【BFS】-跳马问题

题目描述与示例

题目描述

输入 `m` 和 `n` 两个数， `m` 和 `n` 表示一个 `m*n` 的棋盘。输入棋盘内的数据。棋盘中存在数字和 `"."` 两种字符，如果是数字表示该位置是一匹马，如果是 `"."` 表示该位置为空的，棋盘内的数字表示为该马能走的最大步数。

例如棋盘内某个位置一个数字为 `k` ，表示该马只能移动 `0~k` 步的距离。

棋盘内的马移动类似于中国象棋中的马移动，先在水平或者垂直方向上移动一格，然后再将其移动到对角线位置。

棋盘内的马可以移动到同一个位置，同一个位置可以有多匹马。

请问能否将棋盘上所有的马移动到同一个位置，若可以请输出移动的最小步数。若不可以输出 `0` 。

输入描述

输入 `m` 和 `n` 两个数， `m` 和 `n` 表示一个 `m*n` 的棋盘。输入棋盘内的数据。

输出描述

能否将棋盘上所有的马移动到同一个位置，若可以请输入移动的最小步数。若不可以输出 `0` 。

示例一

输入

```
3 2
. .
2 .
. .
```

输出

```
0
```

示例二

输入

```
3 5
4 7 . 4 8
4 7 4 4 .
7 . . . .
```

输出

```
17
```

解题思路

单匹马的跳跃情况

假设已知某匹马的坐标和最大跳跃步数，则可以用BFS计算出该匹马能够到达地图上某个点的最小步数。比如对于以下初始位于 `(0, 0)` 位置最多能跳 `4` 步的马

```
4 . . . .
. . . . .
. . . . .
```

考虑它跳跃到地图上各个点所花费的步数

```
跳跃0步
0 . . . .
. . . . .
. . . . .

跳跃1步
0 . . . .
. . 1 . .
. 1 . . .

```

```
跳跃2步
0 . 2 . 2
. . 1 2 .
2 1 . . 2

跳跃3步
0 3 2 3 2
3 . 1 2 3
2 1 . 3 2

跳跃4步
0 3 2 3 2
3 4 1 2 3
2 1 4 3 2
```

因此可以通过BFS过程得到这匹马可以到达的最终状态。其代码如下

```python
from collections import deque

DIERECTIONS = [(1, 2), (1, -2), (-1, 2), (-1, -2), (2, 1), (2, -1), (-2, 1), (-2, -1)]

def bfs4SingleHorse(i, j, m, n, step):
    mat = [[-1] * n for _ in range*(m)]
    mat[i][j] = 0
    q = deque()
    q.append((i, j))
    level = 0
    while q:
        level += 1
        if level > step:
            break
        qSize = len(q)
        for _ in range(qSize):
            cur_i, cur_j = q.popleft()
            for di, dj in DIERECTIONS:
                nxt_i, nxt_j = cur_i + di, cur_j + dj
                if 0 <= nxt_i < m and 0 <= nxt_j < n and mat[nxt_i][nxt_j] == -1:
                    mat[nxt_i][nxt_j] = level
                    q.append((nxt_i, nxt_j))

    return mat
```

多匹马的跳跃情况

对于每一匹马，都可以计算出对应的二维矩阵 `mat` 。考虑多匹马的情况，将所有马的 `mat` 叠加成一个总的二维矩阵 `ans_mat` ，对于每一个点 `(x, y)` 而言，其逻辑如下

- 若 `ans_mat[x][y]` 已经为 `-1` ，说明有其他马无法到达点 `(x,y)`
- 若某匹马的 `mat[x][y]` 为 `-1` ，说明这匹马无法到达点 `(x,y)` ，将 `ans_mat[x][y]` 改为 `-1`
- `ans_mat[x][y]` 和 `mat[x][y]` 均不为 `-1` ，则将 `mat[x][y]` 叠加到 `ans_mat[x][y]` 中

考虑 `2` 匹马的简单例子，可以从以下例子看出上述逻辑。假设初始矩阵为

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

那么位置为 `(0, 0)` 的马的最终可到达情况矩阵 `mat` 为

```
1 0 3 2 3 2
2 3 . 1 2 3
3 2 1 . 3 2
```

位置为 `(1, 2)` 的马的最终可到达情况矩阵 `mat` 为

```
1 1 . . . 1
2 . . 0 . .
3 1 . . . 1
```

其中 `-1` 用 `.` 来表示。两者的叠加结果为

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

可以看出，所有马跳到同一个位置的最小的步数就为 `1` 。

代码

Python

```python
# 题目：【BFS】2024E-跳马问题
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：BFS
# 代码看不懂的地方，请直接在群上提问


from collections import deque
from math import inf


# 马走”日“字型的八个方向数组
DIERECTIONS = [(1, 2), (1, -2), (-1, 2), (-1, -2), (2, 1), (2, -1), (-2, 1),
(-2, -1)]


# 单匹马进行BFS的函数
# (i, j)为马的起始位置
# step为马能够走的最大步数
def bfs4SingleHorse(i, j, m, n, step):
    # 记录这匹🐎最终的跳跃情况的数组，
    # 初始化每一个位置为-1，表示暂且无法到达
    # mat也同时可以作为check_list的作用
    mat = [[-1] * n for _ in range(m)]
    # 马所在的初始位置(i,j)设置到达步数为0
    mat[i][j] = 0
    q = deque()
    q.append((i, j))
    # BFS的层数，表示跳到某个位置需要的步数
    level = 0
    # 进行BFS
    while q:
        # 层数+1
        level += 1
        # 如果此时BFS的层数已经超过了这匹马能够跳跃的最大步数step
        # 则直接退出循环
        if level > step:
```

```
37              break
38          qSize = len(q)
39          # 遍历该层的所有点
40          for _ in range(qSize):
41              # 弹出队头元素，获得当前点(cur_i, cur_j)
42              cur_i, cur_j = q.popleft()
43              # 考虑当前点走“日”字型的八个方向
44              for di, dj in DIERECTIONS:
45                  # 计算下一个点的到达位置(nxt_i, nxt_j)
46                  nxt_i, nxt_j = cur_i + di, cur_j + dj
47                  # 如果下一个点没有越界，且之前尚未经过（mat起到check_list的作用）
48                  if 0 <= nxt_i < m and 0 <= nxt_j < n and mat[nxt_i][nxt_j] ==
-1:
49                      # 把mat[nxt_i][nxt_j]修改为到达该点(nxt_i, nxt_j)的最小步数
50                      mat[nxt_i][nxt_j] = level
51                      # 同时该点也需要加入队列中，继续做BFS
52                      q.append((nxt_i, nxt_j))
53
54      # 做完BFS，将mat传出函数外
55      return mat
56
57
58  # 输入地图的长m，宽n
59  m, n = map(int, input().split())
60  grid = list()
61  for _ in range(m):
62      # 输入地图，由于存在字符'.'
63      # 所以不需要转化成int整数，储存字符串数组即可
64      grid.append(list(input().split()))
65
66
67  # 初始化ans_mat，
68  # ans_mat[x][y]表示【所有马】到达点(x,y)所需的总步数
69  # 如果无法到达，则最终会被标记为-1
70  ans_mat = [[0] * n for _ in range(m)]
71
72  # 双重循环，遍历原grid中每一个点
73  for i in range(m):
74      for j in range(n):
75          # 如果这个点是数字，则可以计算这匹马最终跳跃状态对应的mat
76          # 其中最大跳跃步数为int(grid[i][j])
77          if grid[i][j] != ".":
78              mat = bfs4SingleHorse(i, j, m, n, int(grid[i][j]))
79              # 对于算出来的mat，再次遍历每一个位置，更新ans_mat
80              for x in range(m):
81                  for y in range(n):
82                      # 如果ans[x][y]已经为-1，说明有其他马无法到达点(x,y)
```

```
                    # 如果mat[x][y]为-1，说明这匹马无法到达点(x,y)
                    # 无论是上述那种情况，都应该把ans_mat[x][y]改为-1
                    if mat[x][y] == -1 or ans_mat[x][y] == -1:
                        ans_mat[x][y] = -1
                    # 否则，将mat[x][y]的值叠加在ans_mat[x][y]中
                    else:
                        ans_mat[x][y] += mat[x][y]

# 最终需要输出的最终答案
ans = inf
# 遍历ans_mat中的每一个点，
# 计算出ans_mat中不为-1的最小值
for i in range(m):
    for j in range(n):
        if ans_mat[i][j] != -1 and ans > ans_mat[i][j]:
            ans = ans_mat[i][j]

print(0 if ans == inf else ans)
```

Java

```
import java.util.*;

class Main {
    static class Pair {
        int first;
        int second;

        Pair(int first, int second) {
            this.first = first;
            this.second = second;
        }
    }

    static final int[][] DIRECTIONS = {{1, 2}, {1, -2}, {-1, 2}, {-1, -2}, {2, 1}, {2, -1}, {-2, 1}, {-2, -1}};

    static int[][] bfs4SingleHorse(int i, int j, int m, int n, int step) {
        int[][] mat = new int[m][n];
        for (int[] row : mat) {
            Arrays.fill(row, -1);
        }

        mat[i][j] = 0;
        Queue<Pair> q = new LinkedList<>();
```

```
24          q.add(new Pair(i, j));
25          int level = 0;
26
27          while (!q.isEmpty()) {
28              level++;
29              if (level > step) {
30                  break;
31              }
32              int qSize = q.size();
33              for (int k = 0; k < qSize; k++) {
34                  Pair cur = q.poll();
35                  for (int[] dir : DIRECTIONS) {
36                      int ni = cur.first + dir[0];
37                      int nj = cur.second + dir[1];
38                      if (0 <= ni && ni < m && 0 <= nj && nj < n && mat[ni][nj]
    == -1) {
39                          mat[ni][nj] = level;
40                          q.add(new Pair(ni, nj));
41                      }
42                  }
43              }
44          }
45          return mat;
46      }
47
48      public static void main(String[] args) {
49          Scanner scanner = new Scanner(System.in);
50          int m = scanner.nextInt();
51          int n = scanner.nextInt();
52
53          scanner.nextLine(); // consume newline
54
55          String[][] grid = new String[m][n];
56          for (int i = 0; i < m; i++) {
57              String line = scanner.nextLine();
58              String[] tokens = line.split(" ");
59              for (int j = 0; j < n; j++) {
60                  grid[i][j] = tokens[j];
61              }
62          }
63
64          int[][] ansMat = new int[m][n];
65          for (int[] row : ansMat) {
66              Arrays.fill(row, 0);
67          }
68
69          for (int i = 0; i < m; i++) {
```

```
            for (int j = 0; j < n; j++) {
                if (!grid[i][j].equals(".")) {
                    int[][] mat = bfs4SingleHorse(i, j, m, n,
Integer.parseInt(grid[i][j]));
                    for (int x = 0; x < m; x++) {
                        for (int y = 0; y < n; y++) {
                            if (mat[x][y] == -1 || ansMat[x][y] == -1) {
                                ansMat[x][y] = -1;
                            } else {
                                ansMat[x][y] += mat[x][y];
                            }
                        }
                    }
                }
            }
        }

        int ans = Integer.MAX_VALUE;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (ansMat[i][j] != -1 && ans > ansMat[i][j]) {
                    ans = ansMat[i][j];
                }
            }
        }

        System.out.println((ans == Integer.MAX_VALUE) ? 0 : ans);
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

using namespace std;

const vector<pair<int, int>> DIRECTIONS = {{1, 2}, {1, -2}, {-1, 2}, {-1, -2},
{2, 1}, {2, -1}, {-2, 1}, {-2, -1}};

vector<vector<int>> bfs4SingleHorse(int i, int j, int m, int n, int step) {
    vector<vector<int>> mat(m, vector<int>(n, -1));
```

```
    mat[i][j] = 0;
    queue<pair<int, int>> q;
    q.push({i, j});
    int level = 0;

    while (!q.empty()) {
        level++;
        if (level > step) {
            break;
        }
        int qSize = q.size();
        for (int k = 0; k < qSize; k++) {
            pair<int, int> cur = q.front();
            q.pop();
            for (auto &dir : DIRECTIONS) {
                int ni = cur.first + dir.first;
                int nj = cur.second + dir.second;
                if (0 <= ni && ni < m && 0 <= nj && nj < n && mat[ni][nj] ==
-1) {
                    mat[ni][nj] = level;
                    q.push({ni, nj});
                }
            }
        }
    }
    return mat;
}

int main() {
    int m, n;
    cin >> m >> n;
    cin.ignore();

    vector<vector<string>> grid(m, vector<string>(n));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }

    vector<vector<int>> ansMat(m, vector<int>(n, 0));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (grid[i][j] != ".") {
                auto mat = bfs4SingleHorse(i, j, m, n, stoi(grid[i][j]));
                for (int x = 0; x < m; x++) {
                    for (int y = 0; y < n; y++) {
```

```
58                          if (mat[x][y] == -1 || ansMat[x][y] == -1) {
59                              ansMat[x][y] = -1;
60                          } else {
61                              ansMat[x][y] += mat[x][y];
62                          }
63                      }
64                  }
65              }
66          }
67      }
68
69      int ans = INT_MAX;
70      for (int i = 0; i < m; i++) {
71          for (int j = 0; j < n; j++) {
72              if (ansMat[i][j] != -1 && ans > ansMat[i][j]) {
73                  ans = ansMat[i][j];
74              }
75          }
76      }
77
78      cout << ((ans == INT_MAX) ? 0 : ans) << endl;
79      return 0;
80  }
81
```

时空复杂度

时间复杂度：`O((NM)^2)`。

空间复杂度：`O(NM)`。