for _ in range(n)]
44
45 BFS(0, 0)
46 printPath((n - 1, m - 1))

```

示例：sy322跨步迷宫

<https://sunnywhy.com/sfbj/8/2/322>

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或两格（两格为同向），且只能移动到平地上（不允许跨越墙壁）。求从迷宫左上角到右下角的最小步数（假设移动两格时算作一步）。

输入

第一行两个整数 n, m ($2 \leq n \leq 100, 2 \leq m \leq 100$)，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

1	3 3
2	0 1 0
3	0 0 0
4	0 1 0

输出

1	3
---	---

解释

假设左上角坐标是，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：=>=>=>。

因此最少需要3步。

样例2

输入

1	3 3
2	0 1 0
3	0 1 0
4	0 1 0

输出

1	-1
---	----

解释

显然从左上角无法到达右下角。

我们使用from collections import deque就满足要求，适用于需要频繁从队列的两端进行操作的场景，如广度优先搜索（BFS）、滑动窗口等问题。

from queue import Queue适用于多线程编程中，需要在多个线程之间安全地共享和传递数据的场景。提供线程安全的特性，内置锁机制，可以在多线程环境中安全地使用。支持阻塞操作，如get和put方法可以设置超时时间，等待队列中有数据可用或空间可用。不支持从队列两端进行操作，只能从一端进行插入和删除。

```
1  from collections import deque
2
3  MAXN = 100
4  MAXD = 8
5
6  dx = [0, 0, 0, 0, 1, -1, 2, -2]
7  dy = [1, -1, 2, -2, 0, 0, 0, 0]
8
9  def canVisit(x, y):
10     return x >= 0 and x < n and y >= 0 and y < m and maze[x][y] == 0 and not
11     inQueue[x][y]
12
13  def bfs(x, y):
14      q = deque()
15      q.append((x, y))
16      inQueue[x][y] = True
17      step = 0
18      while q:
19          cnt = len(q)
20          while cnt > 0:
21              front = q.popleft()
22              cnt -= 1
23              if front[0] == n - 1 and front[1] == m - 1:
24                  return step
25              for i in range(MAXD):
26                  nextX = front[0] + dx[i]
27                  nextY = front[1] + dy[i]
28                  nextHalfX = front[0] + dx[i] // 2
29                  nextHalfY = front[1] + dy[i] // 2
30                  if canVisit(nextX, nextY) and maze[nextHalfX][nextHalfY] == 0:
31                      inQueue[nextX][nextY] = True
32                      q.append((nextX, nextY))
33              step += 1
34      return -1
35
36  n, m = map(int, input().split())
37  maze = []
38  inQueue = [[False] * m for _ in range(n)]
39  for _ in range(n):
40      maze.append(list(map(int, input().split())))
41  step = bfs(0, 0)
42  print(step)
```

示例：sy323字符迷宫

<https://sunnywhy.com/sfbj/8/2/323>

现有一个 $n \times m$ 大小的迷宫，其中 * 表示不可通过的墙壁，. 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。求从起点 s 到终点 t 的最小步数。

输入

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

接下来 n 行，每行一个长度为 m 的字符串，表示迷宫。

输出

输出一个整数，表示最小步数。如果无法从 s 到达 t，那么输出 -1。

样例1

输入

```
1 | 5 5
2 | .....
3 | .*.*.
4 | .*S*.
5 | .***.
6 | ...T*
```

输出

```
1 | 11
```

解释

假设左上角坐标是，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。

起点的坐标为，终点的坐标为。

可以得到从 s 到 t 的前进路线：=>=>=>=>=>=>=>=>=>。

样例2

输入

复制

```
1 | 5 5
2 | .....
3 | .*.*.
4 | .*S*.
5 | .***.
6 | ..*T*
```

输出

```
1 | -1
```

解释

显然终点 `T` 被墙壁包围，无法到达。

```
1 from collections import deque
2
3 MAXN = 100
4 MAXD = 4
5
6 dx = [0, 0, 1, -1]
7 dy = [1, -1, 0, 0]
8
9 def canVisit(x, y):
10     return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and not inQueue[x][y]
11
12 def BFS(start, target):
13     q = deque([start])
14     inQueue[start[0]][start[1]] = True
15     step = 0
16     while q:
17         cnt = len(q)
18         while cnt > 0:
19             front = q.popleft()
20             cnt -= 1
21             if front == target:
22                 return step
23             for i in range(MAXD):
24                 nextX = front[0] + dx[i]
25                 nextY = front[1] + dy[i]
26                 if canVisit(nextX, nextY):
27                     inQueue[nextX][nextY] = True
28                     q.append((nextX, nextY))
29             step += 1
30     return -1
31
32 n, m = map(int, input().split())
33 maze = []
34 inQueue = [[False] * m for _ in range(n)]
35 start, target = None, None
36
37 for i in range(n):
38     row = input().strip()
39     maze_row = []
40     for j in range(m):
```

```

41     if row[j] == '.':
42         maze_row.append(0)
43     elif row[j] == '*':
44         maze_row.append(1)
45     elif row[j] == 'S':
46         start = (i, j)
47         maze_row.append(0)
48     elif row[j] == 'T':
49         target = (i, j)
50         maze_row.append(0)
51     maze.append(maze_row)
52
53 if start is None or target is None:
54     print(-1)
55 else:
56     step = BFS(start, target)
57     print(step)

```

示例：sy324多终点迷宫问题

<https://sunnywhy.com/sfbj/8/2/324>

现有一个 $n \times m$ 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。求从迷宫左上角到迷宫中每个位置的最小步数。

输入

第一行两个整数 n, m ($2 \leq n \leq 100, 2 \leq m \leq 100$)，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 n 行 m 列个整数，表示从左上角到迷宫中每个位置需要的最小步数。如果无法到达，那么输出 -1。注意，整数之间用空格隔开，行末不允许有多余的空格。

样例1

输入

1	3 3
2	0 0 0
3	1 0 0
4	0 1 0

输出

1	0 1 2
2	-1 2 3
3	-1 -1 4

解释

假设左上角坐标是，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所有点的前进路线：=>=>或=>=>。

左下角的三个位置无法到达。

```
1 from collections import deque
2 import sys
3
4 INF = sys.maxsize
5 MAXN = 100
6 MAXD = 4
7
8 dx = [0, 0, 1, -1]
9 dy = [1, -1, 0, 0]
10
11 def canVisit(x, y):
12     return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and not inQueue[x][y]
13
14 def BFS(x, y):
15     minStep = [[-1] * m for _ in range(n)]
16     q = deque([(x, y)])
17     inQueue[x][y] = True
18     minStep[x][y] = 0
19     step = 0
20     while q:
21         cnt = len(q)
22         while cnt > 0:
23             front = q.popleft()
24             cnt -= 1
25             for i in range(MAXD):
26                 nextX = front[0] + dx[i]
27                 nextY = front[1] + dy[i]
28                 if canVisit(nextX, nextY):
29                     inQueue[nextX][nextY] = True
30                     minStep[nextX][nextY] = step + 1
31                     q.append((nextX, nextY))
32             step += 1
33     return minStep
34
35 n, m = map(int, input().split())
36 maze = []
37 inQueue = [[False] * m for _ in range(n)]
38
39 for _ in range(n):
40     maze.append(list(map(int, input().split())))
41
42 minStep = BFS(0, 0)
43 for i in range(n):
```

```
44 |     print(' '.join(map(str, minStep[i])))
```

示例：sy325迷宫问题-传送点

<https://sunnywhy.com/sfbj/8/2/325>

现有一个 $n \times m$ 大小的迷宫，其中1表示不可通过的墙壁，0表示平地，2表示传送点。每次移动只能向上下左右移动一格，且只能移动到平地或传送点上。当位于传送点时，可以选择传送到另一个2处（传送不计入步数），也可以选择不传送。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 ($2 \leq n \leq 100, 2 \leq m \leq 100$)，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或2），表示迷宫。数据保证有且只有两个2，且传送点不会在起始点出现。

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

复制

1	3 3
2	0 1 2
3	0 1 0
4	2 1 0

输出

1	4
---	---

解释

假设左上角坐标是，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： $=>=>=>=>=$ ，其中 $=>$ 属于传送，不计入步数。

因此最少需要4步。

样例2

输入

1	3 3
2	0 1 0
3	2 1 0
4	2 1 0

输出

```
1 | -1
```

解释

显然从左上角无法到达右下角。

将 transVector 中的第一个位置映射到第二个位置，并将第二个位置映射到第一个位置。这样，就建立了传送门的双向映射关系。

在 BFS 函数中，当遇到传送门时，通过映射表 transMap 找到传送门的另一侧位置，并将其加入队列，以便继续进行搜索。

```
1 from collections import deque
2
3 MAXN = 100
4 MAXD = 4
5
6 dx = [0, 0, 1, -1]
7 dy = [1, -1, 0, 0]
8
9 def canVisit(x, y):
10     return 0 <= x < n and 0 <= y < m and (maze[x][y] == 0 or maze[x][y] == 2) and not
11     inQueue[x][y]
12
13 def BFS(x, y):
14     q = deque([(x, y)])
15     inQueue[x][y] = True
16     step = 0
17     while q:
18         cnt = len(q)
19         while cnt > 0:
20             front = q.popleft()
21             cnt -= 1
22             if front[0] == n - 1 and front[1] == m - 1:
23                 return step
24             for i in range(MAXD):
25                 nextX = front[0] + dx[i]
26                 nextY = front[1] + dy[i]
27                 if canVisit(nextX, nextY):
28                     inQueue[nextX][nextY] = True
29                     q.append((nextX, nextY))
30                     if maze[nextX][nextY] == 2:
31                         transPosition = transMap[(nextX, nextY)]
32                         inQueue[transPosition[0]][transPosition[1]] = True
33                         q.append(transPosition)
34             step += 1
35     return -1
```

```

36 n, m = map(int, input().split())
37 maze = []
38 inQueue = [[False] * m for _ in range(n)]
39 transMap = {}
40 transVector = []
41
42 for i in range(n):
43     row = list(map(int, input().split()))
44     maze.append(row)
45
46     if 2 in row:
47         for j, val in enumerate(row):
48             if val == 2:
49                 transVector.append((i, j))
50
51     if len(transVector) == 2:
52         transMap[transVector[0]] = transVector[1]
53         transMap[transVector[1]] = transVector[0]
54         transVector = [] # 清空 transVector 以便处理下一对传送点
55
56 step = BFS(0, 0)
57 print(step)

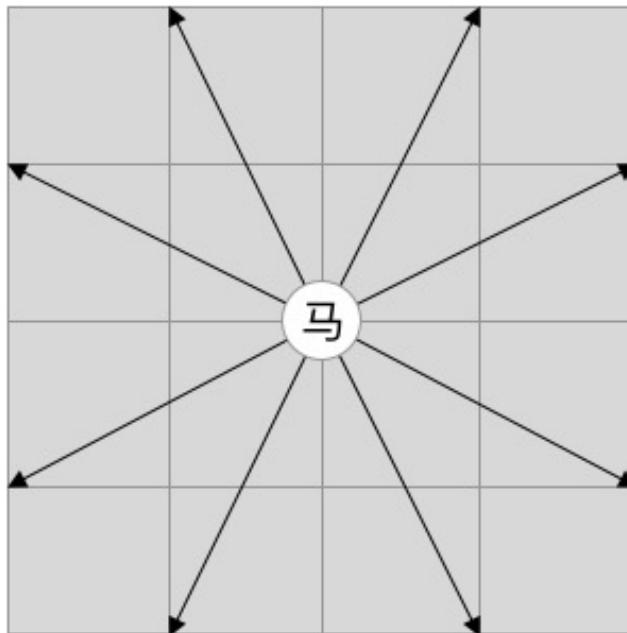
```

示例：sy326中国象棋-马-无障碍

<https://sunnywhy.com/sfbj/8/2/326>

现有一个n*m大小的棋盘，在棋盘的第行第列的位置放置了一个棋子，其他位置都未放置棋子。棋子的走位参照中国象棋的“马”。求该棋子到棋盘上每个位置的最小步数。

注：中国象棋中“马”的走位为“日”字形，如下图所示。



输入

四个整数 n 、 m 、 x 、 y ($2 \leq n \leq 100, 2 \leq m \leq 100, 1 \leq x \leq n, 1 \leq y \leq m$)，分别表示棋盘的行数和列数、棋子的所在位置。

输出

输出行个整数，表示从棋子到棋盘上每个位置需要的最小步数。如果无法到达，那么输出 -1 。注意，整数之间用空格隔开，行末不允许有多余的空格。

样例1

输入

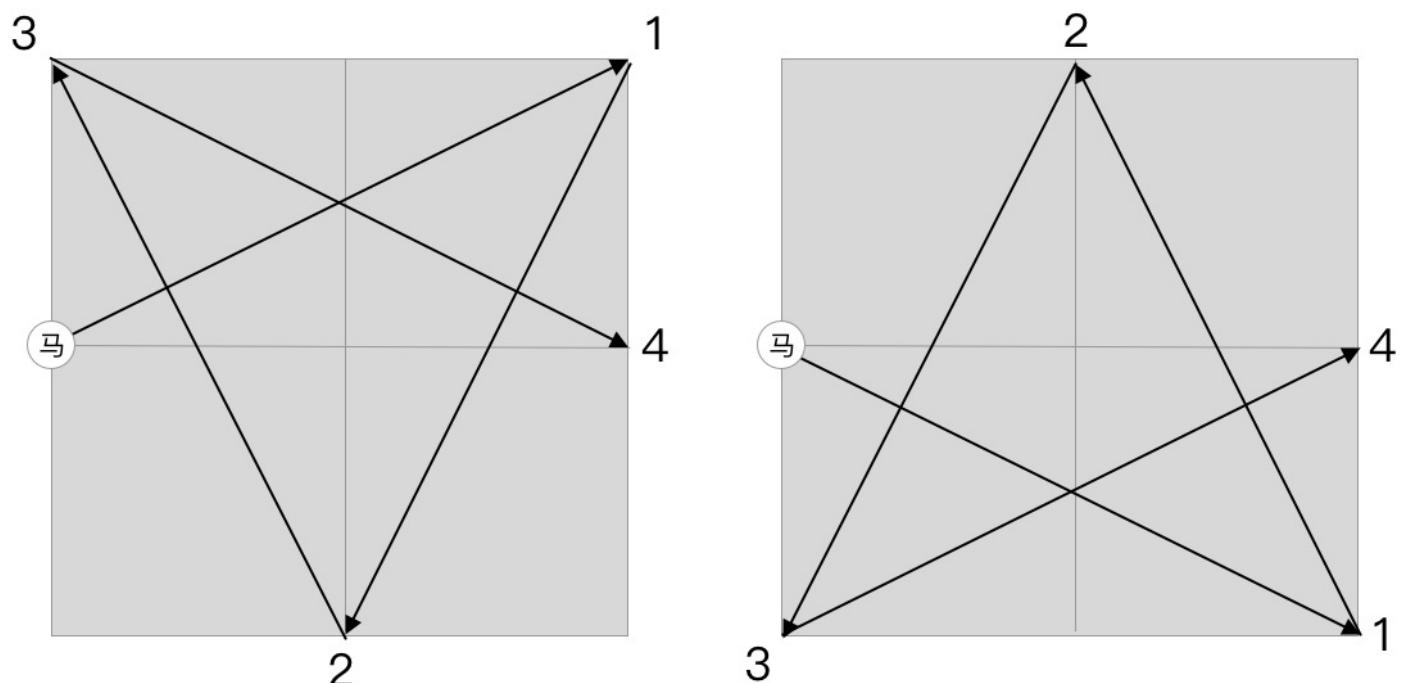
```
1 | 3 3 2 1
```

输出

```
1 | 3 2 1  
2 | 0 -1 4  
3 | 3 2 1
```

解释

共 3 行 3 列，“马”在第 2 行第 1 列的位置，由此可得“马”能够前进的路线如下图所示。



```
1 from collections import deque  
2  
3 MAXN = 100  
4 MAXD = 8  
5
```

```

6  dx = [-2, -1, 1, 2, -2, -1, 1, 2]
7  dy = [1, 2, 2, 1, -1, -2, -2, -1]
8
9  def canVisit(x, y):
10     return 0 <= x < n and 0 <= y < m and not inQueue[x][y]
11
12 def BFS(x, y):
13     minStep = [[-1] * m for _ in range(n)]
14     queue = deque()
15     queue.append((x, y))
16     inQueue[x][y] = True
17     minStep[x][y] = 0
18     step = 0
19     while queue:
20         cnt = len(queue)
21         while cnt > 0:
22             front = queue.popleft()
23             cnt -= 1
24             for i in range(MAXD):
25                 nextX = front[0] + dx[i]
26                 nextY = front[1] + dy[i]
27                 if canVisit(nextX, nextY):
28                     inQueue[nextX][nextY] = True
29                     minStep[nextX][nextY] = step + 1
30                     queue.append((nextX, nextY))
31             step += 1
32     return minStep
33
34
35 n, m, x, y = map(int, input().split())
36 inQueue = [[False] * m for _ in range(n)]
37 minStep = BFS(x - 1, y - 1)
38 for row in minStep:
39     print(' '.join(map(str, row)))

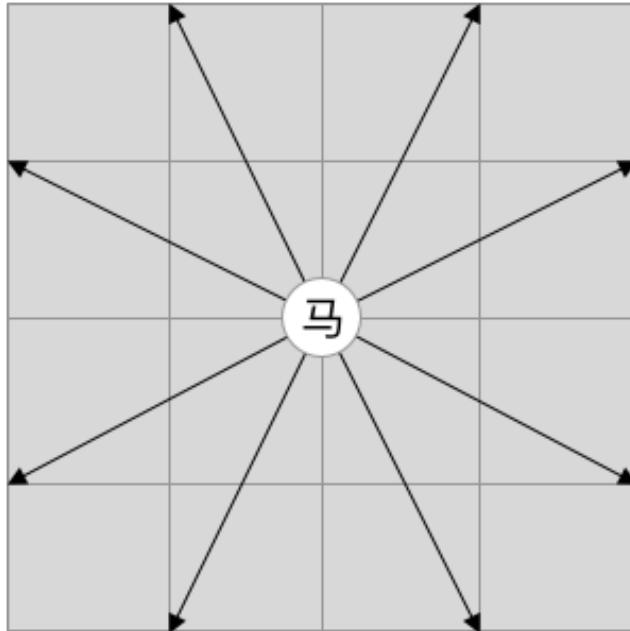
```

示例：sy327中国象棋-马-有障碍

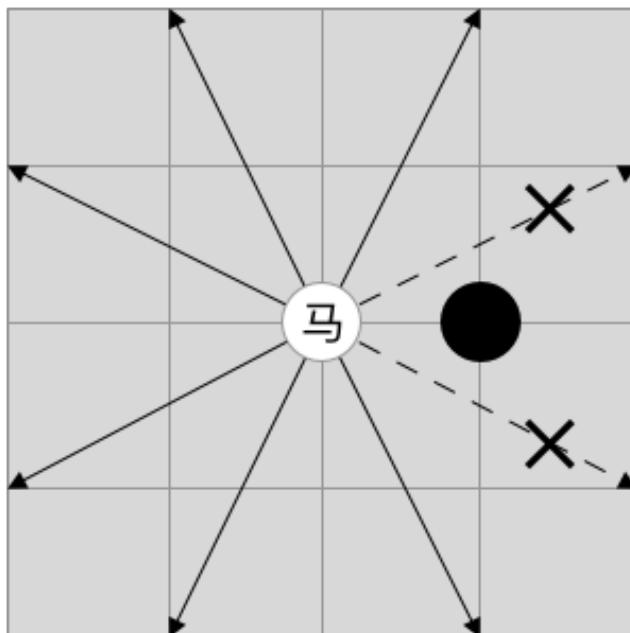
<https://sunnywhy.com/sfbj/8/2/327>

现有一个大小的棋盘，在棋盘的第行第列的位置放置了一个棋子，其他位置中的一部分放置了障碍棋子。棋子的走位参照中国象棋的“马”（障碍棋子将成为“马脚”）。求该棋子到棋盘上每个位置的最小步数。

注₁：中国象棋中“马”的走位为“日”字形，如下图所示。



注 2：与“马”直接相邻的棋子会成为“马脚”，“马”不能往以“马”=>“马脚”为长边的方向前进，如下图所示。



输入

第一行四个整数 n 、 m 、 x 、 y ($2 \leq n \leq 100, 2 \leq m \leq 100, 1 \leq x \leq n, 1 \leq y \leq m$)，分别表示棋盘的行数和列数、棋子的所在位置；

第二行一个整数 k ($1 \leq k \leq 10$)，表示障碍棋子的个数；

接下来 k 行，每行两个整数 x_i 、 y_i ($1 \leq x_i \leq n, 1 \leq y_i \leq m$)，表示第 i 个障碍棋子的所在位置。数据保证不存在相同位置的障碍棋子。

输出

输出 n 行 m 列个整数，表示从棋子到棋盘上每个位置需要的最小步数。如果无法到达，那么输出 -1 。注意，整数之间用空格隔开，行末不允许有多余的空格。

样例1

输入

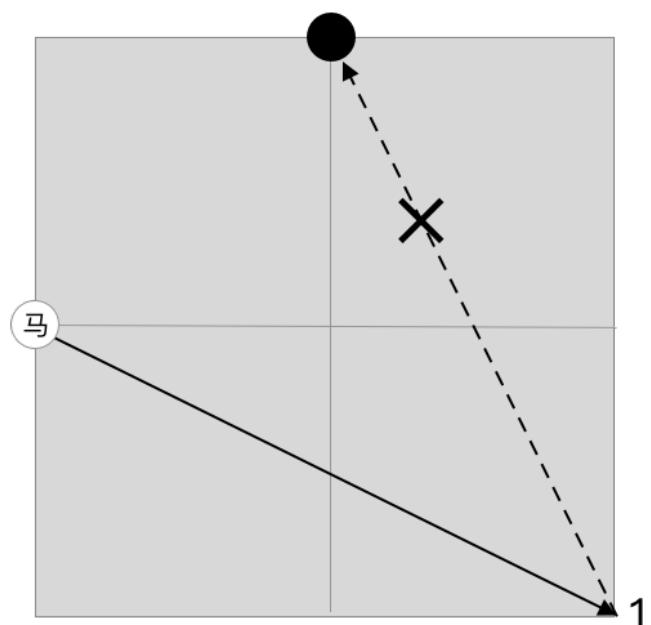
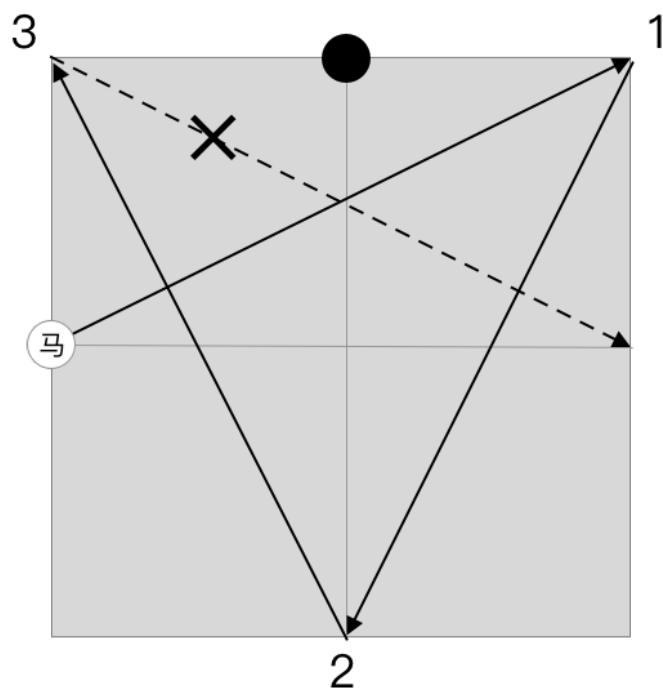
1	3	3	2	1
2	1			
3	1	2		

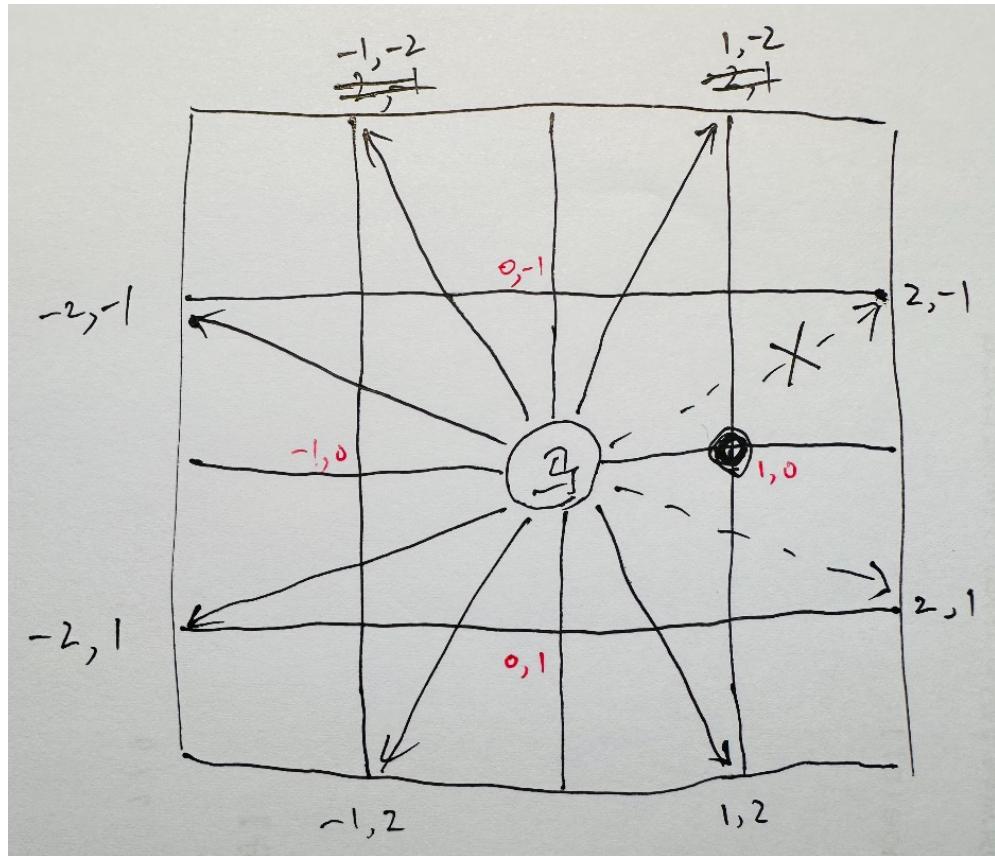
输出

1	3	-1	1
2	0	-1	-1
3	-1	2	1

解释

共 3 行 3 列，“马”在第 2 行第 1 列的位置，障碍棋子在第 1 行第 2 列的位置，由此可得“马”能够前进的路线如下图所示。





```

1  from collections import deque
2
3  MAXD = 8
4  dx = [-2, -2, -1, 1, 2, 2, 1, -1]
5  dy = [1, -1, -2, -2, -1, 1, 2, 2]
6
7
8  def canVisit(x, y):
9      return x >= 0 and x < n and y >= 0 and y < m and not isBlock.get((x, y), False)
10     and not inQueue[x][y]
11
12 def BFS(x, y):
13     minStep = [[-1] * m for _ in range(n)]
14     queue = deque()
15     queue.append((x, y))
16     inQueue[x][y] = True
17     minStep[x][y] = 0
18     step = 0
19     while queue:
20         cnt = len(queue)
21         for _ in range(cnt):
22             front = queue.popleft()
23             wx, wy = [-1, 0, 1, 0], [0, -1, 0, 1]
24             for i in range(MAXD):
25                 nextX = front[0] + dx[i]
26                 nextY = front[1] + dy[i]
27                 footX, footY = front[0] + wx[i//2], front[1] + wy[i//2]

```

```

28
29         if canVisit(nextX, nextY) and not isBlock.get((footX, footY), False):
30             inQueue[nextX][nextY] = True
31             minStep[nextX][nextY] = step + 1
32             queue.append((nextX, nextY))
33
34
35         step += 1
36     return minStep
37
38 n, m, x, y = map(int, input().split())
39 inQueue = [[False] * m for _ in range(n)]
40 isBlock = {}
41
42 k = int(input())
43 for _ in range(k):
44     blockX, blockY = map(int, input().split())
45     isBlock[(blockX - 1, blockY - 1)] = True
46
47 minStep = BFS(x - 1, y - 1)
48
49 for row in minStep:
50     print(' '.join(map(str, row)))

```

```

1 from collections import deque
2
3 MAXD = 8
4 dx = [-2, -1, 1, 2, -2, -1, 1, 2]
5 dy = [1, 2, 2, 1, -1, -2, -2, -1]
6
7
8 def canVisit(x, y):
9     return x >= 0 and x < n and y >= 0 and y < m and not isBlock.get((x, y), False)
10    and not inQueue[x][y]
11
12 def BFS(x, y):
13     minStep = [[-1] * m for _ in range(n)]
14     queue = deque()
15     queue.append((x, y))
16     inQueue[x][y] = True
17     minStep[x][y] = 0
18     step = 0
19     while queue:
20         cnt = len(queue)
21         for _ in range(cnt):
22             front = queue.popleft()
23             for i in range(MAXD):

```

```

24         nextX = front[0] + dx[i]
25         nextY = front[1] + dy[i]
26         if dx[i] == -1 and dy[i] == -1: #如果dx=-1, -1//2=-1, 期望得到0
27             footX, footY = front[0], front[1]
28         elif dx[i] == -1 and dy[i] != -1:
29             footX, footY = front[0], front[1] + dy[i] // 2
30         elif dx[i] != -1 and dy[i] == -1:
31             footX, footY = front[0] + dx[i] // 2, front[1]
32         else:
33             footX, footY = front[0] + dx[i] // 2, front[1] + dy[i] // 2
34
35         if canVisit(nextX, nextY) and not isBlock.get((footX, footY), False):
36             inQueue[nextX][nextY] = True
37             minStep[nextX][nextY] = step + 1
38             queue.append((nextX, nextY))
39
40
41     step += 1
42
43
44 n, m, x, y = map(int, input().split())
45 inQueue = [[False] * m for _ in range(n)]
46 isBlock = {}
47
48 k = int(input())
49 for _ in range(k):
50     blockX, blockY = map(int, input().split())
51     isBlock[(blockX - 1, blockY - 1)] = True
52
53 minStep = BFS(x - 1, y - 1)
54
55 for row in minStep:
56     print(' '.join(map(str, row)))

```

3 相关题目

02287: Tian Ji -- The Horse Racing

greedy, <http://cs101.openjudge.cn/practice/02287>

Here is a famous story in Chinese history.

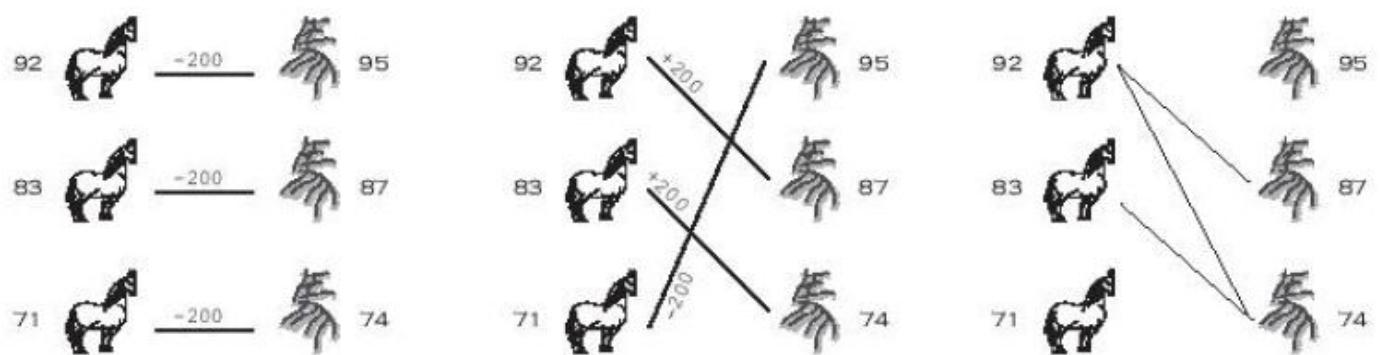
That was about 2300 years ago. General Tian Ji was a high official in the country Qi. He likes to play horse racing with the king and others.

Both of Tian and the king have three horses in different classes, namely, regular, plus, and super. The rule is to have three rounds in a match; each of the horses must be used in one round. The winner of a single round takes two hundred silver dollars from the loser.

Being the most powerful man in the country, the king has so nice horses that in each class his horse is better than Tian's. As a result, each time the king takes six hundred silver dollars from Tian.

Tian Ji was not happy about that, until he met Sun Bin, one of the most famous generals in Chinese history. Using a little trick due to Sun, Tian Ji brought home two hundred silver dollars and such a grace in the next match.

It was a rather simple trick. Using his regular class horse race against the super class from the king, they will certainly lose that round. But then his plus beat the king's regular, and his super beat the king's plus. What a simple trick. And how do you think of Tian Ji, the high ranked official in China?



Were Tian Ji lives in nowadays, he will certainly laugh at himself. Even more, were he sitting in the ACM contest right now, he may discover that the horse racing problem can be simply viewed as finding the maximum matching in a bipartite graph. Draw Tian's horses on one side, and the king's horses on the other. Whenever one of Tian's horses can beat one from the king, we draw an edge between them, meaning we wish to establish this pair. Then, the problem of winning as many rounds as possible is just to find the maximum matching in this graph. If there are ties, the problem becomes more complicated, he needs to assign weights 0, 1, or -1 to all the possible edges, and find a maximum weighted perfect matching...

However, the horse racing problem is a very special case of bipartite matching. The graph is decided by the speed of the horses -- a vertex of higher speed always beat a vertex of lower speed. In this case, the weighted bipartite matching algorithm is a too advanced tool to deal with the problem.

In this problem, you are asked to write a program to solve this special case of matching problem.

输入

The input consists of up to 50 test cases. Each case starts with a positive integer n ($n \leq 1000$) on the first line, which is the number of horses on each side. The next n integers on the second line are the speeds of Tian's horses. Then the next n integers on the third line are the speeds of the king's horses. The input ends with a line that has a single '0' after the last test case.

输出

For each input case, output a line containing a single number, which is the maximum money Tian Ji will get, in silver dollars.

样例输入

```
1 3
2 92 83 71
3 95 87 74
4 2
5 20 20
6 20 20
7 2
8 20 19
9 22 18
10 0
```

样例输出

```
1 200
2 0
3 0
```

来源: Shanghai 2004

dfs

```
1 # 赵时阳-数院23
2
3 from functools import lru_cache
4 import sys
5 sys.setrecursionlimit(1 << 30)
6
7
8 def compare(a, b):
9     if a > b:
10         return 1
11     elif a == b:
12         return 0
13     else:
14         return -1
15
16 while True:
17     n = int(input())
18     if n == 0:
19         break
20
21     tian_values = list(map(int, input().split()))
22     king_values = list(map(int, input().split()))
23     tian_values.sort()
24     king_values.sort()
25
26 @lru_cache(maxsize=2048)
```

```
27     def dfs(start, end, i):
28         if i < n:
29             tian_value = tian_values[i]
30             king_value_start = king_values[start]
31             x1 = dfs(start + 1, end, i + 1) + compare(tian_value, king_value_start)
32
33             king_value_end = king_values[end]
34             x2 = dfs(start, end - 1, i + 1) + compare(tian_value, king_value_end)
35             x = max(x1, x2)
36             return x
37         else:
38             return 0
39
40     result = dfs(0, n - 1, 0)
41     print(200 * result)
42
```

20140: 今日化学论文

<http://cs101.openjudge.cn/practice/20140/>

常凯申同学发现自己今日化学论文字数抄上限了，决定采取如下的压缩方法蒙混过关：

把连续的x个字符串s记为[xs]。($1 \leq x \leq 100$)

但这样的方法当然骗不过lwh老师啦。老师非常生气，但出于好奇，还是想看一看常凯申同学写了什么。
请你帮老师还原出原始的论文。

输入

仅一行，由小写英文字母、数字和[]组成的字符串（其中不含空格）

输出

一行，原始的字符串。

样例输入

```
1 | [2b[3a]c]
```

样例输出

```
1 | baaacbaaac
```

来源: cs101-2019 柏敬尧v0.2

本地调试可以用`sys.stderr.write(checkpoint)`。如果精神不集中写程序太难受了，调试的print忘记注释造成WA，会耽误很久。

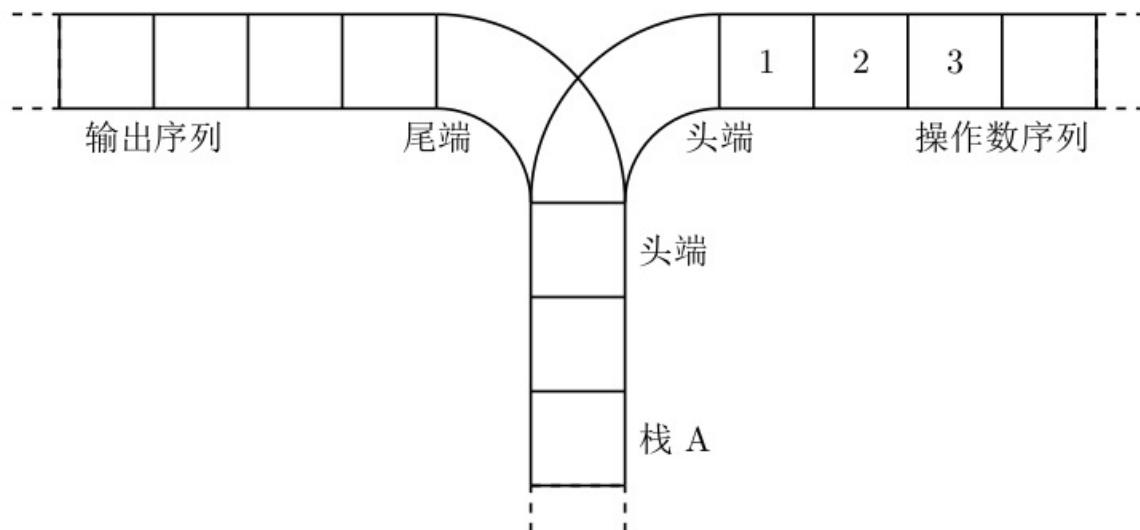
递归实现，避免使用全局变量。

```
1 import sys
2
3 def recursive_decode(s, idx):
4     stack = []
5     numstr = ""
6
7     while idx < len(s):
8         sys.stderr.write(s)
9         sys.stderr.write(s[idx])
10        if s[idx] == "[":
11            decoded_str, next_idx = recursive_decode(s, idx + 1)
12            stack.extend(decoded_str)
13            idx = next_idx
14        elif s[idx] == "]":
15            num = int(numstr)
16            return stack * num, idx
17        elif s[idx].isdigit():
18            numstr += s[idx]
19        else:
20            stack.append(s[idx])
21        idx += 1
22
23    return stack, idx
24
25
26 s = input()
27 #s = "[2b[3a]c]"
28 decoded_str, _ = recursive_decode(s, 0)
29 print(*decoded_str, sep="")
```

27217: 有多少种合法的出栈顺序

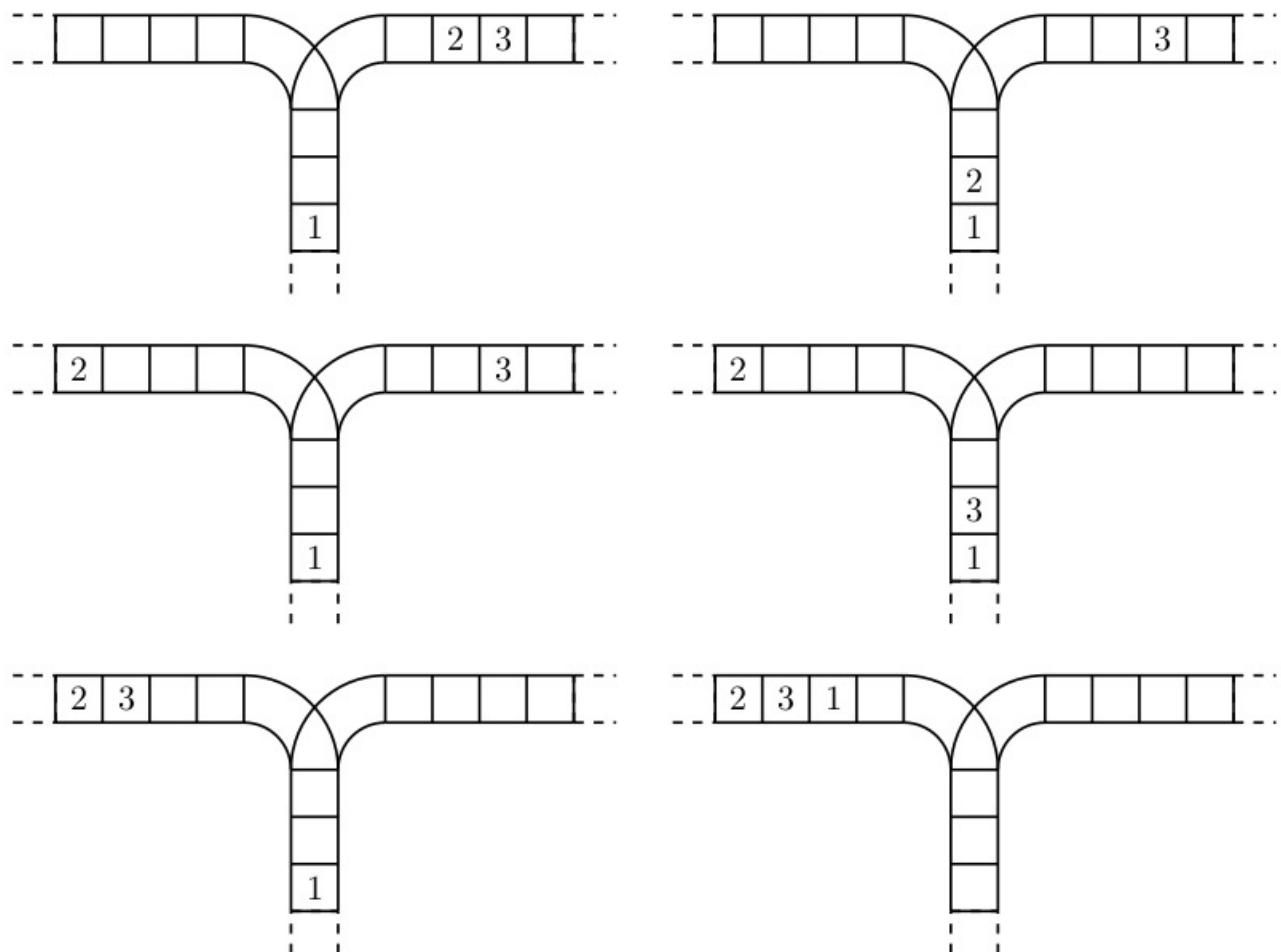
<http://cs101.openjudge.cn/practice/27217/>

栈是计算机中经典的数据结构，简单的说，栈就是限制在一端进行插入删除操作的线性表。栈有两种最重要的操作，即 pop（从栈顶弹出一个元素）和 push（将一个元素进栈）。栈的重要性不言自明，任何一门数据结构的课程都会介绍栈。宁宁同学在复习栈的基本概念时，想到了一个书上没有讲过的问题，而他自己无法给出答案，所以需要你的帮忙。



洛谷

宁宁考虑的是这样一个问题：一个操作数序列， $1, 2, \dots, n$ （图示为 1 到 3 的情况），栈 A 的深度大于 n 。现在可以进行两种操作，将一个数，从操作数序列的头端移到栈的头端（对应数据结构栈的 push 操作）将一个数，从栈的头端移到输出序列的尾端（对应数据结构栈的 pop 操作）使用这两种操作，由一个操作数序列就可以得到一系列的输出序列，下图所示为由 `1 2 3` 生成序列 `2 3 1` 的过程。



洛谷

(原始状态如上图所示) 你的程序将对给定的 n ，计算并输出由操作数序列 $1, 2, \dots, n$ 经过操作可能得到的输出序列

的总数。

输入

输入文件只含一个整数 n ($1 \leq n \leq 1000$) 。

输出

输出文件只有一行，即可能输出序列的总数目。

样例输入

3

样例输出

5

来源：洛谷 1044

```
1 ...
2 递归/记忆化搜索, https://www.luogu.com.cn/problem/solution/P1044
3 1) 二维数组f[i,j], 用下标 i 表示队列里还有几个待排的数, j 表示栈里有 j 个数,
4 f[i,j]表示此时的情况数
5 2) 那么, 更加自然的, 只要f[i,j]有值就直接返回;
6 3) 然后递归如何实现呢? 首先, 可以想到, 要是数全在栈里了, 就只剩1种情况了, 所以: i=0时, 返回1;
7 4) 然后, 有两种情况: 一种栈空, 一种栈不空: 在栈空时, 我们不可以弹出栈里的元素, 只能进入,
8 所以队列里的数-1, 栈里的数+1, 即加上 f[i-1,j+1] ; 另一种是栈不空,
9 那么此时有出栈1个或者进1个再出1个 2种情况, 分别加上 f[i-1,j+1] 和 f[i,j-1]
10 ...
11 import sys
12 sys.setrecursionlimit(1<<30)
13
14 def dfs(i, j, f):
15     if f[i][j] != -1:
16         return f[i][j]
17
18     if i == 0:
19         f[i][j] = 1
20         return 1
21
22     if j == 0:
23         f[i][j] = dfs(i - 1, j + 1, f)
24         return f[i][j]
25
26     f[i][j] = dfs(i - 1, j + 1, f) + dfs(i, j - 1, f)
27     return f[i][j]
28
29 n = int(input())
30 f = [[-1] * (n + 1) for _ in range(n + 1)]
31
32 result = dfs(n, 0, f)
33 print(result)
```

20123: 7-友好数

brute force, math, dfs similar, <http://cs101.openjudge.cn/practice/20123/>

黑板上写了一个正整数N，其首位不为0，位数不超过 10^5 。

N被称为7-友好数，如果可以擦掉若干位，使得剩下的数字构成的数为7的倍数。

要求不能擦掉所有数字，但允许只剩下一个数字0。

请你编写程序，判断N是不是7-友好数。

输入

一个正整数N，其位数 $\leq 10^5$ 。

输出

如果N是7-友好数，那么输出YES；否则输出NO

样例输入

```
1 输入样例1:  
2 123364315  
3  
4 输出样例1:  
5 YES  
6  
7 解释：可以使得剩下的数为35，因此满足要求。
```

样例输出

```
1 输入样例2:  
2 31116  
3  
4 输出样例2:  
5 NO
```

来源：cs101-2019 金及凯

```
1 #20123:7-友好数, http://cs101.openjudge.cn/practice/20123/  
2 #  
3 # 陈威宇: >=7位就一定YES了，因为所有后缀%7有两个相等的（抽屉原理），  
4 # 取这两个后缀里长的那个去掉短的那个即可?  
5 ...  
6 通过递归地尝试不同的子串来寻找符合条件的解。  
7 `dfs(n, i)` 函数是进行深度优先搜索的核心部分。它接受两个参数：`n` 代表当前搜索到的子串，  
8 `i` 代表当前处理到的位置索引。在函数内部，通过不断拼接字符来生成不同的子串，
```

```

9 然后检查是否满足能够被7整除的条件。
10
11 ...
12 def dfs(n, i):
13     global bo
14     if len(n) > 0 and int(n) % 7 == 0:
15         bo = True
16     if bo:
17         return
18     if i >= l:
19         return
20     dfs(n, i+1)
21     dfs(n+s[i], i+1)
22
23
24 s = input()
25 l = len(s)
26 if l >= 7:
27     print('YES')
28     exit()
29 bo = False
30 dfs('', 0)
31 if bo:
32     print('YES')
33 else:
34     print('NO')
35

```

550C. Divisibility by Eight

Brute force, dp, math, 1500, <https://codeforces.com/contest/550/problem/C>

You are given a non-negative integer n , its decimal representation consists of at most 100 digits and doesn't contain leading zeroes.

Your task is to determine if it is possible in this case to remove some of the digits (possibly not remove any digit at all) so that the result contains at least one digit, forms a non-negative integer, doesn't have leading zeroes and is divisible by 8. After the removing, it is forbidden to rearrange the digits.

If a solution exists, you should print it.

Input

The single line of the input contains a non-negative integer n . The representation of number n doesn't contain any leading zeroes and its length doesn't exceed 100 digits.

Output

Print "NO" (without quotes), if there is no such way to remove some digits from number n .

Otherwise, print "YES" in the first line and the resulting number after removing digits from number n in the second line. The printed number must be divisible by 8.

If there are multiple possible answers, you may print any of them.

Examples

input

1	3454
---	------

output

1	YES
2	344

input

1	10
---	----

output

1	YES
2	0

input

1	111111
---	--------

output

1	NO
---	----

记忆式搜索，20-21行是分叉。

```
1 ...
2 应该递归后三位，而不是所有的位数。因为
3 A number is divisible by 8 if its last three digits are also divisible by 8
4 ...
5 from functools import lru_cache
6
7 @lru_cache(maxsize=None)
8 def dfs(n, i, depth):
9     global bo, result
10    if depth > 3 or bo:
11        return
12    if len(n) > 0 and int(n) % 8 == 0:
```

```

13     result = n
14     bo = True
15     return
16 if bo:
17     return
18 if i >= l:
19     return
20 dfs(n, i+1, depth)
21 dfs(n+s[i], i+1, depth+1)
22
23
24 s = input()
25 l = len(s)
26 bo = False
27 result = ""
28 dfs('', 0, 0)
29 if bo:
30     print('YES\n', result)
31 else:
32     print('NO')

```

1843D. Apple Tree

Combinatorics, dfs and similar, dp, math, trees, *1200

<https://codeforces.com/problemset/problem/1843/D>

Timofey has an apple tree growing in his garden; it is a rooted tree of n vertices with the root in vertex 1 (the vertices are numbered from 1 to n). A tree is a connected graph without loops and multiple edges.

This tree is very unusual — it grows with its root upwards. However, it's quite normal for programmer's trees.

The apple tree is quite young, so only two apples will grow on it. Apples will grow in certain vertices (these vertices may be the same). After the apples grow, Timofey starts shaking the apple tree until the apples fall. Each time Timofey shakes the apple tree, the following happens to each of the apples:

Let the apple now be at vertex u .

- If a vertex u has a child, the apple moves to it (if there are several such vertices, the apple can move to any of them).
- Otherwise, the apple falls from the tree.

It can be shown that after a finite time, both apples will fall from the tree.

Timofey has q assumptions in which vertices apples can grow. He assumes that apples can grow in vertices x and y , and wants to know the number of pairs of vertices (a, b) from which apples can fall from the tree, where a — the vertex from which an apple from vertex x will fall, b — the vertex from which an apple from vertex y will fall. Help him do this.

Input

The first line contains integer t ($1 \leq t \leq 10^4$) — the number of test cases.

The first line of each test case contains integer n ($2 \leq n \leq 2 \cdot 10^5$) — the number of vertices in the tree.

Then there are $n-1$ lines describing the tree. In line i there are two integers ui and vi ($1 \leq ui, vi \leq n$, $ui \neq vi$) — edge in tree.

The next line contains a single integer q ($1 \leq q \leq 2 \cdot 10^5$) — the number of Timofey's assumptions.

Each of the next q lines contains two integers xi and yi ($1 \leq xi, yi \leq n$) — the supposed vertices on which the apples will grow for the assumption .

It is guaranteed that the sum of n does not exceed $2 \cdot 10^5$. Similarly, It is guaranteed that the sum of q does not exceed $2 \cdot 10^5$.

Output

For each Timofey's assumption output the number of ordered pairs of vertices from which apples can fall from the tree if the assumption is true on a separate line.

Examples

input

```
1 2
2 5
3 1 2
4 3 4
5 5 3
6 3 2
7 4
8 3 4
9 5 1
10 4 4
11 1 3
12 3
13 1 2
14 1 3
15 3
16 1 1
17 2 3
18 3 1
```

output

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

input

1	2
2	5
3	5 1
4	1 2
5	2 3
6	4 3
7	2
8	5 5
9	5 1
10	5
11	3 2
12	5 3
13	2 1
14	4 2
15	3
16	4 3
17	2 1
18	4 2

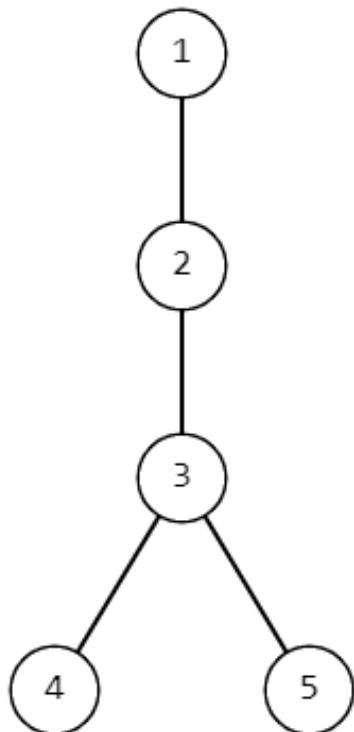
output

1	1
2	2
3	1
4	4
5	2

Note

In the first example:

- For the first assumption, there are two possible pairs of vertices from which apples can fall from the tree: (4,4),(5,4)(4,4),(5,4).
- For the second assumption there are also two pairs: (5,4),(5,5)(5,4),(5,5).
- For the third assumption there is only one pair: (4,4)(4,4).
- For the fourth assumption, there are 44 pairs: (4,4),(4,5),(5,4),(5,5)(4,4),(4,5),(5,4),(5,5).



Tree from the first example.

For the second example, there are 44 of possible pairs of vertices from which apples can fall: (2,3),(2,2),(3,2),(3,3)(2,3),(2,2),(3,2),(3,3). For the second assumption, there is only one possible pair: (2,3)(2,3). For the third assumption, there are two pairs: (3,2),(3,3)(3,2),(3,3).

蒋子轩23工学院 清晰明了的程序， custom stack.

```

1 def build_tree(edges):
2     tree = {}
3     for edge in edges:
4         u, v = edge
5         tree.setdefault(u, []).append(v)
6         tree.setdefault(v, []).append(u)
7     return tree
8
9 def count_leaves(tree, leaves_count):
10    stack = [(1, 0, 0)] # 节点, 阶段标志, 父节点
11    while stack:
12        vertex, stage, parent = stack.pop()
13
14        if stage == 0:
15            stack.append((vertex, 1, parent))
16            for child in tree[vertex]:
17                if child != parent:
18                    stack.append((child, 0, vertex))
19        else:
20            if len(tree[vertex]) == 1 and vertex != 1:
21                leaves_count[vertex] = 1
22            else:
23                child_count = 0
  
```

```

24         for child in tree[vertex]:
25             if child != parent:
26                 child_count += leaves_count[child]
27
28     leaves_count[vertex] = child_count # 当前节点的叶子节点数等于其子节点的叶
子节点数之和
29
30 def process_assumptions(tree, leaves_count, assumptions):
31     for x, y in assumptions:
32         result = leaves_count[x] * leaves_count[y]
33         print(result)
34
35 t = int(input())
36 for _ in range(t):
37     n = int(input())
38     edges = []
39     for _ in range(n - 1):
40         edges.append(tuple(map(int, input().split())))
41
42     tree = build_tree(edges)
43     leaves_count = {node: 0 for node in range(1, n + 1)}
44     count_leaves(tree, leaves_count)
45     # print(tree, leaves_count)
46     q = int(input())
47     assumptions = []
48     for _ in range(q):
49         assumptions.append(tuple(map(int, input().split())))
50
51     process_assumptions(tree, leaves_count, assumptions)
52

```

蒋子轩23工学院 清晰明了的程序，dfs with thread.

```

1 import sys
2 import threading
3 sys.setrecursionlimit(1 << 30)
4 threading.stack_size(2*10**8)
5
6
7 def main():
8     def build_tree(edges):
9         tree = {}
10        for edge in edges:
11            u, v = edge
12            tree.setdefault(u, []).append(v)
13            tree.setdefault(v, []).append(u)
14        return tree
15
16    def count_leaves(tree, vertex, parent, leaves_count):
17        child_count = 0

```

```

18     for child in tree[vertex]:
19         if child != parent:
20             child_count += count_leaves(tree, child, vertex, leaves_count)
21     #if len(tree[vertex]) == 1 and vertex != parent: # 当前节点是叶子节点
22     if len(tree[vertex]) == 1 and vertex != 1:
23         leaves_count[vertex] = 1
24     return 1
25     leaves_count[vertex] = child_count # 当前节点的叶子节点数等于其子节点的叶子节点数之
26     和
27
28 def process_assumptions(tree, leaves_count, assumptions):
29     for x, y in assumptions:
30         result = leaves_count[x] * leaves_count[y]
31         print(result)
32
33 t = int(input())
34 for _ in range(t):
35     n = int(input())
36     edges = []
37     for _ in range(n - 1):
38         edges.append(tuple(map(int, input().split())))
39
40     tree = build_tree(edges)
41     leaves_count = {node: 0 for node in range(1, n + 1)}
42     count_leaves(tree, 1, 0, leaves_count) # 从根节点开始遍历计算叶子节点数量
43     #print(tree, leaves_count)
44     q = int(input())
45     assumptions = []
46     for _ in range(q):
47         assumptions.append(tuple(map(int, input().split())))
48
49     process_assumptions(tree, leaves_count, assumptions)
50
51
52 thread = threading.Thread(target=main)
53 thread.start()
54 thread.join()

```

02754: 八皇后

dfs and similar, <http://cs101.openjudge.cn/practice/02754>

描述：会下国际象棋的人都很清楚：皇后可以在横、竖、斜线上不限步数地吃掉其他棋子。如何将8个皇后放在棋盘上（有 8×8 个方格），使它们谁也不能被吃掉！这就是著名的八皇后问题。

对于某个满足要求的8皇后的摆放方法，定义一个皇后串 a 与之对应，即 $a = b_1 b_2 \dots b_8$ ，其中 b_i 为相应摆法中第 i 行皇后所处的列数。已经知道8皇后问题一共有92组解（即92个不同的皇后串）。

给出一个数 b ，要求输出第 b 个串。串的比较是这样的：皇后串 x 置于皇后串 y 之前，当且仅当将 x 视为整数时比 y 小。

八皇后是一个古老的经典问题：如何在一张国际象棋的棋盘上，摆放8个皇后，使其任意两个皇后互相不受攻击。该问题由一位德国国际象棋排局家 Max Bezzel 于 1848年提出。严格来说，那个年代，还没有“德国”这个国家，彼时称作“普鲁士”。1850年，Franz Nauck 给出了第一个解，并将其扩展成了“n皇后”问题，即在一张 $n \times n$ 的棋盘上，如何摆放 n 个皇后，使其两两互不攻击。历史上，八皇后问题曾惊动过“数学王子”高斯(Gauss)，而且正是 Franz Nauck 写信找高斯请教的。

02773: 采药

dp, <http://cs101.openjudge.cn/practice/02773>

辰辰是个很有潜能、天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗？

输入

输入的第一行有两个整数 T ($1 \leq T \leq 1000$) 和 M ($1 \leq M \leq 100$)， T 代表总共能够用来采药的时间， M 代表山洞里的草药的数目。接下来的 M 行每行包括两个在 1 到 100 之间（包括 1 和 100）的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出

输出只包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

样例输入

```
1 70 3
2 71 100
3 69 1
4 1 2
```

样例输出

```
1 | 3
```

来源：NOIP 2005

记忆式搜索，13行是分叉。

```
1 import math
2 from functools import lru_cache
3
4 @lru_cache(maxsize = None)
5 def fn(i, s):
```

```

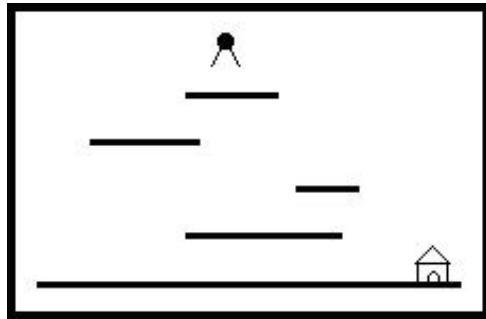
6  # i-th item, knapsack with available capacity s
7
8  if (s < 0):
9      return -math.inf
10 if (i == len(vs)):
11     return 0
12
13 return max(fn(i+1, s), vs[i] + fn(i+1, s-ws[i]))
14
15
16 T, M = map(int, input().split())
17 ws = []
18 vs = []
19 for _ in range(M):
20     t, v = map(int, input().split())
21     ws.append(t)
22     vs.append(v)
23
24 print(fn(0, T))

```

01661: Help Jimmy (难题)

dfs/dp, <http://cs101.openjudge.cn/practice/01661>

"Help Jimmy" 是在下图所示的场景上完成的游戏：



场景中包括多个长度和高度各不相同的平台。地面是最低的平台，高度为零，长度无限。

Jimmy老鼠在时刻0从高于所有平台的某处开始下落，它的下落速度始终为1米/秒。当Jimmy落到某个平台上时，游戏者选择让它向左还是向右跑，它跑动的速度也是1米/秒。当Jimmy跑到平台的边缘时，开始继续下落。Jimmy每次下落的高度不能超过MAX米，不然就会摔死，游戏也会结束。

设计一个程序，计算Jimmy到底地面时可能的最早时间。

输入

第一行是测试数据的组数t ($0 \leq t \leq 20$)。每组测试数据的第一行是四个整数N, X, Y, MAX, 用空格分隔。N是平台的数目（不包括地面），X和Y是Jimmy开始下落的位置的横竖坐标，MAX是一次下落的最大高度。接下来的N行每行描述一个平台，包括三个整数，X1[i], X2[i]和H[i]。H[i]表示平台的高度，X1[i]和X2[i]表示平台左右端点的横坐标。 $1 \leq N \leq 1000$, $-20000 \leq X, X1[i], X2[i] \leq 20000$, $0 < H[i] < Y \leq 20000$ ($i = 1..N$)。所有坐标的单位都是米。

Jimmy的大小和平台的厚度均忽略不计。如果Jimmy恰好落在某个平台的边缘，被视为落在平台上。所有的平台均不重叠或相连。测试数据保证问题一定有解。

输出

对输入的每组测试数据，输出一个整数，Jimmy到底地面时可能的最早时间。

样例输入

```
1 1
2 3 8 17 20
3 0 10 8
4 0 10 13
5 4 14 3
```

样例输出

```
1 23
```

来源：POJ Monthly--2004.05.15, CEOI 2000, POJ 1661, 程序设计实习2007

```
1 # 查达闻 2300011813
2 from functools import lru_cache
3
4 @lru_cache
5 def dfs(x, y, z):
6     for i in range(z+1, N+1):
7         if y - MaxVal > p[i][2]:
8             return 1 << 30
9         elif p[i][0] <= x <= p[i][1]:
10            left = x - p[i][0] + dfs(p[i][0], p[i][2], i)
11            right = p[i][1] - x + dfs(p[i][1], p[i][2], i)
12            return min(left,right)
13
14     if y <= MaxVal:
15         return 0
16     else:
17         return 1 << 30
18
19
20 for _ in range(int(input())):
21     N, ini_x, ini_y, MaxVal = map(int, input().split())
22
23     p = []      #platform
24     p.append([0, 0, 1 << 30]) # 1<<30 大于 20000*2*1000
25     for _ in range(N):
26         p.append([int(x) for x in input().split()])
27     p.sort(key = lambda x:-x[2])
```

```
29 |     print(ini_y + dfs(ini_x, ini_y, 0))
```

02386: Lake Counting

dfs similar, <http://cs101.openjudge.cn/practice/02386>

Due to recent rains, water has pooled in various places in Farmer John's field, which is represented by a rectangle of $N \times M$ ($1 \leq N \leq 100$; $1 \leq M \leq 100$) squares. Each square contains either water ('W') or dry land ('.'). Farmer John would like to figure out how many ponds have formed in his field. A pond is a connected set of squares with water in them, where a square is considered adjacent to all eight of its neighbors.

Given a diagram of Farmer John's field, determine how many ponds he has.

输入

* Line 1: Two space-separated integers: N and M

* Lines 2.. $N+1$: M characters per line representing one row of Farmer John's field. Each character is either 'W' or '.'. The characters do not have spaces between them.

输出

* Line 1: The number of ponds in Farmer John's field.

样例输入

```
1 10 12
2 W.....WW.
3 .WWW.....WWW
4 ....WW....WW.
5 .....WW..
6 .....W..
7 ..W.....W..
8 ..W.W.....WW.
9 W.W.W.....W.
10 ..W.W.....W.
11 ...W.....W.
```

样例输出

```
1 | 3
```

提示

OUTPUT DETAILS:

There are three ponds: one in the upper left, one in the lower left, and one along the right side.

来源: USACO 2004 November

```

1 #1.dfs
2 import sys
3 sys.setrecursionlimit(20000)
4 def dfs(x,y):
5     #标记，避免再次访问
6     field[x][y]='.'
7     for k in range(8):
8         nx,ny=x+dx[k],y+dy[k]
9         #范围内且未访问的lake
10        if 0<=nx<n and 0<=ny<m\
11            and field[nx][ny]=='W':
12            #继续搜索
13            dfs(nx,ny)
14 n,m=map(int,input().split())
15 field=[list(input()) for _ in range(n)]
16 cnt=0
17 dx=[-1,-1,-1,0,0,1,1,1]
18 dy=[-1,0,1,-1,1,-1,0,1]
19 for i in range(n):
20     for j in range(m):
21         if field[i][j]=='W':
22             dfs(i,j)
23             cnt+=1
24 print(cnt)

```

05585: 晶矿的个数

matrices/dfs similar, <http://cs101.openjudge.cn/practice/05585>

在某个区域发现了一些晶矿，已经探明这些晶矿总共有分为两类，为红晶矿和黑晶矿。现在要统计该区域内红晶矿和黑晶矿的个数。假设可以用二维地图m[][]来描述该区域，若m[i][j]为#表示该地点是非晶矿地点，若m[i][j]为r表示该地点是红晶矿地点，若m[i][j]为b表示该地点是黑晶矿地点。一个晶矿是由相同类型的并且上下左右相通的晶矿点组成。现在给你该区域的地图，求红晶矿和黑晶矿的个数。

输入

第一行为k，表示有k组测试输入。

每组第一行为n，表示该区域由n*n个地点组成， $3 \leq n \leq 30$

接下来n行，每行n个字符，表示该地点的类型。

输出

对每组测试数据输出一行，每行两个数字分别是红晶矿和黑晶矿的个数，一个空格隔开。

样例输入

```
1 2
2 6
3 r##bb#
4 ####b##
5 #r##b#
6 #r##b#
7 #r#####
8 ######
9 4
10 #####
11 #rrb
12 #rr#
13 ##bb
```

样例输出

```
1 | 2 2
2 | 1 2
```

```
1 dire = [[-1,0], [1,0], [0,-1], [0,1]]
2
3 def dfs(x, y, c):
4     m[x][y] = '#'
5     for i in range(len(dire)):
6         tx = x + dire[i][0]
7         ty = y + dire[i][1]
8         if m[tx][ty] == c:
9             dfs(tx, ty, c)
10
11 for _ in range(int(input())):
12     n = int(input())
13     m = [[0 for _ in range(n+2)] for _ in range(n+2)]
14
15     for i in range(1, n+1):
16         m[i][1:-1] = input()
17
18     r = 0 ; b=0
19     for i in range(1, n+1):
20         for j in range(1, n+1):
21             if m[i][j] == 'r':
22                 dfs(i, j, 'r')
23                 r += 1
24             if m[i][j] == 'b':
25                 dfs(i,j,'b')
26                 b += 1
27
28 print(r, b)
```

19930: 寻宝

bfs, <http://cs101.openjudge.cn/practice/19930>

Billy获得了一张藏宝图，图上标记了普通点（0），藏宝点（1）和陷阱（2）。按照藏宝图，Billy只能上下左右移动，每次移动一格，且途中不能经过陷阱。现在Billy从藏宝图的左上角出发，请问他是否能到达藏宝点？如果能，所需最短步数为多少？

输入

第一行为两个整数m,n，分别表示藏宝图的行数和列数。(m<=50,n<=50)

此后m行，每行n个整数（0, 1, 2），表示藏宝图的内容。

输出

如果不能到达，输出'NO'。

如果能到达，输出所需的最短步数（一个整数）。

样例输入

```
1 样例输入1:  
2 3 4  
3 0 0 2 0  
4 0 2 1 0  
5 0 0 0 0  
6  
7 样例输出1:  
8 5
```

样例输出

```
1 样例输入2:  
2 2 2  
3 0 2  
4 2 1  
5  
6 样例输出2:  
7 NO
```

提示

每张藏宝图有且仅有一个藏宝点。

输入保证左上角（起点）不是陷阱。

来源：by cs101-2009 邵天泽

其实所有求最短、最长的问题都能用heapq实现，在图搜索中搭配bfs尤其好用。

```

2 import heapq
3 def bfs(x,y):
4     d=[[-1,0],[1,0],[0,1],[0,-1]]
5     queue=[]
6     heapq.heappush(queue,[0,x,y])
7     check=set()
8     check.add((x,y))
9     while queue:
10         step,x,y=map(int,heapq.heappop(queue))
11         if martix[x][y]==1:
12             return step
13         for i in range(4):
14             dx,dy=x+d[i][0],y+d[i][1]
15             if martix[dx][dy]!=2 and (dx,dy) not in check:
16                 heapq.heappush(queue,[step+1,dx,dy])
17                 check.add((dx,dy))
18     return "NO"
19
20 m,n=map(int,input().split())
21 martix=[[2]*(n+2)]+[[2]+list(map(int,input().split()))+[2] for i in range(m)]+[[2]*
22 (n+2)]
23 print(bfs(1,1))

```

20106: 走山路

<http://cs101.openjudge.cn/routine/20106/>

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

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

样例输出

```
1 2
2 3
3 NO
4
5 解释:
6 第一组: 从左上角到右下角, 要上1再下来, 所需体力为2
7 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
8 第三组: 左下角周围都是"#", 不可以经过, 因此到不了
```

来源: cs101-2019 张翔宇

注意 line 9: v.add(..), 在heappop之后, 保证最优的才入v。

```
1 # 23 蒋子轩
2 from heapq import heappop, heappush
3
4 def bfs(x1, y1):
5     q = [(0, x1, y1)]
6     v = set()
7     while q:
8         t, x, y = heappop(q)
9         v.add((x, y))
10        if x == x2 and y == y2:
11            return t
12        for dx, dy in dir:
13            nx, ny = x+dx, y+dy
14            if 0 <= nx < m and 0 <= ny < n and ma[nx][ny] != '#' and (nx, ny) not in
15                v:
16                nt = t+abs(int(ma[nx][ny])-int(ma[x][y]))
17                heappush(q, (nt, nx, ny))
18    return 'NO'
19
20 m, n, p = map(int, input().split())
21 ma = [list(input().split()) for _ in range(m)]
22 dir = [(1, 0), (-1, 0), (0, 1), (0, -1)]
23 for _ in range(p):
24     x1, y1, x2, y2 = map(int, input().split())
```

```

25     if ma[x1][y1] == '#' or ma[x2][y2] == '#':
26         print('NO')
27         continue
28     print(bfs(x1, y1))

```

```

1 # 23 苏王捷
2
3 import heapq
4 m, n, p = map(int, input().split())
5 martix = [list(input().split())for i in range(m)]
6 dir = [(-1, 0), (1, 0), (0, 1), (0, -1)]
7 for _ in range(p):
8     sx, sy, ex, ey = map(int, input().split())
9     if martix[sx][sy] == "#" or martix[ex][ey] == "#":
10        print("NO")
11        continue
12     vis, heap, ans = set(), [], []
13     heapq.heappush(heap, (0, sx, sy))
14     vis.add((sx, sy, -1))
15     while heap:
16         tire, x, y = heapq.heappop(heap)
17         if x == ex and y == ey:
18             ans.append(tire)
19             for i in range(4):
20                 dx, dy = dir[i]
21                 x1, y1 = dx+x, dy+y
22                 if 0 <= x1 < m and 0 <= y1 < n and martix[x1][y1] != "#" and (x1, y1, i)
23                 not in vis:
24                     t1 = tire+abs(int(martix[x][y])-int(martix[x1][y1]))
25                     heapq.heappush(heap, (t1, x1, y1))
26                     vis.add((x1, y1, i))
27     print(min(ans) if ans else "NO")

```

04115: 鸣人和佐助

bfs, <http://cs101.openjudge.cn/practice/04115/>

佐助被大蛇丸诱骗走了，鸣人在多少时间内能追上他呢？



NARUTO 火影忍者
Narutoen.52pk.com

有时间来对我穷追不舍

已知一张地图（以二维矩阵的形式表示）以及佐助和鸣人的位置。地图上的每个位置都可以走到，只不过有些位置上有大蛇丸的手下，需要先打败大蛇丸的手下才能到这些位置。鸣人有一定数量的查克拉，每一个单位的查克拉可以打败一个大蛇丸的手下。假设鸣人可以往上下左右四个方向移动，每移动一个距离需要花费1个单位时间，打败大蛇丸的手下不需要时间。如果鸣人查克拉消耗完了，则只可以走到没有大蛇丸手下的位置，不可以再移动到有大蛇丸手下的位置。佐助在此期间不移动，大蛇丸的手下也不移动。请问，鸣人要追上佐助最少需要花费多少时间？

输入

输入的第一行包含三个整数：M，N，T。代表M行N列的地图和鸣人初始的查克拉数量T。 $0 < M, N < 200$, $0 \leq T < 10$

后面是M行N列的地图，其中@代表鸣人，+代表佐助。*代表通路，#代表大蛇丸的手下。

输出

输出包含一个整数R，代表鸣人追上佐助最少需要花费的时间。如果鸣人无法追上佐助，则输出-1。

样例输入

```
1 样例输入1
2 4 4 1
3 #@##
4 * * ##
5 # ## +
6 * * *
7
8 样例输入2
9 4 4 2
10 #@##
11 * * ##
12 # ## +
13 * * *
```

样例输出

```
1 样例输出1
2 6
3
4 样例输出2
5 4
```

```
1 # 2300011075 苏王捷 工学院
2 ...
3 这段代码是一个迷宫求解的问题。主要使用了广度优先搜索算法来找到从起点到终点的最短路径。
4
5 首先，代码导入了deque模块来实现队列数据结构。
6 然后，定义了一个Node类，表示迷宫中的一个节点。它有四个属性：x和y表示节点的坐标，
7 tools表示节点当前拥有的工具数，steps表示从起点到达该节点的步数。
8
9 接下来，读取输入的迷宫信息，包括迷宫的大小M和N，以及可以使用的工具数T。maze是一个二维列表，
10 表示迷宫的格子，其中'@'表示起点，'+'表示终点，'*'表示障碍物。
11
12 创建了一个visit列表，用于记录节点是否被访问过。visit是一个三维列表，
13 三个维度分别表示行、列和工具数。
14
15 定义了directions列表，包含四个方向的偏移量。
16
17 通过遍历迷宫，找到起点和终点的位置，并设置起点节点的属性。
18
19 使用广度优先搜索算法，通过一个队列queue来依次处理节点。从起点开始，判断四个方向的相邻节点：
20 如果是'*'表示可以直接通过，将其加入队列并标记为已访问；如果是'#'表示需要使用一个工具，
21 才能通过，判断当前节点是否还有工具可用，如果有则减少一个工具，并将该节点加入队列并标记为已访问。
22
23 如果队列为空，即无法到达终点，则输出"-1"；如果找到终点，输出步数，并将flag标记为1，退出循环。
24
25 最后，判断flag是否为0，如果是说明无法找到终点，输出"-1"。
26 ...
27
28 from collections import deque
29
30
31 class Node:
32     def __init__(self, x, y, tools, steps):
33         self.x = x
34         self.y = y
35         self.tools = tools
36         self.steps = steps
37
38
39 M, N, T = map(int, input().split())
40 maze = [list(input()) for _ in range(M)]
41 visit = [[[0] * (T+1) for _ in range(N)] for _ in range(M)]
```

```

42 directions = [[-1, 0], [1, 0], [0, -1], [0, 1]]
43 start = end = None
44 flag = 0
45 for i in range(M):
46     for j in range(N):
47         if maze[i][j] == '@':
48             start = Node(i, j, T, 0)
49             visit[i][j][T] = 1
50         if maze[i][j] == '+':
51             end = (i, j)
52             maze[i][j] = '*'
53
54 queue = deque([start])
55 while queue:
56     node = queue.popleft()
57     if (node.x, node.y) == end:
58         print(node.steps)
59         flag = 1
60         break
61     for direction in directions:
62         nx, ny = node.x+direction[0], node.y+direction[1]
63         if 0 <= nx < M and 0 <= ny < N:
64             if maze[nx][ny] == '*':
65                 if not visit[nx][ny][node.tools]:
66                     queue.append(Node(nx, ny, node.tools, node.steps+1))
67                     visit[nx][ny][node.tools] = 1
68             elif maze[nx][ny] == '#':
69                 if node.tools > 0 and not visit[nx][ny][node.tools-1]:
70                     queue.append(Node(nx, ny, node.tools-1, node.steps+1))
71                     visit[nx][ny][node.tools-1] = 1
72
73 if not flag:
74     print("-1")
75

```

04116: 拯救行动

bfs, <http://cs101.openjudge.cn/practice/04116/>

公主被恶人抓走，被关押在牢房的某个地方。牢房用 $N \times M$ ($N, M \leq 200$) 的矩阵来表示。矩阵中的每项可以代表道路 (@)、墙壁 (#)、和守卫 (x)。

英勇的骑士 (r) 决定孤身一人去拯救公主 (a)。我们假设拯救成功的表示是“骑士到达了公主所在的位置”。由于在通往公主所在位置的道路中可能遇到守卫，骑士一旦遇到守卫，必须杀死守卫才能继续前进。

现假设骑士可以向上、下、左、右四个方向移动，每移动一个位置需要1个单位时间，杀死一个守卫需要花费额外的1个单位时间。同时假设骑士足够强壮，有能力杀死所有的守卫。

给定牢房矩阵，公主、骑士和守卫在矩阵中的位置，请你计算拯救行动成功需要花费最短时间。

输入

第一行为一个整数S, 表示输入的数据的组数 (多组输入)

随后有S组数据，每组数据按如下格式输入

1、两个整数代表N和M, (N, M <= 200).

2、随后N行，每行有M个字符。 "@"代表道路， "a"代表公主， "r"代表骑士， "x"代表守卫， "#"代表墙壁。

输出

如果拯救行动成功，输出一个整数，表示行动的最短时间。

如果不可能成功，输出"Impossible"

样例输入

```
1 2
2 7 8
3 #@#####@#
4 #@a#@@r@
5 #@@#x@@@#
6 @@#@#@#@#
7 #@@@#@@@#
8 @#@@@@@@#
9 @@@@@@@@#
10 13 40
11 @x@@##x@#x@x#xxxx##@#x@x@#x#@#x#@#x@#x@#x@#
12 xx##@#x@x@#@##xx@@#@#@x@#@#x@xxx@@#@#x@#x@#@#
13 #@x#@x#@x#@##@#@x#@xx#@xx@#@x##@#@#@#@#x@#@x@x@#
14 @##@x@#@x@x#@#@#xxxx#@@x@x@#@#x@#@x@#@x@#@#x@##
15 @#xxxx##@#@x##@x@xxx@#@#x@x####@#@#@x@#x##@#@#
16 #@xxx#@#x####xxx@#@#xx@#@#@x@xxx#@#xxx@x#######
17 #x@xxxx#@x@#@#@##@#@x#@xx#@xxx@#@xx#@######x#@x#
18 xx##@#@x##@x##@x#@#@x#@#xx@#@#x@#@##@#@##@xx@#@#
19 x#@x#@#x#@#x#@##@#@xrx@x#@xxxx#@##@x##@xx#@#x@xx@#
20 #x@#@#@##@##@##@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#
21 x#@xx@x####@#@xxx#@#@#x@#@##@#@#@##@#@x#@#@#@#@#
22 #@#@x#@x#@x#@x####@#@#@#@#@#@#@#@#@#@#@#@#@#@#
23 #x#@x#@x######@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#@#
```

样例输出

1	13
2	7

```
1 # 用时间来扩展 bfs 的下一个节点
2 #from collections import deque
3 from heapq import heappush, heappop
4
5 dx = [-1, 1, 0, 0]
6 dy = [0, 0, -1, 1]
```

```

7
8
9 def bfs(matrix, start):
10    n, m = len(matrix), len(matrix[0])
11    visited = [[False for _ in range(m)] for _ in range(n)]
12    #q = deque([(start[0], start[1], 0)])
13    q = []
14    heappush(q, (0, start[0], start[1]))
15    visited[start[0]][start[1]] = True
16    while len(q) != 0:
17        #x, y, time = q.popleft()
18        time, x, y = heappop(q)
19        for i in range(4):
20            nx, ny = x + dx[i], y + dy[i]
21            if 0 <= nx < n and 0 <= ny < m and not visited[nx][ny]:
22                if matrix[nx][ny] == "a":
23                    #ans.append(time+1)
24                    return time + 1
25                elif matrix[nx][ny] == "@":
26                    #q.append((nx, ny, time + 1))
27                    heappush(q, (time + 1, nx, ny))
28                    visited[nx][ny] = True
29                elif matrix[nx][ny] == "x":
30                    #q.append((nx, ny, time + 2))
31                    heappush(q, (time + 2, nx, ny))
32                    visited[nx][ny] = True
33
34    return "Impossible"
35
36
37 S = int(input())
38 for _ in range(S):
39    N, M = map(int, input().split())
40    matrix = [list(input()) for _ in range(N)]
41    start = None
42    ans = []
43    for i in range(N):
44        for j in range(M):
45            if matrix[i][j] == "r":
46                start = (i, j)
47                break
48    print(bfs(matrix, start))
49    # if ans == []:
50    #     print("Impossible")
51    # else:
52    #     print(min(ans))
53

```

04129: 变换的迷宫

bfs, <http://cs101.openjudge.cn/practice/04129>

你现在身处一个 $R \times C$ 的迷宫中，你的位置用 "S" 表示，迷宫的出口用 "E" 表示。

迷宫中有一些石头，用 "#" 表示，还有一些可以随意走动的区域，用 "." 表示。

初始时间为 0 时，你站在地图中标记为 "S" 的位置上。你每移动一步（向上下左右方向移动）会花费一个单位时间。你必须一直保持移动，不能停留在原地不走。

当前时间是 K 的倍数时，迷宫中的石头就会消失，此时你可以走到这些位置上。在其余的时间里，你不能走到石头所在的位置。

求你从初始位置走到迷宫出口最少需要花费多少个单位时间。

如果无法走到出口，则输出 "Oop!"。

输入

第一行是一个正整数 T，表示有 T 组数据。

每组数据的第一行包含三个用空格分开的正整数，分别为 R、C、K。

接下来的 R 行中，每行包含了 C 个字符，分别可能是 "S"、"E"、"#" 或 "."。

其中， $0 < T \leq 20$, $0 < R, C \leq 100$, $2 \leq K \leq 10$ 。

输出

对于每组数据，如果能够走到迷宫的出口，则输出一个正整数，表示最少需要花费的单位时间，否则输出 "Oop!"。

样例输入

```
1 1
2 6 6 2
3 ...S..
4 ...#..
5 .#.....
6 ...#..
7 ...#..
8 ...#E#.
```

样例输出

```
1 | 7
```

容易想到广搜，但是数据大，会超时，需要剪枝。由于每过 k 单位时间，石头就会消失一次，那么当我们站在某点 (x, y) 时，时间为 $t+k$ 和 t 时，它们之后行走面临的情境是完全一样的，那就意味着，对于某个状态的时间，我们可以取模后作为 $\text{visited}[x][y][\text{time}]$ 的第三个变量，如果取模后的值代入发现已经访问过，那说明之前已经有更优秀的情况出现过，不必再继续搜索了。思路参考：<https://blog.csdn.net/dhc65376/article/details/101555903>

```
1 arr2 = lambda m,n : [ [ ' ' for j in range(n)] for i in range(m) ]
```

```

2 arr3 = lambda m,n,l : [ [ [False for k in range(l)] for j in range(n)] for i in
3   range(m) ]
4
5 N = 100
6 K = 10
7
8 class Node:
9     def __init__(self, r=0, c=0, t=0):
10        self.row = r
11        self.col = c
12        self.time = t
13
14 dr = [-1, 1, 0, 0]
15 dc = [0, 0, -1, 1]
16
17 for _ in range(int(input())):
18     maze = arr2(N, N)          # 注意不同数据组之间的初始化
19     vis = arr3(N, N, K)
20     q = []
21     r,c,k = map(int, input().split())
22     for i in range(r):
23         maze[i][:c] = list(input())
24
25     tr = tc = cnt = 0;
26     for i in range(r):
27         for j in range(c):
28             if maze[i][j] == 'S':
29                 q.append(Node(i, j))
30                 vis[i][j][0] = True
31                 cnt += 1
32                 if cnt == 2: break
33             elif maze[i][j] == 'E':
34                 tr = i
35                 tc = j
36                 cnt += 1
37                 if cnt == 2: break
38
39     while(len(q)):
40         t = q[0] # t : Node
41         if t.row == tr and t.col == tc: break
42         q.pop(0)
43         for i in range(4):
44             nrow = t.row + dr[i]
45             ncol = t.col + dc[i]
46
47             if nrow < 0 or nrow >= r or ncol < 0 or ncol >= c:
48                 continue
49
50             # 剪枝很容易能知道，由于每过k单位时间，石头就会消失一次，那么当我们站在某点 (x,y) 时，
51             # 时间为 t+k 和 t 时，它们之后行走面临的情境是完全一样的，那就意味着，
52             # 对于某个状态的时间，我们可以取模后作为 visited[x][y][time] 的第三个变量，

```

```
52     # 如果取模后的值代入发现已经访问过，那说明之前已经有更优越的情况出现过，不必再继续搜索
53     了。
54
55     if vis[nrow][ncol][(t.time + 1) % k]:
56         continue
57
58     # 时间是k 的倍数时，迷宫中的石头就会消失
59     if (t.time + 1) % k and maze[nrow][ncol] == '#':
60         continue;
61     vis[nrow][ncol][(t.time + 1) % k] = True
62     q.append(Node(nrow, ncol, t.time + 1))
63
64 if len(q) == 0:
65     print("Oop!")
66 else:
67     print(q[0].time)
```