

t0825

[华为OD机试2025C卷 - 最大男生相连数 / 学生方阵 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 流浪地球 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 发广播 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 区间连接器 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 水果摊小买卖 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 贪吃蛇 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 符号运算 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 空栈压数 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 生成回文素数 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 战场索敌 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 字符串拼接 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 攀登者2 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 高效的任务规划 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

华为OD机试2025C卷 - 最大男生相连数 / 学生方阵 (C++ & Python & JAVA & JS & GO)-CSDN博客

学生方阵

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

学校组织活动，将学生排成一个矩形方阵。

请在矩形方阵中找到最大的位置相连的男生数量。

这个相连位置在一个直线上，方向可以是水平的，垂直的，成对角线的或者呈反对角线的。

注：学生个数不会超过10000

输入描述

输入的第一行为矩阵的行数和列数，接下来的n行为矩阵元素，元素间用”，”分隔。

输出描述

输出一个整数，表示矩阵中最长的位置相连的男生个数。

示例1

输入

```
▼ Plain Text |  
1 3,4  
2 F,M,M,F  
3 F,M,M,F  
4 F,F,F,M
```

输出

▼

Plain Text |

1 3

题解

思路： **DFS**

1. 接收输入数据，使用 **字符串分割** 方式解析出学生矩阵并使用二维数组保存。
2. 循环遍历数组，当遇到 **M** 时，采用DFS算法递归往分别往 **水平、垂直、主对角线、副对角线** 进行递归，直到遇到非 **M** 结束，记录出现的最长连续M数目。
3. 解释一下下面代码为什么对于 **水平、垂直、主对角线、副对角线** 只考虑一种延伸方式。一条线段存在两个端点，端点的距离就是这条线段的长度，通过2的逻辑进行处理肯定是找到每条线段的端点的，对于端点只会往单个方向延伸就能求出长度。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5 #include <algorithm>
6
7 using namespace std;
8
9 // 四个搜索方向: 水平、垂直、主对角线、副对角线
10 const vector<vector<int>> DIRECTIONS = {{0, 1}, {1, 0}, {1, 1}, {-1, 1}};
11
12 // 计算从 (row, col) 出发, 在四个方向上的最大连续 'M' 数量
13 int findMaxConnectedM(const vector<vector<string>>& grid, int row, int col) {
14     // 至少起点为M
15     int maxCount = 1;
16     int rows = grid.size();
17     int cols = grid[0].size();
18
19     for (const auto& dir : DIRECTIONS) {
20         int count = 1, r = row, c = col;
21         while (r + dir[0] >= 0 && r + dir[0] < rows && c + dir[1] >= 0 &&
22               c + dir[1] < cols
23               && grid[r + dir[0]][c + dir[1]] == "M") {
24             r += dir[0];
25             c += dir[1];
26             count++;
27         }
28         maxCount = max(maxCount, count);
29     }
30     return maxCount;
31 }
32
33 int main() {
34     string input;
35     getline(cin, input);
36
37     // 解析行数和列数
38     stringstream ss(input);
39     string temp;
40     getline(ss, temp, ',');
41     int rows = stoi(temp);
42     getline(ss, temp);
43     int cols = stoi(temp);
```

```
44
45 // 读取矩阵数据
46 vector<vector<string>> grid(rows, vector<string>(cols));
47 for (int i = 0; i < rows; ++i) {
48     getline(cin, input);
49     stringstream rowStream(input);
50     for (int j = 0; j < cols; ++j) {
51         getline(rowStream, grid[i][j], ',');
52     }
53 }
54
55 int maxMCount = 0; // 记录最大的连续 'M' 数量
56 for (int i = 0; i < rows; ++i) {
57     for (int j = 0; j < cols; ++j) {
58         if (grid[i][j] == "M") {
59             maxMCount = max(maxMCount, findMaxConnectedM(grid, i, j));
60         }
61     }
62 }
63
64 cout << maxMCount << endl; // 输出最大的连续 'M' 数量
65
66 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 四个方向: 水平、垂直、主对角线、副对角线
5     private static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {1, 1}, {-1, 1}};
6
7     // 计算从 (row, col) 出发, 在四个方向上的最大连续 'M' 数量
8     private static int findMaxConnectedM(String[][] grid, int row, int col) {
9         int maxCount = 1;
10        int rows = grid.length, cols = grid[0].length;
11
12        for (int[] dir : DIRECTIONS) {
13            int count = 1, r = row, c = col;
14            while (r + dir[0] >= 0 && r + dir[0] < rows && c + dir[1] >= 0 && c + dir[1] < cols
15                && grid[r + dir[0]][c + dir[1]].equals("M")) {
16                r += dir[0];
17                c += dir[1];
18                count++;
19            }
20            maxCount = Math.max(maxCount, count);
21        }
22        return maxCount;
23    }
24
25    public static void main(String[] args) {
26        Scanner scanner = new Scanner(System.in);
27        String[] dimensions = scanner.nextLine().split(",");
28        int rows = Integer.parseInt(dimensions[0]);
29        int cols = Integer.parseInt(dimensions[1]);
30
31        // 读取矩阵数据
32        String[][] grid = new String[rows][cols];
33        for (int i = 0; i < rows; i++) {
34            grid[i] = scanner.nextLine().split(",");
35        }
36        scanner.close();
37
38        int maxMCount = 0;
39        for (int i = 0; i < rows; i++) {
40            for (int j = 0; j < cols; j++) {
41                if (grid[i][j].equals("M")) {
```

```
43     d, i, j));
44         }
45     }
46 }
47
48     System.out.println(maxMCount);
49 }
}
```

Python

```
1 import sys
2
3 # 四个方向: 水平、垂直、主对角线、副对角线
4 DIRECTIONS = [(0, 1), (1, 0), (1, 1), (-1, 1)]
5
6 # 计算从 (row, col) 出发, 在四个方向上的最大连续 'M' 数量
7 def find_max_connected_m(grid, row, col):
8     max_count = 1
9     rows, cols = len(grid), len(grid[0])
10
11    for dr, dc in DIRECTIONS:
12        count, r, c = 1, row, col
13        while 0 <= r + dr < rows and 0 <= c + dc < cols and grid[r + dr][c + dc] == "M":
14            r += dr
15            c += dc
16            count += 1
17        max_count = max(max_count, count)
18
19    return max_count
20
21 def main():
22     # 读取矩阵尺寸
23     rows, cols = map(int, sys.stdin.readline().strip().split(","))
24
25     # 读取矩阵数据
26     grid = [sys.stdin.readline().strip().split(",") for _ in range(rows)]
27
28     max_m_count = 0
29     for i in range(rows):
30         for j in range(cols):
31             if grid[i][j] == "M":
32                 max_m_count = max(max_m_count, find_max_connected_m(grid,
33 i, j))
34
35     print(max_m_count)
36
37 if __name__ == "__main__":
38     main()
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 const DIRECTIONS = [[0, 1], [1, 0], [1, 1], [-1, 1]];
9
10 // 计算从 (row, col) 出发, 在四个方向上的最大连续 'M' 数量
11 function findMaxConnectedM(grid, row, col) {
12     let maxCount = 1;
13     const rows = grid.length, cols = grid[0].length;
14
15     for (const [dr, dc] of DIRECTIONS) {
16         let count = 1, r = row, c = col;
17         while (r + dr >= 0 && r + dr < rows && c + dc >= 0 && c + dc < cols && grid[r + dr][c + dc] === "M") {
18             r += dr;
19             c += dc;
20             count++;
21         }
22         maxCount = Math.max(maxCount, count);
23     }
24     return maxCount;
25 }
26
27 const inputLines = [];
28 rl.on("line", (line) => {
29     inputLines.push(line);
30 }).on("close", () => {
31     const [rows, cols] = inputLines[0].split(",").map(Number);
32     const grid = inputLines.slice(1).map(row => row.split(","));
33
34     let maxMCount = 0;
35     for (let i = 0; i < rows; i++) {
36         for (let j = 0; j < cols; j++) {
37             if (grid[i][j] === "M") {
38                 maxMCount = Math.max(maxMCount, findMaxConnectedM(grid,
39 i, j));
40             }
41         }
42     }
43     console.log(maxMCount);
44 }
```

44 });

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 四个方向: 水平、垂直、主对角线、副对角线
12 var directions = [][]int{{0, 1}, {1, 0}, {1, 1}, {-1, 1}}
13
14 // 计算从 (row, col) 出发, 在四个方向上的最大连续 'M' 数量
15 func findMaxConnectedM(grid [][]string, row, col int) int {
16     maxCount := 1
17     rows, cols := len(grid), len(grid[0])
18
19     for _, dir := range directions {
20         count, r, c := 1, row, col
21         for r+dir[0] >= 0 && r+dir[0] < rows && c+dir[1] >= 0 && c+dir[1] < cols && grid[r+dir[0]][c+dir[1]] == "M" {
22             r += dir[0]
23             c += dir[1]
24             count++
25         }
26         if count > maxCount {
27             maxCount = count
28         }
29     }
30     return maxCount
31 }
32
33 func main() {
34     scanner := bufio.NewScanner(os.Stdin)
35
36     // 读取矩阵尺寸
37     scanner.Scan()
38     dimensions := strings.Split(scanner.Text(), ",")
39     rows, _ := strconv.Atoi(dimensions[0])
40     cols, _ := strconv.Atoi(dimensions[1])
41
42     // 读取矩阵数据
43     grid := make([][]string, rows)
44     for i := 0; i < rows; i++ {
```

```
45     scanner.Scan()
46     grid[i] = strings.Split(scanner.Text(), ",")
47 }
48
49 maxMCount := 0
50 for i := 0; i < rows; i++ {
51     for j := 0; j < cols; j++ {
52         if grid[i][j] == "M" {
53             maxMCount = max(maxMCount, findMaxConnectedM(grid, i, j))
54         }
55     }
56 }
57
58 fmt.Println(maxMCount)
59 }
60
61 // 返回较大值
62 func max(a, b int) int {
63     if a > b {
64         return a
65     }
66     return b
67 }
```

来自: 华为OD机试2025C卷 – 最大男生相连数 / 学生方阵 (C++ & Python & JAVA & JS & GO)–
[CSDN博客](#)

华为OD机试2025C卷 - 流浪地球 (C++ & Python & JAVA & JS & GO)-CSDN博客

流浪地球

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

流浪地球计划在赤道上均匀部署了N个转向发动机，按位置顺序编号为0~N-1。

1. 初始状态下所有的发动机都是未启动状态；
2. 发动机启动的方式分为“手动启动”和“关联启动”两种方式；
3. 如果在时刻1一个发动机被启动，下一个时刻2与之相邻的两个发动机就会被“关联启动”；
4. 如果准备启动某个发动机时，它已经被启动了，则什么都不用做；
5. 发动机0与发动机N-1是相邻的；

地球联合政府准备挑选某些发动机在某些时刻进行“手动启动”。当然最终所有的发动机都会被启动。

哪些发动机最晚被启动呢？

输入描述

- 第一行两个数字N和E，中间有空格，N代表部署发动机的总个数，E代表计划手动启动的发动机总个数
 $1 < N \leq 1000, 1 \leq E \leq 1000, E \leq N$
- 接下来共E行，每行都是两个数字T和P，中间有空格。T代表发动机的手动启动时刻，P代表此发动机的位置编号。
 $0 \leq T \leq N, 0 \leq P \leq N$

输出描述

- 第一行一个数字N，以回车结束。N代表最后被启动的发动机个数
- 第二行N个数字，中间有空格，以回车结束。每个数字代表发动机的位置编号，从小到大排序

示例1

输入

```
1 8 2  
2 0 2  
3 0 6
```

Plain Text |

输出

```
1 2  
2 0 4
```

Plain Text |

说明

| 8个发动机，时刻0启动2和6号发动机

示例2

输入

```
1 8 2  
2 0 0  
3 1 7
```

Plain Text |

输出

```
1 1  
2 4
```

Plain Text |

说明

| 8个发动机，时刻0手动启动0，时刻1手动启动7.

题解

思路： `BFS` 实现，

1. 使用队列模拟进行 `BFS` 遍历，定义 `count` 记录已经启动过的发动机数量,初始定义 `count = 0`。并使用 `curTime` 记录所处 **当前时间**

2. `curTime` 启动的发动机分为：到达指定启动时间的发动机(注意发动机可能在指定时间之前已经被关联启动了)；上一秒启动的发动机关联启动周围的发动机。使用队列 `queue` 保存这一秒启动的发动机，用于下一秒进行关联启动。
3. 当 `queue.size() + count == N` 说明这是最后一次启动，队列中中的id就是需要的结果。转换为数组，进行升序排序。输出数组长度和数组中的元素即可

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<queue>
4 #include<algorithm>
5 #include<unordered_map>
6 using namespace std;
7
8 //获取环形数组中节点x的邻居节点
9 vector<int> getNeighbors(int x, int N)
10 {
11     if (x == 0) return {N - 1, 1}; //如果是第一个节点，则返回它的左右邻居
12     else if (x == N - 1) return {N - 2, 0}; //如果是最后一个节点，则返回它的左右邻居
13     else return {x - 1, x + 1}; //返回一般节点的左右邻居
14 }
15
16 int main()
17 {
18     int N, E;
19     cin >> N >> E;
20
21     //哈希表，用于存储每个启动时刻启动的发动机列表
22     unordered_map<int, vector<int>> table;
23
24     //遍历每条启动信息
25     for (int i = 0; i < E; i++)
26     {
27         int T, P;
28         cin >> T >> P;
29         table[T].push_back(P); //将发动机编号添加到相应的启动时刻中
30     }
31
32     deque<int> q; //队列，用于BFS
33     vector<bool> checkList(N, false); //用于标记发动机是否已经启动
34     int total = 0; //已启动的发动机数量
35     int curTime = min_element(table.begin(), table.end(), [] (const auto& a, const auto& b) {
36         return a.first < b.first; //获取最早的启动时刻
37     })->first;
38
39     vector<int> ans; //储存结果的数组
40
41     //进行BFS搜索，知道所有发动机都启动
42     while (total < N)
43     {
```

```

44     if (table.find(curTime) != table.end())
45     {
46         //如果当前时刻有发动机要启动
47         for (int x : table[curTime])
48         {
49             if (!checkList[x])
50             {
51                 checkList[x] = true;
52                 q.push_back(x);
53             }
54         }
55     }
56
57     int qSize = q.size();
58     total += qSize;
59
60     //如果启动的发动机数量达到了总数，则退出循环
61     if (total == N)
62     {
63         ans.assign(q.begin(), q.end());
64         sort(ans.begin(), ans.end());
65         break;
66     }
67
68     //关联启动当前启动的发动机周边的发动机
69     for (int i = 0; i < qSize; ++i)
70     {
71         int x = q.front(); //获取队列中的第一个发动机编号
72         q.pop_front(); //从队列中删除该发动机编号
73         for (int nx : getNeighbors(x, N))
74         {
75             if (!checkList[nx])
76             {
77                 checkList[nx] = true; //标记为已启动
78                 q.push_back(nx); //加入到队列中
79             }
80         }
81     }
82     curTime++; //增加当前时刻
83 }
84
85 //输出结果
86 cout << ans.size() << endl;
87 for(int x : ans)
88 {
89     cout << x << " ";
90 }
91

```

```
92     return 0;  
93 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 获取环形数组中节点 x 的邻居节点
5     static List<Integer> getNeighbors(int x, int N) {
6         if (x == 0) return Arrays.asList(N - 1, 1);
7         else if (x == N - 1) return Arrays.asList(N - 2, 0);
8         return Arrays.asList(x - 1, x + 1);
9     }
10
11    public static void main(String[] args) {
12        Scanner scanner = new Scanner(System.in);
13        int N = scanner.nextInt();
14        int E = scanner.nextInt();
15        //哈希表，用于存储每个启动时刻启动的发动机列表
16        Map<Integer, List<Integer>> table = new HashMap<>();
17        int minKey = Integer.MAX_VALUE;
18        //遍历每条启动信息
19        for (int i = 0; i < E; i++) {
20            int T = scanner.nextInt();
21            int P = scanner.nextInt();
22            table.putIfAbsent(T, new ArrayList<>());
23            //将发动机编号添加到相应的启动时刻中
24            table.get(T).add(P);
25            minKey = Math.min(minKey, T);
26        }
27        scanner.close();
28
29        Queue<Integer> queue = new LinkedList<>();
30        //用于标记发动机是否已经启动
31        boolean[] checkList = new boolean[N];
32        // 已经启动发动机数量
33        int total = 0;
34        int curTime = minKey;
35        List<Integer> lastStarted = new ArrayList<>();
36        //BFS
37        while (total < N) {
38            //如果当前时刻有发动机要启动
39            if (table.containsKey(curTime)) {
40                for (int x : table.get(curTime)) {
41                    if (!checkList[x]) {
42                        checkList[x] = true;
43                        queue.add(x);
44                    }
45                }
46            }
47        }
48    }
49}
```

```
46     }
47
48     int qSize = queue.size();
49     total += qSize;
50     // 全部启动
51     if (total == N) {
52         lastStarted.addAll(queue);
53         break;
54     }
55     //关联启动当前启动的发动机周边的发动机
56     for (int i = 0; i < qSize; i++) {
57         int x = queue.poll();
58         for (int nx : getNeighbors(x, N)) {
59             if (!checkList[nx]) {
60                 checkList[nx] = true;
61                 queue.add(nx);
62             }
63         }
64     }
65     // 时间增加
66     curTime++;
67 }
68
69 Collections.sort(lastStarted);
70 System.out.println(lastStarted.size());
71 for (int x : lastStarted) {
72     System.out.print(x + " ");
73 }
74 }
75 }
```

Python

```
1 import sys
2 from collections import deque
3
4 def get_neighbors(x, N):
5     """获取环形数组中节点 x 的邻居"""
6     if x == 0:
7         return [N - 1, 1]
8     elif x == N - 1:
9         return [N - 2, 0]
10    return [x - 1, x + 1]
11
12 def main():
13     N, E = map(int, sys.stdin.readline().split())
14     # 哈希表，用于存储每个启动时刻启动的发动机列表
15     table = {}
16     min_key = float('inf')
17     # 遍历每条启动信息
18     for _ in range(E):
19         T, P = map(int, sys.stdin.readline().split())
20         if T not in table:
21             table[T] = []
22         # 将发动机编号添加到相应的启动时刻中
23         table[T].append(P)
24         min_key = min(min_key, T)
25
26     queue = deque()
27     # 用于标记发动机是否已经启动
28     check_list = [False] * N
29     # 已经启动的数量
30     total = 0
31     cur_time = min_key
32     last_started = []
33
34     # BFS
35     while total < N:
36         # 如果当前时刻有发动机要启动
37         if cur_time in table:
38             for x in table[cur_time]:
39                 if not check_list[x]:
40                     check_list[x] = True
41                     queue.append(x)
42
43         q_size = len(queue)
44         total += q_size
45         # 全部启动完毕
```

```
46     if total == N:
47         last_started.extend(queue)
48         break
49     # 关联启动当前启动的发动机周边的发动机
50     for _ in range(q_size):
51         x = queue.popleft()
52         for nx in get_neighbors(x, N):
53             if not check_list[nx]:
54                 check_list[nx] = True
55                 queue.append(nx)
56     # 时间增加
57     cur_time += 1
58
59     last_started.sort()
60     print(len(last_started))
61     print(" ".join(map(str, last_started)))
62
63 if __name__ == "__main__":
64     main()
```

JavaScript

```

1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  let input = [];
9
10 rl.on("line", (line) => {
11     input.push(line);
12 }).on("close", () => {
13     const [N, E] = input[0].split(" ").map(Number);
14     // 哈希表，用于存储每个启动时刻启动的发动机列表
15     let table = new Map();
16     let minKey = Infinity;
17     // 遍历每条启动信息
18     for (let i = 1; i <= E; i++) {
19         let [T, P] = input[i].split(" ").map(Number);
20         if (!table.has(T)) table.set(T, []);
21         // 将发动机编号添加到相应的启动时刻中
22         table.get(T).push(P);
23         minKey = Math.min(minKey, T);
24     }
25
26     let queue = [];
27     let checkList = new Array(N).fill(false);
28     // 已经启动的数量
29     let total = 0;
30     let curTime = minKey;
31     let lastStarted = [];
32     // 获取环形数组中节点 x 的邻居
33     function getNeighbors(x, N) {
34         return x === 0 ? [N - 1, 1] : x === N - 1 ? [N - 2, 0] : [x - 1,
35         x + 1];
36     }
37     // BFS 模拟
38     while (total < N) {
39         // 如果当前时刻有发动机要启动
40         if (table.has(curTime)) {
41             for (let x of table.get(curTime)) {
42                 if (!checkList[x]) {
43                     checkList[x] = true;
44                     queue.push(x);
45                 }
46             }
47         }
48     }
49
50     // 打印结果
51     console.log(lastStarted.join(" "));
52 }

```

```
45         }
46     }
47
48     let qSize = queue.length;
49     total += qSize;
50     // 全部启动完毕
51     if (total === N) {
52         lastStarted = [...queue];
53         break;
54     }
55     //关联启动当前启动的发动机周边的发动机
56     for (let i = 0; i < qSize; i++) {
57         let x = queue.shift();
58         for (let nx of getNeighbors(x, N)) {
59             if (!checkList[nx]) {
60                 checkList[nx] = true;
61                 queue.push(nx);
62             }
63         }
64     }
65     // 时间增加
66     curTime++;
67 }
68
69 lastStarted.sort((a, b) => a - b);
70 console.log(lastStarted.length);
71 console.log(lastStarted.join(" "));
72});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10    )
11
12 // 获取环形数组中节点 x 的邻居
13 func getNeighbors(x, N int) []int {
14     if x == 0 {
15         return []int{N - 1, 1}
16     } else if x == N-1 {
17         return []int{N - 2, 0}
18     }
19     return []int{x - 1, x + 1}
20 }
21
22 func main() {
23     scanner := bufio.NewScanner(os.Stdin)
24     scanner.Scan()
25     parts := strings.Split(scanner.Text(), " ")
26     N, _ := strconv.Atoi(parts[0])
27     E, _ := strconv.Atoi(parts[1])
28     //哈希表，用于存储每个启动时刻启动的发动机列表
29     table := make(map[int][]int)
30     minKey := int(^uint(0) >> 1)
31     //遍历每条启动信息
32     for i := 0; i < E; i++ {
33         scanner.Scan()
34         parts = strings.Split(scanner.Text(), " ")
35         T, _ := strconv.Atoi(parts[0])
36         P, _ := strconv.Atoi(parts[1])
37         //将发动机编号添加到相应的启动时刻中
38         table[T] = append(table[T], P)
39         if T < minKey {
40             minKey = T
41         }
42     }
43
44     queue := []int{}
45     checkList := make([]bool, N)
```

```
46     // 已经启动的数量
47     total := 0
48     // 当前时间
49     curTime := minKey
50     lastStarted := []int{}
51
52     // 使用BFS进行模拟
53     for total < N {
54         //如果当前时刻有发动机要启动
55         if engines, exists := table[curTime]; exists {
56             for _, x := range engines {
57                 if !checkList[x] {
58                     checkList[x] = true
59                     queue = append(queue, x)
60                 }
61             }
62         }
63
64         qSize := len(queue)
65         total += qSize
66         // 全部启动完毕
67         if total == N {
68             lastStarted = append(lastStarted, queue...)
69             break
70         }
71         newQueue := []int{}
72         //关联启动当前启动的发动机周边的发动机
73         for _, x := range queue {
74             for _, nx := range getNeighbors(x, N) {
75                 if !checkList[nx] {
76                     checkList[nx] = true
77                     newQueue = append(newQueue, nx)
78                 }
79             }
80         }
81         queue = newQueue
82         curTime++
83     }
84
85     sort.Ints(lastStarted)
86     fmt.Println(len(lastStarted))
87     fmt.Println(strings.Trim(fmt.Sprint(lastStarted), "[]"))
88 }
```


华为OD机试2025C卷 - 发广播 (C++ & Python & JAVA & JS & GO)-CSDN博客

发广播

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

服务器连接方式包括直接相连，间接连接。

A和B直接连接，B和C直接连接，则A和C间接连接。

直接连接和间接连接都可以发送广播。

给出一个N*N数组，代表N个服务器，

`matrix[i][j] == 1`，

则代表i和j直接连接；不等于 1 时，代表i和j不直接连接。

`matrix[i][i] == 1`，

即自己和自己直接连接。`matrix[i][j] == matrix[j][i]`。

计算初始需要给几台服务器广播，才可以使每个服务器都收到广播。

输入描述

输入为N行，每行有N个数字，为0或1，由空格分隔，

构成N*N的数组，N的范围为 $1 \leq N \leq 40$

输出描述

输出一个数字，为需要广播的服务器的数量

示例1

输入

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

Plain Text

输出

Plain Text

1 3

说明

3 台服务器互不连接，所以需要分别广播这 3 台服务器

示例2

输入

Plain Text

1 1 1
2 1 1

输出

Plain Text

1 1

说明

2 台服务器相互连接，所以只需要广播其中一台服务器

题解

思路： 并查集

1. 直接相连和间接相连的服务器可以分到一个组，一组服务器内部只需要在一个服务器发送广播，这个组中的所有服务器都能收到广播。所以结果就是 服务器可以分为多少个组。
2. 对于这种经典问题可以使用 并查集 算法来实现。基本逻辑如下：
 - a. 对于每个组选取一个服务器id作为这个组的id，初始任务每个服务器单独属于一个组，这个组的代表为它本身。
 - b. 遍历输入矩阵，当遇到 `matrix[i][j] = 1` 时将这两个组进行合并。`下面代码逻辑` 是合并组之后，选取 `i j` 组的之前代表节点小的作为新组的代表。
 - c. 按照2的逻辑处理之后，有多少个代表就有多少个组。统计有多少个组长就是需要广播的服务器的数量。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include <sstream>
4 using namespace std;
5
6 // 找到组中代表id
7 int find(int a, vector<int> &ans) {
8     if (ans[a] != a) {
9         ans[a] = find(ans[a], ans);
10    }
11    return ans[a];
12 }
13
14 // 根据delimiter分割字符串
15 vector<string> split(const string& str, char delimiter) {
16     vector<string> result;
17     stringstream ss(str);
18     string item;
19
20     // 使用流读取，每次读取到分隔符
21     while (getline(ss, item, delimiter)) {
22         result.push_back(item);
23     }
24     return result;
25 }
26
27 // 合并组
28 void merge(int a, int b, vector<int> &ans) {
29     int rootA = find(a, ans);
30     int rootB = find(b, ans);
31     // 选取两个组代表小的作为新组的代表
32     int newRoot = min(rootA, rootB);
33     ans[rootA] = newRoot;
34     ans[rootB] = newRoot;
35 }
36 int main() {
37     vector<vector<string>> ans;
38     string s;
39
40     while (getline(cin, s)) {
41         if (s == "") {
42             break;
43         }
44         vector<string> tmp = split(s, ' ');
45         ans.push_back(tmp);
```

```
46     }
47     int n = ans.size();
48
49     vector<int> root(n+1);
50     // 初始化
51     for (int i = 1; i <=n ; i++) {
52         root[i] = i;
53     }
54     // 尝试合并组
55     for (int i = 0; i < n; i++) {
56         for (int j = 0; j < n; j++) {
57             if (ans[i][j] == "1" && i != j) {
58                 merge(i+1, j+1, root);
59             }
60         }
61     }
62 }
63 // 统计组的数量
64 int res = 0;
65 for (int i = 1; i <=n ; i++) {
66     if (find(i, root) == i) {
67         res++;
68     }
69 }
70 cout << res;
71 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 并查集模板
5     static int find(int x, int[] parent) {
6         if (parent[x] != x) {
7             parent[x] = find(parent[x], parent);
8         }
9         return parent[x];
10    }
11    // 并查集模板 合并组
12    static void merge(int a, int b, int[] parent) {
13        int rootA = find(a, parent);
14        int rootB = find(b, parent);
15        // 选取两个组代表小的作为新组的代表
16        int newRoot = Math.min(rootA, rootB);
17        parent[rootA] = newRoot;
18        parent[rootB] = newRoot;
19    }
20
21    public static void main(String[] args) {
22        Scanner scanner = new Scanner(System.in);
23        List<String> lines = new ArrayList<>();
24
25        while (scanner.hasNextLine()) {
26            String line = scanner.nextLine().trim();
27            if (line.isEmpty()) break;
28            lines.add(line);
29        }
30        scanner.close();
31
32        int n = lines.size();
33        int[] parent = new int[n + 1];
34        // 初始化
35        for (int i = 1; i <= n; i++) parent[i] = i;
36
37        String[][] matrix = new String[n][n];
38        for (int i = 0; i < n; i++) {
39            matrix[i] = lines.get(i).split(" ");
40        }
41        // 尝试合并
42        for (int i = 0; i < n; i++) {
43            for (int j = 0; j < n; j++) {
44                if (matrix[i][j].equals("1") && i != j) {
45                    merge(i + 1, j + 1, parent);
46                }
47            }
48        }
49    }
50}
```

```
46         }
47     }
48 }
49 // 统计组的数量 队长数量 == 组的数量
50 int count = 0;
51 for (int i = 1; i <= n; i++) {
52     if (find(i, parent) == i) count++;
53 }
54
55 System.out.println(count);
56 }
57 }
```

Python

```
1 import sys
2
3 # 并查集查找
4 def find(x, parent):
5     if parent[x] != x:
6         parent[x] = find(parent[x], parent)
7     return parent[x]
8
9 # 并查集合并
10 def merge(a, b, parent):
11     rootA = find(a, parent)
12     rootB = find(b, parent)
13     # 选取两个代表小的作为新组的代表
14     newRoot = min(rootA, rootB)
15     parent[rootA] = newRoot
16     parent[rootB] = newRoot
17
18 def main():
19     lines = sys.stdin.read().strip().split("\n")
20     n = len(lines)
21
22     # 初始化并查集
23     parent = list(range(n + 1))
24
25     # 读取矩阵并执行合并
26     matrix = [line.split() for line in lines]
27
28     for i in range(n):
29         for j in range(n):
30             if matrix[i][j] == "1" and i != j:
31                 merge(i + 1, j + 1, parent)
32
33     # 计算组的数量
34     result = sum(1 for i in range(1, n + 1) if find(i, parent) == i)
35     print(result)
36
37 if __name__ == "__main__":
38     main()
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let lines = [];
9
10 rl.on("line", (line) => {
11     if (line.trim() !== "") {
12         lines.push(line.trim());
13     }
14 }).on("close", () => {
15     const n = lines.length;
16     let parent = Array.from({ length: n + 1 }, (_, i) => i);
17     // 找到组中的代表
18     function find(x) {
19         if (parent[x] !== x) {
20             parent[x] = find(parent[x]);
21         }
22         return parent[x];
23     }
24     // 并查集模板 合并组
25     function merge(a, b) {
26         let rootA = find(a);
27         let rootB = find(b);
28         // 选取两个代表小的作为新组的代表
29         let newRoot = Math.min(rootA, rootB);
30         parent[rootA] = newRoot;
31         parent[rootB] = newRoot;
32     }
33
34     const matrix = lines.map(line => line.split(" "));
35     // 尝试合并组
36     for (let i = 0; i < n; i++) {
37         for (let j = 0; j < n; j++) {
38             if (matrix[i][j] === "1" && i !== j) {
39                 merge(i + 1, j + 1);
40             }
41         }
42     }
43     // 统计组的数量
44     let count = 0;
45     for (let i = 1; i <= n; i++) {
```

```
46         if (find(i) === i) count++;
47     }
48     console.log(count);
50 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // 并查集查找
11 func find(x int, parent []int) int {
12     if parent[x] != x {
13         parent[x] = find(parent[x], parent)
14     }
15     return parent[x]
16 }
17
18 // 并查集合并
19 func merge(a, b int, parent []int) {
20     rootA := find(a, parent)
21     rootB := find(b, parent)
22     // 选取两个组代表小的作为新组的代表
23     newRoot := min(rootA, rootB)
24     parent[rootA] = newRoot
25     parent[rootB] = newRoot
26 }
27
28 func min(a, b int) int {
29     if a < b {
30         return a
31     }
32     return b
33 }
34
35 func main() {
36     scanner := bufio.NewScanner(os.Stdin)
37     var lines []string
38
39     for scanner.Scan() {
40         line := strings.TrimSpace(scanner.Text())
41         if line == "" {
42             break
43         }
44         lines = append(lines, line)
45     }
}
```

```
46
47     n := len(lines)
48     parent := make([]int, n+1)
49         // 尝试
50     for i := 1; i <= n; i++ {
51         parent[i] = i
52     }
53
54     matrix := make([][]string, n)
55     for i := 0; i < n; i++ {
56         matrix[i] = strings.Fields(lines[i])
57     }
58         //尝试合并组
59     for i := 0; i < n; i++ {
60         for j := 0; j < n; j++ {
61             if matrix[i][j] == "1" && i != j {
62                 merge(i+1, j+1, parent)
63             }
64         }
65     }
66         // 统计组的数量
67     count := 0
68     for i := 1; i <= n; i++ {
69         if find(i, parent) == i {
70             count++
71         }
72     }
73
74     fmt.Println(count)
75 }
```

| 来自: 华为OD机试2025C卷 – 发广播 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 区间连接器 (C++ & Python & JAVA & JS & GO)-CSDN博客

区间连接器

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 200分题型

题目描述

有一组区间 $[a_0, b_0]$, $[a_1, b_1]$, ... (a, b表示起点, 终点), 区间有可能重叠、相邻, 重叠或相邻则可以合并为更大的区间;

给定一组连接器 $[x_1, x_2, x_3, \dots]$ (x 表示连接器的最大可连接长度, 即 $x \geq \text{gap}$) , 可用于将分离的区间连接起来, 但两个分离区间之间只能使用1个连接器;

请编程实现使用连接器后, 最少的区间数结果。

备注:

- 区间数量 < 10000 , a,b均 ≤ 10000
- 连接器数量 < 10000 ; $x \leq 10000$

输入描述

无

输出描述

无

用例1

输入

```
1 [1,10],[15,20],[18,30],[33,40]
2 [5,4,3,2]
```

输出

Plain Text

```
Plain Text |  
1 1
```

说明

合并后: [1,10], [15,30], [33,40], 使用5, 3两个连接器连接后只剩下[1,40]。

用例2

输入

```
Plain Text |  
1 [1,2],[3,5],[7,10],[15,20],[30,100]  
2 [5,4,3,2,1]
```

输出

```
Plain Text |  
1 2
```

说明

无重叠和相邻, 使用1, 2, 5三个连接器连接后只剩下[1,20], [30,100]

题解

思路: 合并区间 + 贪心分配

1. 合并输入的区间, 得到一组不重叠的区间。得到按照起始值升序排序不重叠的区间 `merge` 数组。
2. 计算相邻区间中间存在空闲距离, 存入 `dist` 数组。将 `dist` 数组进行排序。
3. 贪心将连接器距离大 贪心 分配给 `dist` 中大的间隔, 使用 `count` 统计可以分配的数量。
4. 结果就为 `merge.size() - count`

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 #include<set>
11 using namespace std;
12
13 // 通用 split 函数
14 vector<string> split(const string& str, const string& delimiter) {
15     vector<string> result;
16     size_t start = 0;
17     size_t end = str.find(delimiter);
18     while (end != string::npos) {
19         result.push_back(str.substr(start, end - start));
20         start = end + delimiter.length();
21         end = str.find(delimiter, start);
22     }
23     // 添加最后一个部分
24     result.push_back(str.substr(start));
25     return result;
26 }
27
28
29 // 区域合并
30 vector<pair<int, int>> merged(vector<pair<int, int>> segs) {
31     // 按起点排序, 方便合并
32     sort(segs.begin(), segs.end());
33     // 合并区间
34     vector<pair<int, int>> merged;
35     pair<int, int> cur = segs[0];
36     int n = segs.size();
37     for (int i = 1; i < n; i++) {
38         if (segs[i].first <= cur.second) {
39             cur.second = max(cur.second, segs[i].second);
40         } else {
41             merged.push_back(cur);
42             cur = segs[i];
43         }
44     }
45     merged.push_back(cur);
}
```

```

46     return merged;
47 }
48
49 int main() {
50     string input1, input2;
51     getline(cin, input1);
52     getline(cin, input2);
53     // 存储区间
54     vector<pair<int, int>> ans;
55
56     // 处理区间输入字符串 [18,30], [33,40] => 18,30], [33,40
57     input1 = input1.substr(1, input1.size() - 2);
58     vector<string> tmpArr = split(input1, "[", "]");
59     for (int i = 0; i < tmpArr.size(); i++) {
60         string tmp = tmpArr[i];
61         vector<string> startAndEnd = split(tmp, ",");
62         int start = stoi(startAndEnd[0]);
63         int end = stoi(startAndEnd[1]);
64         ans.push_back({start, end});
65     }
66
67     // 处理连接器输入字符串
68     input2 = input2.substr(1, input2.size() - 2);
69     vector<string> joinStrArr = split(input2, ",");
70     vector<int> joinArr;
71     for (int i = 0; i < joinStrArr.size(); i++) {
72         joinArr.push_back(stoi(joinStrArr[i]));
73     }
74
75     // 输入数据区间进行区间合并
76     vector<pair<int, int>> merge = merged(ans);
77
78     // 计算每个相邻区间之间的距离
79     vector<int> dist;
80     for (int i = 1; i < merge.size(); i++) {
81         dist.push_back(merge[i].first - merge[i-1].second);
82     }
83     sort(dist.begin(), dist.end());
84     sort(joinArr.begin(), joinArr.end());
85
86
87     int i = dist.size() - 1, j = joinArr.size() - 1;
88     // 合并次数
89     int count = 0;
90     // 贪心优先把连接器数量大的分配给距离大
91     while (i >= 0 && j >= 0) {
92         if (dist[i] <= joinArr[j]) {
93             count++;

```

```
94         i--;
95         j--;
96     } else {
97         i--;
98     }
99 }
100 // 区间数 - 合并
101 cout << dist.size() + 1 - count;
102 return 0;
103 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 区间合并函数
5     public static List<int[]> merged(List<int[]> segs) {
6         // 按起点排序
7         segs.sort(Comparator.comparingInt(a -> a[0]));
8         List<int[]> merged = new ArrayList<>();
9         int[] cur = segs.get(0);
10        int n = segs.size();
11        for (int i = 1; i < n; i++) {
12            if (segs.get(i)[0] <= cur[1]) {
13                cur[1] = Math.max(cur[1], segs.get(i)[1]);
14            } else {
15                merged.add(cur);
16                cur = segs.get(i);
17            }
18        }
19        merged.add(cur);
20        return merged;
21    }
22
23    public static void main(String[] args) {
24        Scanner sc = new Scanner(System.in);
25        String input1 = sc.nextLine();
26        String input2 = sc.nextLine();
27
28        // 去除开头和结尾的中括号
29        input1 = input1.substring(1, input1.length() - 1);
30        String[] tmpArr = input1.split("\\],\\[" );
31
32        // 解析区间
33        List<int[]> ans = new ArrayList<>();
34        for (String s : tmpArr) {
35            String[] parts = s.split(",");
36            int start = Integer.parseInt(parts[0]);
37            int end = Integer.parseInt(parts[1]);
38            ans.add(new int[]{start, end});
39        }
40
41        // 解析连接器
42        input2 = input2.substring(1, input2.length() - 1);
43        String[] joinStrArr = input2.split(",");
44        List<Integer> joinArr = new ArrayList<>();
45        for (String s : joinStrArr) {
```

```
46         joinArr.add(Integer.parseInt(s));
47     }
48
49     // 合并区间
50     List<int[]> merge = merged(ans);
51
52     // 计算每对相邻区间之间的间距
53     List<Integer> dist = new ArrayList<>();
54     for (int i = 1; i < merge.size(); i++) {
55         dist.add(merge.get(i)[0] - merge.get(i - 1)[1]);
56     }
57
58     // 排序准备贪心合并
59     Collections.sort(dist);
60     Collections.sort(joinArr);
61
62     int i = dist.size() - 1, j = joinArr.size() - 1;
63     int count = 0;
64     while (i >= 0 && j >= 0) {
65         if (dist.get(i) <= joinArr.get(j)) {
66             count++;
67             i--;
68             j--;
69         } else {
70             i--;
71         }
72     }
73
74     // 原始段数 - 合并次数
75     System.out.println(dist.size() + 1 - count);
76 }
77 }
```

Python

```
1 # 合并区间函数
2 def merged(segs):
3     segs.sort()
4     merged_list = []
5     cur = segs[0]
6     for i in range(1, len(segs)):
7         if segs[i][0] <= cur[1]:
8             cur[1] = max(cur[1], segs[i][1])
9         else:
10             merged_list.append(cur)
11             cur = segs[i]
12     merged_list.append(cur)
13     return merged_list
14
15 # 主程序逻辑
16 input1 = input().strip()
17 input2 = input().strip()
18
19 # 去除外层中括号
20 input1 = input1[1:-1]
21 segments = input1.split("], [")
22 ans = []
23 for s in segments:
24     start, end = map(int, s.split(','))
25     ans.append([start, end])
26
27 # 解析连接器
28 input2 = input2[1:-1]
29 joinArr = list(map(int, input2.split(',')))
30
31 # 合并区间
32 merge = merged(ans)
33
34 # 计算每对相邻区间之间的距离
35 dist = []
36 for i in range(1, len(merge)):
37     dist.append(merge[i][0] - merge[i-1][1])
38
39 dist.sort()
40 joinArr.sort()
41
42 # 贪心分配连接器
43 i, j = len(dist) - 1, len(joinArr) - 1
44 count = 0
45 while i >= 0 and j >= 0:
```

```
46     if dist[i] <= joinArr[j]:  
47         count += 1  
48         i -= 1  
49         j -= 1  
50     else:  
51         i -= 1  
52  
53 # 原始区间数 - 合并次数  
54 print(len(dist) + 1 - count)
```

JavaScript

```
1 const readline = require("readline");
2 const rl = readline.createInterface({
3     input: process.stdin,
4     output: process.stdout
5 });
6
7 let inputs = [];
8 rl.on("line", line => {
9     inputs.push(line.trim());
10    if (inputs.length === 2) rl.close();
11 });
12
13 rl.on("close", () => {
14     let input1 = inputs[0];
15     let input2 = inputs[1];
16
17     // 去掉最外层中括号
18     input1 = input1.slice(1, -1);
19     const segStrs = input1.split("[", "]");
20     let segments = segStrs.map(s => s.split(",")).map(Number));
21
22     // 合并区间
23     segments.sort((a, b) => a[0] - b[0]);
24     let merged = [];
25     let cur = segments[0];
26     for (let i = 1; i < segments.length; i++) {
27         if (segments[i][0] <= cur[1]) {
28             cur[1] = Math.max(cur[1], segments[i][1]);
29         } else {
30             merged.push(cur);
31             cur = segments[i];
32         }
33     }
34     merged.push(cur);
35
36     // 解析连接器
37     input2 = input2.slice(1, -1);
38     let joinArr = input2.split(",").map(Number);
39
40     // 计算相邻区间之间的距离
41     let dist = [];
42     for (let i = 1; i < merged.length; i++) {
43         dist.push(merged[i][0] - merged[i - 1][1]);
44     }
45
```

```
46     dist.sort((a, b) => a - b);
47     joinArr.sort((a, b) => a - b);
48
49     let i = dist.length - 1, j = joinArr.length - 1, count = 0;
50     while (i >= 0 && j >= 0) {
51         if (dist[i] <= joinArr[j]) {
52             count++;
53             i--;
54             j--;
55         } else {
56             i--;
57         }
58     }
59
60     console.log(dist.length + 1 - count);
61 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10    )
11
12 // 合并区间函数
13 func merged(segs [][][2]int) [][][2]int {
14     sort.Slice(segs, func(i, j int) bool {
15         return segs[i][0] < segs[j][0]
16     })
17     var result [][][2]int
18     cur := segs[0]
19     for i := 1; i < len(segs); i++ {
20         if segs[i][0] <= cur[1] {
21             if segs[i][1] > cur[1] {
22                 cur[1] = segs[i][1]
23             }
24         } else {
25             result = append(result, cur)
26             cur = segs[i]
27         }
28     }
29     result = append(result, cur)
30     return result
31 }
32
33 func main() {
34     reader := bufio.NewReader(os.Stdin)
35     input1, _ := reader.ReadString('\n')
36     input2, _ := reader.ReadString('\n')
37     input1 = strings.TrimSpace(input1)
38     input2 = strings.TrimSpace(input2)
39
40     // 去掉最外层中括号
41     input1 = input1[1 : len(input1)-1]
42     parts := strings.Split(input1, "], [")
43     var segments [][][2]int
44     for _, s := range parts {
45         nums := strings.Split(s, ",")
```

```

46     start, _ := strconv.Atoi(nums[0])
47     end, _ := strconv.Atoi(nums[1])
48     segments = append(segments, [2]int{start, end})
49 }
50
51 // 解析连接器
52 input2 = input2[1 : len(input2)-1]
53 joinStrs := strings.Split(input2, ",")
54 var joinArr []int
55 for _, s := range joinStrs {
56     num, _ := strconv.Atoi(s)
57     joinArr = append(joinArr, num)
58 }
59
60 // 合并区间
61 mergedSegs := merged(segments)
62
63 // 计算相邻区间之间的距离
64 var dist []int
65 for i := 1; i < len(mergedSegs); i++ {
66     d := mergedSegs[i][0] - mergedSegs[i-1][1]
67     dist = append(dist, d)
68 }
69
70 sort.Ints(dist)
71 sort.Ints(joinArr)
72
73 i, j := len(dist)-1, len(joinArr)-1
74 count := 0
75 for i >= 0 && j >= 0 {
76     if dist[i] <= joinArr[j] {
77         count++
78         i--
79         j--
80     } else {
81         i--
82     }
83 }
84
85 // 输出最终可合并后的区间数
86 fmt.Println(len(dist) + 1 - count)
87 }

```

来自: 华为OD机试2025C卷 – 区间连接器 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 水果摊小买卖 (C++ & Python & JAVA & JS & GO)-CSDN博客

水果摊小买卖

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 200分题型

题目描述

小王手里有点闲钱，想着做点卖水果的小买卖。给出两个数组m, n，用 $m[i]$ 代表第 i 个水果的成本价， $n[i]$ 代表第 i 个水果能卖出的价钱，假如现在有本钱 k ，试问最后最多能赚多少钱？

说明：

1. 每种水果只需买一次，只能卖一次。
2. 数组 m 、 n 大小不超过 50。
3. 数组元素为正整数，不超过 1000。

输入描述

数组 m 、 n

本钱 k

备注：

第一行输入逗号分隔的数组 m 的元素值

第二行输入逗号分隔的数组 n 的元素值

第三行输入本钱

输出描述

最多能赚取多少钱。

用例1

输入

```
1 4,2,6,4
2 5,3,8,7
3 15
```

输出

▼ Plain Text |

```
1 22
```

用例2

输入

▼ Plain Text |

```
1 3,4,5
2 3,3,5
3 10
```

输出

▼ Plain Text |

```
1 10
```

说明

| 三个水果中利润均不大于0（利润分别为0, -1, 0），因此不进行交易，最终资金仍为10。

用例3

输入

▼ Plain Text |

```
1 1,2,3,4
2 2,4,6,8
3 1
```

输出

▼ Plain Text |

```
1 11
```

说明

| 初始资金1:

第1个水果：成本1，利润1，购买后资金变为2；

第2个水果：成本2，利润2，购买后资金变为4；

第3个水果：成本3，利润3，购买后资金变为7；

第4个水果：成本4，利润4，购买后资金变为11。

题解

思路：**贪心** 算法实现，优先购买盈利更多的水果。

1. 通过用例3可以看出，随着不断买入卖出水果本钱增加过程中可以购买水果是动态变化的。
2. 具体处理逻辑如下
 - a. 根据输入数据，仅考虑有利润的水果。
 - b. 将水果按照成本的大小升序排序，方便处理后序盈利过程中**可以购买水果的动态变化**的情况。
 - c. 按照当前的本钱大小，将获得一堆成本价小于等于本钱的水果作为本次的考虑对象。
 - d. 贪心选择可以购买的最大盈利的水果。下面代码使用了优先队列进行快速选取（预先将3中筛选出的放入优先队列）。选取出来最大利润的水果之后，更新本钱大小。
 - e. 重复3 4，直到没有可购买的水果为止。
 - f. 最后的钱就是结果。

本题描述其实不太清晰，按照用例的意思，最终结果其实是能获得最多的钱。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 using namespace std;
11
12
13 struct Node{
14     int profit;
15     int cost;
16     int sell;
17     Node(int cost, int sell, int profit):cost(cost),sell(sell),profit(profit) {}
18 };
19
20 // 通用 split 函数
21 vector<int> split(const string& str, const string& delimiter) {
22     vector<int> result;
23     size_t start = 0;
24     size_t end = str.find(delimiter);
25     while (end != string::npos) {
26         result.push_back(stoi(str.substr(start, end - start)));
27         start = end + delimiter.length();
28         end = str.find(delimiter, start);
29     }
30     // 添加最后一个部分
31     result.push_back(stoi(str.substr(start)));
32     return result;
33 }
34
35 bool cmp(Node a, Node b) {
36     return a.cost < b.cost;
37 }
38
39 // 自定义比较函数 (小顶堆)
40 struct compare {
41     bool operator()(Node a, Node b) {
42         return a.profit < b.profit;
43     }
44 };
```

```

45
46
47 int main() {
48     string mstr,nstr;
49     int money;
50     getline(cin, mstr);
51     getline(cin, nstr);
52     cin >> money;
53     vector<int> mAns = split(mstr, ",");
54     vector<int> nAns = split(nstr, ",");
55
56     int n = mAns.size();
57     vector<Node> ans;
58     for (int i = 0; i < n; i++) {
59         int cost = mAns[i];
60         int sell = nAns[i];
61         int profit = sell -cost;
62         // 只考虑盈利的水果
63         if (profit > 0) {
64             ans.push_back({cost, sell, profit});
65         }
66     }
67     // 按照成本从小到大排序
68     sort(ans.begin(), ans.end(), cmp);
69     // 设置为可供选择苹果的数量
70     n = ans.size();
71     priority_queue<Node, vector<Node>, compare> pq;
72     int index = 0;
73     int current = money;
74     while (true) {
75         // 当前money可以进的水果
76         while (index < n && ans[index].cost <= current) {
77             pq.push(ans[index]);
78             index++;
79         }
80         // 没有可购买的水果了
81         if (pq.empty()) {
82             break;
83         }
84         // 当前可以购买的最大盈利的水果
85         Node tmp = pq.top();
86         pq.pop();
87         // 盈利的钱
88         current += tmp.profit;
89     }
90
91     cout << current;
92     return 0;

```

93 }

JAVA

```
1 import java.util.*;
2
3 class Node {
4     int cost, sell, profit;
5
6     Node(int cost, int sell, int profit) {
7         this.cost = cost;
8         this.sell = sell;
9         this.profit = profit;
10    }
11 }
12
13 public class Main {
14     public static void main(String[] args) {
15         Scanner scanner = new Scanner(System.in);
16         // 读取输入
17         String[] mStr = scanner.nextLine().split(",");
18         String[] nStr = scanner.nextLine().split(",");
19         int money = scanner.nextInt();
20
21         int n = mStr.length;
22         List<Node> fruits = new ArrayList<>();
23
24         for (int i = 0; i < n; i++) {
25             int cost = Integer.parseInt(mStr[i]);
26             int sell = Integer.parseInt(nStr[i]);
27             int profit = sell - cost;
28             if (profit > 0) { // 只考虑盈利的水果
29                 fruits.add(new Node(cost, sell, profit));
30             }
31         }
32
33         // 按成本排序
34         fruits.sort(Comparator.comparingInt(a -> a.cost));
35
36         // 使用优先队列（大顶堆），按照利润排序
37         PriorityQueue<Node> pq = new PriorityQueue<>((a, b) -> b.profit -
38             a.profit);
39
40         int index = 0;
41         int current = money;
42
43         while (true) {
44             // 选择当前资金可以买入的水果
```

```
45     ent) {
46         while (index < fruits.size() && fruits.get(index).cost <= curr
47             pq.offer(fruits.get(index));
48             index++;
49         }
50         if (pq.isEmpty()) {
51             break;
52         }
53         // 购买利润最高的水果
54         Node bestChoice = pq.poll();
55         current += bestChoice.profit;
56     }
57     System.out.println(current);
58 }
```

Python

```
1 import sys
2 import heapq
3
4 # 读取输入
5 m_values = list(map(int, sys.stdin.readline().strip().split(",")))
6 n_values = list(map(int, sys.stdin.readline().strip().split(",")))
7 money = int(sys.stdin.readline().strip())
8
9 # 构造有效的水果集合
10 fruits = []
11 for cost, sell in zip(m_values, n_values):
12     profit = sell - cost
13     if profit > 0: # 只考虑盈利的水果
14         fruits.append((cost, profit)) # 只存储必要数据 (成本, 利润)
15
16 # 按成本升序排序
17 fruits.sort()
18
19 # 使用大顶堆存储利润最高的水果
20 pq = []
21 index = 0
22 current = money
23
24 while True:
25     # 将当前资金可以买入的水果加入堆中 (使用负值模拟最大堆)
26     while index < len(fruits) and fruits[index][0] <= current:
27         heapq.heappush(pq, (-fruits[index][1], fruits[index][0])) # (负利润, 成本)
28         index += 1
29
30     if not pq:
31         break
32
33     # 取出利润最高的水果
34     max_profit, _ = heapq.heappop(pq)
35     current += -max_profit # 恢复利润的正值
36
37 print(current)
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9
10 rl.on("line", (line) => {
11     inputLines.push(line);
12     if (inputLines.length === 3) {
13         rl.close();
14     }
15 });
16
17 rl.on("close", () => {
18     let mValues = inputLines[0].split(",").map(Number);
19     let nValues = inputLines[1].split(",").map(Number);
20     let money = parseInt(inputLines[2]);
21
22     let fruits = [];
23     for (let i = 0; i < mValues.length; i++) {
24         let cost = mValues[i];
25         let sell = nValues[i];
26         let profit = sell - cost;
27         if (profit > 0) {
28             fruits.push({ cost, sell, profit });
29         }
30     }
31
32     // 按成本排序
33     fruits.sort((a, b) => a.cost - b.cost);
34
35     // 使用最大堆
36     let pq = [];
37     let index = 0;
38     let current = money;
39
40     while (true) {
41         // 将当前资金可以买入的水果加入堆中
42         while (index < fruits.length && fruits[index].cost <= current) {
43             pq.push(fruits[index]);
44             pq.sort((a, b) => b.profit - a.profit); // 模拟大顶堆
45             index++;
46     }
47 }
```

```
46     }
47
48     if (pq.length === 0) {
49         break;
50     }
51
52     // 购买利润最高的水果
53     let bestChoice = pq.shift();
54     current += bestChoice.profit;
55 }
56
57     console.log(current);
58 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9     "container/heap"
10    "sort"
11 )
12
13 // Node 结构体
14 type Node struct {
15     cost, sell, profit int
16 }
17
18 // 定义优先队列
19 type PriorityQueue []Node
20
21 func (pq PriorityQueue) Len() int          { return len(pq) }
22 func (pq PriorityQueue) Less(i, j int) bool { return pq[i].profit > pq[j].profit } // 大顶堆
23 func (pq PriorityQueue) Swap(i, j int)      { pq[i], pq[j] = pq[j], pq[i] }
24 func (pq *PriorityQueue) Push(x interface{}) { *pq = append(*pq, x.(Node)) }
25 func (pq *PriorityQueue) Pop() interface{} {
26     old := *pq
27     n := len(old)
28     x := old[n-1]
29     *pq = old[0 : n-1]
30     return x
31 }
32
33 // 字符串切割转换为整数数组
34 func splitToInts(s string, delimiter string) []int {
35     parts := strings.Split(s, delimiter)
36     result := make([]int, len(parts))
37     for i, p := range parts {
38         result[i], _ = strconv.Atoi(p)
39     }
40     return result
41 }
42 }
```

```
43 func main() {
44     scanner := bufio.NewScanner(os.Stdin)
45
46     // 读取输入
47     scanner.Scan()
48     mValues := splitToInts(scanner.Text(), ",")
49     scanner.Scan()
50     nValues := splitToInts(scanner.Text(), ",")
51     scanner.Scan()
52     money, _ := strconv.Atoi(scanner.Text())
53
54     // 构造有效的水果集合
55     var fruits []Node
56     for i := 0; i < len(mValues); i++ {
57         cost := mValues[i]
58         sell := nValues[i]
59         profit := sell - cost
60         if profit > 0 {
61             fruits = append(fruits, Node{cost, sell, profit})
62         }
63     }
64
65     // 按照成本排序
66     sort.Slice(fruits, func(i, j int) bool {
67         return fruits[i].cost < fruits[j].cost
68     })
69
70     // 使用优先队列（最大堆）
71     pq := &PriorityQueue{}
72     heap.Init(pq)
73
74     index := 0
75     current := money
76
77     for {
78         // 添加可以购买的水果
79         for index < len(fruits) && fruits[index].cost <= current {
80             heap.Push(pq, fruits[index])
81             index++
82         }
83
84         if pq.Len() == 0 {
85             break
86         }
87
88         // 购买利润最高的水果
89         bestChoice := heap.Pop(pq).(Node)
90         current += bestChoice.profit
```

```
91     }
92     fmt.Println(current)
93 }
94 }
```

| 来自: 华为OD机试2025C卷 – 水果摊小买卖 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 贪吃蛇 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 贪吃蛇

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 200分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

贪吃蛇是一个经典游戏，蛇的身体由若干方格连接而成，身体随蛇头移动。蛇头触碰到食物时，蛇的长度会增加一格。蛇头和身体的任一方格或者游戏版图边界碰撞时，游戏结束。

下面让我们来完成贪吃蛇游戏的模拟。

给定一个NM的数组arr，代表NM个方格组成的版图，贪吃蛇每次移动一个方格。

若 $\text{arr}[i][j] == \text{'H}'$ ，表示该方格为贪吃蛇的起始位置；

若 $\text{arr}[i][j] == \text{'F}'$ ，表示该方格为食物，

若 $\text{arr}[i][j] == \text{'E}'$ ，表示该方格为空格。

贪吃蛇初始长度为1，初始移动方向为向左。

为给定一系列贪吃蛇的移动操作，返回操作后蛇的长度，如果在操作执行完之前已经游戏结束，返回游戏结束时蛇的长度。

贪吃蛇移动、吃食物和碰撞处理的细节见下面图示：

		F	H	1	
				2	
				3	

图1

		H	1	2	
				3	
				4	

图2

		H	1	2	3
					4

图3

	H	1	2	3	4
					X

图4

	5	6	7	8	
	4		H		
	3	2	1		

图5

	6	7	H8		
	5			1	X
	4	3	2		

图6

图1：截取了贪吃蛇移动的一个中间状态，H表示蛇头，F表示食物，数字为蛇身体各节的编号，蛇为向左移动，此时蛇头和食物已经相邻

图2：蛇头向左移动一格，蛇头和食物重叠，注意此时食物的格子成为了新的蛇头，第1节身体移动到蛇头位置，第2节身体移动到第1节身体位置，以此类推，最后添加第4节身体到原来第3节身体的位置。

图3：蛇头继续向左移动一格，身体的各节按上述规则移动，此时蛇头已经和边界相邻，但还未碰撞。

图4：蛇头继续向左移动一格，此时蛇头已经超过边界，发生碰撞，游戏结束。

图5和图6给出一个蛇头和身体碰撞的例子，蛇为向上移动。

图5时蛇头和第7节身体相邻，但还未碰撞；

图6蛇头向上移动一格，此时蛇头和第8节身体都移动到了原来第7节身体的位置，发生碰撞，游戏结束。

输入描述

输入第一行为空格分隔的字母，代表贪吃蛇的移动操作。

字母取值为U、D、L、R和G，

U、D、L、R分别表示贪吃蛇往上、下、左、右和转向，转向时贪吃蛇不移动，G表示贪吃蛇按当前的方向移动一格。

用例保证输入的操作正确。

第二行为空格分隔的两个数，指定N和M，为数组的行和列数。

余下N行每行是空格分隔的M个字母。字母取值为H、F和E，H表示贪吃蛇的起始位置，F表示食物，E表示该方格为空。

用例保证有且只有一个H，而F和E会有多个。

输出描述

输出一个数字，为蛇的长度。

用例1

输入

```
▼ Plain Text |  
1 D G G  
2 3 3  
3 F F F  
4 F F H  
5 E F E
```

输出

```
▼ Plain Text |  
1 1
```

说明

地图表示为：

- 蛇头 H(Head)
- 食物 F(Food)
- E表示该方格为空

四个方向分别表示为：

- 向上 U(up)
- 向下 D(down)
- 向左 L(Left)
- 向右 R(Right)

题解

思路： 模拟

1. 这道题主要是要想明白当蛇头移动时，其它位置的移动规律。蛇头移动时，蛇头到达新位置，第二段占据原先第一段的位置，依此类推。如果吃到果实，新的长度部分会占据原先蛇尾位置。根据这个规律可以发现，模拟过程中其实操作只会发生在蛇头和蛇尾。接下来模拟过程可以使用链表结构去模拟。
2. 碰到边界和碰到身体的情形只会发生在蛇头移动的时候。对于越界还是很好判断的。对于碰到身体的情形，可以将蛇占据的位置全部设置为 H。这样就比较好判断碰撞身体的情况。如果蛇头下一步位置为 H，并且不是蛇尾位置，那么就会发生碰撞到身体的情况。
3. 明白 1 2 规律之后，直接模拟输入的操作序列之后，结束条件为 完成所有操作或者 发生越界 或者 碰撞到身体情形。结束之后，链表长度就是蛇的最终长度。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<list>
9 using namespace std;
10
11 // 通用 split 函数
12 vector<string> split(const string& str, const string& delimiter) {
13     vector<string> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(str.substr(start, end - start));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(str.substr(start));
23     return result;
24 }
25
26 int simulation(vector<string>& operators, vector<vector<string>>& grid, l
ist<pair<int,int>>& snake, int n , int m) {
27     int offsets[2] = {0, -1};
28     for (int i = 0; i < operators.size(); i++) {
29
30         string handle = operators[i];
31         if (handle == "U") {
32             offsets[0] = -1;
33             offsets[1] = 0;
34         } else if (handle == "D") {
35             offsets[0] = 1;
36             offsets[1] = 0;
37         } else if (handle == "L") {
38             offsets[0] = 0;
39             offsets[1] = -1;
40         } else if (handle == "R") {
41             offsets[0] = 0;
42             offsets[1] = 1;
43         } else if (handle == "G") {
44             pair<int,int> head = snake.front();
```

```

45         int nextHeadX = head.first + offsets[0];
46         int nextHeadY = head.second + offsets[1];
47         // 越界
48         if (nextHeadX < 0 || nextHeadX >= n || nextHeadY < 0 || nextHe
49 adY >= m) {
50             return snake.size();
51         }
52
53         pair<int,int> tail = snake.back();
54         // 空格 相当于移除蛇尾 添加新的头
55         if (grid[nextHeadX][nextHeadY] == "E") {
56             grid[tail.first][tail.second] = "E";
57             snake.pop_back();
58             // 更新蛇头位置
59             grid[nextHeadX][nextHeadY] = "H";
60             snake.push_front({nextHeadX, nextHeadY});
61             // 原先位置不变 添加一个蛇头
62         } else if (grid[nextHeadX][nextHeadY] == "F") {
63             grid[nextHeadX][nextHeadY] = "H";
64             snake.push_front({nextHeadX, nextHeadY});
65         } else if (grid[nextHeadX][nextHeadY] == "H"){
66             // 原先蛇尾位置 不会发生碰撞
67             if (nextHeadX == tail.first && nextHeadY == tail.second)
68                 {
69                     snake.pop_back();
70                     snake.push_front({nextHeadX, nextHeadY});
71                 } else {
72                     return snake.size();
73                 }
74             }
75         }
76         return snake.size();
77     }
78
79
80     int main() {
81         string oper;
82         getline(cin , oper);
83         vector<string> operators = split(oper, " ");
84         int n,m;
85         cin >> n >> m;
86         vector<vector<string>> grid(n, vector<string>(m));
87
88         list<pair<int,int>> snake;
89         for (int i = 0; i < n; i++) {
90             for (int j = 0; j < m; j++) {

```

```
91         cin >> grid[i][j];
92         if (grid[i][j] == "H") {
93             snake.push_front({i, j});
94         }
95     }
96 }
97
98 int res = simulation(operators, grid, snake, n, m);
99 cout << res;
100 return 0;
101 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4
5     public static int simulation(List<String> operators, String[][] grid,
6         LinkedList<int[]> snake, int n, int m) {
7         int[] offsets = {0, -1}; // 初始方向 (左)
8
9         for (String handle : operators) {
10            switch (handle) {
11                case "U": offsets = new int[]{-1, 0}; break;
12                case "D": offsets = new int[]{1, 0}; break;
13                case "L": offsets = new int[]{0, -1}; break;
14                case "R": offsets = new int[]{0, 1}; break;
15                case "G":
16                    int[] head = snake.getFirst();
17                    int nextX = head[0] + offsets[0];
18                    int nextY = head[1] + offsets[1];
19                    // 越界
20                    if (nextX < 0 || nextX >= n || nextY < 0 || nextY >=
m)
21                        return snake.size();
22
23                    int[] tail = snake.getLast();
24
25                    switch (grid[nextX][nextY]) {
26                        // 空格 相当于移除蛇尾 添加新的头
27                        case "E":
28                            grid[tail[0]][tail[1]] = "E";
29                            snake.removeLast();
30                            // 更新蛇头位置
31                            grid[nextX][nextY] = "H";
32                            snake.addFirst(new int[]{nextX, nextY});
33                            break;
34                        // 原先位置不变 添加一个蛇头
35                        case "F":
36                            grid[nextX][nextY] = "H";
37                            snake.addFirst(new int[]{nextX, nextY});
38                            break;
39                        case "H":
40                            // 原先蛇尾位置 不会发生碰撞
41                            if (nextX == tail[0] && nextY == tail[1]) {
42                                snake.removeLast();
43                                snake.addFirst(new int[]{nextX, nextY});
44                            } else {
```

```
44                     return snake.size();
45                 }
46             break;
47         }
48     }
49 }
50 }
51
52     return snake.size();
53 }
54
55 public static void main(String[] args) {
56     Scanner sc = new Scanner(System.in);
57     List<String> operators = Arrays.asList(sc.nextLine().split(" "));
58     int n = sc.nextInt(), m = sc.nextInt();
59     sc.nextLine(); // 消耗换行
60
61     String[][] grid = new String[n][m];
62     LinkedList<int[]> snake = new LinkedList<>();
63
64     for (int i = 0; i < n; i++) {
65         String[] row = sc.nextLine().split(" ");
66         for (int j = 0; j < m; j++) {
67             grid[i][j] = row[j];
68             if (grid[i][j].equals("H")) {
69                 snake.addFirst(new int[]{i, j});
70             }
71         }
72     }
73
74     System.out.println(simulation(operators, grid, snake, n, m));
75 }
76 }
```

Python

```
1 # 模拟
2 def simulation(operators, grid, snake, n, m):
3     dx, dy = 0, -1 # 初始方向为左
4
5     for op in operators:
6         if op == "U":
7             dx, dy = -1, 0
8         elif op == "D":
9             dx, dy = 1, 0
10        elif op == "L":
11            dx, dy = 0, -1
12        elif op == "R":
13            dx, dy = 0, 1
14        elif op == "G":
15            head = snake[0]
16            nx, ny = head[0] + dx, head[1] + dy
17            # 越界
18            if nx < 0 or nx >= n or ny < 0 or ny >= m:
19                return len(snake)
20
21        tail = snake[-1]
22        # 空格 相当于移除蛇尾 添加新的头
23        if grid[nx][ny] == "E":
24            grid[tail[0]][tail[1]] = "E"
25            snake.pop()
26            # 更新蛇头位置
27            grid[nx][ny] = "H"
28            snake.insert(0, (nx, ny))
29        # 原先位置不变 添加一个蛇头
30        elif grid[nx][ny] == "F":
31            grid[nx][ny] = "H"
32            snake.insert(0, (nx, ny))
33        elif grid[nx][ny] == "H":
34            # 原先蛇尾位置 不会发生碰撞
35            if (nx, ny) == tail:
36                snake.pop()
37                snake.insert(0, (nx, ny))
38            else:
39                return len(snake)
40
41    return len(snake)
42
43 # 主程序入口
44 operators = input().split()
45 n, m = map(int, input().split())
```

```
46
47     grid = []
48     snake = []
49     for i in range(n):
50         row = input().split()
51         for j, val in enumerate(row):
52             if val == "H":
53                 snake.insert(0, (i, j))
54     grid.append(row)
55
56 print(simulation(operators, grid, snake, n, m))
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9 rl.on('line', line => {
10     inputLines.push(line.trim());
11     if (inputLines.length >= 3) {
12         const [n, m] = inputLines[1].split(' ').map(Number);
13         if (inputLines.length === 2 + n) {
14             rl.close();
15         }
16     }
17 });
18
19 rl.on('close', () => {
20     const operators = inputLines[0].split(' ');
21     const [n, m] = inputLines[1].split(' ').map(Number);
22     const grid = [];
23     const snake = [];
24
25     for (let i = 0; i < n; i++) {
26         const row = inputLines[2 + i].split(' ');
27         grid.push(row);
28         for (let j = 0; j < m; j++) {
29             if (row[j] === 'H') {
30                 snake.unshift([i, j]);
31             }
32         }
33     }
34
35     function simulate(ops, grid, snake, n, m) {
36         //初始方向为左
37         let dx = 0, dy = -1;
38
39         for (let op of ops) {
40             if (op === 'U') [dx, dy] = [-1, 0];
41             else if (op === 'D') [dx, dy] = [1, 0];
42             else if (op === 'L') [dx, dy] = [0, -1];
43             else if (op === 'R') [dx, dy] = [0, 1];
44             else if (op === 'G') {
45                 const [hx, hy] = snake[0];
```

```

46         const nx = hx + dx, ny = hy + dy;
47         // 越界
48         if (nx < 0 || nx >= n || ny < 0 || ny >= m)
49             return snake.length;
50
51         const [tx, ty] = snake[snake.length - 1];
52         // 空格 相当于移除蛇尾 添加新的头
53         if (grid[nx][ny] === 'E') {
54             grid[tx][ty] = 'E';
55             snake.pop();
56             // 更新蛇头位置
57             grid[nx][ny] = 'H';
58             snake.unshift([nx, ny]);
59             // 原先位置不变 添加一个蛇头
60         } else if (grid[nx][ny] === 'F') {
61             grid[nx][ny] = 'H';
62             snake.unshift([nx, ny]);
63         } else if (grid[nx][ny] === 'H') {
64             // 原先蛇尾位置 不会发生碰撞
65             if (nx === tx && ny === ty) {
66                 snake.pop();
67                 snake.unshift([nx, ny]);
68             } else {
69                 return snake.length;
70             }
71         }
72     }
73
74     return snake.length;
75 }
76
77     console.log(simulate(operators, grid, snake, n, m));
78 });
79

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8     "strconv"
9 )
10
11 type Point struct{ x, y int }
12
13 func simulation(operators []string, grid [][]string, snake []Point, n, m int) int {
14     dx, dy := 0, -1 // 初始方向为左
15
16     for _, op := range operators {
17         switch op {
18             case "U":
19                 dx, dy = -1, 0
20             case "D":
21                 dx, dy = 1, 0
22             case "L":
23                 dx, dy = 0, -1
24             case "R":
25                 dx, dy = 0, 1
26             case "G":
27                 head := snake[0]
28                 nx, ny := head.x+dx, head.y+dy
29                     // 越界
30                 if nx < 0 || nx >= n || ny < 0 || ny >= m {
31                     return len(snake)
32                 }
33
34                 tail := snake[len(snake)-1]
35
36                 switch grid[nx][ny] {
37                     // 空格 相当于移除蛇尾 添加新的头
38                     case "E":
39                         grid[tail.x][tail.y] = "E"
40                         snake = snake[:len(snake)-1]
41                         // 更新蛇头位置
42                         grid[nx][ny] = "H"
43                         snake = append([]Point{{nx, ny}}, snake...)
44                         // 原先位置不变 添加一个蛇头
```

```

45     case "F":
46         grid[nx][ny] = "H"
47         snake = append([]Point{{nx, ny}}, snake...)
48     case "H":
49         // 原先蛇尾位置 不会发生碰撞
50         if nx == tail.x && ny == tail.y {
51             snake = snake[:len(snake)-1]
52             snake = append([]Point{{nx, ny}}, snake...)
53         } else {
54             return len(snake)
55         }
56     }
57 }
58 }
59 return len(snake)
60 }
61
62 func main() {
63     scanner := bufio.NewScanner(os.Stdin)
64     scanner.Scan()
65     operators := strings.Split(scanner.Text(), " ")
66     scanner.Scan()
67     dims := strings.Fields(scanner.Text())
68     n, _ := strconv.Atoi(dims[0])
69     m, _ := strconv.Atoi(dims[1])
70
71     grid := make([][]string, n)
72     snake := []Point{}
73
74     for i := 0; i < n; i++ {
75         scanner.Scan()
76         row := strings.Fields(scanner.Text())
77         grid[i] = row
78         for j, val := range row {
79             if val == "H" {
80                 snake = append([]Point{{i, j}}, snake...)
81             }
82         }
83     }
84
85     fmt.Println(simulation(operators, grid, snake, n, m))
86 }

```

来自: 华为OD机试2025C卷 – 贪吃蛇 (C++ & Python & JAVA & JS & GO)–CSDN博客

华为OD机试2025C卷 - 符号运算 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 符号运算

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 100分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

给定一个表达式，求其分数计算结果。

表达式的限制如下：

1. 所有的输入数字皆为正整数（包括0）
2. 仅支持四则运算 ($\pm*/$) 和括号
3. 结果为整数或分数，分数必须化为最简格式（比如6, $3/4$, $7/8$, $90/7$ ）
4. 除数可能为0，如果遇到这种情况，直接输出"ERROR"
5. 输入和最终计算结果中的数字都不会超出整型范围

用例输入一定合法，不会出现括号匹配的情况

输入描述

字符串格式的表达式，仅支持 $\pm*/$ ，数字可能超过两位，可能带有空格，没有负数
长度小于200个字符

输出描述

表达式结果，以最简格式表达

- 如果结果为整数，那么直接输出整数
- 如果结果为负数，那么分子分母不可再约分，可以为假分数，不可表达为带分数
- 结果可能是负数，符号放在前面

用例1

输入

```
1 1 + 5 * 7 / 8
```

输出

Plain Text |

1 43/8

用例2

输入

Plain Text |

1 1 / (0 - 5)

输出

Plain Text |

1 -1/5

说明

| 符号需要提到最前面

用例3

输入

Plain Text |

1 1 * (3*4/(8-(7+0)))

输出

Plain Text |

1 12

说明

| 注意括号可以多重嵌套

题解

思路： 模拟 题，使用两个栈来进行处理

- 一个用来记录操作数
- 一个用来记录操作符

具体步骤如下：

1. 正常情况下，解析表达式，将操作数和操作符按照顺序压入对应栈中，操作数的存储可以采用 **分子 分母** 分开存储，方便进行乘数和除数的运算。。中途可以由于优先级可能发生计算，可以参考2的说明。
2. 计算可以发生在以下情况：
 - 当前操作符优先级低于上个操作符时
 - 遇到 **)** 时，需要计算 **)** 代表的 **(subexpress)** 子表达式中的值
 - 遍历完输入表达式时，最终求解
3. 计算的具体过程是从操作数栈中弹出两个数 **b a**，操作符栈中弹出一个元素，代表的含义其实是 **a 操作符 b**，计算结果之后重新压入操作符栈中。具体计算可以定义一个函数处理。
4. 额外注意是否出现 除以 0 的情况。
5. 如果计算过程中出现 **除以0** 的非法运算，直接输出 **ERROR**。否则对最终结果的分子和分母进行最大公约数通分得到最简分数，输出。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 #include<cmath>
11 #include<stack>
12 using namespace std;
13
14 struct Fractions{
15     int fa; // 分母
16     int ch; // 分子
17     Fractions() {}
18     Fractions(int fa, int ch): fa(fa), ch(ch) {}
19 };
20
21 // 操作数栈
22 stack<Fractions> operaorNum;
23 // 操作符栈
24 stack<char> operatorSign;
25 // 记录除0的非常操作
26 bool invalidFlag = false;
27 // 使用操作数栈和操作符栈进行计算
28 void calc() {
29     Fractions b = operaorNum.top();
30     operaorNum.pop();
31
32     Fractions a = operaorNum.top();
33     operaorNum.pop();
34
35     char op = operatorSign.top();
36     operatorSign.pop();
37     Fractions result;
38     // 四则运算
39     switch (op) {
40         case '+':
41             result.fa = a.fa * b.fa;
42             result.ch = a.ch * b.fa + b.ch * a.fa;
43             break;
44         case '-':
45             result.fa = a.fa * b.fa;
```

```

46         result.ch = a.ch * b.fa - b.ch * a.fa;
47         break;
48     case '*':
49         result.fa = a.fa * b.fa;
50         result.ch = a.ch * b.ch ;
51         break;
52     case '/':
53         result.fa = a.fa * b.ch;
54         result.ch = a.ch * b.fa;
55         break;
56     }
57     if (result.fa == 0) {
58         invalidFlag = true;
59     }
60     operaorNum.push(result);
61 }
62
63 // 求最大公约数
64 int gcd(int a, int b) {
65     while (b != 0) {
66         int temp = b;
67         b = a % b;
68         a = temp;
69     }
70     return a < 0 ? -a : a; // 确保结果非负
71 }
72
73
74 string solve(string express) {
75     map<char, int> opertorPriorty;
76     opertorPriorty['+'] = 1;
77     opertorPriorty['-'] = 1;
78     opertorPriorty['*'] = 2;
79     opertorPriorty['/] = 2;
80
81     string numStr = "";
82
83     int pos = 0;
84     while (pos < express.size()) {
85         if (invalidFlag) {
86             break;
87         }
88         char c = express[pos];
89         if (c >= '0' && c <= '9') {
90             // 获取完整数字
91             while (c >= '0' && c <= '9') {
92                 numStr += c;
93                 if (pos + 1 >= express.size()) {

```

```

94         break;
95     }
96     pos++;
97     c = express[pos];
98 }
99 // 压入操作数栈
100 operaorNum.push({1, stoi(numStr)});
101 // 清空
102 numStr = "";
103 }
104
105 if (c == '+' || c == '-' || c == '*' || c == '/') {
106     while (!operatorSign.empty() && operatorSign.top() != '(' &&
107     opertorPriorty[c] <= opertorPriorty[operatorSign.top()]) {
108         calc();
109     }
110     operatorSign.push(c);
111 } else if (c == ')') {
112     while (operatorSign.top() != '(') {
113         calc();
114     }
115     // 移除对应的(
116     operatorSign.pop();
117 } else if (c == '(') {
118     // 压栈
119     operatorSign.push(c);
120 }
121 pos++;
122 }
123 while (operaorNum.size() > 1) {
124     calc();
125 }
126
127 Fractions result = operaorNum.top();
128 operaorNum.pop();
129 // 分母为零
130 if (result.fa == 0 || invalidFlag) {
131     return "ERROR";
132 }
133 // 最简分数处理
134 int k = gcd(result.fa, result.ch);
135 result.fa /= k;
136 result.ch /= k;
137
138 string sign = result.fa * result.ch < 0 ? "-" : "";
139
140 int fa = abs(result.fa);

```

```
141     int ch = abs(result.ch);
142
143     if (fa == 1) {
144         return sign + to_string(ch);
145     } else {
146         return sign + to_string(ch) + "/" + to_string(fa);
147     }
148 }
149
150
151 int main() {
152     string express;
153     getline(cin ,express);
154     string result = solve(express);
155
156     cout<<result;
157     return 0;
158 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4
5     static class Fractions {
6         int fa; // 分母
7         int ch; // 分子
8
9         Fractions() {}
10        Fractions(int fa, int ch) {
11            this.fa = fa;
12            this.ch = ch;
13        }
14    }
15
16    // 操作数栈
17    static Stack<Fractions> operaorNum = new Stack<>();
18    // 操作符栈
19    static Stack<Character> operatorSign = new Stack<>();
20    // 记录除0的非常操作
21    static boolean invalidFlag = false;
22
23    // 四则运算
24    static void calc() {
25        Fractions b = operaorNum.pop();
26        Fractions a = operaorNum.pop();
27        char op = operatorSign.pop();
28        Fractions result = new Fractions();
29
30        switch (op) {
31            case '+':
32                result.fa = a.fa * b.fa;
33                result.ch = a.ch * b.fa + b.ch * a.fa;
34                break;
35            case '-':
36                result.fa = a.fa * b.fa;
37                result.ch = a.ch * b.fa - b.ch * a.fa;
38                break;
39            case '*':
40                result.fa = a.fa * b.fa;
41                result.ch = a.ch * b.ch;
42                break;
43            case '/':
44                result.fa = a.fa * b.ch;
45                result.ch = a.ch * b.fa;
```

```

46                     break;
47     }
48     if (result.fa == 0) {
49         invalidFlag = true;
50     }
51     operaorNum.push(result);
52 }
53
54 // 求最大公约数
55 static int gcd(int a, int b) {
56     while (b != 0) {
57         int temp = b;
58         b = a % b;
59         a = temp;
60     }
61     return a < 0 ? -a : a; // 确保结果非负
62 }
63
64 // 解析表达式
65 static String solve(String express) {
66     Map<Character, Integer> opertorPriorty = new HashMap<>();
67     opertorPriorty.put('+', 1);
68     opertorPriorty.put('-', 1);
69     opertorPriorty.put('*', 2);
70     opertorPriorty.put('/', 2);
71
72     StringBuilder numStr = new StringBuilder();
73     int pos = 0;
74
75     while (pos < express.length()) {
76         if (invalidFlag) {
77             break;
78         }
79         char c = express.charAt(pos);
80         if (Character.isDigit(c)) {
81             // 获取完整数字
82             while (Character.isDigit(c)) {
83                 numStr.append(c);
84                 if (pos + 1 >= express.length()) {
85                     break;
86                 }
87                 pos++;
88                 c = express.charAt(pos);
89             }
90             // 压入操作数栈
91             operaorNum.push(new Fractions(1, Integer.parseInt(numStr.
92                         toString())));
93             numStr.setLength(0); // 清空

```

```

93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
    }
    if (c == '+' || c == '-' || c == '*' || c == '/') {
        while (!operatorSign.isEmpty() && operatorSign.peek() != ')'
            && opertorPriority.get(c) <= opertorPriority.get(operatorSign.peek())))
    {
        calc();
    }
    operatorSign.push(c);
} else if (c == ')') {
    while (operatorSign.peek() != '(') {
        calc();
    }
    // 移除对应的(
    operatorSign.pop();
} else if (c == '(') {
    // 压栈
    operatorSign.push(c);
}
pos++;
}

while (operaorNum.size() > 1) {
    calc();
}

Fractions result = operaorNum.pop();
if (result.fa == 0 || invalidFlag) {
    return "ERROR";
}
// 最简分数处理
int k = gcd(result.fa, result.ch);
result.fa /= k;
result.ch /= k;

String sign = result.fa * result.ch < 0 ? "-" : "";
int fa = Math.abs(result.fa);
int ch = Math.abs(result.ch);

if (fa == 1) {
    return sign + ch;
} else {
    return sign + ch + "/" + fa;
}
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
}

```

```
139     String express = sc.nextLine();
140     String result = solve(express);
141     System.out.println(result);
142 }
143 }
```

Python

```
1 # 定义分数类
2 class Fractions:
3     def __init__(self, fa=0, ch=0):
4         self.fa = fa # 分母
5         self.ch = ch # 分子
6
7     # 操作数栈
8     operaorNum = []
9     # 操作符栈
10    operatorSign = []
11    # 记录除0的非常操作
12    invalidFlag = False
13
14    # 四则运算
15    def calc():
16        global invalidFlag
17        b = operaorNum.pop()
18        a = operaorNum.pop()
19        op = operatorSign.pop()
20        result = Fractions()
21
22        if op == '+':
23            result.fa = a.fa * b.fa
24            result.ch = a.ch * b.fa + b.ch * a.fa
25        elif op == '-':
26            result.fa = a.fa * b.fa
27            result.ch = a.ch * b.fa - b.ch * a.fa
28        elif op == '*':
29            result.fa = a.fa * b.fa
30            result.ch = a.ch * b.ch
31        elif op == '/':
32            result.fa = a.fa * b.ch
33            result.ch = a.ch * b.fa
34
35        if result.fa == 0:
36            invalidFlag = True
37
38        operaorNum.append(result)
39
40    # 求最大公约数
41    def gcd(a, b):
42        while b != 0:
43            a, b = b, a % b
44        return abs(a)
45
```

```

46     # 解析表达式
47     def solve(express):
48         global invalidFlag
49         opertorPriority = {'+": 1, "-": 1, "*": 2, "/": 2}
50
51         numStr = ""
52         pos = 0
53
54         while pos < len(express):
55             if invalidFlag:
56                 break
57             c = express[pos]
58             if c.isdigit():
59                 # 获取完整数字
60                 while c.isdigit():
61                     numStr += c
62                     pos += 1
63                     if pos >= len(express):
64                         break
65                     c = express[pos]
66                 # 压入操作数栈
67                 operaorNum.append(Fractions(1, int(numStr)))
68                 numStr = ""
69                 continue
70             elif c in opertorPriority:
71                 while operatorSign and operatorSign[-1] != '(' and opertorPriority[c] <= opertorPriority[operatorSign[-1]]:
72                     calc()
73                     operatorSign.append(c)
74             elif c == ')':
75                 while operatorSign[-1] != '(':
76                     calc()
77                 operatorSign.pop() # 移除左括号
78             elif c == '(':
79                 operatorSign.append(c)
80
81             pos += 1
82
83         while len(operaorNum) > 1:
84             calc()
85
86         result = operaorNum.pop()
87         if result.fa == 0 or invalidFlag:
88             return "ERROR"
89
90         # 最简分数处理
91         k = gcd(result.fa, result.ch)
92         result.fa //= k

```

```
93     result.ch //= k
94
95     sign = '-' if result.fa * result.ch < 0 else ''
96
97     fa = abs(result.fa)
98     ch = abs(result.ch)
99
100    if fa == 1:
101        return f"{sign}{ch}"
102    else:
103        return f"{sign}{ch}/{fa}"
104
105    # 主函数
106    if __name__ == "__main__":
107        express = input()
108        result = solve(express)
109        print(result)
```

JavaScript

```
1 const fs = require("fs");
2
3 // 分数类
4 class Fraction {
5     constructor(fa = 0, ch = 0) {
6         this.fa = fa; // 分母
7         this.ch = ch; // 分子
8     }
9 }
10
11 // 操作数栈
12 let operandStack = [];
13 // 操作符栈
14 let operatorStack = [];
15 // 记录除0的非法操作
16 let invalidFlag = false;
17
18 // 计算四则运算
19 function calc() {
20     if (operandStack.length < 2 || operatorStack.length === 0) return;
21     let b = operandStack.pop();
22     let a = operandStack.pop();
23     let op = operatorStack.pop();
24     let result = new Fraction();
25
26     if (op === "+") {
27         result.fa = a.fa * b.fa;
28         result.ch = a.ch * b.fa + b.ch * a.fa;
29     } else if (op === "-") {
30         result.fa = a.fa * b.fa;
31         result.ch = a.ch * b.fa - b.ch * a.fa;
32     } else if (op === "*") {
33         result.fa = a.fa * b.fa;
34         result.ch = a.ch * b.ch;
35     } else if (op === "/") {
36         if (b.ch === 0) {
37             invalidFlag = true;
38             return;
39         }
40         result.fa = a.fa * b.ch;
41         result.ch = a.ch * b.fa;
42     }
43
44     if (result.fa === 0) {
45         invalidFlag = true;
46     }
47 }
```

```

46     }
47
48     operandStack.push(result);
49 }
50
51 // 求最大公约数
52 function gcd(a, b) {
53     while (b !== 0) {
54         [a, b] = [b, a % b];
55     }
56     return Math.abs(a);
57 }
58
59 // 解析表达式并计算结果
60 function solve(expression) {
61     invalidFlag = false;
62     operandStack = [];
63     operatorStack = [];
64
65     const operatorPriority = { "+": 1, "-": 1, "*": 2, "/": 2 };
66
67     let numStr = "";
68     let pos = 0;
69
70     while (pos < expression.length) {
71         if (invalidFlag) break;
72         let c = expression[pos];
73
74         if (/^\d/.test(c)) {
75             // 获取完整数字
76             while (/^\d/.test(c)) {
77                 numStr += c;
78                 pos++;
79                 if (pos >= expression.length) break;
80                 c = expression[pos];
81             }
82             // 压入操作数栈
83             operandStack.push(new Fraction(1, parseInt(numStr, 10)));
84             numStr = "";
85             continue;
86         } else if (c in operatorPriority) {
87             while (
88                 operatorStack.length &&
89                 operatorStack[operatorStack.length - 1] !== "(" &&
90                 operatorPriority[c] <= operatorPriority[operatorStack[ope
ratorStack.length - 1]]
91             ) {
92                 calc();

```

```

93         }
94         operatorStack.push(c);
95     } else if (c === ")") {
96         while (operatorStack.length && operatorStack[operatorStack.length - 1] !== "(") {
97             calc();
98         }
99         operatorStack.pop(); // 移除左括号
100    } else if (c === "(") {
101        operatorStack.push(c);
102    }
103
104    pos++;
105}
106
107 while (operandStack.length > 1) {
108     calc();
109 }
110
111 if (operandStack.length === 0 || invalidFlag) return "ERROR";
112
113 let result = operandStack.pop();
114
115 // 最简分数处理
116 let k = gcd(result.fa, result.ch);
117 result.fa /= k;
118 result.ch /= k;
119
120 let sign = result.fa * result.ch < 0 ? "-" : "";
121 let fa = Math.abs(result.fa);
122 let ch = Math.abs(result.ch);
123
124 return fa === 1 ? `${sign}${ch}` : `${sign}${ch}/${fa}`;
125 }
126
127 // 读取输入并计算
128 if (require.main === module) {
129     const input = fs.readFileSync(0, "utf-8").trim();
130     console.log(solve(input));
131 }

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 分数结构体
12 type Fraction struct {
13     fa int // 分母
14     ch int // 分子
15 }
16
17 // 操作数栈
18 var operatorNum []Fraction
19 // 操作符栈
20 var operatorSign []rune
21 // 记录除0的非法操作
22 var invalidFlag bool = false
23
24 // 计算四则运算
25 func calc() {
26     b := operatorNum[len(operatorNum)-1]
27     operatorNum = operatorNum[:len(operatorNum)-1]
28
29     a := operatorNum[len(operatorNum)-1]
30     operatorNum = operatorNum[:len(operatorNum)-1]
31
32     op := operatorSign[len(operatorSign)-1]
33     operatorSign = operatorSign[:len(operatorSign)-1]
34
35     var result Fraction
36     switch op {
37     case '+':
38         result = Fraction{a.fa * b.fa, a.ch*b.fa + b.ch*a.fa}
39     case '-':
40         result = Fraction{a.fa * b.fa, a.ch*b.fa - b.ch*a.fa}
41     case '*':
42         result = Fraction{a.fa * b.fa, a.ch * b.ch}
43     case '/':
44         if b.ch == 0 {
45             invalidFlag = true
46         }
47     }
48 }
```

```

46         return
47     }
48     result = Fraction{a.fa * b.ch, a.ch * b.fa}
49 }
50
51     if result.fa == 0 {
52         invalidFlag = true
53     }
54     operatorNum = append(operatorNum, result)
55 }
56
57 // 求最大公约数
58 func gcd(a, b int) int {
59     for b != 0 {
60         a, b = b, a%b
61     }
62     if a < 0 {
63         return -a
64     }
65     return a
66 }
67
68 // 解析并计算表达式
69 func solve(express string) string {
70     operatorPriority := map[rune]int{'+": 1, '-': 1, '*': 2, '/': 2}
71
72     numStr := ""
73     for pos := 0; pos < len(express); pos++ {
74         if invalidFlag {
75             break
76         }
77         c := rune(express[pos])
78
79         if c >= '0' && c <= '9' {
80             numStr += string(c)
81             if pos+1 >= len(express) || express[pos+1] < '0' || express[pos+1]
82             > '9' {
83                 val, _ := strconv.Atoi(numStr)
84                 operatorNum = append(operatorNum, Fraction{1, val})
85                 numStr = ""
86             }
87         } else if c == '+' || c == '-' || c == '*' || c == '/' {
88             for len(operatorSign) > 0 && operatorSign[len(operatorSign)-1] !=
89             '(' &&
90                 operatorPriority[c] <= operatorPriority[operatorSign[len(operator
91             Sign)-1]] {
92                     calc()
93                 }
94             }

```

```

91         operatorSign = append(operatorSign, c)
92     } else if c == ')' {
93         for operatorSign[len(operatorSign)-1] != '(' {
94             calc()
95         }
96         operatorSign = operatorSign[:len(operatorSign)-1] // 移除 ')'
97     } else if c == '(' {
98         operatorSign = append(operatorSign, c)
99     }
100    }
101
102    for len(operatorNum) > 1 {
103        calc()
104    }
105
106    result := operatorNum[0]
107    if result.fa == 0 || invalidFlag {
108        return "ERROR"
109    }
110
111    k := gcd(result.fa, result.ch)
112    result.fa /= k
113    result.ch /= k
114
115    sign := ""
116    if result.fa*result.ch < 0 {
117        sign = "-"
118    }
119
120    fa, ch := abs(result.fa), abs(result.ch)
121    if fa == 1 {
122        return fmt.Sprintf("%s%d", sign, ch)
123    } else {
124        return fmt.Sprintf("%s%d/%d", sign, ch, fa)
125    }
126}
127
128 // 绝对值
129 func abs(x int) int {
130     if x < 0 {
131         return -x
132     }
133     return x
134 }
135
136 func main() {
137     reader := bufio.NewReader(os.Stdin)
138     express, _ := reader.ReadString('\n')

```

```
139     express = strings.TrimSpace(express)
140
141     fmt.Println(solve(express))
142 }
```

| 来自: 华为OD机试2025C卷 – 符号运算 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 空栈压数 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 空栈压数

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 200分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

向一个空栈中依次存入正整数，假设入栈元素 $n(1 \leq n \leq 2^{31}-1)$ 按顺序依次为 $n_4 \dots n_1$ ，每当元素入栈时，如果 $n_1 = n_2 + \dots + n_y$ (y 的范围 $[2, x]$, $1 \leq x \leq 1000$)，则 $n_1 \sim n_y$ 全部元素出栈，重新入栈新元素 $m(m=2*n_1)$ 。

如：依次向栈存入 6、1、2、3，当存入 6、1、2 时，栈底至栈顶依次为 [6、1、2]；当存入 3 时， $3=2+1$ ，3、2、1 全部出栈，重新入栈元素 6($6=2*3$)，此时栈中有元素 6；

因为 $6=6$ ，所以两个 6 全部出栈，存入 12，最终栈中只剩一个元素 12。

输入描述

使用单个空格隔开的正整数的字符串，如”5 6 7 8”，左边的数字先入栈，输入的正整数个数为 x ， $1 \leq x \leq 1000$ 。

输出描述

最终栈中存留的元素值，元素值使用空格隔开，如”8 7 6 5”，栈顶数字在左边。 6 1 2 3

用例1

输入

```
1 5 10 20 50 85 1
```

输出

```
1 1 170
```

说明

$5+10+20+50=85$, 输入 85 时, 5、10、20、50、85 全部出栈, 入栈 170, 最终依次出栈的数字为 1 和 170。

用例2

输入

Plain Text |

```
1 6 7 8 13 9
```

输出

Plain Text |

```
1 9 13 8 7 6
```

用例3

输入

Plain Text |

```
1 1 2 5 7 9 1 2 2
```

输出

Plain Text |

```
1 4 1 9 14 1
```

题解

思路： 模拟

1. 按照题目使用数组进行模拟就行。当一个元素 x 被压入栈的时候，判断是否存在连续后缀和等于当前元素，如果存在的话弹出连续后缀，压入 $2 * x + 1$ 。若不存在压入 x 即可。
2. 这道题主要考点就是 1 弹出连续后缀的操作，需要递归进行判断，例如压入 $2 * x + 1$ 后，又存在一个新的连续后缀和等于 $2 * x + 1$ 。
3. 遍历完所有元素之后，输出数组中保留的元素即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<stack>
8 using namespace std;
9
10 int main() {
11     vector<int> ans;
12     int tmp;
13     while (cin >> tmp) {
14         ans.push_back(tmp);
15     }
16
17     int n = ans.size();
18     vector<int> stk(n, 0);
19     // 下一个要放入的位置
20     int pos = 0;
21     for (int i = 0; i < n; i++) {
22         long sum = ans[i];
23         long tmp = sum;
24
25         // 递归进行压栈判断
26         while (pos > 0 && tmp >= 0) {
27             int j = pos - 1;
28             for (; j >= 0; j--) {
29                 tmp -= stk[j];
30                 if (tmp <= 0) {
31                     break;
32                 }
33             }
34             // 满足条件，更新元素
35             if (tmp == 0) {
36                 sum = 2 * sum;
37                 pos = j;
38                 tmp = sum;
39             } else {
40                 break;
41             }
42         }
43         stk[pos++] = sum;
45     }
}
```

```
46
47     for (int i = pos -1; i >=0;i--) {
48         cout << stk[i] << " ";
49     }
50     return 0;
51 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         List<Integer> ans = new ArrayList<>();
7
8         // 读取输入直到结束
9         while (sc.hasNextInt()) {
10             ans.add(sc.nextInt());
11         }
12
13         int n = ans.size();
14
15         int[] stk = new int[n];
16         int pos = 0;
17
18         for (int i = 0; i < n; i++) {
19             long sum = ans.get(i);
20             long tmp = sum;
21
22             // 递归进行压栈判断
23             while (pos > 0 && tmp >= 0) {
24                 int j = pos - 1;
25                 for (; j >= 0; j--) {
26                     tmp -= stk[j];
27                     if (tmp <= 0) {
28                         break;
29                     }
30                 }
31                 // 满足条件，更新元素
32                 if (tmp == 0) {
33                     sum = 2 * sum;
34                     pos = j;
35                     tmp = sum;
36                 } else {
37                     break;
38                 }
39             }
40             stk[pos++] = (int) sum;
41         }
42
43         // 输出栈中的元素
44         for (int i = pos - 1; i >= 0; i--) {
```

```
46             System.out.print(stk[i] + " ");
47         }
48     }
49 }
```

Python

```
1 def main():
2     ans = []
3
4     # 读取输入数据，直到没有更多输入
5     try:
6         while True:
7             # 读取一行输入
8             line = input().strip()
9             # 拆分输入的字符串并将每个数字转换为整数
10            ans.extend(map(int, line.split()))
11    except EOFError:
12        pass
13
14    n = len(ans)
15    stk = [0] * n
16    pos = 0
17
18    for i in range(n):
19        sum_val = ans[i]
20        tmp = sum_val
21
22        # 递归进行压栈判断
23        while pos > 0 and tmp >= 0:
24            j = pos - 1
25            while j >= 0:
26                tmp -= stk[j]
27                if tmp <= 0:
28                    break
29                j -= 1
30
31            # 满足条件，更新元素
32            if tmp == 0:
33                sum_val = 2 * sum_val
34                pos = j
35                tmp = sum_val
36            else:
37                break
38
39            stk[pos] = sum_val
40            pos += 1
41
42        # 输出栈中的元素
43        for i in range(pos - 1, -1, -1):
44            print(stk[i], end=" ")
```

```
46  if __name__ == "__main__":
47      main()
```

JavaScript

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require('readline');
3
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout
7 });
8
9  function main() {
10     let ans = [];
11
12     // 读取输入数据
13     rl.on('line', (input) => {
14         if (input.trim() === '') {
15             // 结束输入
16             rl.close();
17         } else {
18             // 将输入的字符串按空格分割，并转换为整数
19             ans.push(...input.split(' ').map(Number));
20         }
21     });
22
23     rl.on('close', () => {
24         let n = ans.length;
25         let stk = new Array(n).fill(0);
26         let pos = 0;
27
28         for (let i = 0; i < n; i++) {
29             let sum = ans[i];
30             let tmp = sum;
31
32             // 递归进行压栈判断
33             while (pos > 0 && tmp >= 0) {
34                 let j = pos - 1;
35                 while (j >= 0) {
36                     tmp -= stk[j];
37                     if (tmp <= 0) {
38                         break;
39                     }
40                     j--;
41                 }
42
43             // 满足条件，更新元素
44             if (tmp === 0) {
45                 sum = 2 * sum;
```

```
46             pos = j;
47             tmp = sum;
48         } else {
49             break;
50         }
51     }
52
53     stk[pos] = sum;
54     pos++;
55 }
56
57 // 输出栈中的元素
58 console.log(stk.slice(0, pos).reverse().join(' '));
59 })
60 }
61
62 main();
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     // 从标准输入读取数据
13     scanner := bufio.NewScanner(os.Stdin)
14
15     var ans []int
16     if scanner.Scan() {
17         line := scanner.Text()
18         inputs := strings.Fields(line)
19         for _, val := range inputs {
20             num, err := strconv.Atoi(val)
21             if err == nil {
22                 ans = append(ans, num)
23             }
24         }
25     }
26
27     n := len(ans)
28     stk := make([]int, n)
29     pos := 0 // 下一个要放入的位置
30
31     for i := 0; i < n; i++ {
32         sum := ans[i]
33         tmp := sum
34
35         // 递归进行压栈判断
36         for pos > 0 && tmp >= 0 {
37             j := pos - 1
38             for j >= 0 {
39                 tmp -= stk[j]
40                 if tmp <= 0 {
41                     break
42                 }
43                 j--
44             }
45         }
```

```
46     // 满足条件，更新元素
47     if tmp == 0 {
48         sum = 2 * sum
49         pos = j
50         tmp = sum
51     } else {
52         break
53     }
54 }
55
56     stk[pos] = sum
57     pos++
58 }
59
60 // 输出结果
61 for i := pos - 1; i >= 0; i-- {
62     fmt.Println(stk[i], " ")
63 }
64 }
```

| 来自: 华为OD机试2025C卷 – 空栈压数 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 生成回文素数 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 生成回文素数

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 100分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

求出大于或等于 N 的最小回文素数。

如果一个数大于 1，且其因数只有 1 和它自身，那么这个数是素数。例如，2, 3, 5, 7, 11 以及 13 是素数。

如果一个数从左往右读与从右往左读是一样的，那么这个数是回文数，例如，12321 是回文数。

输入描述

一个数

输出描述

一个数

用例1

输入

▼	Plain Text
1 6	

输出

▼	Plain Text
1 7	

用例2

输入

Plain Text |

1 8

输出

Plain Text |

1 11

题解

思路： 模拟 题，

1. 从N开始枚举，每次枚举值 + 1，直到找到一个数既是素数又是回文数结束。
2. 回文数判断：基于回文数 从左往右读与从右往左读是一样的 特征，可以将数字转换为字符串，然后使用 双指针 进行判断。
3. 素数判断：
 - a. 常规验证方法：枚举除数范围[2,sqrt(n)]，判断n能否被其中一个数整除，如果存在则不是素数。
 - b. 基于数学原理：将所有数都表示成 $6k + m$ 其中 $6k + 0$, $6k + 2$, $6k + 3$, $6k + 4$ 形式的数肯定不是素数，对于[2,sqrt(n)]的被除数枚举范围可以缩小为[2,sqrt(n)]中符合 $6k+5$, $6k+1$ 特征的数。
4. 根据 1 2 3 的逻辑执行，直到找到一个既是素数又是回文数的数字时结束循环，该数字即为结果。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 using namespace std;
11
12 // 判断是否为素数
13 bool isPrime(int n) {
14     if (n < 2) return false;
15     if (n == 2 || n == 3) return true;
16     if (n % 2 == 0 || n % 3 == 0) return false;
17     // 只需要检查6k + 1 和 6k - 1
18     for (int i = 5; i * i <= n; i += 6) {
19         if (n % i == 0 || n % (i + 2) == 0)
20             return false;
21     }
22     return true;
23 }
24
25 // 判断是否为回文数 双指针
26 bool isSymmetryNum(int num) {
27     string s = to_string(num);
28     int i = 0, j = s.size() - 1;
29     while (i < j) {
30         if (s[i] != s[j]) {
31             return false;
32         }
33         i++;
34         j--;
35     }
36     return true;
37 }
38
39
40 int main() {
41     long n;
42     cin >> n;
43     while (true) {
44         if (isSymmetryNum(n) && isPrime(n)) {
45             cout << n;
```

```
46         break;
47     }
48     // 尝试下一个数
49     n++;
50 }
51 return 0;
52 }
```

JAVA

```
1 import java.util.Scanner;
2
3 public class Main {
4
5     // 判断是否为素数
6     public static boolean isPrime(long n) {
7         if (n < 2) return false;
8         if (n == 2 || n == 3) return true;
9         if (n % 2 == 0 || n % 3 == 0) return false;
10        // 只需要检查 6k ± 1 形式的数
11        for (long i = 5; i * i <= n; i += 6) {
12            if (n % i == 0 || n % (i + 2) == 0)
13                return false;
14        }
15        return true;
16    }
17
18    // 判断是否为回文数
19    public static boolean isSymmetryNum(long num) {
20        String s = Long.toString(num);
21        int i = 0, j = s.length() - 1;
22        while (i < j) {
23            if (s.charAt(i) != s.charAt(j)) {
24                return false;
25            }
26            i++;
27            j--;
28        }
29        return true;
30    }
31
32    public static void main(String[] args) {
33        Scanner scanner = new Scanner(System.in);
34        long n = scanner.nextLong();
35        scanner.close();
36
37        while (true) {
38            if (isSymmetryNum(n) && isPrime(n)) {
39                System.out.println(n);
40                break;
41            }
42            n++; // 尝试下一个数
43        }
44    }
45}
```

Python

Plain Text |

```
1 import sys
2
3 # 判断是否为素数
4 def is_prime(n: int) -> bool:
5     if n < 2:
6         return False
7     if n in (2, 3):
8         return True
9     if n % 2 == 0 or n % 3 == 0:
10        return False
11    # 只需要检查 6k ± 1 形式的数
12    i = 5
13    while i * i <= n:
14        if n % i == 0 or n % (i + 2) == 0:
15            return False
16        i += 6
17    return True
18
19 # 判断是否为回文数
20 def is_symmetry_num(num: int) -> bool:
21     return str(num) == str(num)[::-1]
22
23 def main():
24     n = int(sys.stdin.readline().strip())
25
26     while True:
27         if is_symmetry_num(n) and is_prime(n):
28             print(n)
29             break
30         n += 1 # 尝试下一个数
31
32 if __name__ == "__main__":
33     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 // 判断是否为素数
9 function isPrime(n) {
10     if (n < 2) return false;
11     if (n === 2 || n === 3) return true;
12     if (n % 2 === 0 || n % 3 === 0) return false;
13     // 只需要检查 6k ± 1 形式的数
14     for (let i = 5; i * i <= n; i += 6) {
15         if (n % i === 0 || n % (i + 2) === 0)
16             return false;
17     }
18     return true;
19 }
20
21 // 判断是否为回文数
22 function isSymmetryNum(num) {
23     const s = num.toString();
24     return s === s.split('').reverse().join('');
25 }
26
27 rl.on('line', (line) => {
28     let n = BigInt(line.trim()); // 使用 BigInt 处理大数
29
30     while (true) {
31         if (isSymmetryNum(n) && isPrime(Number(n))) {
32             console.log(n.toString());
33             break;
34         }
35         n++; // 尝试下一个数
36     }
37
38     rl.close();
39 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8 )
9
10 // 判断是否为素数
11 func isPrime(n int) bool {
12     if n < 2 {
13         return false
14     }
15     if n == 2 || n == 3 {
16         return true
17     }
18     if n%2 == 0 || n%3 == 0 {
19         return false
20     }
21     // 只需要检查 6k ± 1 形式的数
22     for i := 5; i*i <= n; i += 6 {
23         if n%i == 0 || n%(i+2) == 0 {
24             return false
25         }
26     }
27     return true
28 }
29
30 // 判断是否为回文数
31 func isSymmetryNum(num int) bool {
32     s := strconv.Itoa(num)
33     i, j := 0, len(s)-1
34     for i < j {
35         if s[i] != s[j] {
36             return false
37         }
38         i++
39         j--
40     }
41     return true
42 }
43
44 func main() {
45     scanner := bufio.NewScanner(os.Stdin)
```

```
46     scanner.Scan()
47     n, _ := strconv.Atoi(scanner.Text())
48
49     for {
50         if isSymmetryNum(n) && isPrime(n) {
51             fmt.Println(n)
52             break
53         }
54         n++ // 尝试下一个数
55     }
56 }
```

华为OD机试2025C卷真题 – 生成回文素数

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 100分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

求出大于或等于 N 的最小回文素数。

如果一个数大于 1，且其因数只有 1 和它自身，那么这个数是素数。例如，2, 3, 5, 7, 11 以及 13 是素数。

如果一个数从左往右读与从右往左读是一样的，那么这个数是回文数，例如，12321 是回文数。

输入描述

一个数

输出描述

一个数

用例1

输入

```
1 6
```

Plain Text

输出

```
1 7
```

Plain Text

用例2

输入

```
Plain Text |  
1 8
```

输出

```
Plain Text |  
1 11
```

题解

思路： 模拟 题，

1. 从N开始枚举，每次枚举值 + 1，直到找到一个数既是素数又是回文数结束。
2. 回文数判断：基于回文数 从左往右读与从右往左读是一样的 特征，可以将数字转换为字符串，然后使用 双指针 进行判断。
3. 素数判断：
 - a. 常规验证方法：枚举除数范围[2,sqrt(n)]，判断n能否被其中一个数整除，如果存在则不是素数。
 - b. 基于数学原理：将所有数都表示成 $6k + m$ 其中 $6k + 0$, $6k + 2$, $6k + 3$, $6k + 4$ 形式的数肯定不是素数，对于[2,sqrt(n)]的被除数枚举范围可以缩小为[2,sqrt(n)]中符合 $6k+5$, $6k + 1$ 特征的数。
4. 根据 1 2 3 的逻辑执行，直到找到一个既是素数又是回文数的数字时结束循环，该数字即为结果。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 using namespace std;
11
12 // 判断是否为素数
13 bool isPrime(int n) {
14     if (n < 2) return false;
15     if (n == 2 || n == 3) return true;
16     if (n % 2 == 0 || n % 3 == 0) return false;
17     // 只需要检查6k + 1 和 6k - 1
18     for (int i = 5; i * i <= n; i += 6) {
19         if (n % i == 0 || n % (i + 2) == 0)
20             return false;
21     }
22     return true;
23 }
24
25 // 判断是否为回文数 双指针
26 bool isSymmetryNum(int num) {
27     string s = to_string(num);
28     int i = 0, j = s.size() - 1;
29     while (i < j) {
30         if (s[i] != s[j]) {
31             return false;
32         }
33         i++;
34         j--;
35     }
36     return true;
37 }
38
39
40 int main() {
41     long n;
42     cin >> n;
43     while (true) {
44         if (isSymmetryNum(n) && isPrime(n)) {
45             cout << n;
```

```
46         break;
47     }
48     // 尝试下一个数
49     n++;
50 }
51 return 0;
52 }
```

JAVA

```
1 import java.util.Scanner;
2
3 public class Main {
4
5     // 判断是否为素数
6     public static boolean isPrime(long n) {
7         if (n < 2) return false;
8         if (n == 2 || n == 3) return true;
9         if (n % 2 == 0 || n % 3 == 0) return false;
10        // 只需要检查 6k ± 1 形式的数
11        for (long i = 5; i * i <= n; i += 6) {
12            if (n % i == 0 || n % (i + 2) == 0)
13                return false;
14        }
15        return true;
16    }
17
18    // 判断是否为回文数
19    public static boolean isSymmetryNum(long num) {
20        String s = Long.toString(num);
21        int i = 0, j = s.length() - 1;
22        while (i < j) {
23            if (s.charAt(i) != s.charAt(j)) {
24                return false;
25            }
26            i++;
27            j--;
28        }
29        return true;
30    }
31
32    public static void main(String[] args) {
33        Scanner scanner = new Scanner(System.in);
34        long n = scanner.nextLong();
35        scanner.close();
36
37        while (true) {
38            if (isSymmetryNum(n) && isPrime(n)) {
39                System.out.println(n);
40                break;
41            }
42            n++; // 尝试下一个数
43        }
44    }
45}
```

Python

```
1 import sys
2
3 # 判断是否为素数
4 def is_prime(n: int) -> bool:
5     if n < 2:
6         return False
7     if n in (2, 3):
8         return True
9     if n % 2 == 0 or n % 3 == 0:
10        return False
11    # 只需要检查 6k ± 1 形式的数
12    i = 5
13    while i * i <= n:
14        if n % i == 0 or n % (i + 2) == 0:
15            return False
16        i += 6
17    return True
18
19 # 判断是否为回文数
20 def is_symmetry_num(num: int) -> bool:
21     return str(num) == str(num)[::-1]
22
23 def main():
24     n = int(sys.stdin.readline().strip())
25
26     while True:
27         if is_symmetry_num(n) and is_prime(n):
28             print(n)
29             break
30         n += 1 # 尝试下一个数
31
32 if __name__ == "__main__":
33     main()
```

Plain Text |

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 // 判断是否为素数
9 function isPrime(n) {
10     if (n < 2) return false;
11     if (n === 2 || n === 3) return true;
12     if (n % 2 === 0 || n % 3 === 0) return false;
13     // 只需要检查 6k ± 1 形式的数
14     for (let i = 5; i * i <= n; i += 6) {
15         if (n % i === 0 || n % (i + 2) === 0)
16             return false;
17     }
18     return true;
19 }
20
21 // 判断是否为回文数
22 function isSymmetryNum(num) {
23     const s = num.toString();
24     return s === s.split('').reverse().join('');
25 }
26
27 rl.on('line', (line) => {
28     let n = BigInt(line.trim()); // 使用 BigInt 处理大数
29
30     while (true) {
31         if (isSymmetryNum(n) && isPrime(Number(n))) {
32             console.log(n.toString());
33             break;
34         }
35         n++; // 尝试下一个数
36     }
37
38     rl.close();
39});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8 )
9
10 // 判断是否为素数
11 func isPrime(n int) bool {
12     if n < 2 {
13         return false
14     }
15     if n == 2 || n == 3 {
16         return true
17     }
18     if n%2 == 0 || n%3 == 0 {
19         return false
20     }
21     // 只需要检查 6k ± 1 形式的数
22     for i := 5; i*i <= n; i += 6 {
23         if n%i == 0 || n%(i+2) == 0 {
24             return false
25         }
26     }
27     return true
28 }
29
30 // 判断是否为回文数
31 func isSymmetryNum(num int) bool {
32     s := strconv.Itoa(num)
33     i, j := 0, len(s)-1
34     for i < j {
35         if s[i] != s[j] {
36             return false
37         }
38         i++
39         j--
40     }
41     return true
42 }
43
44 func main() {
45     scanner := bufio.NewScanner(os.Stdin)
```

```
46     scanner.Scan()
47     n, _ := strconv.Atoi(scanner.Text())
48
49     for {
50         if isSymmetryNum(n) && isPrime(n) {
51             fmt.Println(n)
52             break
53         }
54         n++ // 尝试下一个数
55     }
56 }
```

| 来自: 华为OD机试2025C卷 – 生成回文素数 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机试2025C卷 – 生成回文素数 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 战场索敌 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 战场索敌

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 100分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

有一个大小是 $N \times M$ 的战场地图，被墙壁‘#’分隔成大小不同的区域，上下左右四个方向相邻的空地‘.’属于同一个区域，只有空地上可能存在敌人‘E’，请求出地图上总共有多少区域里的敌人数小于 K 。

输入描述

第一行输入为 N, M, K ；

- N 表示地图的行数， M 表示地图的列数， K 表示目标敌人数量
- $N, M \leq 100$

之后为一个 $N \times M$ 大小的字符数组。

输出描述

敌人数小于 K 的区域数量

示例1

输入

```
1 3 5 2
2 ..#EE
3 E.#E.
4 ###..


Plain Text
```

输出

```
1 1


Plain Text
```

说明

| 地图被墙壁分为两个区域，左边区域有1个敌人，右边区域有3个敌人，符合条件的区域数量是1

题解

思路： **DFS/BFS**

1. 接收输入的战场地图，以及指定要求的K。
2. 接下来从上往下 从左往右顺序遍历输入的地图，当遇到 **E** 字符时使用 **DFS/BFS** 统计当前区域的敌人数量。如果敌人数量 $\leq k$,则满足要求区域 + 1. BFS访问过程中可以把访问过的 **E** 重新赋值为 **.** 来防止重复访问 (也可以定义visited数组来防止重复访问)。
3. 通过2逻辑，遍历完地图之后。输出满足要求的区域数量即可。

C++

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 int rowCount, colCount, enemyThreshold; // 地图的行数、列数和目标敌人数量
7 vector<vector<char>> grid; // 存储地图
8 vector<vector<int>> visited; // 访问标记
9 int currentEnemyCount; // 记录当前区域的敌人数
10
11 // 深度优先搜索，统计当前区域的敌人数
12 void exploreRegion(int x, int y) {
13     visited[x][y] = 1; // 标记当前格子已访问
14
15     if (grid[x][y] == 'E') {
16         currentEnemyCount++; // 统计敌人
17     }
18
19     // 上、下、左、右四个方向
20     vector<vector<int>> directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
21
22     for (const auto& dir : directions) {
23         int newX = x + dir[0];
24         int newY = y + dir[1];
25
26         // 检查边界条件，确保新位置在地图范围内，未访问，且不是墙壁
27         if (newX >= 0 && newX < rowCount && newY >= 0 && newY < colCount &
28             &
29             visited[newX][newY] == 0 && grid[newX][newY] != '#') {
30             exploreRegion(newX, newY); // 递归探索
31         }
32     }
33 }
34
35 int main() {
36     cin >> rowCount >> colCount >> enemyThreshold;
37
38     grid.resize(rowCount, vector<char>(colCount));
39     visited.resize(rowCount, vector<int>(colCount));
40
41     // 读取地图数据
42     for (int i = 0; i < rowCount; i++) {
43         string row;
44         cin >> row;
45         for (int j = 0; j < colCount; j++) {
```

```
45         grid[i][j] = row[j];
46     }
47 }
48
49 int validRegions = 0; // 记录符合条件的区域数量
50
51 // 遍历整个地图，查找未访问的区域
52 for (int i = 0; i < rowCount; i++) {
53     for (int j = 0; j < colCount; j++) {
54         if (visited[i][j] != 0 || grid[i][j] == '#') {
55             continue; // 跳过已访问的格子或墙壁
56         }
57         currentEnemyCount = 0; // 初始化敌人数
58         exploreRegion(i, j); // 进行深度优先搜索
59
60         // 如果当前区域的敌人数小于阈值，则计入有效区域
61         if (currentEnemyCount < enemyThreshold) {
62             validRegions++;
63         }
64     }
65 }
66
67 cout << validRegions << endl; // 输出符合条件的区域数量
68
69 return 0;
}
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     static int rowCount, colCount, enemyThreshold;
5     static char[][] grid;
6     static boolean[][] visited;
7     static int currentEnemyCount;
8
9     // 深度优先搜索，统计当前区域的敌人数
10    static void exploreRegion(int x, int y) {
11        visited[x][y] = true; // 标记当前格子已访问
12
13        if (grid[x][y] == 'E') {
14            currentEnemyCount++; // 统计敌人
15        }
16
17        // 上、下、左、右四个方向
18        int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
19
20        for (int[] dir : directions) {
21            int newX = x + dir[0];
22            int newY = y + dir[1];
23
24            // 确保新位置在地图范围内，未访问，且不是墙壁
25            if (newX >= 0 && newX < rowCount && newY >= 0 && newY < colCount &&
26                !visited[newX][newY] && grid[newX][newY] != '#') {
27                exploreRegion(newX, newY);
28            }
29        }
30    }
31
32    public static void main(String[] args) {
33        Scanner scanner = new Scanner(System.in);
34        rowCount = scanner.nextInt();
35        colCount = scanner.nextInt();
36        enemyThreshold = scanner.nextInt();
37        scanner.nextLine(); // 读取换行符
38
39        grid = new char[rowCount][colCount];
40        visited = new boolean[rowCount][colCount];
41
42        // 读取地图数据
43        for (int i = 0; i < rowCount; i++) {
44            String row = scanner.nextLine();
45        }
46    }
47}
```

```
45         grid[i] = row.toCharArray();
46     }
47
48     int validRegions = 0; // 记录符合条件的区域数量
49
50     // 遍历整个地图，查找未访问的区域
51     for (int i = 0; i < rowCount; i++) {
52         for (int j = 0; j < colCount; j++) {
53             if (!visited[i][j] && grid[i][j] != '#') {
54                 currentEnemyCount = 0; // 初始化敌人数
55                 exploreRegion(i, j); // 进行深度优先搜索
56
57                 if (currentEnemyCount < enemyThreshold) {
58                     validRegions++;
59                 }
60             }
61         }
62     }
63
64     System.out.println(validRegions); // 输出符合条件的区域数量
65 }
66 }
```

Python

```
1 import sys
2
3 # 读取地图的行数、列数和目标敌人数量
4 rowCount, colCount, enemyThreshold = map(int, sys.stdin.readline().split())
5
6 # 读取地图数据
7 grid = [list(sys.stdin.readline().strip()) for _ in range(rowCount)]
8 visited = [[False] * colCount for _ in range(rowCount)]
9
10 def explore_region(x, y):
11     """ 深度优先搜索，统计当前区域的敌人数 """
12     global current_enemy_count
13     visited[x][y] = True # 标记为已访问
14
15     if grid[x][y] == 'E':
16         current_enemy_count += 1 # 统计敌人
17
18     # 上、下、左、右四个方向
19     directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
20
21     for dx, dy in directions:
22         new_x, new_y = x + dx, y + dy
23
24         # 确保新位置在地图范围内，未访问，且不是墙壁
25         if 0 <= new_x < rowCount and 0 <= new_y < colCount and not visited[new_x][new_y] and grid[new_x][new_y] != '#':
26             explore_region(new_x, new_y)
27
28 valid_regions = 0
29
30 # 遍历整个地图，查找未访问的区域
31 for i in range(rowCount):
32     for j in range(colCount):
33         if not visited[i][j] and grid[i][j] != '#':
34             current_enemy_count = 0
35             explore_region(i, j)
36
37             if current_enemy_count < enemyThreshold:
38                 valid_regions += 1
39
40 print(valid_regions) # 输出符合条件的区域数量
```

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9 let rowCount, colCount, enemyThreshold;
10 let grid = [];
11 let visited = [];
12 let currentEnemyCount = 0;
13
14 // 读取输入
15 rl.on("line", (line) => {
16     inputLines.push(line);
17 }).on("close", () => {
18     [rowCount, colCount, enemyThreshold] = inputLines[0].split(" ").map(Number);
19
20     grid = inputLines.slice(1, rowCount + 1).map(row => row.split(""));
21     visited = Array.from({ length: rowCount }, () => Array(colCount).fill(false));
22     // DFS 统计区域敌人数量
23     function exploreRegion(x, y) {
24         visited[x][y] = true;
25
26         if (grid[x][y] === 'E') {
27             currentEnemyCount++;
28         }
29         // 四个方向偏移数组
30         const directions = [[-1, 0], [1, 0], [0, -1], [0, 1]];
31
32         for (const [dx, dy] of directions) {
33             let newX = x + dx, newY = y + dy;
34             // 不越界 且 未访问过 以及 不是墙
35             if (newX >= 0 && newX < rowCount && newY >= 0 && newY < colCount &&
36                 !visited[newX][newY] && grid[newX][newY] !== "#") {
37                 exploreRegion(newX, newY);
38             }
39         }
40     }
41
42     let validRegions = 0;
```

```
43
44     for (let i = 0; i < rowCount; i++) {
45         for (let j = 0; j < colCount; j++) {
46             if (!visited[i][j] && grid[i][j] !== "#") {
47                 currentEnemyCount = 0;
48                 exploreRegion(i, j);
49                 // 满足要求区域 + 1
50                 if (currentEnemyCount < enemyThreshold) {
51                     validRegions++;
52                 }
53             }
54         }
55     }
56
57     console.log(validRegions);
58 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 var rowCount, colCount, enemyThreshold int
10 var grid [][]byte
11 var visited [][]bool
12 var currentEnemyCount int
13
14 // 深度优先搜索
15 func exploreRegion(x, y int) {
16     visited[x][y] = true
17
18     if grid[x][y] == 'E' {
19         currentEnemyCount++
20     }
21     // 四个方向偏移数组
22     directions := []int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}
23
24     for _, dir := range directions {
25         newX, newY := x+dir[0], y+dir[1]
26             // 不越界 且 未访问过 以及 不是墙
27         if newX >= 0 && newX < rowCount && newY >= 0 && newY < colCount &&
28             !visited[newX][newY] && grid[newX][newY] != '#' {
29             exploreRegion(newX, newY)
30         }
31     }
32 }
33
34 func main() {
35     scanner := bufio.NewScanner(os.Stdin)
36     fmt.Scan(&rowCount, &colCount, &enemyThreshold)
37
38     grid = make([][]byte, rowCount)
39     visited = make([][]bool, rowCount)
40
41     for i := 0; i < rowCount; i++ {
42         scanner.Scan()
43         grid[i] = []byte(scanner.Text())
44         visited[i] = make([]bool, colCount)
45     }
}
```

```
46
47     validRegions := 0
48
49     for i := 0; i < rowCount; i++ {
50         for j := 0; j < colCount; j++ {
51             if !visited[i][j] && grid[i][j] != '#' {
52                 currentEnemyCount = 0
53                 exploreRegion(i, j)
54                     // 判断是否该区域是否满足要求
55                 if currentEnemyCount < enemyThreshold {
56                     validRegions++
57                 }
58             }
59         }
60     }
61
62     fmt.Println(validRegions)
63 }
```

| 来自: 华为OD机试2025C卷 – 战场索敌 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 字符串拼接 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 字符串拼接

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 200分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

给定 M ($0 < M \leq 30$) 个字符 (a-z) , 从中取出任意字符 (每个字符只能用一次) 拼接成长度为 N ($0 < N \leq 5$) 的字符串,

要求相同的字符不能相邻, 计算给出定的字符列表能拼接出多少种满足条件的字符串,
输入非法或者无法拼接出满足条件的字符串则返回0。

输入描述

给定的字符列表和结果字符串长度, 中间使用空格(" ")拼接

输出描述

满足条件的字符串个数

示例1

输入

Plain Text

```
1 aab 2
```

输出

Plain Text

```
1 2
```

说明

只能构成ab,ba。

示例2

输入

```
Plain Text |  
1 abc 2
```

输出

```
Plain Text |  
1 6
```

说明

| 可以构成: ab ac ba bc ca cb。

题解

思路: 由于题目的数据范围比较小, 直接采用 递归回溯 枚举所有组合情况就行。主要注意这两点:

1. 去重: 可以采用集合去进行去重。
2. 相同的字符不能相邻: 额外判断当前要加入字符和上一个选择字符是否相同即可。

| 具体思路可参照下面代码。

C++

```
1 #include <iostream>
2 #include <unordered_set>
3 #include <vector>
4 #include <sstream>
5
6 using namespace std;
7
8 // 递归生成不同的符合条件的字符串
9 void generateCombinations(const string& input, int targetLength, string cu
10 rrent, unordered_set<string>& results, vector<bool>& used) {
11     if (current.length() == targetLength) {
12         results.insert(current);
13         return;
14     }
15     for (int i = 0; i < input.length(); i++) {
16         if (used[i] || (!current.empty() && current.back() == input[i])) {
17             continue; // 跳过已使用或连续重复的字符
18         }
19         used[i] = true;
20         generateCombinations(input, targetLength, current + input[i], resu
21 lts, used);
22         // 回溯
23         used[i] = false;
24     }
25 }
26 // 计算符合条件的不同字符串数量
27 int countUniqueCombinations(const string& input, int targetLength) {
28     // 使用集合去重
29     unordered_set<string> uniqueCombinations;
30     // 标记字符是否已经使用
31     vector<bool> used(input.length(), false);
32     generateCombinations(input, targetLength, "", uniqueCombinations, use
33 d);
34     return uniqueCombinations.size();
35 }
36
37 int main() {
38     string inputLine;
39     getline(cin, inputLine);
40
41     string inputStr;
42     int targetLength;
43     istringstream iss(inputLine);
```

```
43     iss >> inputStr >> targetLength;
44
45     cout << countUniqueCombinations(inputStr, targetLength) << endl;
46
47     return 0;
48 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 递归生成不同的符合条件的字符串
5     private static void generateCombinations(String input, int targetLength, String current, Set<String> results, boolean[] used) {
6         if (current.length() == targetLength) {
7             results.add(current);
8             return;
9         }
10
11         for (int i = 0; i < input.length(); i++) {
12             if (used[i] || (!current.isEmpty() && current.charAt(current.length() - 1) == input.charAt(i))) {
13                 continue; // 跳过已使用或连续重复的字符
14             }
15             used[i] = true;
16             generateCombinations(input, targetLength, current + input.charAt(i), results, used);
17             used[i] = false; // 回溯
18         }
19     }
20
21     // 计算符合条件的不同字符串数量
22     private static int countUniqueCombinations(String input, int targetLength) {
23         // 使用集合去重
24         Set<String> uniqueCombinations = new HashSet<>();
25         // 标记字符是否已经使用
26         boolean[] used = new boolean[input.length()];
27         generateCombinations(input, targetLength, "", uniqueCombinations, used);
28         return uniqueCombinations.size();
29     }
30
31     public static void main(String[] args) {
32         Scanner scanner = new Scanner(System.in);
33         String inputStr = scanner.next();
34         int targetLength = scanner.nextInt();
35         scanner.close();
36
37         System.out.println(countUniqueCombinations(inputStr, targetLength));
38     }
39 }
```

Python

```
Plain Text |  
1 import sys  
2 # 递归生成不同的符合条件的字符串  
3 def generate_combinations(input_str, target_length, current, results, used):  
4     if len(current) == target_length:  
5         results.add(current)  
6         return  
7  
8     for i in range(len(input_str)):  
9         if used[i] or (current and current[-1] == input_str[i]):  
10             continue # 跳过已使用或连续重复的字符  
11         used[i] = True  
12         generate_combinations(input_str, target_length, current + input_st  
r[i], results, used)  
13         used[i] = False # 回溯  
14  
15 def count_unique_combinations(input_str, target_length):  
16     # 使用集合去重  
17     unique_combinations = set()  
18     # 标记该字符是否已经使用  
19     used = [False] * len(input_str)  
20     generate_combinations(input_str, target_length, "", unique_combination  
s, used)  
21     return len(unique_combinations)  
22  
23 if __name__ == "__main__":  
24     input_line = sys.stdin.read().strip().split()  
25     input_str, target_length = input_line[0], int(input_line[1])  
26     print(count_unique_combinations(input_str, target_length))
```

JavaScript

```
1 const readline = require("readline");
2
3 // 递归生成不同的符合条件的字符串
4 function generateCombinations(input, targetLength, current, results, used) {
5     if (current.length === targetLength) {
6         results.add(current);
7         return;
8     }
9
10    for (let i = 0; i < input.length; i++) {
11        if (used[i] || (current.length > 0 && current[current.length - 1]
12 === input[i])) {
13            continue; // 跳过已使用或连续重复的字符
14        }
15        used[i] = true;
16        generateCombinations(input, targetLength, current + input[i], results, used);
17        used[i] = false; // 回溯
18    }
19}
20
21 // 计算符合条件的不同字符串数量
22 function countUniqueCombinations(input, targetLength) {
23     // 使用集合去重
24     let uniqueCombinations = new Set();
25     // 标记字符是否已经使用
26     let used = new Array(input.length).fill(false);
27     generateCombinations(input, targetLength, "", uniqueCombinations, used);
28     return uniqueCombinations.size;
29}
30
31 // 读取输入并处理
32 const rl = readline.createInterface({
33     input: process.stdin,
34     output: process.stdout
35 });
36
37 rl.on("line", (line) => {
38     const [inputStr, targetLength] = line.split(" ");
39     console.log(countUniqueCombinations(inputStr, parseInt(targetLength)));
40     rl.close();
41});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // 递归生成不同的符合条件的字符串
11 func generateCombinations(input string, targetLength int, current string,
12 results map[string]bool, used []bool) {
13     if len(current) == targetLength {
14         results[current] = true
15         return
16     }
17
18     for i := 0; i < len(input); i++ {
19         if used[i] || (len(current) > 0 && current[len(current)-1] == input[i]) {
20             continue // 跳过已使用或连续重复的字符
21         }
22         used[i] = true
23         generateCombinations(input, targetLength, current+string(input[i]), re
24         sults, used)
25         used[i] = false // 回溯
26     }
27 }
28
29 // 计算符合条件的不同字符串数量
30 func countUniqueCombinations(input string, targetLength int) int {
31     // 使用集合去重
32     results := make(map[string]bool)
33     // 标记字符是否已经使用
34     used := make([]bool, len(input))
35     generateCombinations(input, targetLength, "", results, used)
36     return len(results)
37 }
38
39 func main() {
40     reader := bufio.NewReader(os.Stdin)
41     line, _ := reader.ReadString('\n')
42     parts := strings.Fields(line)
43     inputStr := parts[0]
```

```
43     var targetLength int
44     fmt.Sscanf(parts[1], "%d", &targetLength)
45
46     fmt.Println(countUniqueCombinations(inputStr, targetLength))
47 }
```

| 来自: 华为OD机试2025C卷 – 字符串拼接 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 攀登者2 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 攀登者2

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 200分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

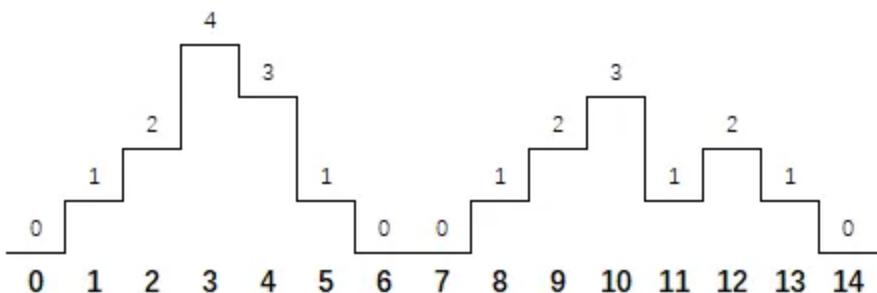
题目描述

攀登者喜欢寻找各种地图，并且尝试攀登到最高的山峰。

地图表示为一维数组，数组的索引代表水平位置，数组的元素代表相对海拔高度。其中数组元素0代表地面。

例如：[0,1,2,4,3,1,0,0,1,2,3,1,2,1,0]，代表如下图所示的地图，地图中有两个山脉位置分别为1,2,3,4,5和8,9,10,11,12,13，最高峰高度分别为4,3。最高峰位置分别为3,10。

一个山脉可能有多座山峰(高度大于相邻位置的高度，或在地图边界且高度大于相邻的高度)。



登山时会消耗登山者的体力(整数)，

- 上山时，消耗相邻高度差两倍的体力
- 下山时，消耗相邻高度差一倍的体力
- 平地不消耗体力

登山者体力消耗到零时会有生命危险。

例如，上图所示的山峰：

- 从索引0，走到索引1，高度差为1，需要消耗 $2 * 1 = 2$ 的体力，
- 从索引2，走到索引3，高度差为2，需要消耗 $2 * 2 = 4$ 的体力。
- 从索引3，走到索引4，高度差为1，需要消耗 $1 * 1 = 1$ 的体力。

攀登者想要评估一张地图内有多少座山峰可以进行攀登，且可以安全返回到地面，且无生命危险。

例如上图中的数组，有3个不同的山峰，登上位置在3的山可以从位置0或者位置6开始，从位置0登上山顶需要消耗体力 $1 * 2 + 1 * 2 + 2 * 2 = 8$ ，从山顶返回到地面0需要消耗体力 $2 * 1 + 1 * 1 + 1 * 1 = 4$ 的体力，按照登山路线 $0 \rightarrow 3 \rightarrow 0$ 需要消耗体力12。攀登者至少需要12以上的体力（大于12）才能安全返回。

输入描述

第一行输入为地图一维数组

第二行输入为攀登者的体力

输出描述

确保可以安全返回地面，且无生命危险的情况下，地图中有多少山峰可以攀登。

用例1

输入

```
Plain Text |  
1 0,1,4,3,1,0,0,1,2,3,1,2,1,0  
2 13
```

输出

```
Plain Text |  
1 3
```

说明

登山者可能登上所有山峰高度分别为 4 3 2，具体为 $0 \rightarrow 2 \rightarrow 0, 6 \rightarrow 9 \rightarrow 6, 13 \rightarrow 11 \rightarrow 13$

输入

```
Plain Text |  
1 1,4,3  
2 999
```

输出

▼

Plain Text |

1 0

说明

| 没有合适的起点和终点

题解

思路：模拟

- 首先找出哪些点是山峰位置，山峰位置的条件为 `(i == 0 || heights[i] > heights[i-1]) && (i == n-1 || heights[i] > heights[i+1])`
- 从左右至右遍历，分别记录 从当前点到左边第一个高度为0的位置所需的体力， 从左边第一个高度为0位置到当前点所需的体力，分别使用 `up`、`down` 数组存储。
- 可以反向遍历，类似于2的处理，去更新 `up[i]`, `down[i]` 的值取两种方向较小的值。
- 循环统计可以到达的山峰，可以到达的山峰满足的条件是 `peaks[i] && (up[i] + down[i] < strength)`

C++

```
1 #include <cmath>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 #include<list>
9 #include<queue>
10 #include<map>
11 #include<set>
12 #include<climits>
13 using namespace std;
14
15
16 // 通用 split 函数
17 vector<string> split(const string& str, const string& delimiter) {
18     vector<string> result;
19     size_t start = 0;
20     size_t end = str.find(delimiter);
21     while (end != string::npos) {
22         result.push_back(str.substr(start, end - start));
23         start = end + delimiter.length();
24         end = str.find(delimiter, start);
25     }
26     // 添加最后一个部分
27     result.push_back(str.substr(start));
28     return result;
29 }
30 //确定山峰位置
31 vector<bool> findPeaks(vector<int>& heights) {
32     int n = heights.size();
33     vector<bool> peaks(n, false);
34     for (int i = 0; i < n; i++) {
35         // 山峰位置判断条件
36         if ((i == 0 || heights[i] > heights[i-1]) && (i == n-1 || heights
37 [i] > heights[i+1])) {
38             peaks[i] = true;
39         }
40     }
41     return peaks;
42 }
43 int solve(vector<int>& heights, int strength) {
44     int n = heights.size();
```

```
45 // 确定山峰位置
46 vector<bool> peaks = findPeaks(heights);
47 // 登上某处所需要花费最小体力
48 vector<int> up(n, INT_MAX / 2 - 1);
49 // 从某处到位置为0所需要花费最小体力
50 vector<int> down(n, INT_MAX / 2 - 1);
51
52 int tempUp = 0;
53 int tempDown = 0;
54 int upFrom = -1;
55 for (int i = 0; i < n; i++) {
56     if (heights[i] == 0) {
57         tempDown = 0;
58         tempUp = 0;
59         upFrom = i;
60         continue;
61     }
62     // 说明前面没有出发点
63     if (upFrom == -1) {
64         continue;
65     }
66
67     int diff = heights[i] - heights[i-1];
68     if (diff > 0) {
69         tempUp += 2 * abs(diff);
70         tempDown += abs(diff);
71     } else {
72         tempUp += abs(diff);
73         tempDown += 2 * abs(diff);
74     }
75
76     up[i] = tempUp;
77     down[i] = tempDown;
78 }
79
80 tempUp = 0;
81 tempDown = 0;
82 upFrom = n;
83 // 反向登山
84 for (int i = n - 1; i >= 0; --i) {
85     if (heights[i] == 0) {
86         upFrom = i;
87         tempDown = 0;
88         tempUp = 0;
89         continue;
90     }
91     // 没有出发点
92     if (upFrom == n) continue;
```

```

93
94     int diff = heights[i] - heights[i + 1];
95     if (diff > 0) {
96         tempUp += 2 * abs(diff);
97         tempDown += abs(diff);
98     }
99     else {
100         tempUp += abs(diff);
101         tempDown += 2 * abs(diff);
102     }
103     up[i] = min(up[i], tempUp);
104     down[i] = min(down[i], tempDown);
105 }
106
107 int cnt = 0;
108 for (int i = 0; i < n; i++) {
109     // 是山峰并且可以安全返回
110     if (peaks[i] && (up[i] + down[i] < strength)) {
111         cnt += 1;
112     }
113 }
114 return cnt;
115 }
116
117
118 int main() {
119     string s;
120     getline(cin, s);
121     vector<string> ans = split(s, ",");
122     int n = ans.size();
123     vector<int> heights(n);
124     for (int i = 0; i < n; i++) {
125         heights[i] = stoi(ans[i]);
126     }
127     int strength;
128     cin >> strength;
129     int res = solve(heights, strength);
130     cout << res;
131     return 0;
132 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 将字符串按指定分隔符拆分为整数数组
5     public static List<Integer> splitString(String s, String delimiter) {
6         String[] parts = s.split(delimiter);
7         List<Integer> result = new ArrayList<>();
8         for (String part : parts) {
9             result.add(Integer.parseInt(part));
10        }
11    return result;
12 }
13
14 // 确定哪些位置是山峰
15 public static boolean[] findPeaks(List<Integer> heights) {
16     int n = heights.size();
17     boolean[] peaks = new boolean[n];
18     for (int i = 0; i < n; i++) {
19         if ((i == 0 || heights.get(i) > heights.get(i - 1)) &&
20             (i == n - 1 || heights.get(i) > heights.get(i + 1))) {
21             peaks[i] = true;
22         }
23     }
24     return peaks;
25 }
26
27 // 计算可以安全返回的山峰数量
28 public static int solve(List<Integer> heights, int strength) {
29     int n = heights.size();
30     boolean[] peaks = findPeaks(heights);
31
32     int[] up = new int[n];
33     int[] down = new int[n];
34     Arrays.fill(up, Integer.MAX_VALUE / 2);
35     Arrays.fill(down, Integer.MAX_VALUE / 2);
36
37     int tempUp = 0, tempDown = 0, upFrom = -1;
38
39     // 从左向右遍历
40     for (int i = 0; i < n; i++) {
41         if (heights.get(i) == 0) {
42             tempUp = tempDown = 0;
43             upFrom = i;
44             continue;
45         }
46     }
47 }
```

```

46         if (upFrom == -1) continue;
47
48         int diff = heights.get(i) - heights.get(i - 1);
49         if (diff > 0) {
50             tempUp += 2 * diff;
51             tempDown += diff;
52         } else {
53             tempUp += -diff;
54             tempDown += -2 * diff;
55         }
56
57         up[i] = tempUp;
58         down[i] = tempDown;
59     }
60
61     // 从右向左遍历
62     tempUp = tempDown = 0;
63     upFrom = n;
64     for (int i = n - 1; i >= 0; i--) {
65         if (heights.get(i) == 0) {
66             upFrom = i;
67             tempUp = tempDown = 0;
68             continue;
69         }
70         if (upFrom == n) continue;
71
72         int diff = heights.get(i) - heights.get(i + 1);
73         if (diff > 0) {
74             tempUp += 2 * diff;
75             tempDown += diff;
76         } else {
77             tempUp += -diff;
78             tempDown += -2 * diff;
79         }
80
81         up[i] = Math.min(up[i], tempUp);
82         down[i] = Math.min(down[i], tempDown);
83     }
84
85     // 计算可返回的山峰数量
86     int count = 0;
87     for (int i = 0; i < n; i++) {
88         if (peaks[i] && (up[i] + down[i] < strength)) {
89             count++;
90         }
91     }
92     return count;
93 }
```

```
94
95     public static void main(String[] args) {
96         Scanner scanner = new Scanner(System.in);
97
98         // 读取高度列表
99         String line = scanner.nextLine();
100        List<Integer> heights = splitString(line, ",");
101
102        // 读取体力值
103        int strength = scanner.nextInt();
104
105        // 输出结果
106        System.out.println(solve(heights, strength));
107    }
108 }
```

Python

```
1 import sys
2
3 # 将字符串按指定分隔符拆分为整数数组
4 def split_string(s, delimiter):
5     return list(map(int, s.split(delimiter)))
6
7 # 确定哪些位置是山峰
8 def find_peaks(heights):
9     n = len(heights)
10    peaks = [False] * n
11    for i in range(n):
12        if (i == 0 or heights[i] > heights[i - 1]) and (i == n - 1 or heights[i] > heights[i + 1]):
13            peaks[i] = True
14    return peaks
15
16 # 计算可以安全返回的山峰数量
17 def solve(heights, strength):
18     n = len(heights)
19     peaks = find_peaks(heights)
20
21     up = [float('inf')] * n
22     down = [float('inf')] * n
23
24     temp_up = temp_down = 0
25     up_from = -1
26
27     # 从左向右遍历
28     for i in range(n):
29         if heights[i] == 0:
30             temp_up = temp_down = 0
31             up_from = i
32             continue
33         if up_from == -1:
34             continue
35
36         diff = heights[i] - heights[i - 1]
37         temp_up += 2 * diff if diff > 0 else -diff
38         temp_down += diff if diff > 0 else -2 * diff
39
40         up[i] = temp_up
41         down[i] = temp_down
42
43     # 从右向左遍历
44     temp_up = temp_down = 0
```

```
45     up_from = n
46     for i in range(n - 1, -1, -1):
47         if heights[i] == 0:
48             up_from = i
49             temp_up = temp_down = 0
50             continue
51         if up_from == n:
52             continue
53
54         diff = heights[i] - heights[i + 1]
55         temp_up += 2 * diff if diff > 0 else -diff
56         temp_down += diff if diff > 0 else -2 * diff
57
58         up[i] = min(up[i], temp_up)
59         down[i] = min(down[i], temp_down)
60
61     return sum(1 for i in range(n) if peaks[i] and (up[i] + down[i] < strength))
62
63 # 读取输入并计算结果
64 heights = split_string(sys.stdin.readline().strip(), ",")
65 strength = int(sys.stdin.readline().strip())
66 print(solve(heights, strength))
```

JavaScript

```
1 const readline = require('readline');
2
3 // 创建 `readline` 接口
4 const rl = readline.createInterface({
5     input: process.stdin,
6     output: process.stdout
7 });
8
9 let inputLines = [];
10
11 // 监听每行输入
12 rl.on('line', (line) => {
13     inputLines.push(line);
14 });
15
16 // 监听输入结束
17 rl.on('close', () => {
18     // 解析输入数据
19     const heights = inputLines[0].split(",").map(Number); // 解析第一行高度
        数据
20     const strength = parseInt(inputLines[1]); // 解析第二行体力值
21
22     console.log(solve(heights, strength)); // 计算并输出结果
23 });
24
25 /**
26 * 确定哪些位置是山峰
27 * @param {number[]} heights - 高度数组
28 * @returns {boolean[]} - 山峰位置的布尔数组
29 */
30 function findPeaks(heights) {
31     return heights.map((h, i) =>
32         (i === 0 || h > heights[i - 1]) &&
33         (i === heights.length - 1 || h > heights[i + 1])
34     );
35 }
36
37 /**
38 * 计算可以安全返回的山峰数量
39 * @param {number[]} heights - 高度数组
40 * @param {number} strength - 体力值
41 * @returns {number} - 可返回的山峰数量
42 */
43 function solve(heights, strength) {
44     const n = heights.length;
```

```

45     const peaks = findPeaks(heights);
46
47     let up = Array(n).fill(Infinity);
48     let down = Array(n).fill(Infinity);
49
50     let tempUp = 0, tempDown = 0, upFrom = -1;
51
52     // 从左向右遍历
53     for (let i = 0; i < n; i++) {
54         if (heights[i] === 0) {
55             tempUp = tempDown = 0;
56             upFrom = i;
57             continue;
58         }
59         if (upFrom === -1) continue;
60
61         let diff = heights[i] - heights[i - 1];
62         tempUp += diff > 0 ? 2 * diff : -diff;
63         tempDown += diff > 0 ? diff : -2 * diff;
64
65         up[i] = tempUp;
66         down[i] = tempDown;
67     }
68
69     // 从右向左遍历
70     tempUp = tempDown = 0;
71     upFrom = n;
72     for (let i = n - 1; i >= 0; i--) {
73         if (heights[i] === 0) {
74             upFrom = i;
75             tempUp = tempDown = 0;
76             continue;
77         }
78         if (upFrom === n) continue;
79
80         let diff = heights[i] - heights[i + 1];
81         tempUp += diff > 0 ? 2 * diff : -diff;
82         tempDown += diff > 0 ? diff : -2 * diff;
83
84         up[i] = Math.min(up[i], tempUp);
85         down[i] = Math.min(down[i], tempDown);
86     }
87
88     // 计算可返回的山峰数量
89     return peaks.filter((p, i) => p && (up[i] + down[i] < strength)).length
90     h;
}

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // splitString 将字符串按给定分隔符拆分并转换为整数数组
12 func splitString(s, delimiter string) []int {
13     parts := strings.Split(s, delimiter)
14     result := make([]int, len(parts))
15     for i, part := range parts {
16         num, _ := strconv.Atoi(part)
17         result[i] = num
18     }
19     return result
20 }
21
22 // findPeaks 确定哪些位置是山峰
23 func findPeaks(heights []int) []bool {
24     n := len(heights)
25     peaks := make([]bool, n)
26     for i := 0; i < n; i++ {
27         // 山峰的判断条件
28         if (i == 0 || heights[i] > heights[i-1]) && (i == n-1 || heights[i] > heights[i+1]) {
29             peaks[i] = true
30         }
31     }
32     return peaks
33 }
34
35 // solve 计算可以安全返回的山峰数量
36 func solve(heights []int, strength int) int {
37     n := len(heights)
38     // 确定山峰位置
39     peaks := findPeaks(heights)
40
41     // 记录到达某点的最小体力
42     up := make([]int, n)
43     down := make([]int, n)
44     inf := int(^uint(0) >> 1) // 最大整数
```

```
45     for i := range up {
46         up[i] = inf
47         down[i] = inf
48     }
49
50     // 变量用于存储当前爬升和下降所需的体力
51     tempUp, tempDown, upFrom := 0, 0, -1
52
53     // 从左到右遍历
54     for i := 0; i < n; i++ {
55         if heights[i] == 0 {
56             tempUp, tempDown = 0, 0
57             upFrom = i
58             continue
59         }
60         if upFrom == -1 {
61             continue
62         }
63
64         diff := heights[i] - heights[i-1]
65         if diff > 0 {
66             tempUp += 2 * diff
67             tempDown += diff
68         } else {
69             tempUp += -diff
70             tempDown += -2 * diff
71         }
72
73         up[i] = tempUp
74         down[i] = tempDown
75     }
76
77     // 从右到左遍历
78     tempUp, tempDown, upFrom = 0, 0, n
79     for i := n - 1; i >= 0; i-- {
80         if heights[i] == 0 {
81             upFrom = i
82             tempUp, tempDown = 0, 0
83             continue
84         }
85         if upFrom == n {
86             continue
87         }
88
89         diff := heights[i] - heights[i+1]
90         if diff > 0 {
91             tempUp += 2 * diff
92             tempDown += diff
93         }
94     }
95 }
```

```
93     } else {
94         tempUp += -diff
95         tempDown += -2 * diff
96     }
97
98     if tempUp < up[i] {
99         up[i] = tempUp
100    }
101    if tempDown < down[i] {
102        down[i] = tempDown
103    }
104}
105
106 // 计算满足条件的山峰数量
107 count := 0
108 for i := 0; i < n; i++ {
109     if peaks[i] && up[i]+down[i] < strength {
110         count++
111     }
112 }
113 return count
114}
115
116 func main() {
117     // 读取输入
118     scanner := bufio.NewScanner(os.Stdin)
119
120     // 读取第一行，解析高度列表
121     scanner.Scan()
122     heights := splitString(scanner.Text(), ",")
123
124     // 读取第二行，解析体力值
125     scanner.Scan()
126     strength, _ := strconv.Atoi(scanner.Text())
127
128     // 计算并输出结果
129     fmt.Println(solve(heights, strength))
130 }
```

| 来自: 华为OD机试2025C卷 – 攀登者2 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 高效的任务规划 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷真题 – 高效的任务规划

2025华为OD机试2025C卷 – 华为OD上机考试2025年C卷 200分题型

华为OD机试2025C卷真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

题目描述

你有 n 台机器，编号为 $1 \sim n$ ，每台都需要完成一项工作，机器经过配置后都能独立完成一项工作。

假设第 i 台机器你需要花 B_i 分钟进行设置，然后开始运行， J_i 分钟后完成任务。

现在，你需要选择布置工作的顺序，使得用最短的时间完成所有工作。

注意，不能同时对两台进行配置，但配置完成的机器们可以同时执行他们各自的工作。

输入描述

第一行输入代表总共有 M 组任务数据 ($1 \leq M \leq 10$)。

每组数第一行为一个整数指定机器的数量 N ($0 < N \leq 1000$)。随后的 N 行每行两个整数，第一个表示 B ($0 \leq B \leq 10000$)，第二个表示 J ($0 \leq J \leq 10000$)。

每组数据连续输入，不会用空行分隔。各组任务单独计时。

输出描述

对于每组任务，输出最短完成时间，且每组的结果独占一行。例如，两组任务就应该有两行输出。

用例1

输入

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

输出

Plain Text |

```
1 4
```

说明

输入共3行数据，
第一行代表只有1组任务；
第二行代表本组任务只有1台机器；
第三行代表本机器：配置需要2分钟，执行任务需要2分钟，
输出共一行数据，代表执行结果为4分钟

用例2

输入

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

Plain Text |

输出

```
1 4  
2 7
```

Plain Text |

说明

第1行 2代表有2组任务；
2~4行代表第1组数据，为2台机器的配置、执行信息
第1台：配置1分钟，执行1分钟
第2台：配置2分钟，执行2分钟
5~8行代表第2组数据，为3台机器的配置、执行信息
第1台：配置1分钟，执行1分钟

第2台：配置2分钟，执行2分钟

第3台：配置3分钟，执行3分钟

输出共2行，分别代表第1组任务共需要4分钟和第2组任务共需要7分钟

先配置3，再配置2，再配置1，执行1分钟，共7分钟

题解

思路：贪心 题

- 贪心规律：让运行时间长的任务优先进行配置。
- 最大为用时 = $\max(\text{“每个机器的配置结束时间} + \text{该机器的运行时间”})$

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 #include<set>
11 using namespace std;
12
13 // 按照运行时间进行降序排序
14 bool cmp(pair<int, int> a, pair<int, int> b) {
15     return a.second > b.second;
16 }
17
18
19 int getMinTime(vector<pair<int, int>> ans) {
20     int res = 0;
21     // 贪心 优先让执行时间长的任务先配置
22     sort(ans.begin(), ans.end(), cmp);
23
24
25     // 记录上个任务配置完后的时间
26     int lastConfigEndTime = 0;
27     for (int i = 0; i < ans.size(); i++) {
28         int configTime = ans[i].first;
29         int runTime = ans[i].second;
30         lastConfigEndTime += configTime;
31         // 获取最大执行结束时间
32         res = max(res, lastConfigEndTime + runTime);
33     }
34
35     return res;
36 }
37
38
39 int main() {
40     int m;
41     cin >> m;
42
43     for (int i = 0; i < m; i++) {
44         int n;
45         cin >> n;
```

```
46     vector<pair<int, int>> ans(n);
47     for (int j = 0; j < n; j++) {
48         int b, a;
49         cin >> b >> a;
50         ans[j] = {b, a};
51     }
52
53     int res = getMinTime(ans);
54     cout << res << endl;
55 }
56 return 0;
57 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4
5     // 按照运行时间降序排序
6     static class Task {
7         int configTime, runTime;
8         Task(int configTime, int runTime) {
9             this.configTime = configTime;
10            this.runTime = runTime;
11        }
12    }
13
14    public static int getMinTime(List<Task> tasks) {
15        // 贪心策略，运行时间长的任务先执行
16        tasks.sort((a, b) -> b.runTime - a.runTime);
17
18        int res = 0;
19        int lastConfigEndTime = 0;
20        for (Task task : tasks) {
21            lastConfigEndTime += task.configTime;
22            res = Math.max(res, lastConfigEndTime + task.runTime);
23        }
24
25        return res;
26    }
27
28    public static void main(String[] args) {
29        Scanner sc = new Scanner(System.in);
30        int m = sc.nextInt(); // 测试组数
31
32        for (int i = 0; i < m; i++) {
33            int n = sc.nextInt(); // 每组任务数
34            List<Task> tasks = new ArrayList<>();
35            for (int j = 0; j < n; j++) {
36                int b = sc.nextInt(); // 配置时间
37                int a = sc.nextInt(); // 运行时间
38                tasks.add(new Task(b, a));
39            }
40            int res = getMinTime(tasks);
41            System.out.println(res);
42        }
43    }
44}
```

Python

Plain Text |

```
1 # -*- coding: utf-8 -*-
2 def get_min_time(tasks):
3     # 贪心: 运行时间长的任务优先执行
4     tasks.sort(key=lambda x: -x[1])
5     res = 0
6     last_config_end_time = 0
7     for config_time, run_time in tasks:
8         last_config_end_time += config_time
9         res = max(res, last_config_end_time + run_time)
10    return res
11
12 m = int(input())
13 for _ in range(m):
14     n = int(input())
15     tasks = [tuple(map(int, input().split())) for _ in range(n)]
16     res = get_min_time(tasks)
17     print(res)
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9 rl.on('line', line => {
10     inputLines.push(line.trim());
11 });
12
13 rl.on('close', () => {
14     let index = 0;
15     const m = parseInt(inputLines[index++]);
16
17     for (let i = 0; i < m; i++) {
18         const n = parseInt(inputLines[index++]);
19         const tasks = [];
20         for (let j = 0; j < n; j++) {
21             const [b, a] = inputLines[index++].split(' ').map(Number);
22             tasks.push([b, a]);
23         }
24
25         // 贪心策略：运行时间长的任务先执行
26         tasks.sort((x, y) => y[1] - x[1]);
27
28         let res = 0;
29         let lastConfigEndTime = 0;
30         for (const [configTime, runTime] of tasks) {
31             lastConfigEndTime += configTime;
32             res = Math.max(res, lastConfigEndTime + runTime);
33         }
34
35         console.log(res);
36     }
37 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10    )
11
12 type Task struct {
13     configTime int
14     runTime     int
15 }
16
17 func getMinTime(tasks []Task) int {
18     // 贪心：运行时间长的优先
19     sort.Slice(tasks, func(i, j int) bool {
20         return tasks[i].runTime > tasks[j].runTime
21     })
22
23     res := 0
24     lastConfigEndTime := 0
25     for _, task := range tasks {
26         lastConfigEndTime += task.configTime
27         if lastConfigEndTime+task.runTime > res {
28             res = lastConfigEndTime + task.runTime
29         }
30     }
31     return res
32 }
33
34 func main() {
35     reader := bufio.NewReader(os.Stdin)
36     line, _ := reader.ReadString('\n')
37     m, _ := strconv.Atoi(strings.TrimSpace(line))
38
39     for i := 0; i < m; i++ {
40         line, _ = reader.ReadString('\n')
41         n, _ := strconv.Atoi(strings.TrimSpace(line))
42
43         tasks := make([]Task, n)
44         for j := 0; j < n; j++ {
45             line, _ = reader.ReadString('\n')
```

```
46     parts := strings.Fields(line)
47     b, _ := strconv.Atoi(parts[0])
48     a, _ := strconv.Atoi(parts[1])
49     tasks[j] = Task{configTime: b, runTime: a}
50 }
51
52     res := getMinTime(tasks)
53     fmt.Println(res)
54 }
55 }
```

| 来自: 华为OD机试2025C卷 – 高效的任务规划 (C++ & Python & JAVA & JS & GO)-CSDN博客