# **E卷-智能驾驶[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**





2,2

10,20

30,40



70



4,4

10,30,30,20

30,30,-1,10

0,20,20,40

10,-1,30,40



70



4,5

10,0,30,-1,10

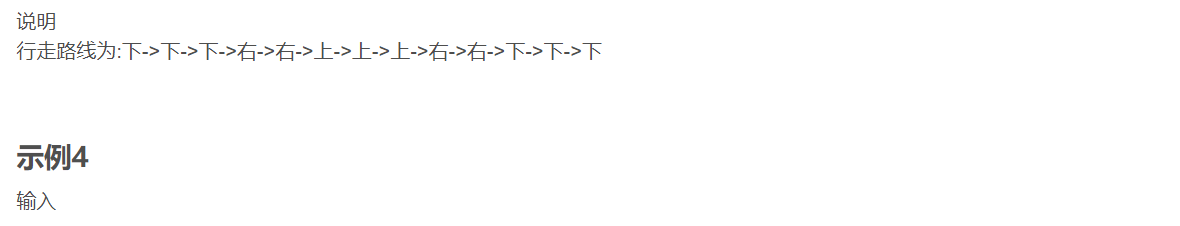
30,0,20,0,20

10,0,10,0,30

10,-1,30,0,10



60



4,4

10,30,30,20

30,30,20,10

10,20,10,40

10,20,30,40



-1





import java.util.PriorityQueue;

import java.util.Scanner;

public class Main {

// 定义一个极大值用来表示无限大

public static int inf = (int) 1e9;

// 地图的行数和列数

public static int n, m;

// 地图数组

public static int[][] g;

// 表示四个方向的移动

public static int[][] nxt = new int[][] { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };

// 表示是否已经访问过的位置

public static Boolean[][][] f;

// 优先队列用于广度优先搜索

public static PriorityQueue<int[]> pq;

// 检查给定的初始油量mid是否能达到终点

public static boolean check(int mid) {

// 如果起点的油耗量大于初始油量，则返回false

if (g[0][0] > mid) {

return false;

}

// 优先队列用于存储当前路径的油耗情况

pq = new PriorityQueue<>((x, y) -> {

return x[0] - y[0];

});

// 初始化访问数组，用于记录在特定油量下，某位置是否被访问过

f = new Boolean[n][m][101];

for (int i = 0; i < n; ++i) {

for (int j = 0; j < m; ++j) {

for (int k = 0; k < 101; ++k) {

f[i][j][k] = false;

}

}

}

// 计算初始剩余油量

int st = mid - g[0][0];

// 将起点加入优先队列

pq.add(new int[] { g[0][0], st, 0, 0 });

f[0][0][st] = true;

// 开始广度优先搜索

while (pq.size() > 0) {

int[] tp = pq.poll();

int val = tp[1], x = tp[2], y = tp[3];

// 如果已经到达终点，返回true

if (x == n - 1 && y == m - 1) {

return true;

}

// 遍历四个方向

for (int k = 0; k < 4; k++) {

int nx = x + nxt[k][0], ny = y + nxt[k][1];

// 检查新的位置是否在地图范围内并且不是障碍物

if (nx < 0 || nx >= n || ny < 0 || ny >= m || g[nx][ny] == 0) {

continue;

}

int dval = val - g[nx][ny];

// 如果当前油量可以到达新位置

if (dval >= 0) {

int need = tp[0] + g[nx][ny];

// 如果新位置是加油站，重置油量

if (g[nx][ny] == -1) {

need += 1;

dval = 100;

}

// 如果新位置以及剩余油量组合没有被访问过，加入优先队列

if (!f[nx][ny][dval]) {

pq.add(new int[] { need, dval, nx, ny });

f[nx][ny][dval] = true;

}

}

}

}

return false;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// 读取地图的行数和列数

String[] tmp = sc.nextLine().split(",");

n = Integer.parseInt(tmp[0]);

m = Integer.parseInt(tmp[1]);

g = new int[n][m];

// 读取地图数据

for (int i = 0; i < n; i++) {

tmp = sc.nextLine().split(",");

for (int j = 0; j < m; j++) {

g[i][j] = Integer.parseInt(tmp[j]);

}

// 起点是障碍物，直接返回-1

if (g[0][0] == 0) {

System.out.println(-1);

return;

}

}

// 二分搜索初始油量

int left = 0, right = 100;

int res = inf;

while (left <= right) {

int mid = left + (right - left) / 2;

if (check(mid)) {

right = mid - 1;

res = mid;

} else {

left = mid + 1;

}

}

// 如果起点是加油站，且找到了结果，输出0

if (g[0][0] == -1 && res != inf) {

System.out.println(0);

} else {

// 输出结果

if (res == inf) {

System.out.println(-1);

} else {

System.out.println(res);

}

}

}

}



import heapq

# 定义一个很大的数作为无穷大

inf = int(1e9)

# 定义四个方向的移动（右、下、左、上）

nxt = [(0, 1), (1, 0), (0, -1), (-1, 0)]

def check(g, n, m, mid):

# 如果起始位置的耗油量大于当前油量，则无法出发

if g[0][0] > mid:

return False

# 优先队列用于存储当前路径的状态

pq = []

# 三维数组用于记录每个位置和剩余油量的访问状态

f = [[[False] \* 101 for \_ in range(m)] for \_ in range(n)]

# 计算初始剩余油量

st = mid - g[0][0]

# 将起始位置加入优先队列

heapq.heappush(pq, (g[0][0], st, 0, 0))

# 标记起始位置和剩余油量的状态

f[0][0][st] = True

while pq:

# 弹出优先队列中耗油量最小的状态

val, st, x, y = heapq.heappop(pq)

# 如果到达终点，返回True

if x == n - 1 and y == m - 1:

return True

# 遍历四个方向

for dx, dy in nxt:

nx, ny = x + dx, y + dy

# 检查新位置是否在地图范围内且不是障碍物

if 0 <= nx < n and 0 <= ny < m and g[nx][ny] != 0:

# 计算新位置的剩余油量

dval = st - g[nx][ny]

# 如果剩余油量大于等于0，则可以到达

if dval >= 0:

# 计算新的耗油量

need = val + g[nx][ny]

# 如果是加油站，将油量加满

if g[nx][ny] == -1:

need += 1

dval = 100

# 如果该状态未被访问过，将其加入队列

if not f[nx][ny][dval]:

heapq.heappush(pq, (need, dval, nx, ny))

f[nx][ny][dval] = True

# 如果无法到达终点，返回False

return False

def main():

# 读取地图大小

data = input().split(",")

n, m = int(data[0]), int(data[1])

# 读取地图数据

g = []

for i in range(n):

row = list(map(int, input().split(",")))

g.append(row)

# 如果起点是障碍物，直接返回-1

if g[0][0] == 0:

print(-1)

return

# 二分查找初始油量

left, right = 0, 100

res = inf

while left <= right:

mid = (left + right) // 2

# 检查当前油量是否可行

if check(g, n, m, mid):

right = mid - 1

res = mid

else:

left = mid + 1

# 输出结果

if g[0][0] == -1 and res != inf:

print(0)

elif res == inf:

print(-1)

else:

print(res)

if \_\_name\_\_ == "\_\_main\_\_":

main()



#include <iostream>

#include <queue>

#include <vector>

using namespace std;

// 定义一个很大的数作为无穷大

const int inf = 1e9;

// 定义地图的行数和列数

int n, m;

// 定义地图矩阵

vector<vector<int>> g;

// 三维数组用于记录每个位置和剩余油量的访问状态

vector<vector<vector<bool>>> f;

// 定义四个方向的移动（右、下、左、上）

int nxt[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

// 检查给定的初始油量是否足够到达终点

bool check(int mid) {

// 如果起始位置的耗油量大于当前油量，则无法出发

if (g[0][0] > mid) return false;

// 优先队列用于存储当前路径的状态，按照耗油量从小到大排序

priority\_queue<vector<int>, vector<vector<int>>, greater<>> pq;

// 初始化访问状态数组

f.assign(n, vector<vector<bool>>(m, vector<bool>(101, false)));

// 计算初始剩余油量

int st = mid - g[0][0];

// 将起始位置加入优先队列

pq.push({g[0][0], st, 0, 0});

// 标记起始位置和剩余油量的状态

f[0][0][st] = true;

while (!pq.empty()) {

// 弹出优先队列中耗油量最小的状态

auto tp = pq.top();

pq.pop();

int val = tp[1], x = tp[2], y = tp[3];

// 如果到达终点，返回true

if (x == n - 1 && y == m - 1) return true;

// 遍历四个方向

for (int k = 0; k < 4; k++) {

int nx = x + nxt[k][0], ny = y + nxt[k][1];

// 检查新位置是否在地图范围内且不是障碍物

if (nx < 0 || nx >= n || ny < 0 || ny >= m || g[nx][ny] == 0) continue;

// 计算新位置的剩余油量

int dval = val - g[nx][ny];

// 如果剩余油量大于等于0，则可以到达

if (dval >= 0) {

int need = tp[0] + g[nx][ny];

// 如果是加油站，将油量加满

if (g[nx][ny] == -1) {

need += 1;

dval = 100;

}

// 如果该状态未被访问过，将其加入队列

if (!f[nx][ny][dval]) {

pq.push({need, dval, nx, ny});

f[nx][ny][dval] = true;

}

}

}

}

// 如果无法到达终点，返回false

return false;

}

// 将输入字符串按逗号分隔并转换为整数数组

vector<int> split(string& s) {

int x = 0, f = 1;

vector<int> res;

for (auto& i : s) {

if (i == ',') {

res.push\_back(x \* f);

x = 0;

f = 1;

} else if (i == '-') {

f = -1;

} else {

x = x \* 10 + i - '0';

}

}

res.push\_back(x \* f);

return res;

}

int main() {

string tmp;

// 读取地图大小

getline(cin, tmp);

vector<int> v = split(tmp);

n = v[0];

m = v[1];

// 读取地图数据

for (int i = 0; i < n; i++) {

getline(cin, tmp);

v = split(tmp);

g.push\_back(v);

}

// 如果起点是障碍物，直接返回-1

if (g[0][0] == 0) {

cout << -1 << endl;

return 0;

}

// 二分查找初始油量

int left = 0, right = 100, res = inf;

while (left <= right) {

int mid = left + (right - left) / 2;

// 检查当前油量是否可行

if (check(mid)) {

right = mid - 1;

res = mid;

} else {

left = mid + 1;

}

}

// 输出结果

if (g[0][0] == -1 && res != inf) {

cout << 0 << endl;

} else {

cout << (res == inf ? -1 : res) << endl;

}

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

// 定义一个很大的数作为无穷大

#define INF 1000000000

// 定义节点结构体，用于存储当前路径的状态

struct Node {

int cost; // 当前路径的总耗油量

int fuel; // 当前剩余油量

int x; // 当前所在的行

int y; // 当前所在的列

};

// 地图大小

int n, m;

// 地图矩阵

int g[100][100];

// 三维数组用于记录每个位置和剩余油量的访问状态

bool f[100][100][101];

// 定义四个方向的移动（右、下、左、上）

int dir[4][2] = { {0, 1}, {1, 0}, {0, -1}, {-1, 0} };

// 检查给定的初始油量是否足够到达终点

bool check(int mid) {

// 如果起始位置的耗油量大于当前油量，则无法出发

if (g[0][0] > mid) return false;

// 优先队列用于存储当前路径的状态

struct Node pq[1000000];

int pq\_size = 0;

// 初始化访问状态数组

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

for (int k = 0; k <= 100; k++)

f[i][j][k] = false;

// 计算初始剩余油量

int startFuel = mid - g[0][0];

// 将起始位置加入优先队列

pq[pq\_size++] = (struct Node) {

g[0][0], startFuel, 0, 0

};

f[0][0][startFuel] = true;

while (pq\_size > 0) {

// 弹出优先队列中最后一个节点

struct Node node = pq[--pq\_size];

int cost = node.cost;

int fuel = node.fuel;

int x = node.x;

int y = node.y;

// 如果到达终点，返回true

if (x == n - 1 && y == m - 1) return true;

// 遍历四个方向

for (int k = 0; k < 4; k++) {

int nx = x + dir[k][0];

int ny = y + dir[k][1];

// 检查新位置是否在地图范围内且不是障碍物

if (nx < 0 || nx >= n || ny < 0 || ny >= m || g[nx][ny] == 0) continue;

// 计算新位置的剩余油量

int remainingFuel = fuel - g[nx][ny];

// 如果剩余油量大于等于0，则可以到达

if (remainingFuel >= 0) {

int newCost = cost + g[nx][ny];

// 如果是加油站，将油量加满

if (g[nx][ny] == -1) {

newCost++;

remainingFuel = 100;

}

// 如果该状态未被访问过，将其加入队列

if (!f[nx][ny][remainingFuel]) {

pq[pq\_size++] = (struct Node) {

newCost, remainingFuel, nx, ny

};

f[nx][ny][remainingFuel] = true;

}

}

}

}

// 如果无法到达终点，返回false

return false;

}

char tmp[100005];

int main() {

// 读取地图大小

scanf("%d,%d", &n, &m);

gets(tmp);

// 读取地图数据

for (int i = 0; i < n; i++) {

gets(tmp);

int sz = strlen(tmp);

int p = 0;

int x = 0;

int f = 1;

for (int j = 0; j < sz; ++j) {

if (tmp[j] == '-') {

f = -1;

} else if (tmp[j] == ',') {

g[i][p] = x \* f;

p += 1;

x = 0;

f = 1;

} else {

x = x \* 10 + tmp[j] - '0';

}

}

g[i][p] = x \* f;

}

// 如果起点是障碍物，直接返回-1

if (g[0][0] == 0) {

printf("-1\n");

return 0;

}

// 二分查找初始油量

int left = 0, right = 100;

int res = INF;

while (left <= right) {

int mid = left + (right - left) / 2;

// 检查当前油量是否可行

if (check(mid)) {

right = mid - 1;

res = mid;

} else {

left = mid + 1;

}

}

// 输出结果

if (g[0][0] == -1 && res != INF) {

printf("0\n");

} else {

printf("%d\n", res == INF ? -1 : res);

}

return 0;

}



const rl = require("readline").createInterface({

input: process.stdin,

output: process.stdout,

});

// 创建一个异步迭代器来读取输入

var iter = rl[Symbol.asyncIterator]();

const readline = async () => (await iter.next()).value;

void (async function () {

const inf = 1e9; // 定义一个很大的数作为无穷大

const directions = [

[0, 1], // 向右移动

[1, 0], // 向下移动

[0, -1], // 向左移动

[-1, 0], // 向上移动

];

let n, m; // 地图的行数和列数

let g = []; // 地图矩阵

let f; // 三维数组用于记录每个位置和剩余油量的访问状态

let pq = []; // 优先队列用于存储当前路径的状态

// 检查给定的初始油量是否足够到达终点

function check(mid) {

// 如果起始位置的耗油量大于当前油量，则无法出发

if (g[0][0] > mid) return false;

pq = []; // 初始化优先队列

f = Array.from({ length: n }, () =>

Array.from({ length: m }, () => Array(101).fill(false))

);

let st = mid - g[0][0]; // 计算初始剩余油量

pq.push([g[0][0], st, 0, 0]); // 将起始位置加入优先队列

f[0][0][st] = true; // 标记起始位置和剩余油量的状态

while (pq.length) {

// 按照耗油量从小到大排序

pq.sort((a, b) => a[0] - b[0]);

let [val, st, x, y] = pq.pop(); // 弹出优先队列中耗油量最小的状态

// 如果到达终点，返回true

if (x === n - 1 && y === m - 1) return true;

// 遍历四个方向

for (let [dx, dy] of directions) {

let nx = x + dx,

ny = y + dy;

// 检查新位置是否在地图范围内且不是障碍物

if (nx < 0 || nx >= n || ny < 0 || ny >= m || g[nx][ny] === 0)

continue;

let dval = st - g[nx][ny]; // 计算新位置的剩余油量

// 如果剩余油量大于等于0，则可以到达

if (dval >= 0) {

let need = val + g[nx][ny];

// 如果是加油站，将油量加满

if (g[nx][ny] === -1) {

need += 1;

dval = 100;

}

// 如果该状态未被访问过，将其加入队列

if (!f[nx][ny][dval]) {

pq.push([need, dval, nx, ny]);

f[nx][ny][dval] = true;

}

}

}

}

// 如果无法到达终点，返回false

return false;

}

// 读取地图大小

let tmp = (await readline()).split(",").map(Number);

n = tmp[0];

m = tmp[1];

// 读取地图数据

for (let i = 0; i < n; i++) {

let row = (await readline()).split(",").map(Number);

g.push(row);

}

// 如果起点是障碍物，直接返回-1

if (g[0][0] === 0) {

console.log(-1);

return;

}

// 二分查找初始油量

let left = 0,

right = 100;

let res = inf;

while (left <= right) {

let mid = Math.floor((left + right) / 2);

// 检查当前油量是否可行

if (check(mid)) {

right = mid - 1;

res = mid;

} else {

left = mid + 1;

}

}

// 输出结果

if (g[0][0] === -1 && res !== inf) {

console.log(0);

} else {

console.log(res === inf ? -1 : res);

}

})();



package main

import (

"container/heap"

"fmt"

"strconv"

"strings"

)

const inf = 1e9 // 定义一个很大的数作为无穷大

// 定义节点结构体，用于存储当前路径的状态

type Node struct {

cost int // 当前路径的总耗油量

fuel int // 当前剩余油量

x int // 当前所在的行

y int // 当前所在的列

}

// 定义优先队列类型

type PriorityQueue []\*Node

// 返回优先队列的长度

func (pq PriorityQueue) Len() int { return len(pq) }

// 比较两个节点的优先级（耗油量小的优先）

func (pq PriorityQueue) Less(i, j int) bool {

return pq[i].cost < pq[j].cost

}

// 交换优先队列中的两个元素

func (pq PriorityQueue) Swap(i, j int) {

pq[i], pq[j] = pq[j], pq[i]

}

// 向优先队列中添加元素

func (pq \*PriorityQueue) Push(x interface{}) {

item := x.(\*Node)

\*pq = append(\*pq, item)

}

// 从优先队列中弹出元素

func (pq \*PriorityQueue) Pop() interface{} {

old := \*pq

n := len(old)

item := old[n-1]

\*pq = old[0 : n-1]

return item

}

var (

dirs = [][2]int{{0, 1}, {1, 0}, {0, -1}, {-1, 0}} // 定义四个方向的移动（右、下、左、上）

n, m int // 地图大小

g [][]int // 地图矩阵

f [][][]bool // 三维数组用于记录每个位置和剩余油量的访问状态

)

// 检查给定的初始油量是否足够到达终点

func check(mid int) bool {

if g[0][0] > mid {

return false // 如果起始位置的耗油量大于当前油量，则无法出发

}

pq := &PriorityQueue{} // 初始化优先队列

heap.Init(pq)

f = make([][][]bool, n) // 初始化访问状态数组

for i := 0; i < n; i++ {

f[i] = make([][]bool, m)

for j := 0; j < m; j++ {

f[i][j] = make([]bool, 101)

}

}

st := mid - g[0][0] // 计算初始剩余油量

heap.Push(pq, &Node{g[0][0], st, 0, 0}) // 将起始位置加入优先队列

f[0][0][st] = true // 标记起始位置和剩余油量的状态

for pq.Len() > 0 {

tp := heap.Pop(pq).(\*Node) // 弹出优先队列中耗油量最小的状态

val, st, x, y := tp.cost, tp.fuel, tp.x, tp.y

if x == n-1 && y == m-1 {

return true // 如果到达终点，返回true

}

for \_, dir := range dirs {

nx, ny := x+dir[0], y+dir[1]

if nx < 0 || nx >= n || ny < 0 || ny >= m || g[nx][ny] == 0 {

continue // 检查新位置是否在地图范围内且不是障碍物

}

dval := st - g[nx][ny] // 计算新位置的剩余油量

if dval >= 0 {

need := val + g[nx][ny]

if g[nx][ny] == -1 {

need++ // 如果是加油站，将油量加满

dval = 100

}

if !f[nx][ny][dval] {

heap.Push(pq, &Node{need, dval, nx, ny}) // 将新状态加入优先队列

f[nx][ny][dval] = true // 标记新状态

}

}

}

}

return false // 如果无法到达终点，返回false

}

func main() {

var input string

fmt.Scan(&input) // 读取地图大小

strSlice := strings.Split(input, ",")

intSlice := make([]int, len(strSlice))

for i, s := range strSlice {

num, err := strconv.Atoi(s)

if err != nil {

return

}

intSlice[i] = num

}

n = intSlice[0]

m = intSlice[1]

g = make([][]int, n)

for i := 0; i < n; i++ {

g[i] = make([]int, m)

fmt.Scanln(&input) // 读取地图数据

strSlice := strings.Split(input, ",")

for j, s := range strSlice {

num, err := strconv.Atoi(s)

if err != nil {

return

}

g[i][j] = num

}

}

if g[0][0] == 0 {

fmt.Println(-1) // 如果起点是障碍物，直接返回-1

return

}

left, right, res := 0, 100, 999999 // 二分查找初始油量

for left <= right {

mid := int((left + right) / 2)

if check(mid) {

right = mid - 1

res = mid

} else {

left = mid + 1

}

}

if g[0][0] == -1 && res != 999999 {

fmt.Println(0) // 如果起点是加油站且可行，输出0

} else {

fmt.Println(res) // 输出结果

}

}