t0721

华为OD机考2025C卷 - 推荐多样性 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025C卷 - 智能驾驶 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD上机考试(2025年C卷) - 电脑病毒感染 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 对称美学 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 围棋的气 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 最小矩阵宽度 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 小朋友来自多少个小区 / 最少有多少个小朋友 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 最富裕的小家庭 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 推荐多样性 (C++ & Python & JAVA & JS & GO)-CSDN博客

推荐多样性

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

推荐多样性需要从多个列表中选择元素，一次性要返回 N 屏数据（窗口数量），每屏展示 K 个元素（窗口大小），选择策略：

1. 各个列表元素需要做穿插处理，即先从第一个列表中为每屏选择一个元素，再从第二个列表中为每屏选择一个元素，依次类推
2. 每个列表的元素尽量均分为 N 份，如果不够 N 个，也要全部分配完，参考样例图：
    a. 从第一个列表中选择 4 条 0 1 2 3，分别放到 4 个窗口中
    b. 从第二个列表中选择 4 条 10 11 12 13，分别放到 4 个窗口中
    c. 从第三个列表中选择 4 条 20 21 22 23，分别放到 4 个窗口中
    d. 再从第一个列表中选择 4 条 4 5 6 7，分别放到 4 个窗口中
    e. 再从第一个列表中选择，由于数量不足 4 条，取剩下的 2 条，放到 窗口1 和 窗口2
    f. 再从第二个列表中选择，由于数量不足 4 条并且总的元素数达到窗口要求，取 18 19 放到 窗口3 和 窗口4

输入

N：4
K：7

recallA	0	1	2	3	4	5	6	7	8	9
recallB	10	11	12	13	14	15	16	17	18	19
recallC	20	21	22	23	24	25	26	27	28	29

处理

window1	window2	window3	window4
0	1	2	3
10	11	12	13
20	21	22	23
4	5	6	7
14	15	16	17
24	25	26	27
8	9	18	19

输出

0 10 20 4 14 24 8 1 11 21 5 15 25 9 2 12 22 6 16 26 18 3 13 23 7 17 27 19

window1 window2 window3 window4

输入描述

第一行输入为 N，表示需要输出的窗口数量，取值范围 [1, 10]

第二行输入为 K，表示每个窗口需要的元素数量，取值范围 [1, 100]

之后的行数不定（行数取值范围 [1, 10]），表示每个列表输出的元素列表。元素之间以空格隔开，已经过排序处理，每个列表输出的元素数量取值范围 [1, 100]

输出描述

输出元素列表，元素数量 = 窗口数量 * 窗口大小，元素之间以空格分隔，多个窗口合并为一个列表输出，参考样例：

先输出窗口1的元素列表，再输出窗口2的元素列表，再输出窗口3的元素列表，最后输出窗口4的元素列表

备注

1. 每个列表会保证元素数量满足窗口要求，不需要考虑元素不足情况
2. 每个列表的元素已去重，不需要考虑元素重复情况
3. 每个列表的元素列表均不为空，不需要考虑列表为空的情况
4. 每个列表的元素列表已经过排序处理，输出结果要保证不改变同一个列表的元素顺序
5. 每个列表的元素数量可能是不同的

示例1

输入

```
4
7
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
```

输出

```
0 10 20 4 14 24 8 1 11 21 5 15 25 9 2 12 22 6 16 26 18 3 13 23 7 17 27 19
```

题解

思路： `纯模拟的题` 。注意以下几个点：

- 后期可能存在列表元素不足。可以采用一维数组去规避这个问题。
- 一次列表最多只能取出N个元素，不管上个列表是否没有填充N。

C++

Plain Text

```
#include <iostream>
#include <vector>
#include <queue>
#include <sstream>
#include <string>
#include <algorithm> // 使用 remove_if

using namespace std;

int main() {
    int numberOfRows, numberOfColumns;
    cin >> numberOfRows >> numberOfColumns;
    cin.ignore(); // 忽略换行符

    vector<queue<int>> queueList;
    string inputLine;

    // 读取输入，每行数据存入队列
    while (getline(cin, inputLine) && !inputLine.empty()) {
        istringstream iss(inputLine);
        queue<int> numberQueue;
        int number;
        while (iss >> number) {
            numberQueue.push(number);
        }
        queueList.push_back(numberQueue);
    }

    vector<int> matrix(numberOfColumns * numberOfRows, 0);
    int matrixIndex = 0;
    int queueIndex = 0;

    // 处理矩阵填充
    while (matrixIndex < matrix.size()) {
        // 每个列表每次最多选取numberOfRows个元素
        for (int i = 0; i < numberOfRows; ++i) {
            if (!queueList[queueIndex].empty()) {
                matrix[matrixIndex++] = queueList[queueIndex].front();
                queueList[queueIndex].pop();
                if (matrixIndex >= matrix.size()) break;
            } else {
                break;
            }

        }
```

```
            // 移动到下一个队列
            queueIndex = (queueIndex + 1) % queueList.size();

        }

        // 按列优先顺序输出矩阵
        for (int row = 0; row < numberOfRows; ++row) {
            for (int col = 0; col < numberOfColumns; ++col) {
                cout << matrix[col * numberOfRows + row] << " ";
            }
        }

        return 0;
    }
```

Java

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 读取矩阵行数和列数
        int numberOfRows = scanner.nextInt();
        int numberOfColumns = scanner.nextInt();
        scanner.nextLine(); // 读取换行符

        List<Queue<Integer>> queueList = new ArrayList<>();

        // 读取输入，每行存入队列
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine().trim();
            if (line.isEmpty()) break;

            Queue<Integer> numberQueue = new LinkedList<>();
            String[] numbers = line.split("\\s+");
            for (String num : numbers) {
                numberQueue.add(Integer.parseInt(num));
            }
            queueList.add(numberQueue);
        }

        int[] matrix = new int[numberOfColumns * numberOfRows];
        int matrixIndex = 0;
        int queueIndex = 0;

        // 处理矩阵填充
        while (matrixIndex < matrix.length) {
            for (int i = 0; i < numberOfRows; i++) {
                if (!queueList.get(queueIndex).isEmpty()) {
                    matrix[matrixIndex++] = queueList.get(queueIndex).poll
();
                    if (matrixIndex >= matrix.length) break;
                } else {
                    break;
                }
            }
            queueIndex = (queueIndex + 1) % queueList.size();
        }

        // 按列优先顺序输出矩阵
```

```
            for (int row = 0; row < numberOfRows; row++) {
                for (int col = 0; col < numberOfColumns; col++) {
                    System.out.print(matrix[col * numberOfRows + row] + " ");
                }
            }
        }
    }
```

Python

Plain Text

```
import sys
from collections import deque

# 读取列数和行数
number_of_rows = int(sys.stdin.readline().strip())
number_of_columns = int(sys.stdin.readline().strip())


queue_list = []

# 读取数据，每行存入队列
for line in sys.stdin:
    line = line.strip()
    if not line:
        break
    queue_list.append(deque(map(int, line.split())))

matrix = [0] * (number_of_columns * number_of_rows)
matrix_index = 0
queue_index = 0

# 处理矩阵填充
while matrix_index < len(matrix):
    for _ in range(number_of_rows):
        if queue_list[queue_index]:
            matrix[matrix_index] = queue_list[queue_index].popleft()
            matrix_index += 1
            if matrix_index >= len(matrix):
                break
        else:
            break
    queue_index = (queue_index + 1) % len(queue_list)

# 按列优先顺序输出矩阵
for row in range(number_of_rows):
    for col in range(number_of_columns):
        print(matrix[col * number_of_rows + row], end=" ")
```

JavaScript

Plain Text

```
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let numberOfColumns, numberOfRows;
let queueList = [];
let inputLines = [];
let readingMetadata = true;

rl.on("line", (line) => {
    if (readingMetadata) {
        inputLines.push(line.trim());
        if (inputLines.length === 2) {
            numberOfColumns = parseInt(inputLines[1]);
            numberOfRows = parseInt(inputLines[0]);
            readingMetadata = false;
        }
    } else {
        if (line.trim() === "") {
            rl.close();
        } else {
            queueList.push(line.split(" ").map(Number));
        }
    }
});

rl.on("close", () => {
    let matrix = new Array(numberOfColumns * numberOfRows).fill(0);
    let matrixIndex = 0;
    let queueIndex = 0;

    // 处理矩阵填充
    while (matrixIndex < matrix.length) {
        for (let i = 0; i < numberOfRows; i++) {
            if (queueList[queueIndex] && queueList[queueIndex].length > 0) {
                matrix[matrixIndex++] = queueList[queueIndex].shift();
                if (matrixIndex >= matrix.length) break;
            } else {
                break;
            }
        }
        queueIndex = (queueIndex + 1) % queueList.length;
    }
```

```
45
46      // 按列优先顺序输出矩阵
47      let output = [];
48      for (let row = 0; row < numberOfRows; row++) {
49          for (let col = 0; col < numberOfColumns; col++) {
50              output.push(matrix[col * numberOfRows + row]);
51          }
52      }
53      console.log(output.join(" "));
54  });
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

func main() {
  reader := bufio.NewReader(os.Stdin)

  // 读取列数和行数
  rowLine, _ := reader.ReadString('\n')
  numberOfRows, _ := strconv.Atoi(strings.TrimSpace(rowLine))


  colLine, _ := reader.ReadString('\n')
  numberOfColumns, _ := strconv.Atoi(strings.TrimSpace(colLine))


  var queueList [][]int
  scanner := bufio.NewScanner(reader)

  // 读取数据
  for scanner.Scan() {
    line := strings.TrimSpace(scanner.Text())
    if line == "" {
      break
    }
    strNumbers := strings.Split(line, " ")
    var numbers []int
    for _, num := range strNumbers {
      val, _ := strconv.Atoi(num)
      numbers = append(numbers, val)
    }
    queueList = append(queueList, numbers)
  }

  matrix := make([]int, numberOfRows*numberOfColumns)
  matrixIndex := 0
  queueIndex := 0

  // 处理矩阵填充
```

```
    for matrixIndex < len(matrix) {
      for i := 0; i < numberOfRows; i++ {
        if len(queueList[queueIndex]) > 0 {
          matrix[matrixIndex] = queueList[queueIndex][0]
          queueList[queueIndex] = queueList[queueIndex][1:]
          matrixIndex++
          if matrixIndex >= len(matrix) {
            break
          }
        } else {
          break
        }
      }
      queueIndex = (queueIndex + 1) % len(queueList)
    }

    // 按列优先顺序输出矩阵
    var result []string
    for row := 0; row < numberOfRows; row++ {
      for col := 0; col < numberOfColumns; col++ {
        result = append(result, strconv.Itoa(matrix[col*numberOfRows+row]))
      }
    }
    fmt.Println(strings.Join(result, " "))
}
```

> 来自: 华为OD机考2025C卷 – 推荐多样性 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025C卷 - 智能驾驶 (C++ & Python & JAVA & JS & GO)-CSDN博客

智能驾驶

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

有一辆汽车需要从 m * n 的地图左上角（起点）开往地图的右下角（终点），去往每一个地区都需要消耗一定的油量，加油站可进行加油。
请你计算汽车确保从从起点到达终点时所需的最少初始油量。说明：

1. 智能汽车可以上下左右四个方向移动
2. 地图上的数字取值是 0 或 -1 或 正整数：
    - -1 ： 表示加油站，可以加满油，汽车的油箱容量最大为100；
    - 0 ： 表示这个地区是障碍物，汽车不能通过
    - 正整数：表示汽车走过这个地区的耗油量

如果汽车无论如何都无法到达终点，则返回 -1

输入描述

第一行为两个数字，M，N，表示地图的大小为 M N。0 < M,N ≤ 200
后面一个 M * N 的矩阵，其中的值是 0 或 -1 或正整数，加油站的总数不超过 200 个

输出描述

如果汽车无论如何都无法到达终点，则返回 -1。如果汽车可以到达终点，则返回最少的初始油量

示例1

输入

Plain Text

```
2,2
10,20
30,40
```

输出

```
70
```

说明

> 行走的路线为：右→下

示例2

输入

```
4,4
10,30,30,20
30,30,-1,10
0,20,20,40
10,-1,30,40
```

输出

```
70
```

说明

> 行走的路线为：右→右→下→下→下→右

题解

思路：`二分 + BFS`

1. 使用 `二分` 去枚举汽车初始油量，利用 `BFS` 算法去判断二分确定的油是否满足车从左上角 -> 右下角。
2. 注意事项(特殊点)：
   - 汽车在经过加油站可以加油。
   - 起点为0，代表绝不可能完成 `左上角 -> 右下角的行动` ，因为0为障碍物。
3. 初始设置 `result = -1` 记录可行的最小结果，二分初始设置 `left = 0, right = 100（汽车最大存储油容量）` ，每次二分枚举 `mid = (left + right) / 2` ，使用 `BFS` 进行判断能否从左上角到右小角
   - 能：更新 `right = mid - 1 , result = mid`
   - 不能: 更新 `left = mid + 1`

4. 执行3操作直到 `left > right` 结束。最后输出result即可。
5. BFS的判断逻辑：使用 优先队列 模拟判断给定初始油量能否从左上角到达右下角。可以使用 `fuelLeft[][]` 数组进行剪枝，`fuelLeft[i][j]` 剩余的最多油量进行剪枝。具体逻辑可以参照下面的代码。

C++

Plain Text

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

using namespace std;

const int MAX_FUEL = 100; // 车辆最大油量

// 判断是否能到达终点
bool canReachDestination(const vector<vector<int>>& grid, int initialFuel, int rows, int cols) {
    if (grid[0][0] == 0) return false; // 起点是障碍物，无法前进

    vector<vector<int>> fuelLeft(rows, vector<int>(cols, -1)); // 记录每个点的最大剩余油量
    fuelLeft[0][0] = (grid[0][0] == -1) ? MAX_FUEL : initialFuel - grid[0][0];
    if (fuelLeft[0][0] < 0) return false; // 初始油量不足

    priority_queue<tuple<int, int, int>> pq; // 优先队列（剩余油量，行，列）
    pq.push({fuelLeft[0][0], 0, 0});

    vector<pair<int, int>> directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}}; // 四个移动方向

    while (!pq.empty()) {
        auto [fuel, r, c] = pq.top();
        pq.pop();

        if (r == rows - 1 && c == cols - 1) return true; // 到达终点

        for (auto& [dr, dc] : directions) {
            int nr = r + dr, nc = c + dc;
            if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] != 0) {
                int newFuel = (grid[nr][nc] == -1) ? MAX_FUEL : fuel - grid[nr][nc];
                if (newFuel > fuelLeft[nr][nc]) { // 更新更优解
                    fuelLeft[nr][nc] = newFuel;
                    pq.push({newFuel, nr, nc});
                }
            }
        }
    }
```

```
    return false;
}

// 二分查找最小初始油量
int findMinimumInitialFuel(const vector<vector<int>>& grid, int rows, int cols) {
    int low = 0, high = MAX_FUEL, optimalFuel = -1;

    while (low <= high) {
        int mid = (low + high) / 2;
        if (canReachDestination(grid, mid, rows, cols)) {
            optimalFuel = mid;
            high = mid - 1; // 继续寻找更小油量
        } else {
            low = mid + 1;
        }
    }
    return optimalFuel;
}

int main() {
    int rows, cols;
    // 处理逗号
    char comma;
    cin >> rows >> comma >> cols;

    vector<vector<int>> grid(rows, vector<int>(cols));
    for (int r = 0; r < rows; ++r) {
        for (int c = 0; c < cols; ++c) {
            if (c < cols - 1) cin >> grid[r][c] >> comma;
            else cin >> grid[r][c];
        }
    }

    cout << findMinimumInitialFuel(grid, rows, cols) << endl;
    return 0;
}
```

Java

Plain Text

```
import java.util.*;

public class Main {
    static final int MAX_FUEL = 100; // 车辆最大油量

    // 判断是否能到达终点
    static boolean canReachDestination(int[][] grid, int initialFuel, int rows, int cols) {
        if (grid[0][0] == 0) return false; // 起点是障碍物，无法前进

        int[][] fuelLeft = new int[rows][cols]; // 记录每个点的最大剩余油量
        for (int[] row : fuelLeft) Arrays.fill(row, -1);

        fuelLeft[0][0] = (grid[0][0] == -1) ? MAX_FUEL : initialFuel - grid[0][0];
        if (fuelLeft[0][0] < 0) return false; // 初始油量不足

        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> Integer.compare(b[0], a[0])); // 最大堆（剩余油量，行，列）
        pq.offer(new int[]{fuelLeft[0][0], 0, 0});

        int[][] directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}}; // 四个移动方向

        while (!pq.isEmpty()) {
            int[] top = pq.poll();
            int fuel = top[0], r = top[1], c = top[2];

            if (r == rows - 1 && c == cols - 1) return true; // 到达终点

            for (int[] dir : directions) {
                int nr = r + dir[0], nc = c + dir[1];
                if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] != 0) {
                    int newFuel = (grid[nr][nc] == -1) ? MAX_FUEL : fuel - grid[nr][nc];
                    if (newFuel > fuelLeft[nr][nc]) { // 更新更优解
                        fuelLeft[nr][nc] = newFuel;
                        pq.offer(new int[]{newFuel, nr, nc});
                    }
                }
            }
        }
        return false;
    }
```

```
40
41      // 二分查找最小初始油量
42      static int findMinimumInitialFuel(int[][] grid, int rows, int cols) {
43          int low = 0, high = MAX_FUEL, optimalFuel = -1;
44
45          while (low <= high) {
46              int mid = (low + high) / 2;
47              if (canReachDestination(grid, mid, rows, cols)) {
48                  optimalFuel = mid;
49                  high = mid - 1; // 继续寻找更小油量
50              } else {
51                  low = mid + 1;
52              }
53          }
54          return optimalFuel;
55      }
56
57      public static void main(String[] args) {
58          Scanner scanner = new Scanner(System.in);
59          String[] size = scanner.nextLine().split(",");
60          int rows = Integer.parseInt(size[0]);
61          int cols = Integer.parseInt(size[1]);
62
63          int[][] grid = new int[rows][cols];
64          for (int r = 0; r < rows; r++) {
65              String[] values = scanner.nextLine().split(",");
66              for (int c = 0; c < cols; c++) {
67                  grid[r][c] = Integer.parseInt(values[c]);
68              }
69          }
70
71          System.out.println(findMinimumInitialFuel(grid, rows, cols));
72      }
73  }
```

Python

Plain Text

```
import sys
import heapq

MAX_FUEL = 100  # 车辆最大油量

# 判断是否能到达终点
def can_reach_destination(grid, initial_fuel, rows, cols):
    if grid[0][0] == 0:
        return False  # 起点是障碍物，无法前进

    fuel_left = [[-1] * cols for _ in range(rows)]  # 记录每个点的最大剩余油量
    fuel_left[0][0] = MAX_FUEL if grid[0][0] == -1 else initial_fuel - gri
d[0][0]
    if fuel_left[0][0] < 0:
        return False  # 初始油量不足

    pq = [(-fuel_left[0][0], 0, 0)]  # 最大堆（负剩余油量，行，列）
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]  # 四个移动方向

    while pq:
        fuel, r, c = heapq.heappop(pq)
        fuel = -fuel

        if r == rows - 1 and c == cols - 1:
            return True  # 到达终点

        for dr, dc in directions:
            nr, nc = r + dr, c + dc
            if 0 <= nr < rows and 0 <= nc < cols and grid[nr][nc] != 0:
                new_fuel = MAX_FUEL if grid[nr][nc] == -1 else fuel - grid
[nr][nc]
                if new_fuel > fuel_left[nr][nc]:
                    fuel_left[nr][nc] = new_fuel
                    heapq.heappush(pq, (-new_fuel, nr, nc))

    return False

# 二分查找最小初始油量
def find_minimum_initial_fuel(grid, rows, cols):
    low, high, optimal_fuel = 0, MAX_FUEL, -1
    while low <= high:
        mid = (low + high) // 2
        if can_reach_destination(grid, mid, rows, cols):
            optimal_fuel = mid
            high = mid - 1  # 继续寻找更小油量
```

```
            else:
                low = mid + 1
        return optimal_fuel

if __name__ == "__main__":
        rows, cols = map(int, sys.stdin.readline().strip().split(","))
        grid = [list(map(int, sys.stdin.readline().strip().split(","))) for _
in range(rows)]
        print(find_minimum_initial_fuel(grid, rows, cols))
```

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const MAX_FUEL = 100; // 车辆最大油量

// 判断是否能到达终点
function canReachDestination(grid, initialFuel, rows, cols) {
    if (grid[0][0] === 0) return false;

    let fuelLeft = Array.from({ length: rows }, () => Array(cols).fill(-1));
    fuelLeft[0][0] = grid[0][0] === -1 ? MAX_FUEL : initialFuel - grid[0][0];
    if (fuelLeft[0][0] < 0) return false;

    let pq = [[fuelLeft[0][0], 0, 0]];
    const directions = [[0, 1], [1, 0], [0, -1], [-1, 0]];

    while (pq.length) {
        pq.sort((a, b) => b[0] - a[0]);
        let [fuel, r, c] = pq.shift();

        if (r === rows - 1 && c === cols - 1) return true;

        for (const [dr, dc] of directions) {
            let nr = r + dr, nc = c + dc;
            if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] !== 0) {
                let newFuel = grid[nr][nc] === -1 ? MAX_FUEL : fuel - grid[nr][nc];
                if (newFuel > fuelLeft[nr][nc]) {
                    fuelLeft[nr][nc] = newFuel;
                    pq.push([newFuel, nr, nc]);
                }
            }
        }
    }
    return false;
}

// 二分查找最小初始油量
```

```
function findMinimumInitialFuel(grid, rows, cols) {
    let low = 0, high = MAX_FUEL, optimalFuel = -1;
    while (low <= high) {
        let mid = Math.floor((low + high) / 2);
        if (canReachDestination(grid, mid, rows, cols)) {
            optimalFuel = mid;
            high = mid - 1; // 继续寻找更小油量
        } else {
            low = mid + 1;
        }
    }
    return optimalFuel;
}

// 读取输入
let input = [];
rl.on("line", (line) => {
    input.push(line);
}).on("close", () => {
    let [rows, cols] = input.shift().split(",").map(Number);
    let grid = input.map(row => row.split(",").map(Number));

    console.log(findMinimumInitialFuel(grid, rows, cols));
});
```

Go

Plain Text

```
package main

import (
  "container/heap"
  "fmt"
  "strings"
  "strconv"
)

const MAX_FUEL = 100 // 车辆最大油量

// PriorityQueue 是一个最大堆，用于优先队列
type PriorityQueue struct {
  items [][]int
}

func (pq *PriorityQueue) Len() int {
  return len(pq.items)
}

func (pq *PriorityQueue) Less(i, j int) bool {
  return pq.items[i][0] > pq.items[j][0]
}

func (pq *PriorityQueue) Swap(i, j int) {
  pq.items[i], pq.items[j] = pq.items[j], pq.items[i]
}

func (pq *PriorityQueue) Push(x interface{}) {
  pq.items = append(pq.items, x.([]int))
}

func (pq *PriorityQueue) Pop() interface{} {
  old := pq.items
  n := len(old)
  x := old[n-1]
  pq.items = old[0 : n-1]
  return x
}

// 判断是否能到达终点
func canReachDestination(grid [][]int, initialFuel, rows, cols int) bool
{
  if grid[0][0] == 0 {
    return false
```

```
  }

  fuelLeft := make([][]int, rows)
  for i := range fuelLeft {
    fuelLeft[i] = make([]int, cols)
    for j := range fuelLeft[i] {
      fuelLeft[i][j] = -1
    }
  }

  if grid[0][0] == -1 {
    fuelLeft[0][0] = MAX_FUEL
  } else {
    fuelLeft[0][0] = initialFuel - grid[0][0]
  }
  if fuelLeft[0][0] < 0 {
    return false
  }

  pq := &PriorityQueue{}
  heap.Init(pq)
  heap.Push(pq, []int{fuelLeft[0][0], 0, 0})

  directions := [][2]int{{0, 1}, {1, 0}, {0, -1}, {-1, 0}} // 四个移动方向

  for pq.Len() > 0 {
    item := heap.Pop(pq).([]int)
    fuel, r, c := item[0], item[1], item[2]

    if r == rows-1 && c == cols-1 {
      return true
    }

    for _, dir := range directions {
      nr, nc := r+dir[0], c+dir[1]
      if nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] !=
0 {
        var newFuel int
        if grid[nr][nc] == -1 {
          newFuel = MAX_FUEL
        } else {
          newFuel = fuel - grid[nr][nc]
        }
        if newFuel > fuelLeft[nr][nc] {
          fuelLeft[nr][nc] = newFuel
          heap.Push(pq, []int{newFuel, nr, nc})
        }
      }
```

```
    }
  }
  return false
}

// 二分查找最小初始油量
func findMinimumInitialFuel(grid [][]int, rows, cols int) int {
  low, high := 0, MAX_FUEL
  var optimalFuel = -1

  for low <= high {
    mid := (low + high) / 2
    if canReachDestination(grid, mid, rows, cols) {
      optimalFuel = mid
      high = mid - 1
    } else {
      low = mid + 1
    }
  }

  return optimalFuel
}

func main() {
  var rows, cols int
  fmt.Scanf("%d,%d\n", &rows, &cols)

  grid := make([][]int, rows)
  for i := 0; i < rows; i++ {
    var line string
    fmt.Scanln(&line)
    values := strings.Split(line, ",")
    grid[i] = make([]int, cols)
    for j, value := range values {
      grid[i][j], _ = strconv.Atoi(value)
    }
  }

  fmt.Println(findMinimumInitialFuel(grid, rows, cols))
}
```

> 来自: 华为OD机试 2025C卷 – 智能驾驶 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD上机考试(2025年C卷) - 电脑病毒感染 (C++ & Python & JAVA & JS & GO)-CSDN博客

电脑病毒感染 (华为OD机试2025C卷真题)

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

一个局域网内有很多台电脑，分别标注为 0 ~ N-1 的数字。相连接的电脑距离不一样，所以感染时间不一样，感染时间用 t 表示。
其中网络内一台电脑被病毒感染，求其感染网络内所有的电脑最少需要多长时间。如果最后有电脑不会感染，则返回-1。
给定一个数组 times 表示一台电脑把相邻电脑感染所用的时间。
如图：path[i] = {i, j, t} 表示：电脑 i->j，电脑 i 上的病毒感染 j，需要时间 t。

输入描述

第一行输入一个整数N ，表示局域网内电脑个数 N ，1 ≤ N ≤ 200 ；
第二行输入一个整数M ,表示有 M 条网络连接;
接下来M行 ,每行输入为 i , j , t 。表示电脑 i 感染电脑j 需要时间 t 。（1 ≤ i , j ≤ N）
最后一行为病毒所在的电脑编号。

输出描述

输出最少需要多少时间才能感染全部电脑，如果不存在输出 -1

示例1

输入

Plain Text

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

输出

```
Plain Text
1  2
```

说明

> 第一个参数：局域网内电脑个数N，1 ⩽ N ⩽ 200；
> 第二个参数：总共多少条网络连接
> 第三个 2 1 1 表示2->1时间为1
> 第六行：表示病毒最开始所在电脑号2

题解

思路：输出最少需要多少时间才能感染全部电脑这句话可以理解为 `从点u到达所有点的最短距离的最大值` ， 这道题就可以转换成求点单点最短路。可以是使用经典的 `dijkstra` (迪杰斯特拉)来解决。可以去详细学学这个算法
简单说一些迪杰斯特拉算法的思路：

- 算法的本质是贪心算法，使用当前找到的最短距离去更新与之相邻的点的距离.
- 假设当前点u距离为最小值，更新与之相邻的点离启动的距离，状态转移方程为 `当(dist[u] + w < dist[v])时， dist[v] = dis[u] + w` 。

C++

Plain Text

```
#include <functional>
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include <limits>
#include<queue>
using namespace std;
const int INF = numeric_limits<int>::max();


// 通用 split 函数
vector<string> split(const string& str, const string& delimiter) {
    vector<string> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(str.substr(start));
    return result;
}

// 迪杰斯特拉求最短路径/ 适用于权重为单点
int dijkstra(int start, const vector<vector<pair<int, int>>>& graph) {
    int n = graph.size();
    vector<int> dist(n, INF);
    dist[start] = 0;
    // 优先队列，优先按照时间进行升序排序
    priority_queue<pair<int,int>, vector<pair<int, int>>, greater<>> pq;

    pq.push({0, start});
    while (!pq.empty()) {
        auto [d,u] = pq.top();
        pq.pop();

        if (d > dist[u]) {
            continue;
        }
        // 更新与点u相连的点
```

```
        for (const auto& [v,w] : graph[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
    int res = 0;
    for (int i =0; i < n; i++) {
        if (dist[i] == INF) {
            res = -1;
            break;
        }
        res = max(res, dist[i]);
    }
    return res;
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int,int>>> g(n);
    while (m--) {
        int x, y, t;
        cin >> x >> y>>t;
        g[x-1].push_back({y-1, t});
    }
    int pos;
    cin >> pos;
    int res = dijkstra(pos-1, g);
    cout << res;
    return 0;
}
```

Java

Plain Text

```
import java.util.*;

public class Main {
    static final int INF = Integer.MAX_VALUE;

    // 迪杰斯特拉求最短路径
    public static int dijkstra(int start, List<List<int[]>> graph) {
        int n = graph.size();
        int[] dist = new int[n];
        Arrays.fill(dist, INF);
        dist[start] = 0;

        // 优先队列，按照距离升序排序
        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparing
Int(a -> a[0]));
        pq.offer(new int[]{0, start});

        while (!pq.isEmpty()) {
            int[] node = pq.poll();
            int d = node[0], u = node[1];

            if (d > dist[u]) continue;

            for (int[] edge : graph.get(u)) {
                int v = edge[0], w = edge[1];
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.offer(new int[]{dist[v], v});
                }
            }
        }

        int res = 0;
        for (int d : dist) {
            if (d == INF) return -1;
            res = Math.max(res, d);
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(), m = scanner.nextInt();
        List<List<int[]>> graph = new ArrayList<>();
        for (int i = 0; i < n; i++) graph.add(new ArrayList<>());
```

```
45
46          for (int i = 0; i < m; i++) {
47              int x = scanner.nextInt() - 1, y = scanner.nextInt() - 1, t =
    scanner.nextInt();
48              graph.get(x).add(new int[]{y, t});
49          }
50
51          int pos = scanner.nextInt() - 1;
52          System.out.println(dijkstra(pos, graph));
53          scanner.close();
54      }
55  }
```

Python

Plain Text

```
import sys
import heapq

INF = float('inf')

def dijkstra(start, graph):
    n = len(graph)
    dist = [INF] * n
    dist[start] = 0

    # 优先队列，按照距离升序排序
    pq = [(0, start)]

    while pq:
        d, u = heapq.heappop(pq)

        if d > dist[u]:
            continue

        for v, w in graph[u]:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                heapq.heappush(pq, (dist[v], v))

    res = max(dist) if all(d != INF for d in dist) else -1
    return res

def main():

    n = int(input())
    m = int(input())
    graph = [[] for _ in range(n)]

    for _ in range(m):
        x, y, t = map(int, sys.stdin.readline().split())
        graph[x - 1].append((y - 1, t))

    pos = int(sys.stdin.readline()) - 1
    print(dijkstra(pos, graph))

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const INF = Number.MAX_SAFE_INTEGER;

function dijkstra(start, graph) {
    const n = graph.length;
    const dist = new Array(n).fill(INF);
    dist[start] = 0;

    // 最小堆（优先队列）
    const pq = [[0, start]];
    pq.sort((a, b) => a[0] - b[0]); // 按距离排序

    while (pq.length > 0) {
        const [d, u] = pq.shift();

        if (d > dist[u]) continue;

        for (const [v, w] of graph[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push([dist[v], v]);
                pq.sort((a, b) => a[0] - b[0]); // 维护最小堆
            }
        }
    }

    const res = Math.max(...dist);
    return res === INF ? -1 : res;
}

let input = [];
rl.on("line", (line) => {
    input.push(line.trim());
}).on("close", () => {
    const n = Number(input[0])
    const m = Number(input[1])
    const graph = Array.from({ length: n }, () => []);

    for (let i = 2; i <= m+1; i++) {
```

```
        const [x, y, t] = input[i].split(" ").map(Number);
        graph[x - 1].push([y - 1, t]);
    }

    const pos = Number(input[m + 2]) - 1;
    console.log(dijkstra(pos, graph));
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

const INF = int(^uint(0) >> 1) // Go 语言中的 int 最大值

type Pair struct {
  weight int
  node   int
}

// 迪杰斯特拉算法，计算从 start 开始的最短路径
func dijkstra(start int, graph [][]Pair) int {
  n := len(graph)
  dist := make([]int, n)
  for i := range dist {
    dist[i] = INF
  }
  dist[start] = 0

  // 最小堆（优先队列），按距离从小到大排序
  pq := make(PriorityQueue, 0)
  pq.Push(Pair{0, start})

  for !pq.Empty() {
    top := pq.Pop()
    d, u := top.weight, top.node

    if d > dist[u] {
      continue
    }

    // 更新相邻节点的最短距离
    for _, edge := range graph[u] {
      v, w := edge.node, edge.weight
      if dist[u]+w < dist[v] {
        dist[v] = dist[u] + w
        pq.Push(Pair{dist[v], v})
      }
```

```
      }
    }

    // 计算最大最短路径
    res := 0
    for _, d := range dist {
      if d == INF {
        return -1
      }
      if d > res {
        res = d
      }
    }
    return res
}

// **优先队列（小顶堆）**
type PriorityQueue []Pair

func (pq *PriorityQueue) Push(p Pair) {
  *pq = append(*pq, p)
  up(*pq, len(*pq)-1)
}

func (pq *PriorityQueue) Pop() Pair {
  n := len(*pq) - 1
  pq.swap(0, n)
  down(*pq, 0, n)
  item := (*pq)[n]
  *pq = (*pq)[:n]
  return item
}

func (pq *PriorityQueue) Empty() bool {
  return len(*pq) == 0
}

func up(pq PriorityQueue, j int) {
  for j > 0 {
    i := (j - 1) / 2
    if pq[i].weight <= pq[j].weight {
      break
    }
    pq.swap(i, j)
    j = i
  }
}
```

```
func down(pq PriorityQueue, i, n int) {
  for {
    j1 := 2*i + 1
    if j1 >= n {
      break
    }
    j2 := j1 + 1
    j := j1
    if j2 < n && pq[j2].weight < pq[j1].weight {
      j = j2
    }
    if pq[i].weight <= pq[j].weight {
      break
    }
    pq.swap(i, j)
    i = j
  }
}

func (pq PriorityQueue) swap(i, j int) {
  pq[i], pq[j] = pq[j], pq[i]
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)

  // 读取点数 n
  scanner.Scan()
  n, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))

  // 读取边数 m
  scanner.Scan()
  m, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))

  // 构造图
  graph := make([][]Pair, n)
  for i := 0; i < m; i++ {
    scanner.Scan()
    line := strings.Fields(scanner.Text()) // 按空格分割
    x, _ := strconv.Atoi(line[0])
    y, _ := strconv.Atoi(line[1])
    t, _ := strconv.Atoi(line[2])
    graph[x-1] = append(graph[x-1], Pair{t, y - 1})
  }

  // 读取起点 pos
  scanner.Scan()
  pos, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))
```

```
    // 计算并输出结果
    fmt.Println(dijkstra(pos-1, graph))
}
```

电脑病毒感染 (华为OD机试2025C卷真题)

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

一个局域网内有很多台电脑，分别标注为 0 ~ N-1 的数字。相连接的电脑距离不一样，所以感染时间不一样，感染时间用 t 表示。

其中网络内一台电脑被病毒感染，求其感染网络内所有的电脑最少需要多长时间。如果最后有电脑不会感染，则返回-1。

给定一个数组 times 表示一台电脑把相邻电脑感染所用的时间。

如图：path[i] = {i, j, t} 表示：电脑 i->j，电脑 i 上的病毒感染 j，需要时间 t。

输入描述

第一行输入一个整数N ，表示局域网内电脑个数 N ，1 ≤ N ≤ 200 ；

第二行输入一个整数M ,表示有 M 条网络连接;

接下来M行 ,每行输入为 i , j , t 。表示电脑 i 感染电脑j 需要时间 t 。（1 ≤ i , j ≤ N）

最后一行为病毒所在的电脑编号。

输出描述

输出最少需要多少时间才能感染全部电脑，如果不存在输出 -1

示例1

输入

Plain Text

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

输出

```
1  2
```

说明

> 第一个参数：局域网内电脑个数N，1 ⩽ N ⩽ 200；
> 第二个参数：总共多少条网络连接
> 第三个 2 1 1 表示2–>1时间为1
> 第六行：表示病毒最开始所在电脑号2

题解

思路：输出最少需要多少时间才能感染全部电脑这句话可以理解为 `从点u到达所有点的最短距离的最大值` ， 这道题就可以转换成求点单点最短路。可以是使用经典的 `dijkstra` (迪杰斯特拉)来解决。可以去详细学学这个算法
简单说一些迪杰斯特拉算法的思路：

- 算法的本质是贪心算法，使用当前找到的最短距离去更新与之相邻的点的距离.
- 假设当前点u距离为最小值，更新与之相邻的点离启动的距离，状态转移方程为 `当(dist[u] + w < dist[v])时， dist[v] = dis[u] + w` 。

C++

Plain Text

```
#include <functional>
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include <limits>
#include<queue>
using namespace std;
const int INF = numeric_limits<int>::max();


// 通用 split 函数
vector<string> split(const string& str, const string& delimiter) {
    vector<string> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(str.substr(start));
    return result;
}

// 迪杰斯特拉求最短路径/ 适用于权重为单点
int dijkstra(int start, const vector<vector<pair<int, int>>>& graph) {
    int n = graph.size();
    vector<int> dist(n, INF);
    dist[start] = 0;
    // 优先队列，优先按照时间进行升序排序
    priority_queue<pair<int,int>, vector<pair<int, int>>, greater<>> pq;

    pq.push({0, start});
    while (!pq.empty()) {
        auto [d,u] = pq.top();
        pq.pop();

        if (d > dist[u]) {
            continue;
        }
        // 更新与点u相连的点
```

```
        for (const auto& [v,w] : graph[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
    int res = 0;
    for (int i =0; i < n; i++) {
        if (dist[i] == INF) {
            res = -1;
            break;
        }
        res = max(res, dist[i]);
    }
    return res;
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int,int>>> g(n);
    while (m--) {
        int x, y, t;
        cin >> x >> y>>t;
        g[x-1].push_back({y-1, t});
    }
    int pos;
    cin >> pos;
    int res = dijkstra(pos-1, g);
    cout << res;
    return 0;
}
```

Java

Plain Text

```
import java.util.*;

public class Main {
    static final int INF = Integer.MAX_VALUE;

    // 迪杰斯特拉求最短路径
    public static int dijkstra(int start, List<List<int[]>> graph) {
        int n = graph.size();
        int[] dist = new int[n];
        Arrays.fill(dist, INF);
        dist[start] = 0;

        // 优先队列，按照距离升序排序
        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparing
Int(a -> a[0]));
        pq.offer(new int[]{0, start});

        while (!pq.isEmpty()) {
            int[] node = pq.poll();
            int d = node[0], u = node[1];

            if (d > dist[u]) continue;

            for (int[] edge : graph.get(u)) {
                int v = edge[0], w = edge[1];
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.offer(new int[]{dist[v], v});
                }
            }
        }

        int res = 0;
        for (int d : dist) {
            if (d == INF) return -1;
            res = Math.max(res, d);
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(), m = scanner.nextInt();
        List<List<int[]>> graph = new ArrayList<>();
        for (int i = 0; i < n; i++) graph.add(new ArrayList<>());
```

```
45
46          for (int i = 0; i < m; i++) {
47              int x = scanner.nextInt() - 1, y = scanner.nextInt() - 1, t =
    scanner.nextInt();
48              graph.get(x).add(new int[]{y, t});
49          }
50
51          int pos = scanner.nextInt() - 1;
52          System.out.println(dijkstra(pos, graph));
53          scanner.close();
54      }
55  }
```

Python

Plain Text

```
import sys
import heapq

INF = float('inf')

def dijkstra(start, graph):
    n = len(graph)
    dist = [INF] * n
    dist[start] = 0

    # 优先队列，按照距离升序排序
    pq = [(0, start)]

    while pq:
        d, u = heapq.heappop(pq)

        if d > dist[u]:
            continue

        for v, w in graph[u]:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                heapq.heappush(pq, (dist[v], v))

    res = max(dist) if all(d != INF for d in dist) else -1
    return res

def main():

    n = int(input())
    m = int(input())
    graph = [[] for _ in range(n)]

    for _ in range(m):
        x, y, t = map(int, sys.stdin.readline().split())
        graph[x - 1].append((y - 1, t))

    pos = int(sys.stdin.readline()) - 1
    print(dijkstra(pos, graph))

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const INF = Number.MAX_SAFE_INTEGER;

function dijkstra(start, graph) {
    const n = graph.length;
    const dist = new Array(n).fill(INF);
    dist[start] = 0;

    // 最小堆（优先队列）
    const pq = [[0, start]];
    pq.sort((a, b) => a[0] - b[0]); // 按距离排序

    while (pq.length > 0) {
        const [d, u] = pq.shift();

        if (d > dist[u]) continue;

        for (const [v, w] of graph[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push([dist[v], v]);
                pq.sort((a, b) => a[0] - b[0]); // 维护最小堆
            }
        }
    }

    const res = Math.max(...dist);
    return res === INF ? -1 : res;
}

let input = [];
rl.on("line", (line) => {
    input.push(line.trim());
}).on("close", () => {
    const n = Number(input[0])
    const m = Number(input[1])
    const graph = Array.from({ length: n }, () => []);

    for (let i = 2; i <= m+1; i++) {
```

```
        const [x, y, t] = input[i].split(" ").map(Number);
        graph[x - 1].push([y - 1, t]);
    }

    const pos = Number(input[m + 2]) - 1;
    console.log(dijkstra(pos, graph));
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

const INF = int(^uint(0) >> 1) // Go 语言中的 int 最大值

type Pair struct {
  weight int
  node   int
}

// 迪杰斯特拉算法，计算从 start 开始的最短路径
func dijkstra(start int, graph [][]Pair) int {
  n := len(graph)
  dist := make([]int, n)
  for i := range dist {
    dist[i] = INF
  }
  dist[start] = 0

  // 最小堆（优先队列），按距离从小到大排序
  pq := make(PriorityQueue, 0)
  pq.Push(Pair{0, start})

  for !pq.Empty() {
    top := pq.Pop()
    d, u := top.weight, top.node

    if d > dist[u] {
      continue
    }

    // 更新相邻节点的最短距离
    for _, edge := range graph[u] {
      v, w := edge.node, edge.weight
      if dist[u]+w < dist[v] {
        dist[v] = dist[u] + w
        pq.Push(Pair{dist[v], v})
      }
```

```
      }
    }

    // 计算最大最短路径
    res := 0
    for _, d := range dist {
      if d == INF {
        return -1
      }
      if d > res {
        res = d
      }
    }
    return res
}

// **优先队列（小顶堆）**
type PriorityQueue []Pair

func (pq *PriorityQueue) Push(p Pair) {
  *pq = append(*pq, p)
  up(*pq, len(*pq)-1)
}

func (pq *PriorityQueue) Pop() Pair {
  n := len(*pq) - 1
  pq.swap(0, n)
  down(*pq, 0, n)
  item := (*pq)[n]
  *pq = (*pq)[:n]
  return item
}

func (pq *PriorityQueue) Empty() bool {
  return len(*pq) == 0
}

func up(pq PriorityQueue, j int) {
  for j > 0 {
    i := (j - 1) / 2
    if pq[i].weight <= pq[j].weight {
      break
    }
    pq.swap(i, j)
    j = i
  }
}
```

```
func down(pq PriorityQueue, i, n int) {
  for {
    j1 := 2*i + 1
    if j1 >= n {
      break
    }
    j2 := j1 + 1
    j := j1
    if j2 < n && pq[j2].weight < pq[j1].weight {
      j = j2
    }
    if pq[i].weight <= pq[j].weight {
      break
    }
    pq.swap(i, j)
    i = j
  }
}

func (pq PriorityQueue) swap(i, j int) {
  pq[i], pq[j] = pq[j], pq[i]
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)

  // 读取点数 n
  scanner.Scan()
  n, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))

  // 读取边数 m
  scanner.Scan()
  m, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))

  // 构造图
  graph := make([][]Pair, n)
  for i := 0; i < m; i++ {
    scanner.Scan()
    line := strings.Fields(scanner.Text()) // 按空格分割
    x, _ := strconv.Atoi(line[0])
    y, _ := strconv.Atoi(line[1])
    t, _ := strconv.Atoi(line[2])
    graph[x-1] = append(graph[x-1], Pair{t, y - 1})
  }

  // 读取起点 pos
  scanner.Scan()
  pos, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))
```

```
142
143
144
145
146
    // 计算并输出结果
    fmt.Println(dijkstra(pos-1, graph))
}
```

> 来自: 华为OD上机考试(2025年C卷) – 电脑病毒感染 (C++ & Python & JAVA & JS & GO)-CSDN博客

电脑病毒感染 (华为OD机试2025C卷真题)

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

一个局域网内有很多台电脑，分别标注为 0 ~ N-1 的数字。相连接的电脑距离不一样，所以感染时间不一样，感染时间用 t 表示。
其中网络内一台电脑被病毒感染，求其感染网络内所有的电脑最少需要多长时间。如果最后有电脑不会感染，则返回-1。
给定一个数组 times 表示一台电脑把相邻电脑感染所用的时间。
如图：path[i] = {i, j, t} 表示：电脑 i->j，电脑 i 上的病毒感染 j，需要时间 t。

输入描述

第一行输入一个整数N ，表示局域网内电脑个数 N ，1 ≤ N ≤ 200 ；
第二行输入一个整数M ,表示有 M 条网络连接;
接下来M行 ,每行输入为 i , j , t 。表示电脑 i 感染电脑j 需要时间 t 。 (1 ≤ i , j ≤ N)
最后一行为病毒所在的电脑编号。

输出描述

输出最少需要多少时间才能感染全部电脑，如果不存在输出 -1

示例1

输入

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

输出

```
2
```

说明

> 第一个参数：局域网内电脑个数N，1 ⩽ N ⩽ 200；
> 第二个参数：总共多少条网络连接
> 第三个 2 1 1 表示2->1时间为1
> 第六行：表示病毒最开始所在电脑号2

题解

思路：输出最少需要多少时间才能感染全部电脑这句话可以理解为 `从点u到达所有点的最短距离的最大值` ， 这道题就可以转换成求点单点最短路。可以是使用经典的 `dijkstra` (迪杰斯特拉)来解决。可以去详细学学这个算法
简单说一些迪杰斯特拉算法的思路：

- 算法的本质是贪心算法，使用当前找到的最短距离去更新与之相邻的点的距离.
- 假设当前点u距离为最小值，更新与之相邻的点离启动的距离，状态转移方程为 `当(dist[u] + w < dist[v])时， dist[v] = dis[u] + w` 。

C++

Plain Text

```
#include <functional>
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include <limits>
#include<queue>
using namespace std;
const int INF = numeric_limits<int>::max();


// 通用 split 函数
vector<string> split(const string& str, const string& delimiter) {
    vector<string> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(str.substr(start));
    return result;
}

// 迪杰斯特拉求最短路径/ 适用于权重为单点
int dijkstra(int start, const vector<vector<pair<int, int>>>& graph) {
    int n = graph.size();
    vector<int> dist(n, INF);
    dist[start] = 0;
    // 优先队列，优先按照时间进行升序排序
    priority_queue<pair<int,int>, vector<pair<int, int>>, greater<>> pq;

    pq.push({0, start});
    while (!pq.empty()) {
        auto [d,u] = pq.top();
        pq.pop();

        if (d > dist[u]) {
            continue;
        }
        // 更新与点u相连的点
```

```
        for (const auto& [v,w] : graph[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
    int res = 0;
    for (int i =0; i < n; i++) {
        if (dist[i] == INF) {
            res = -1;
            break;
        }
        res = max(res, dist[i]);
    }
    return res;
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<pair<int,int>>> g(n);
    while (m--) {
        int x, y, t;
        cin >> x >> y>>t;
        g[x-1].push_back({y-1, t});
    }
    int pos;
    cin >> pos;
    int res = dijkstra(pos-1, g);
    cout << res;
    return 0;
}
```

Java

Plain Text

```
import java.util.*;

public class Main {
    static final int INF = Integer.MAX_VALUE;

    // 迪杰斯特拉求最短路径
    public static int dijkstra(int start, List<List<int[]>> graph) {
        int n = graph.size();
        int[] dist = new int[n];
        Arrays.fill(dist, INF);
        dist[start] = 0;

        // 优先队列，按照距离升序排序
        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparing
Int(a -> a[0]));
        pq.offer(new int[]{0, start});

        while (!pq.isEmpty()) {
            int[] node = pq.poll();
            int d = node[0], u = node[1];

            if (d > dist[u]) continue;

            for (int[] edge : graph.get(u)) {
                int v = edge[0], w = edge[1];
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.offer(new int[]{dist[v], v});
                }
            }
        }

        int res = 0;
        for (int d : dist) {
            if (d == INF) return -1;
            res = Math.max(res, d);
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(), m = scanner.nextInt();
        List<List<int[]>> graph = new ArrayList<>();
        for (int i = 0; i < n; i++) graph.add(new ArrayList<>());
```

```
45
46          for (int i = 0; i < m; i++) {
47              int x = scanner.nextInt() - 1, y = scanner.nextInt() - 1, t =
    scanner.nextInt();
48              graph.get(x).add(new int[]{y, t});
49          }
50
51          int pos = scanner.nextInt() - 1;
52          System.out.println(dijkstra(pos, graph));
53          scanner.close();
54      }
55  }
```

Python

Plain Text

```
import sys
import heapq

INF = float('inf')

def dijkstra(start, graph):
    n = len(graph)
    dist = [INF] * n
    dist[start] = 0

    # 优先队列，按照距离升序排序
    pq = [(0, start)]

    while pq:
        d, u = heapq.heappop(pq)

        if d > dist[u]:
            continue

        for v, w in graph[u]:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                heapq.heappush(pq, (dist[v], v))

    res = max(dist) if all(d != INF for d in dist) else -1
    return res

def main():

    n = int(input())
    m = int(input())
    graph = [[] for _ in range(n)]

    for _ in range(m):
        x, y, t = map(int, sys.stdin.readline().split())
        graph[x - 1].append((y - 1, t))

    pos = int(sys.stdin.readline()) - 1
    print(dijkstra(pos, graph))

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const INF = Number.MAX_SAFE_INTEGER;

function dijkstra(start, graph) {
    const n = graph.length;
    const dist = new Array(n).fill(INF);
    dist[start] = 0;

    // 最小堆（优先队列）
    const pq = [[0, start]];
    pq.sort((a, b) => a[0] - b[0]); // 按距离排序

    while (pq.length > 0) {
        const [d, u] = pq.shift();

        if (d > dist[u]) continue;

        for (const [v, w] of graph[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push([dist[v], v]);
                pq.sort((a, b) => a[0] - b[0]); // 维护最小堆
            }
        }
    }

    const res = Math.max(...dist);
    return res === INF ? -1 : res;
}

let input = [];
rl.on("line", (line) => {
    input.push(line.trim());
}).on("close", () => {
    const n = Number(input[0])
    const m = Number(input[1])
    const graph = Array.from({ length: n }, () => []);

    for (let i = 2; i <= m+1; i++) {
```

```
        const [x, y, t] = input[i].split(" ").map(Number);
        graph[x - 1].push([y - 1, t]);
    }

    const pos = Number(input[m + 2]) - 1;
    console.log(dijkstra(pos, graph));
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

const INF = int(^uint(0) >> 1) // Go 语言中的 int 最大值

type Pair struct {
  weight int
  node   int
}

// 迪杰斯特拉算法，计算从 start 开始的最短路径
func dijkstra(start int, graph [][]Pair) int {
  n := len(graph)
  dist := make([]int, n)
  for i := range dist {
    dist[i] = INF
  }
  dist[start] = 0

  // 最小堆（优先队列），按距离从小到大排序
  pq := make(PriorityQueue, 0)
  pq.Push(Pair{0, start})

  for !pq.Empty() {
    top := pq.Pop()
    d, u := top.weight, top.node

    if d > dist[u] {
      continue
    }

    // 更新相邻节点的最短距离
    for _, edge := range graph[u] {
      v, w := edge.node, edge.weight
      if dist[u]+w < dist[v] {
        dist[v] = dist[u] + w
        pq.Push(Pair{dist[v], v})
      }
```

```
      }
    }

    // 计算最大最短路径
    res := 0
    for _, d := range dist {
      if d == INF {
        return -1
      }
      if d > res {
        res = d
      }
    }
    return res
}

// **优先队列（小顶堆）**
type PriorityQueue []Pair

func (pq *PriorityQueue) Push(p Pair) {
  *pq = append(*pq, p)
  up(*pq, len(*pq)-1)
}

func (pq *PriorityQueue) Pop() Pair {
  n := len(*pq) - 1
  pq.swap(0, n)
  down(*pq, 0, n)
  item := (*pq)[n]
  *pq = (*pq)[:n]
  return item
}

func (pq *PriorityQueue) Empty() bool {
  return len(*pq) == 0
}

func up(pq PriorityQueue, j int) {
  for j > 0 {
    i := (j - 1) / 2
    if pq[i].weight <= pq[j].weight {
      break
    }
    pq.swap(i, j)
    j = i
  }
}
```

```
func down(pq PriorityQueue, i, n int) {
  for {
    j1 := 2*i + 1
    if j1 >= n {
      break
    }
    j2 := j1 + 1
    j := j1
    if j2 < n && pq[j2].weight < pq[j1].weight {
      j = j2
    }
    if pq[i].weight <= pq[j].weight {
      break
    }
    pq.swap(i, j)
    i = j
  }
}

func (pq PriorityQueue) swap(i, j int) {
  pq[i], pq[j] = pq[j], pq[i]
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)

  // 读取点数 n
  scanner.Scan()
  n, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))

  // 读取边数 m
  scanner.Scan()
  m, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))

  // 构造图
  graph := make([][]Pair, n)
  for i := 0; i < m; i++ {
    scanner.Scan()
    line := strings.Fields(scanner.Text()) // 按空格分割
    x, _ := strconv.Atoi(line[0])
    y, _ := strconv.Atoi(line[1])
    t, _ := strconv.Atoi(line[2])
    graph[x-1] = append(graph[x-1], Pair{t, y - 1})
  }

  // 读取起点 pos
  scanner.Scan()
  pos, _ := strconv.Atoi(strings.TrimSpace(scanner.Text()))
```

```
    // 计算并输出结果
    fmt.Println(dijkstra(pos-1, graph))
}
```

> 来自: 华为OD上机考试(2025年C卷) – 电脑病毒感染 (C++ & Python & JAVA & JS & GO)-CSDN博客

> 来自: 华为OD上机考试(2025年C卷) – 电脑病毒感染 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 对称美学 (C++ & Python & JAVA & JS & GO)-CSDN博客

对称美学

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

对称就是最大的美学，现有一道关于对称字符串的美学。已知：

- 第1个字符串：R
- 第2个字符串：BR
- 第3个字符串：RBBR
- 第4个字符串：BRRBRBBR
- 第5个字符串：RBBRBRRBBRRBRBBR

相信你已经发现规律了，没错！就是第 i 个字符串 = 第 i – 1 号字符串取反 + 第 i – 1 号字符串；
取反（R–>B, B–>R）；
现在告诉你n和k，让你求得第n个字符串的第k个字符是多少。（k的编号从0开始）

输入描述

第一行输入一个T，表示有T组用例；
解析来输入T行，每行输入两个数字，表示n，k

- 1 ⩽ T ⩽ 100；
- 1 ⩽ n ⩽ 64；
- 0 ⩽ k ＜ 2^(n–1)；

输出描述

输出T行表示答案；
输出“blue”表示字符是B；
输出“red”表示字符是R。
备注：输出字符串区分大小写，请注意输出小写字符串，不带双引号。

用例1

输入

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

输出

```
red
red
blue
blue
blue
```

说明

> 第 1 个字符串：R -> 第 0 个字符为R
> 第 2 个字符串：BR -> 第 1 个字符为R
> 第 3 个字符串：RBBR -> 第 2 个字符为B
> 第 4 个字符串：BRRBRBBR -> 第 6 个字符为B
> 第 5 个字符串：RBBRBRRBBRRBRBBR -> 第 8 个字符为B

用例2

输入

```
1
64 73709551616
```

输出

```
red
```

题解

思路: 使用 `DFS` 的求解

1. 这道题可以通过题目描述找出规律来。
2. 每一个行的数量为2^(n-1).并且下一行的位置和上一行位置的关系,当k >= 2^(n-2)时, currentLine[k] = lastLine[k-2^(n-2)],当 k < 2^(n-2)时,currentLine[k] = !lastLine[k-2^(n-2)], 取相反值。

```
R
BR
RBBR
BRRBRBBR
RBBRBRRBBRRBRBBR
```

- 可以观察上面这个例子以及理解第2点说的规律。然后用代码实现就行。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<math.h>
using namespace std;

int solove(unsigned long long n, unsigned long long  k) {
    if (n == 1) {
        return 0;
    }
    if (n == 2) {
        if (k == 0) {
            return 1;
        } else if (k ==1) {
            return 0;
        }
    }
    // 每一行
    unsigned long long half = 1ULL << (n - 2);
    // 后半部分和上一行的 k -half相同
    if (k >= half) {
        return solove(n-1, k - half);
    } else {
      //和上一行的 k 取反
        return solove(n-1, k) == 0 ? 1 : 0;
    }
}


int main() {
    int t;
    cin >> t;
    while (t--) {
        // 无符号整数，保证 2^63 不溢出
        unsigned long long  n , k;
        cin >> n >> k;
        int res = solove(n, k);
        cout << ((res == 0) ? "red"  : "blue") << endl;
    }
    return 0;
```

```
}
```

JAVA

Plain Text

```
import java.util.Scanner;

public class Main {
    public static int solve(long n, long k) {
        if (n == 1) {
            return 0;
        }
        if (n == 2) {
            return (k == 0) ? 1 : 0;
        }

        long half = 1L << (n - 2);
        // 后半部分和上一行的 k -half相同
        if (k >= half) {
            return solve(n - 1, k - half);
        // 和上一行的 k 取反
        } else {
            return solve(n - 1, k) == 0 ? 1 : 0;
        }
    }

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

        while (t-- > 0) {
            // long 保证 2^63 不溢出
            long n = sc.nextLong();
            long k = sc.nextLong();
            int res = solve(n, k);
            System.out.println(res == 0 ? "red" : "blue");
        }

        sc.close();
    }
}
```

Python

```
import sys

def solve(n, k):
    if n == 1:
        return 0
    if n == 2:
        return 1 if k == 0 else 0

    half = 1 << (n - 2)
    # 后半部分和上一行的 k -half相同
    if k >= half:
        return solve(n - 1, k - half)
    # 和上一行的 k 取反
    else:
        return 1 if solve(n - 1, k) == 0 else 0

# 读取输入
t = int(sys.stdin.readline().strip())
for _ in range(t):
    n, k = map(int, sys.stdin.readline().split())
    res = solve(n, k)
    print("red" if res == 0 else "blue")
```

JavaScript

```
Plain Text
```

```
function solve(n, k) {
    n = BigInt(n); // 确保 n 是 BigInt
    k = BigInt(k); // 确保 k 是 BigInt

    if (n === BigInt(1)) {
        return "red";
    }
    if (n === BigInt(2)) {
        return k === BigInt(0) ? "blue" : "red";
    }

    let half = BigInt(1) << (n - BigInt(2));
    // 后半部分和上一行的 k -half相同
    if (k >= half) {
        return solve(n - BigInt(1), k - half);
    // 和上一行的 k 取反
    } else {
        return solve(n - BigInt(1), k) === "red" ? "blue" : "red";
    }
}

// 处理输入
const readline = require("readline");
const rl = readline.createInterface({ input: process.stdin });

let input = [];
rl.on("line", (line) => {
    input.push(line.trim());
}).on("close", () => {
    let t = parseInt(input[0], 10);
    for (let i = 1; i <= t; i++) {
        // BigInt 保证 2^63 不溢出
        let [n, k] = input[i].split(" ").map(BigInt);
        console.log(solve(n, k));
    }
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

// 递归求解
func solve(n uint64, k uint64) int {
  if n == 1 {
    return 0
  }
  if n == 2 {
    if k == 0 {
      return 1
    } else {
      return 0
    }
  }

  half := uint64(1) << (n - 2)
    // 后半部分和上一行的 k -half相同
  if k >= half {
    return solve(n-1, k-half)
    //和上一行的 k 取反
  } else {
    if solve(n-1, k) == 0 {
      return 1
    } else {
      return 0
    }
  }
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  t, _ := strconv.Atoi(scanner.Text())

  for i := 0; i < t; i++ {
    scanner.Scan()
    parts := strings.Split(scanner.Text(), " ")
```

```
            // 64 保证 2^63 不溢出
        n, _ := strconv.ParseUint(parts[0], 10, 64)
        k, _ := strconv.ParseUint(parts[1], 10, 64)

        res := solve(n, k)
        if res == 0 {
            fmt.Println("red")
        } else {
            fmt.Println("blue")
        }
    }
}
```

对称美学

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

对称就是最大的美学，现有一道关于对称字符串的美学。已知：

- 第1个字符串：R
- 第2个字符串：BR
- 第3个字符串：RBBR
- 第4个字符串：BRRBRBBR
- 第5个字符串：RBBRBRRBBRRBRBBR

相信你已经发现规律了，没错！就是第 i 个字符串 = 第 i – 1 号字符串取反 + 第 i – 1 号字符串；

取反（R–>B, B–>R）；

现在告诉你n和k，让你求得第n个字符串的第k个字符是多少。（k的编号从0开始）

输入描述

第一行输入一个T，表示有T组用例；

解析来输入T行，每行输入两个数字，表示n，k

- 1 ⩽ T ⩽ 100；
- 1 ⩽ n ⩽ 64；
- 0 ⩽ k ＜ 2^(n–1)；

输出描述

输出T行表示答案；

输出“blue”表示字符是B；

输出“red”表示字符是R。

备注：输出字符串区分大小写，请注意输出小写字符串，不带双引号。

用例1

输入

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

输出

```
red
red
blue
blue
blue
```

说明

> 第 1 个字符串：R -> 第 0 个字符为R
> 第 2 个字符串：BR -> 第 1 个字符为R
> 第 3 个字符串：RBBR -> 第 2 个字符为B
> 第 4 个字符串：BRRBRBBR -> 第 6 个字符为B
> 第 5 个字符串：RBBRBRRBBRRBRBBR -> 第 8 个字符为B

用例2

输入

```
1
64 73709551616
```

输出

```
red
```

题解

思路: 使用 `DFS` 的求解

1. 这道题可以通过题目描述找出规律来。
2. 每一个行的数量为2^(n-1).并且下一行的位置和上一行位置的关系,当k >= 2^(n-2)时, currentLine[k] = lastLine[k-2^(n-2)],当 k < 2^(n-2)时,currentLine[k] = !lastLine[k-2^(n-2)], 取相反值。

```
R
BR
RBBR
BRRBRBBR
RBBRBRRBBRRBRBBR
```

- 可以观察上面这个例子以及理解第2点说的规律。然后用代码实现就行。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<math.h>
using namespace std;

int solove(unsigned long long n, unsigned long long  k) {
    if (n == 1) {
        return 0;
    }
    if (n == 2) {
        if (k == 0) {
            return 1;
        } else if (k ==1) {
            return 0;
        }
    }
    // 每一行
    unsigned long long half = 1ULL << (n - 2);
    // 后半部分和上一行的 k -half相同
    if (k >= half) {
        return solove(n-1, k - half);
    } else {
      //和上一行的 k 取反
        return solove(n-1, k) == 0 ? 1 : 0;
    }
}


int main() {
    int t;
    cin >> t;
    while (t--) {
        // 无符号整数，保证 2^63 不溢出
        unsigned long long  n , k;
        cin >> n >> k;
        int res = solove(n, k);
        cout << ((res == 0) ? "red"  : "blue") << endl;
    }
    return 0;
```

```
}
```

JAVA

Plain Text

```
import java.util.Scanner;

public class Main {
    public static int solve(long n, long k) {
        if (n == 1) {
            return 0;
        }
        if (n == 2) {
            return (k == 0) ? 1 : 0;
        }

        long half = 1L << (n - 2);
        // 后半部分和上一行的 k -half相同
        if (k >= half) {
            return solve(n - 1, k - half);
        // 和上一行的 k 取反
        } else {
            return solve(n - 1, k) == 0 ? 1 : 0;
        }
    }

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

        while (t-- > 0) {
            // long 保证 2^63 不溢出
            long n = sc.nextLong();
            long k = sc.nextLong();
            int res = solve(n, k);
            System.out.println(res == 0 ? "red" : "blue");
        }

        sc.close();
    }
}
```

Python

```
import sys

def solve(n, k):
    if n == 1:
        return 0
    if n == 2:
        return 1 if k == 0 else 0

    half = 1 << (n - 2)
    # 后半部分和上一行的 k -half相同
    if k >= half:
        return solve(n - 1, k - half)
    # 和上一行的 k 取反
    else:
        return 1 if solve(n - 1, k) == 0 else 0

# 读取输入
t = int(sys.stdin.readline().strip())
for _ in range(t):
    n, k = map(int, sys.stdin.readline().split())
    res = solve(n, k)
    print("red" if res == 0 else "blue")
```

JavaScript

```
Plain Text
```

```
function solve(n, k) {
    n = BigInt(n); // 确保 n 是 BigInt
    k = BigInt(k); // 确保 k 是 BigInt

    if (n === BigInt(1)) {
        return "red";
    }
    if (n === BigInt(2)) {
        return k === BigInt(0) ? "blue" : "red";
    }

    let half = BigInt(1) << (n - BigInt(2));
    // 后半部分和上一行的 k -half相同
    if (k >= half) {
        return solve(n - BigInt(1), k - half);
    // 和上一行的 k 取反
    } else {
        return solve(n - BigInt(1), k) === "red" ? "blue" : "red";
    }
}

// 处理输入
const readline = require("readline");
const rl = readline.createInterface({ input: process.stdin });

let input = [];
rl.on("line", (line) => {
    input.push(line.trim());
}).on("close", () => {
    let t = parseInt(input[0], 10);
    for (let i = 1; i <= t; i++) {
        // BigInt 保证 2^63 不溢出
        let [n, k] = input[i].split(" ").map(BigInt);
        console.log(solve(n, k));
    }
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

// 递归求解
func solve(n uint64, k uint64) int {
  if n == 1 {
    return 0
  }
  if n == 2 {
    if k == 0 {
      return 1
    } else {
      return 0
    }
  }

  half := uint64(1) << (n - 2)
    // 后半部分和上一行的 k -half相同
  if k >= half {
    return solve(n-1, k-half)
    //和上一行的 k 取反
  } else {
    if solve(n-1, k) == 0 {
      return 1
    } else {
      return 0
    }
  }
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  t, _ := strconv.Atoi(scanner.Text())

  for i := 0; i < t; i++ {
    scanner.Scan()
    parts := strings.Split(scanner.Text(), " ")
```

```
            // 64 保证 2^63 不溢出
        n, _ := strconv.ParseUint(parts[0], 10, 64)
        k, _ := strconv.ParseUint(parts[1], 10, 64)

        res := solve(n, k)
        if res == 0 {
            fmt.Println("red")
        } else {
            fmt.Println("blue")
        }
    }
}
```

> 来自: 华为OD机考2025C卷 - 对称美学 (C++ & Python & JAVA & JS & GO)-CSDN博客

对称美学

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

对称就是最大的美学，现有一道关于对称字符串的美学。已知：

- 第1个字符串：R
- 第2个字符串：BR
- 第3个字符串：RBBR
- 第4个字符串：BRRBRBBR
- 第5个字符串：RBBRBRRBBRRBRBBR

相信你已经发现规律了，没错！就是第 i 个字符串 = 第 i - 1 号字符串取反 + 第 i - 1 号字符串；

取反（R->B, B->R）；

现在告诉你n和k，让你求得第n个字符串的第k个字符是多少。（k的编号从0开始）

输入描述

第一行输入一个T，表示有T组用例；

解析来输入T行，每行输入两个数字，表示n，k

- 1 ≤ T ≤ 100；
- 1 ≤ n ≤ 64；
- 0 ≤ k < 2^(n-1)；

输出描述

输出T行表示答案；

输出“blue”表示字符是B；
输出“red”表示字符是R。
备注：输出字符串区分大小写，请注意输出小写字符串，不带双引号。

用例1

输入

```Plain Text
5
1 0
2 1
3 2
4 6
5 8
```

输出

```Plain Text
red
red
blue
blue
blue
```

说明

> 第 1 个字符串：R -> 第 0 个字符为R
> 第 2 个字符串：BR -> 第 1 个字符为R
> 第 3 个字符串：RBBR -> 第 2 个字符为B
> 第 4 个字符串：BRRBRBBR -> 第 6 个字符为B
> 第 5 个字符串：RBBRBRRBBRRBRBBR -> 第 8 个字符为B

用例2

输入

```Plain Text
1
64 73709551616
```

输出

```
red
```

题解

思路: 使用 `DFS` 的求解

1. 这道题可以通过题目描述找出规律来。
2. 每一个行的数量为2^(n-1).并且下一行的位置和上一行位置的关系,当k >= 2^(n-2)时, currentLine[k] = lastLine[k-2^(n-2)],当 k < 2^(n-2)时,currentLine[k] = !lastLine[k-2^(n-2)], 取相反值。

```
R
BR
RBBR
BRRBRBBR
RBBRBRRBBRRBRBBR
```

- 可以观察上面这个例子以及理解第2点说的规律。然后用代码实现就行。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<math.h>
using namespace std;

int solove(unsigned long long n, unsigned long long  k) {
    if (n == 1) {
        return 0;
    }
    if (n == 2) {
        if (k == 0) {
            return 1;
        } else if (k ==1) {
            return 0;
        }
    }
    // 每一行
    unsigned long long half = 1ULL << (n - 2);
    // 后半部分和上一行的 k -half相同
    if (k >= half) {
        return solove(n-1, k - half);
    } else {
      //和上一行的 k 取反
        return solove(n-1, k) == 0 ? 1 : 0;
    }
}


int main() {
    int t;
    cin >> t;
    while (t--) {
        // 无符号整数，保证 2^63 不溢出
        unsigned long long  n , k;
        cin >> n >> k;
        int res = solove(n, k);
        cout << ((res == 0) ? "red"  : "blue") << endl;
    }
    return 0;
```

```
}
```

JAVA

Plain Text

```
import java.util.Scanner;

public class Main {
    public static int solve(long n, long k) {
        if (n == 1) {
            return 0;
        }
        if (n == 2) {
            return (k == 0) ? 1 : 0;
        }

        long half = 1L << (n - 2);
        // 后半部分和上一行的 k -half相同
        if (k >= half) {
            return solve(n - 1, k - half);
        // 和上一行的 k 取反
        } else {
            return solve(n - 1, k) == 0 ? 1 : 0;
        }
    }

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

        while (t-- > 0) {
            // long 保证 2^63 不溢出
            long n = sc.nextLong();
            long k = sc.nextLong();
            int res = solve(n, k);
            System.out.println(res == 0 ? "red" : "blue");
        }

        sc.close();
    }
}
```

Python

```
import sys

def solve(n, k):
    if n == 1:
        return 0
    if n == 2:
        return 1 if k == 0 else 0

    half = 1 << (n - 2)
    # 后半部分和上一行的 k -half相同
    if k >= half:
        return solve(n - 1, k - half)
    # 和上一行的 k 取反
    else:
        return 1 if solve(n - 1, k) == 0 else 0

# 读取输入
t = int(sys.stdin.readline().strip())
for _ in range(t):
    n, k = map(int, sys.stdin.readline().split())
    res = solve(n, k)
    print("red" if res == 0 else "blue")
```

JavaScript

```
Plain Text
```

```
function solve(n, k) {
    n = BigInt(n); // 确保 n 是 BigInt
    k = BigInt(k); // 确保 k 是 BigInt

    if (n === BigInt(1)) {
        return "red";
    }
    if (n === BigInt(2)) {
        return k === BigInt(0) ? "blue" : "red";
    }

    let half = BigInt(1) << (n - BigInt(2));
    // 后半部分和上一行的 k -half相同
    if (k >= half) {
        return solve(n - BigInt(1), k - half);
    // 和上一行的 k 取反
    } else {
        return solve(n - BigInt(1), k) === "red" ? "blue" : "red";
    }
}

// 处理输入
const readline = require("readline");
const rl = readline.createInterface({ input: process.stdin });

let input = [];
rl.on("line", (line) => {
    input.push(line.trim());
}).on("close", () => {
    let t = parseInt(input[0], 10);
    for (let i = 1; i <= t; i++) {
        // BigInt 保证 2^63 不溢出
        let [n, k] = input[i].split(" ").map(BigInt);
        console.log(solve(n, k));
    }
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

// 递归求解
func solve(n uint64, k uint64) int {
  if n == 1 {
    return 0
  }
  if n == 2 {
    if k == 0 {
      return 1
    } else {
      return 0
    }
  }

  half := uint64(1) << (n - 2)
    // 后半部分和上一行的 k -half相同
  if k >= half {
    return solve(n-1, k-half)
    //和上一行的 k 取反
  } else {
    if solve(n-1, k) == 0 {
      return 1
    } else {
      return 0
    }
  }
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  t, _ := strconv.Atoi(scanner.Text())

  for i := 0; i < t; i++ {
    scanner.Scan()
    parts := strings.Split(scanner.Text(), " ")
```

```
            // 64 保证 2^63 不溢出
        n, _ := strconv.ParseUint(parts[0], 10, 64)
        k, _ := strconv.ParseUint(parts[1], 10, 64)

        res := solve(n, k)
        if res == 0 {
          fmt.Println("red")
        } else {
          fmt.Println("blue")
        }
      }
    }
```

> 来自: 华为OD机考2025C卷 – 对称美学 (C++ & Python & JAVA & JS & GO)-CSDN博客

> 来自: 华为OD机考2025C卷 – 对称美学 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 围棋的气 (C++ & Python & JAVA & JS & GO)-CSDN博客

围棋的气

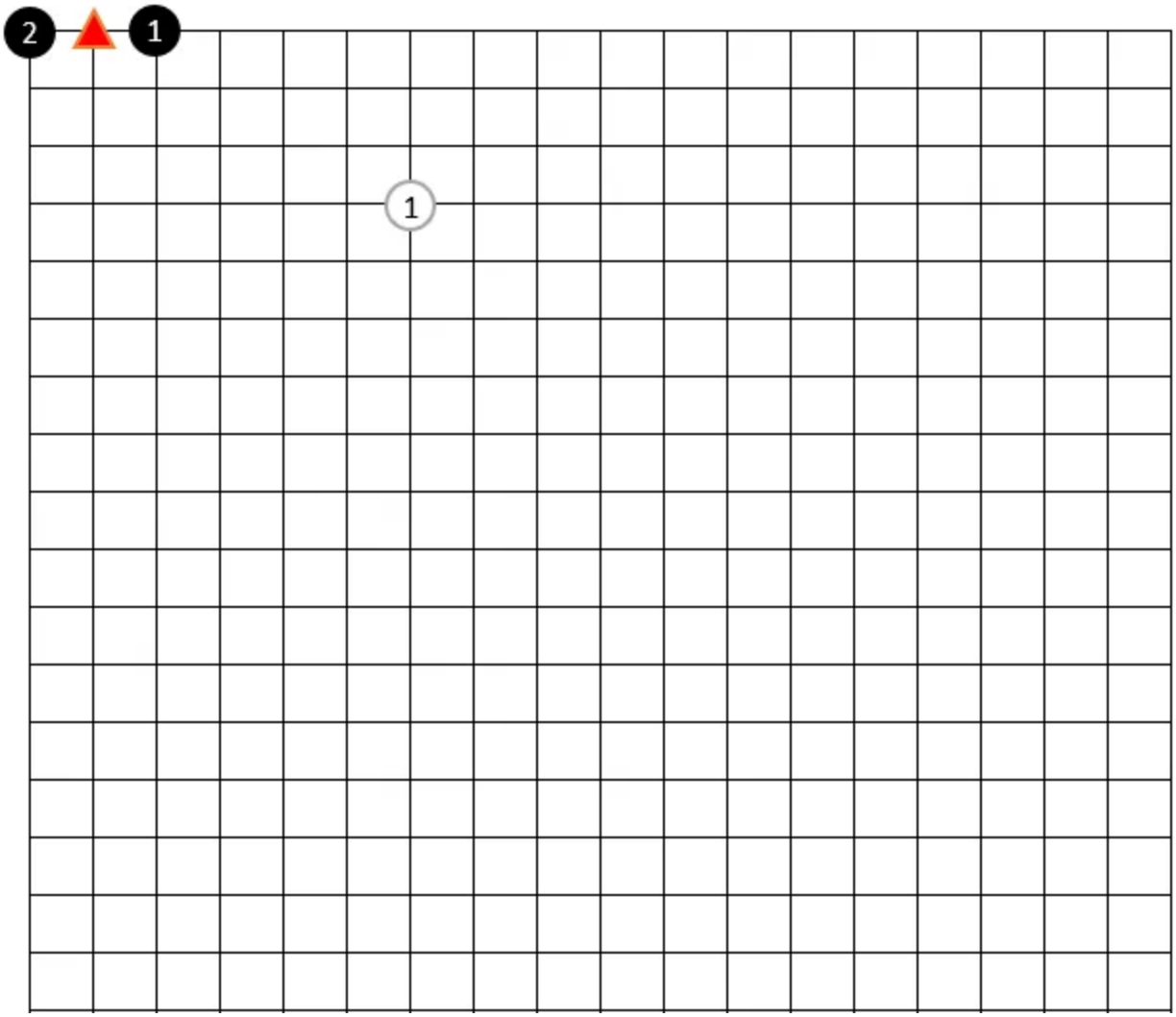
华为OD机试真题目录点击查看: 华为OD机试2025C卷真题题库目录｜机考题库 + 算法考点详解

> 华为OD机试2025C卷 100分题型

题目描述

围棋棋盘由纵横各19条线垂直相交组成，棋盘上一共19 x 19 = 361 个交点，对弈双方一方执白棋，一方执黑棋，落子时只能将棋子置于交点上。
“气”是围棋中很重要的一个概念，某个棋子有几口气，是指其上下左右方向四个相邻的交叉点中，有几个交叉点没有棋子，由此可知：

1. 在棋盘的边缘上的棋子最多有 3 口气（黑1），在棋盘角点的棋子最多有2口气（黑2），其他情况最多有4口气（白1）
2. 所有同色棋子的气之和叫做该色棋子的气，需要注意的是，同色棋子重合的气点，对于该颜色棋子来说，只能计算一次气，比如下图中，黑棋一共4口气，而不是5口气，因为黑1和黑2中间红色三角标出来的气是两个黑棋共有的，对于黑棋整体来说只能算一个气。
3. 本题目只计算气，对于眼也按气计算，如果您不清楚“眼”的概念，可忽略，按照前面描述的规则计算即可。



现在，请根据输入的黑棋和白棋得到坐标位置，计算黑棋和白棋一共各有多少气?

输入描述

输入包含两行数据，如：

```
0 5 8 9 9 10
5 0 9 9 9 8
```

- 每行数据以空格分隔，数据个数是2的整数倍，每两个数是一组，代表棋子在棋盘上的坐标；
- 坐标的原点在棋盘左上角点，第一个值是行号，范围从0到18；第二个值是列号，范围从0到18。
- 举例说明：第一行数据表示三个坐标（0, 5）、(8, 9)、(9, 10)
- 第一行表示黑棋的坐标，第二行表示白棋的坐标。
- 题目保证输入两行数据，无空行且每行按前文要求是偶数个，每个坐标不会超出棋盘范围。

输出描述

```
8 7
```

两个数字以空格分隔，第一个数代表黑棋的气数，第二个数代表白棋的气数。

用例1

输入

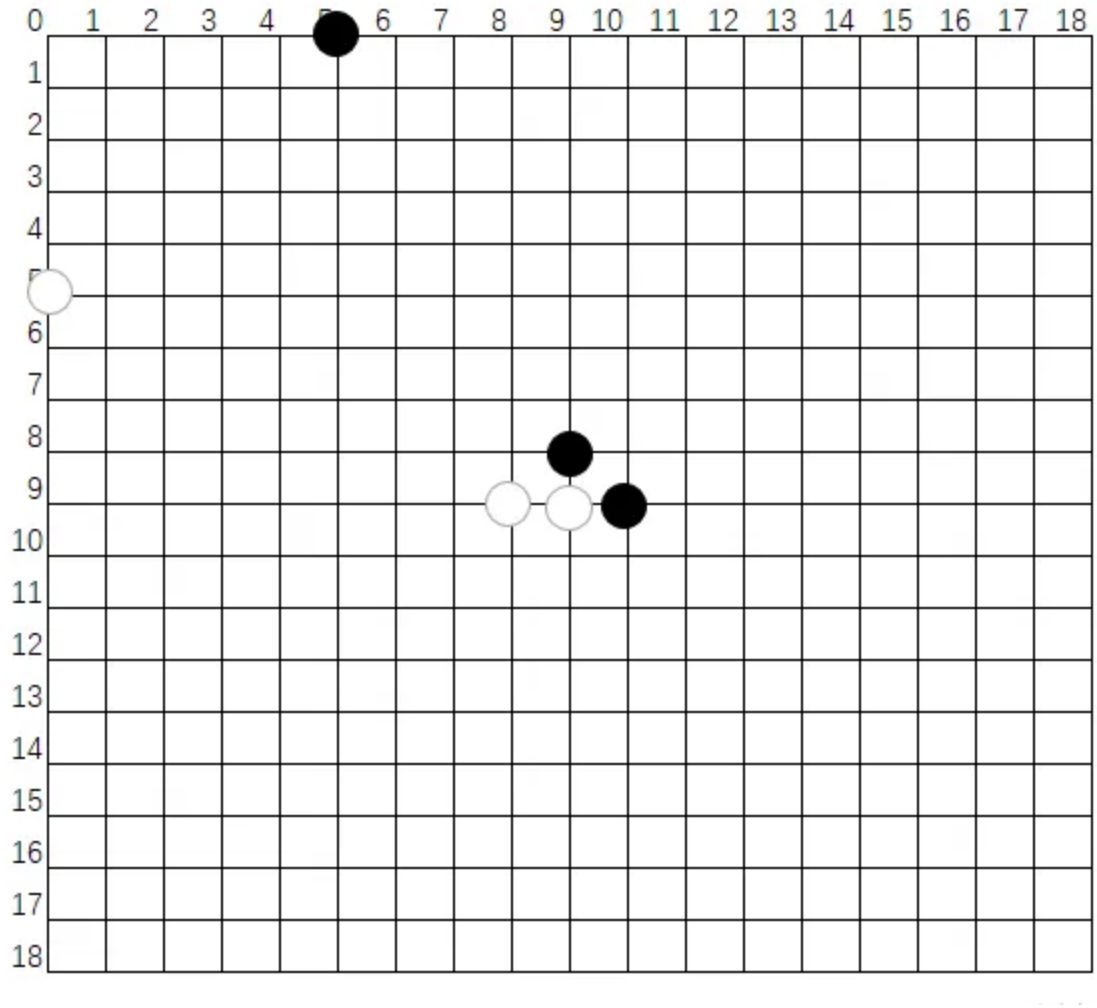
```
0 5 8 9 9 10
5 0 9 9 9 8
```

输出

```
8 7
```

说明

如果将输入数据放到棋盘上



数数黑棋一共8口气，数数白棋一共7口气。

题解

思路： 逻辑分析

1. 做这道题不要被题目描述所误导从黑棋和白棋角度出发，这样会需要额外去考虑去重等因素。而是从空的位置出发，这样可以使得问题简化很多。
2. 一个位置是空的，如果相邻四周有棋子就能给对应颜色棋子添加一个且仅最多添加一个气。所以可以使用两个布尔变量分别表示每个空位置四周是否有白棋、黑棋。如果存在则给对应颜色气的数量 + 1即可。
3. 通过2的逻辑，遍历完所有空的位置，就能分别得到白色和黑色气的数量。输出即可。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<cmath>
#include<map>
using namespace std;


// 通用 split 函数
vector<int> split(const string& str, const string& delimiter) {
    vector<int> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(stoi(str.substr(start, end - start)));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(stoi(str.substr(start)));
    return result;
}

int main() {
    // 构建棋盘
    vector<vector<int>> grid(19, vector<int>(19, 0));
    // 接受输入
    string blackCol,whiteCol;
    getline(cin, blackCol);
    getline(cin, whiteCol);
    vector<int> black = split(blackCol, " ");
    vector<int> white = split(whiteCol, " ");

    // 填充棋子
    for (int i = 0; i < black.size(); i+=2) {
        grid[black[i]][black[i+1]] = 1;
    }

    for (int i = 0; i < white.size(); i+=2) {
        grid[white[i]][white[i+1]] = 2;
    }

```

```
    int blackCount,whiteCount;
    blackCount = whiteCount = 0;

    int direct[4][2] = {{-1,0},{1, 0}, {0, -1}, {0,1}};
    for (int i = 0; i <  19; i++) {
        for (int j = 0; j < 19; j++) {
            if (grid[i][j] != 0) {
                continue;
            }
            // 空的四周是否有黑棋
            bool blackFlag = false;
            // 空的四周是否有白棋
            bool whiteFlag=  false;

            for (int m = 0; m < 4; m++) {
                int x = i + direct[m][0];
                int y = j + direct[m][1];

                // 越界
                if (x < 0 || x >= 19 || y < 0 || y >= 19) {
                    continue;
                }
                if (grid[x][y] == 1) {
                    blackFlag = true;
                }
                if (grid[x][y] == 2) {
                    whiteFlag = true;
                }
            }
            // 四周有黑棋
            if (blackFlag) {
                blackCount++;
            }
            // 四周有白棋
            if (whiteFlag) {
                whiteCount++;
            }
        }
    }
    cout << blackCount << " " << whiteCount;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[][] grid = new int[19][19];

        // 读入黑白棋坐标行
        String[] blackInput = sc.nextLine().trim().split(" ");
        String[] whiteInput = sc.nextLine().trim().split(" ");

        // 填充黑棋：1
        for (int i = 0; i < blackInput.length; i += 2) {
            int x = Integer.parseInt(blackInput[i]);
            int y = Integer.parseInt(blackInput[i + 1]);
            grid[x][y] = 1;
        }

        // 填充白棋：2
        for (int i = 0; i < whiteInput.length; i += 2) {
            int x = Integer.parseInt(whiteInput[i]);
            int y = Integer.parseInt(whiteInput[i + 1]);
            grid[x][y] = 2;
        }

        int blackCount = 0, whiteCount = 0;
        // 四个方向
        int[][] direct = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

        for (int i = 0; i < 19; i++) {
            for (int j = 0; j < 19; j++) {
                if (grid[i][j] != 0) continue;
                // 四周是否有对应颜色棋子标志
                boolean blackFlag = false;
                boolean whiteFlag = false;

                for (int[] d : direct) {
                    int x = i + d[0], y = j + d[1];
                    // 越界
                    if (x < 0 || x >= 19 || y < 0 || y >= 19) continue;
                    if (grid[x][y] == 1) blackFlag = true;
                    if (grid[x][y] == 2) whiteFlag = true;
                }

                if (blackFlag) blackCount++;
```

```
                    if (whiteFlag) whiteCount++;
                }
            }

            System.out.println(blackCount + " " + whiteCount);
        }
    }
```

Python

Plain Text

```
# 构建棋盘
grid = [[0] * 19 for _ in range(19)]

# 接收输入
black_input = list(map(int, input().split()))
white_input = list(map(int, input().split()))

# 填充黑棋：1
for i in range(0, len(black_input), 2):
    x, y = black_input[i], black_input[i + 1]
    grid[x][y] = 1

# 填充白棋：2
for i in range(0, len(white_input), 2):
    x, y = white_input[i], white_input[i + 1]
    grid[x][y] = 2

black_count = 0
white_count = 0
# 四个方向
directions = [(-1,0),(1,0),(0,-1),(0,1)]

# 遍历所有空格，判断周围的黑白棋
for i in range(19):
    for j in range(19):
        if grid[i][j] != 0:
            continue
        # 是否有对应颜色棋子标志
        black_flag = False
        white_flag = False
        for dx, dy in directions:
            x, y = i + dx, j + dy
            if 0 <= x < 19 and 0 <= y < 19:
                if grid[x][y] == 1:
                    black_flag = True
                if grid[x][y] == 2:
                    white_flag = True
        if black_flag:
            black_count += 1
        if white_flag:
            white_count += 1

print(f"{black_count} {white_count}")
```

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];

rl.on('line', line => {
    inputLines.push(line.trim());
    if (inputLines.length === 2) {
        rl.close();
    }
});

rl.on('close', () => {
    // 初始化棋盘
    const grid = Array.from({ length: 19 }, () => Array(19).fill(0));
    // 处理输入
    const black = inputLines[0].split(" ").map(Number);
    const white = inputLines[1].split(" ").map(Number);

    for (let i = 0; i < black.length; i += 2) {
        grid[black[i]][black[i + 1]] = 1;
    }
    for (let i = 0; i < white.length; i += 2) {
        grid[white[i]][white[i + 1]] = 2;
    }
    // 四个方向
    const directions = [[-1,0],[1,0],[0,-1],[0,1]];
    let blackCount = 0, whiteCount = 0;

    for (let i = 0; i < 19; i++) {
        for (let j = 0; j < 19; j++) {
            if (grid[i][j] !== 0) continue;
            // 四周是否有对应颜色棋子
            let blackFlag = false, whiteFlag = false;

            for (const [dx, dy] of directions) {
                const x = i + dx, y = j + dy;
                if (x < 0 || x >= 19 || y < 0 || y >= 19) continue;
                if (grid[x][y] === 1) blackFlag = true;
                if (grid[x][y] === 2) whiteFlag = true;
            }
```

```
            if (blackFlag) blackCount++;
            if (whiteFlag) whiteCount++;
        }
    }

    console.log(`${blackCount} ${whiteCount}`);
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

func main() {
  grid := make([][]int, 19)
  for i := range grid {
    grid[i] = make([]int, 19)
  }

  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  blackInput := strings.Fields(scanner.Text())
  scanner.Scan()
  whiteInput := strings.Fields(scanner.Text())

  // 填充黑棋
  for i := 0; i < len(blackInput); i += 2 {
    x, _ := strconv.Atoi(blackInput[i])
    y, _ := strconv.Atoi(blackInput[i+1])
    grid[x][y] = 1
  }

  // 填充白棋
  for i := 0; i < len(whiteInput); i += 2 {
    x, _ := strconv.Atoi(whiteInput[i])
    y, _ := strconv.Atoi(whiteInput[i+1])
    grid[x][y] = 2
  }
    // 四个方向
  dirs := [4][2]int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}
  blackCount, whiteCount := 0, 0

  for i := 0; i < 19; i++ {
    for j := 0; j < 19; j++ {
      if grid[i][j] != 0 {
        continue
      }
      // 四周是否有对应棋子
```

```
        blackFlag, whiteFlag := false, false
        for _, d := range dirs {
          x, y := i+d[0], j+d[1]
          if x >= 0 && x < 19 && y >= 0 && y < 19 {
            if grid[x][y] == 1 {
              blackFlag = true
            }
            if grid[x][y] == 2 {
              whiteFlag = true
            }
          }
        }
        if blackFlag {
          blackCount++
        }
        if whiteFlag {
          whiteCount++
        }
      }
    }

    fmt.Printf("%d %d\n", blackCount, whiteCount)
}
```

来自: 华为OD机考2025C卷 - 围棋的气 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 最小矩阵宽度 (C++ & Python & JAVA & JS & GO)-CSDN博客

最小矩阵宽度 (华为OD机试2025C卷真题)

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

给定一个矩阵，包含 N * M 个整数，和一个包含 K 个整数的数组。
现在要求在这个矩阵中找一个宽度最小的子矩阵，要求子矩阵包含数组中所有的整数。

输入描述

第一行输入两个正整数 N，M，表示矩阵大小。
接下来 N 行 M 列表示矩阵内容。
下一行包含一个正整数 K。
下一行包含 K 个整数，表示所需包含的数组，K 个整数可能存在重复数字。
所有输入数据小于1000。

输出描述

输出包含一个整数，表示满足要求子矩阵的最小宽度，若找不到，输出-1。

用例1

输入

Plain Text

```
2 5
1 2 2 3 1
2 3 2 3 2
3
1 2 3
```

输出

```
2
```

说明

> 矩阵第0、1列包含了1，2，3，矩阵第3，4列包含了1，2，3

用例2

输入

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

输出

```
5
```

说明

> 矩阵第1、2、3、4、5列包含了1、1、4

题解

思路： `双指针` 处理，基本逻辑如下，判断双指针区域中的数是否能完全匹配上待匹配数组

- 不能：让右指针右移，增加区域。
- 能：让左指针右移，尝试获取更少的宽度。

> 代码可以使用哈希表来加速判断双指针区域是否已经完全匹配。

定义 `count` 表示双指针区域中已经匹配上 待匹配数字的数量， 使用 `map<int, int> mp` 存储 统计待匹配数组中各个数字出现次数，使用 `map<int, int> total`
统计双指针区域中出现数字的数量，处理逻辑如下：

1. 增加右边界，新增一列时，假设当前新增这一列元素为 `nums1 nums2..numsn` 需要依次去变更 `total` 值，例如处理这一列的第一个值时，会进行 `total[num1]++` ，同时判断 `total[num1] <= mp[num1]` ，如果满足条件则说明额外匹配上一个元素，count+= 1。为这一列重复执行前面的操

作。如果新增这一列之后 `count == k` 说明已经完整匹配上了。没有完整匹配则右边界继续右移，完整匹配考虑缩小区域，进行左指针右移。

2. 左指针右移，类似于增加右边界的反操作。减少 `total` 的值， `total[nums1]--` , 如果此时 `total[num1] < mp[num1]` 则count --=1. 左指针右移的结束条件为 `count < k` 。
3. 重复1 2 操作，直到right == m时结束。输出记录的最小宽度即可。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<climits>
using namespace std;

int main() {
    int n , m;
    cin >> n >> m;
    vector<vector<int>> grid(n, vector<int>(m, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }
    int k;
    cin >> k;
    vector<int> ans(k);
    // 统计待匹配数组中各个数字出现次数
    map<int, int> mp;
    for (int i = 0; i < k; i++) {
        cin>>ans[i];
        mp[ans[i]]++;
    }
    // 统计双指针区域中出现数字的数量
    map<int, int> total;
    int res = INT_MAX;
    // 双指针区域中已经匹配上 待匹配数字的数量
    int count  = 0;

    // 统计第一列出现的各个数字的数量  以及更新匹配上的数量
    for (int i = 0; i < n; i++) {
        int num = grid[i][0];
        total[num]++;
        if (total[num] <= mp[num]) {
            count++;
        }
    }
    // 已经完全匹配
```

```
    if (count == k) {
        cout << 1;
        return 0;
    }

    int left = 0, right = 0;
    while (right < m) {
        // 增加右边界 双指针区域并没有完全匹配上所有元素，右边界右移，增加区域
        if (count < k) {
            right++;
            if (right >= m) {
                continue;
            }
            for (int i = 0; i < n; i++) {
                int num = grid[i][right];
                total[num]++;
                if (total[num] <= mp[num]) {
                    count++;
                }
            }
            if (count == k) {
                res = min(res, right - left + 1);
            }
        // 尝试减少左边界  已经完全匹配的情况下，尝试左边界右移缩小宽度
        } else {
            for (int i = 0; i < n; i++) {
                int num = grid[i][left];
                total[num]--;
                if (total[num] < mp[num]) {
                    count--;
                }
            }
            left++;
            if (count == k) {
                res = min(res, right - left + 1);
            }
        }
    }

    if (res != INT_MAX) {
        cout << res;
    } else {
        cout << -1;
    }

    return 0;
}
```

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 读取 n 和 m
        int n = scanner.nextInt();
        int m = scanner.nextInt();

        // 读取矩阵 grid
        int[][] grid = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

        // 读取 k
        int k = scanner.nextInt();
        int[] ans = new int[k];
        Map<Integer, Integer> mp = new HashMap<>();

        // 统计待匹配数组中各个数字出现次数
        for (int i = 0; i < k; i++) {
            ans[i] = scanner.nextInt();
            mp.put(ans[i], mp.getOrDefault(ans[i], 0) + 1);
        }

        // 统计双指针区域中出现数字的数量
        Map<Integer, Integer> total = new HashMap<>();
        int count = 0, res = Integer.MAX_VALUE;

        // 统计第一列出现的各个数字的数量，并更新匹配上的数量
        for (int i = 0; i < n; i++) {
            int num = grid[i][0];
            total.put(num, total.getOrDefault(num, 0) + 1);
            if (total.get(num) <= mp.getOrDefault(num, 0)) {
                count++;
            }
        }

        // 已经完全匹配
        if (count == k) {
            System.out.println(1);
```

```
                return;
            }

            int left = 0, right = 0;
            while (right < m) {
                // 增加右边界，双指针区域并没有完全匹配上所有元素
                if (count < k) {
                    right++;
                    if (right >= m) continue;

                    for (int i = 0; i < n; i++) {
                        int num = grid[i][right];
                        total.put(num, total.getOrDefault(num, 0) + 1);
                        if (total.get(num) <= mp.getOrDefault(num, 0)) {
                            count++;
                        }
                    }

                    if (count == k) {
                        res = Math.min(res, right - left + 1);
                    }
                // 尝试减少左边界，已经完全匹配的情况下尝试缩小宽度
                } else {
                    for (int i = 0; i < n; i++) {
                        int num = grid[i][left];
                        total.put(num, total.get(num) - 1);
                        if (total.get(num) < mp.getOrDefault(num, 0)) {
                            count--;
                        }
                    }
                    left++;
                    if (count == k) {
                        res = Math.min(res, right - left + 1);
                    }
                }
            }

            System.out.println(res != Integer.MAX_VALUE ? res : -1);
        }
    }
```

Python

Plain Text

```
import sys

# 读取输入
n, m = map(int, sys.stdin.readline().split())

# 读取矩阵 grid
grid = [list(map(int, sys.stdin.readline().split())) for _ in range(n)]

# 读取 k 和 ans
k = int(sys.stdin.readline())
ans = list(map(int, sys.stdin.readline().split()))

# 统计待匹配数组中各个数字出现次数
mp = {}
for num in ans:
    mp[num] = mp.get(num, 0) + 1

# 统计双指针区域中出现数字的数量
total = {}
count = 0
res = float('inf')

# 统计第一列出现的各个数字的数量，并更新匹配上的数量
for i in range(n):
    num = grid[i][0]
    total[num] = total.get(num, 0) + 1
    if total[num] <= mp.get(num, 0):
        count += 1

# 已经完全匹配
if count == k:
    print(1)
    sys.exit()

left, right = 0, 0
while right < m:
    # 增加右边界，双指针区域并没有完全匹配上所有元素
    if count < k:
        right += 1
        if right >= m:
            continue

        for i in range(n):
            num = grid[i][right]
            total[num] = total.get(num, 0) + 1
```

```
                if total[num] <= mp.get(num, 0):
                    count += 1

            if count == k:
                res = min(res, right - left + 1)
        # 尝试减少左边界，已经完全匹配的情况下尝试缩小宽度
        else:
            for i in range(n):
                num = grid[i][left]
                total[num] -= 1
                if total[num] < mp.get(num, 0):
                    count -= 1

            left += 1
            if count == k:
                res = min(res, right - left + 1)

    print(res if res != float('inf') else -1)
```

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let input = [];
rl.on('line', (line) => {
    input.push(line);
}).on('close', () => {
    // 读取 n 和 m
    let [n, m] = input[0].split(' ').map(Number);

    // 读取矩阵 grid
    let grid = [];
    for (let i = 1; i <= n; i++) {
        grid.push(input[i].split(' ').map(Number));
    }

    // 读取 k
    let k = Number(input[n + 1]);
    let ans = input[n + 2].split(' ').map(Number);

    // 统计待匹配数组中各个数字出现次数
    let mp = {};
    ans.forEach(num => mp[num] = (mp[num] || 0) + 1);

    // 统计双指针区域中出现数字的数量
    let total = {};
    let count = 0, res = Infinity;

    // 统计第一列出现的各个数字的数量，并更新匹配上的数量
    for (let i = 0; i < n; i++) {
        let num = grid[i][0];
        total[num] = (total[num] || 0) + 1;
        if (total[num] <= (mp[num] || 0)) {
            count++;
        }
    }

    // 已经完全匹配
    if (count === k) {
        console.log(1);
        return;
```

```
    }

    let left = 0, right = 0;
    while (right < m) {
        // 增加右边界，双指针区域并没有完全匹配上所有元素
        if (count < k) {
            right++;
            if (right >= m) continue;

            for (let i = 0; i < n; i++) {
                let num = grid[i][right];
                total[num] = (total[num] || 0) + 1;
                if (total[num] <= (mp[num] || 0)) {
                    count++;
                }
            }

            if (count === k) {
                res = Math.min(res, right - left + 1);
            }
        // 尝试减少左边界，已经完全匹配的情况下尝试缩小宽度
        } else {
            for (let i = 0; i < n; i++) {
                let num = grid[i][left];
                total[num]--;
                if (total[num] < (mp[num] || 0)) {
                    count--;
                }
            }
            left++;
            if (count === k) {
                res = Math.min(res, right - left + 1);
            }
        }
    }

    console.log(res !== Infinity ? res : -1);
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

func main() {
  reader := bufio.NewReader(os.Stdin)

  // 读取 n 和 m
  nm, _ := reader.ReadString('\n')
  nmSlice := splitToInt(nm)
  n, m := nmSlice[0], nmSlice[1]

  // 读取矩阵 grid
  grid := make([][]int, n)
  for i := 0; i < n; i++ {
    line, _ := reader.ReadString('\n')
    grid[i] = splitToInt(line)
  }

  // 读取 k
  kStr, _ := reader.ReadString('\n')
  k, _ := strconv.Atoi(strings.TrimSpace(kStr))

  // 读取 ans 数组
  ansStr, _ := reader.ReadString('\n')
  ans := splitToInt(ansStr)

  // 统计待匹配数组中各个数字出现次数
  mp := make(map[int]int)
  for _, num := range ans {
    mp[num]++
  }

  // 统计双指针区域中出现数字的数量
  total := make(map[int]int)
  count := 0
  res := int(^uint(0) >> 1) // 设置 res 为 int 最大值

  // 统计第一列出现的各个数字的数量，并更新匹配上的数量
```

```
    for i := 0; i < n; i++ {
      num := grid[i][0]
      total[num]++
      if total[num] <= mp[num] {
        count++
      }
    }

    // 已经完全匹配
    if count == k {
      fmt.Println(1)
      return
    }

    left, right := 0, 0
    for right < m {
      // 增加右边界，双指针区域并没有完全匹配上所有元素
      if count < k {
        right++
        if right >= m {
          continue
        }

        for i := 0; i < n; i++ {
          num := grid[i][right]
          total[num]++
          if total[num] <= mp[num] {
            count++
          }
        }

        if count == k {
          res = min(res, right-left+1)
        }
      } else { // 尝试减少左边界，已经完全匹配的情况下尝试缩小宽度
        for i := 0; i < n; i++ {
          num := grid[i][left]
          total[num]--
          if total[num] < mp[num] {
            count--
          }
        }
        left++
        if count == k {
          res = min(res, right-left+1)
        }
      }
    }
```

```
  if res != int(^uint(0)>>1) {
    fmt.Println(res)
  } else {
    fmt.Println(-1)
  }
}

// splitToInt 将输入字符串按空格拆分并转换为整数数组
func splitToInt(s string) []int {
  fields := strings.Fields(s)
  res := make([]int, len(fields))
  for i, v := range fields {
    res[i], _ = strconv.Atoi(v)
  }
  return res
}

// min 返回两个整数中的最小值
func min(a, b int) int {
  if a < b {
    return a
  }
  return b
}
```

> 来自: 华为OD机考2025C卷 – 最小矩阵宽度 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 小朋友来自多少个小区 / 最少有多少个小朋友 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 小朋友来自多少个小区 / 最少有多少个小朋友

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

幼儿园组织活动，老师布置了一个任务：
每个小朋友去了解与自己同一个小区的小朋友还有几个。
我们将这些数量汇总到数组 garden 中。
请根据这些小朋友给出的信息，计算班级小朋友至少来自几个小区？

输入描述

输入：garden[] = {2, 2, 3}

备注

- garden 数组长度最大为 999
- 每个小区的小朋友数量最多 1000 人，也就是 garden[i] 的范围为 [0, 999]

输出描述

输出：7

用例1

输入

Plain Text

```
2 2 3
```

输出

```
7
```

说明

> 第一个小朋友反馈有两个小朋友和自己同一小区，即此小区有3个小朋友。第二个小朋友反馈有两个小朋友和自己同一小区，即此小区有3个小朋友。这两个小朋友，可能是同一小区的，且此小区的小朋友只有3个人。第三个小区反馈还有3个小朋友与自己同一小区，则这些小朋友只能是另外一个小区的。这个小区有4个小朋友。

题解

> 这个题目描述的有点坑，从测试案例来看是求班级最少有多少个人。

思路: 贪心

- 对于每个小朋友说的值(value)，实际小区的人数应该是(value + 1)，贪心 我们尽可能让值相同的小朋友处于同一个小区.
- 实际处理过程中使用 `map<int,int>` cnts统计出现相同值的数量。比如这个值为2，所有说2的小朋友能得出的最小人数为 `(ceil(cnts[2] * 1.0 / (2 + 1))) * (2 + 1)` 。
- 根据上述贪心思维，计算不通值的最少人数，累加即可。

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<cmath>
using namespace std;

// 通用 split 函数
vector<string> split(const string& str, const string& delimiter) {
    vector<string> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(str.substr(start));
    return result;
}

int main() {
    string s;
    getline(cin, s);
    vector<string> tmp = split(s, " ");
    int n = tmp.size();
    vector<int> ans(n);
    map<int, int> cnts;
    for (int i = 0; i < n; i++) {
        ans[i] = stoi(tmp[i]);
        cnts[ans[i]]++;
    }
    int res = 0;
    for (auto p : cnts) {
        // 代表小区应该有的人数
        int total = p.first + 1;
        // 贪心，让拥有朋友相同的情况，尽可能处于一个小区，向上取整
        res += (ceil(p.second * 1.0 / total)) * total ;
    }
```

```
    cout << res;
    return 0;

}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] input = scanner.nextLine().split(" ");
        scanner.close();

        int n = input.length;
        int[] ans = new int[n];
        Map<Integer, Integer> cnts = new HashMap<>();

        // 统计每个数的出现次数
        for (int i = 0; i < n; i++) {
            ans[i] = Integer.parseInt(input[i]);
            cnts.put(ans[i], cnts.getOrDefault(ans[i], 0) + 1);
        }

        int res = 0;
        for (Map.Entry<Integer, Integer> entry : cnts.entrySet()) {
            // 小区实际的人数
            int total = entry.getKey() + 1;
            // 贪心，尽可能让小朋友处于一个或多个小区。  Math.ceil(entry.getValue() * 1.0 / total) 最少小区的数量
            res += Math.ceil(entry.getValue() * 1.0 / total) * total;
        }

        System.out.println(res);
    }
}
```

Python

```
import sys
import math

def main():
    # 读取输入并转换为整数列表
    ans = list(map(int, sys.stdin.readline().strip().split()))

    # 统计每个数的出现次数
    cnts = {}
    for num in ans:
        cnts[num] = cnts.get(num, 0) + 1

    res = 0
    for k, v in cnts.items():
        # 小区人数数量
        total = k + 1
        # 贪心尽可能让小朋友处于一个或多个小区内  math.ceil(v / total) 最少小区数量
        res += math.ceil(v / total) * total

    print(res)

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.on("line", (line) => {
    const ans = line.trim().split(" ").map(Number);

    // 统计每个数的出现次数
    const cnts = new Map();
    ans.forEach(num => {
        cnts.set(num, (cnts.get(num) || 0) + 1);
    });

    let res = 0;
    cnts.forEach((value, key) => {
        // 小区实际人数
        let total = key + 1;
        // 贪心尽可能让这些小朋友分配到一个或多个小区内   Math.ceil(value / total) 最小需要的小区数量
        res += Math.ceil(value / total) * total;
    });

    console.log(res);
    rl.close();
});
```

Go

```
Plain Text
```

```
package main

import (
  "bufio"
  "fmt"
  "math"
  "os"
  "strconv"
  "strings"
)

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  input := strings.Split(scanner.Text(), " ")

  // 统计每个数的出现次数
  cnts := make(map[int]int)
  for _, s := range input {
    num, _ := strconv.Atoi(s)
    cnts[num]++
  }

  res := 0
  for k, v := range cnts {
    // 小区实际人数
    total := k + 1
    // 尽可能让这些小朋友处于一个或多个小区 int(math.Ceil(float64(v)/float64(total))) 需要最少的小区数量
    res += int(math.Ceil(float64(v)/float64(total))) * total
  }

  fmt.Println(res)
}
```

华为OD机考2025C卷 小朋友来自多少个小区 / 最少有多少个小朋友

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

幼儿园组织活动，老师布置了一个任务：
每个小朋友去了解与自己同一个小区的小朋友还有几个。
我们将这些数量汇总到数组 garden 中。
请根据这些小朋友给出的信息，计算班级小朋友至少来自几个小区？

输入描述

输入：garden[] = {2, 2, 3}

备注

- garden 数组长度最大为 999
- 每个小区的小朋友数量最多 1000 人，也就是 garden[i] 的范围为 [0, 999]

输出描述

输出：7

用例1

输入

```
Plain Text
1  2 2 3
```

输出

```
Plain Text
1  7
```

说明

> 第一个小朋友反馈有两个小朋友和自己同一小区，即此小区有3个小朋友。第二个小朋友反馈有两个小朋友和自己同一小区，即此小区有3个小朋友。这两个小朋友，可能是同一小区的，且此小区的小朋友只有3个人。第三个小区反馈还有3个小朋友与自己同一小区，则这些小朋友只能是另外一个小区的。这个小区有4个小朋友。

题解

> 这个题目描述的有点坑，从测试案例来看是求班级最少有多少个人。

思路: `贪心`

- 对于每个小朋友说的值(value)，实际小区的人数应该是(value + 1)，`贪心` 我们尽可能让值相同的小朋友处于同一个小区.

- 实际处理过程中使用 `map<int,int>` cnts统计出现相同值的数量。比如这个值为2，所有说2的小朋友能得出的最小人数为 `(ceil(cnts[2] * 1.0 / (2 + 1))) * (2 + 1)` 。
- 根据上述贪心思维，计算不通值的最少人数，累加即可。

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<cmath>
using namespace std;

// 通用 split 函数
vector<string> split(const string& str, const string& delimiter) {
    vector<string> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(str.substr(start));
    return result;
}

int main() {
    string s;
    getline(cin, s);
    vector<string> tmp = split(s, " ");
    int n = tmp.size();
    vector<int> ans(n);
    map<int, int> cnts;
    for (int i = 0; i < n; i++) {
        ans[i] = stoi(tmp[i]);
        cnts[ans[i]]++;
    }
    int res = 0;
    for (auto p : cnts) {
        // 代表小区应该有的人数
        int total = p.first + 1;
        // 贪心，让拥有朋友相同的情况，尽可能处于一个小区，向上取整
        res += (ceil(p.second * 1.0 / total)) * total ;
    }
```

```
    cout << res;
    return 0;

}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] input = scanner.nextLine().split(" ");
        scanner.close();

        int n = input.length;
        int[] ans = new int[n];
        Map<Integer, Integer> cnts = new HashMap<>();

        // 统计每个数的出现次数
        for (int i = 0; i < n; i++) {
            ans[i] = Integer.parseInt(input[i]);
            cnts.put(ans[i], cnts.getOrDefault(ans[i], 0) + 1);
        }

        int res = 0;
        for (Map.Entry<Integer, Integer> entry : cnts.entrySet()) {
            // 小区实际的人数
            int total = entry.getKey() + 1;
            // 贪心，尽可能让小朋友处于一个或多个小区。  Math.ceil(entry.getValue() * 1.0 / total) 最少小区的数量
            res += Math.ceil(entry.getValue() * 1.0 / total) * total;
        }

        System.out.println(res);
    }
}
```

Python

```
import sys
import math

def main():
    # 读取输入并转换为整数列表
    ans = list(map(int, sys.stdin.readline().strip().split()))

    # 统计每个数的出现次数
    cnts = {}
    for num in ans:
        cnts[num] = cnts.get(num, 0) + 1

    res = 0
    for k, v in cnts.items():
        # 小区人数数量
        total = k + 1
        # 贪心尽可能让小朋友处于一个或多个小区内  math.ceil(v / total) 最少小区数量
        res += math.ceil(v / total) * total

    print(res)

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.on("line", (line) => {
    const ans = line.trim().split(" ").map(Number);

    // 统计每个数的出现次数
    const cnts = new Map();
    ans.forEach(num => {
        cnts.set(num, (cnts.get(num) || 0) + 1);
    });

    let res = 0;
    cnts.forEach((value, key) => {
        // 小区实际人数
        let total = key + 1;
        // 贪心尽可能让这些小朋友分配到一个或多个小区内   Math.ceil(value / total) 最小需要的小区数量
        res += Math.ceil(value / total) * total;
    });

    console.log(res);
    rl.close();
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "math"
  "os"
  "strconv"
  "strings"
)

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  input := strings.Split(scanner.Text(), " ")

  // 统计每个数的出现次数
  cnts := make(map[int]int)
  for _, s := range input {
    num, _ := strconv.Atoi(s)
    cnts[num]++
  }

  res := 0
  for k, v := range cnts {
    // 小区实际人数
    total := k + 1
    // 尽可能让这些小朋友处于一个或多个小区 int(math.Ceil(float64(v)/float64(total))) 需要最少的小区数量
    res += int(math.Ceil(float64(v)/float64(total))) * total
  }

  fmt.Println(res)
}
```

> 来自: 华为OD机考2025C卷 – 小朋友来自多少个小区 / 最少有多少个小朋友 (C++ & Python & JAVA & JS & GO)-CSDN博客

> 来自: 华为OD机考2025C卷 – 小朋友来自多少个小区 / 最少有多少个小朋友 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 最富裕的小家庭 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025C卷 最富裕的小家庭

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

在一颗树中，每个节点代表一个家庭成员，节点的数字表示其个人的财富值，一个节点及其直接相连的子节点被定义为一个小家庭。
现给你一颗树，请计算出最富裕的小家庭的财富和。

输入描述

第一行为一个数 N，表示成员总数，成员编号 1~N。1 ⩽ N ⩽ 1000
第二行为 N 个空格分隔的数，表示编号 1~N 的成员的财富值。0 ⩽ 财富值 ⩽ 1000000
接下来 N −1 行，每行两个空格分隔的整数（N1, N2），表示 N1 是 N2 的父节点。

输出描述

最富裕的小家庭的财富和

用例1

输入

Plain Text

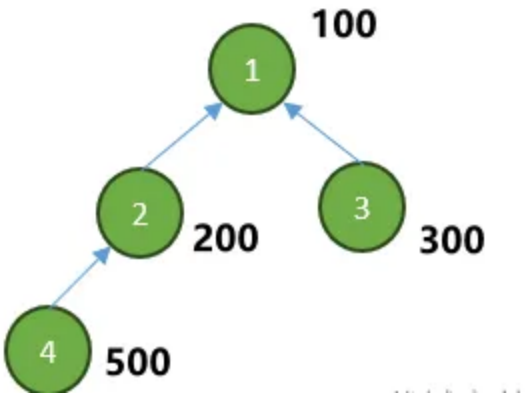
```
4
100 200 300 500
1 2
1 3
2 4
```

输出

Plain Text

```
700
```

说明



> 成员1，2，3 组成的小家庭财富值为600
> 成员2，4 组成的小家庭财富值为700

题解

思路：简单的 `数据结构` 练习题。

1. 使用哈希表 `idValue` (key 为编号 value 为财富值)映射id和财富的值，使用哈希表 `childrenMp` (key 为编号 value 为指定编号的子节点编号集合)映射id和子节点的关系。
2. 接下来就是遍历所有id，计算每个家庭财富(借助 `idValue` 和 `childrenMp` 实现) 自身的子节点财富和，记录其中的最大值，就是结果。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<set>
#include<map>
using namespace std;

int main() {
    int n;
    cin >> n;
    // 映射id 和财富
    map<int, int> idValue;
    for (int i = 1; i <= n; i++) {
        int value;
        cin >> value;
        idValue[i] = value;
    }
    // 映射 id和子节点关系
    map<int, set<int>> childrenMp;
    for (int i = 0; i < n - 1; i++) {
        int n1, n2;
        cin >> n1 >> n2;
        childrenMp[n1].insert(n2);
    }

    int res = 0;
    for (auto p : idValue) {
        int tmp = p.second;
        set<int> children = childrenMp[p.first];
        // 额外加上子节点的财富
        if (!children.empty()) {
            for (auto id : children) {
                tmp += idValue[id];
            }
        }
        res = max(res, tmp);
    }

    cout << res;
    return 0;
```

```
46    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        // 映射id 和财富
        Map<Integer, Integer> idValue = new HashMap<>();
        for (int i = 1; i <= n; i++) {
            int value = sc.nextInt();
            idValue.put(i, value);
        }

        // 映射 id和子节点关系
        Map<Integer, Set<Integer>> childrenMap = new HashMap<>();
        for (int i = 0; i < n - 1; i++) {
            int n1 = sc.nextInt();
            int n2 = sc.nextInt();
            childrenMap.putIfAbsent(n1, new HashSet<>());
            childrenMap.putIfAbsent(n2, new HashSet<>());
            childrenMap.get(n1).add(n2);
        }

        int res = 0;
        // 遍历每个节点计算财富
        for (Map.Entry<Integer, Integer> entry : idValue.entrySet()) {
            int tmp = entry.getValue();
            Set<Integer> children = childrenMap.get(entry.getKey());
            // 额外加上子节点的财富
            if (children != null) {
                for (int id : children) {
                    tmp += idValue.get(id);
                }
            }
            res = Math.max(res, tmp);
        }

        System.out.println(res);
    }
}
```

Python

Plain Text

```
def main():
    n = int(input())

    # 映射id 和财富
    id_value = {}
    wealth = list(map(int, input().split()))
    for i in range(1, n + 1):
        id_value[i] = wealth[i - 1]

    # 映射 id和子节点关系
    children_map = {i: [] for i in range(1, n + 1)}
    for _ in range(n - 1):
        n1, n2 = map(int, input().split())
        children_map[n1].append(n2)

    res = 0
    # 遍历每个节点计算财富
    for node, value in id_value.items():
        tmp = value
        children = children_map[node]
        # 额外加上子节点的财富
        for child in children:
            tmp += id_value[child]
        res = max(res, tmp)

    print(res)

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputs = [];

rl.on("line", (line) => {
    inputs.push(line.trim());
    if (inputs.length === Number(inputs[0]) + 1) {
        main(inputs);
        rl.close();
    }
});

function main(inputs) {
    const n = parseInt(inputs[0]);

    // 映射id 和财富
    const idValue = {};
    const wealth = inputs[1].split(" ").map(Number);
    for (let i = 1; i <= n; i++) {
        idValue[i] = wealth[i - 1];
    }

    // 映射 id和子节点关系
    const childrenMap = {};
    for (let i = 0; i < n - 1; i++) {
        const [n1, n2] = inputs[2 + i].split(" ").map(Number);
        if (!childrenMap[n1]) childrenMap[n1] = [];
        if (!childrenMap[n2]) childrenMap[n2] = [];
        childrenMap[n1].push(n2);
    }

    let res = 0;
    // 遍历每个节点计算财富
    for (let node in idValue) {
        let tmp = idValue[node];
        const children = childrenMap[node] || [];
        // 额外加上子节点的财富
        children.forEach(child => {
            tmp += idValue[child];
        });
```

```
            res = Math.max(res, tmp);
        }

        console.log(res);
    }
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math"
)

func main() {
  var n int
  fmt.Scan(&n)

  // 映射id 和财富
  idValue := make(map[int]int)
  var wealth []int
  for i := 1; i <= n; i++ {
    var value int
    fmt.Scan(&value)
    wealth = append(wealth, value)
    idValue[i] = wealth[i-1]
  }

  // 映射 id和子节点关系
  childrenMap := make(map[int][]int)
  for i := 0; i < n-1; i++ {
    var n1, n2 int
    fmt.Scan(&n1, &n2)
    childrenMap[n1] = append(childrenMap[n1], n2)
  }

  res := 0
  // 遍历每个节点计算财富
  for node, value := range idValue {
    tmp := value
    children := childrenMap[node]
    // 额外加上子节点的财富
    for _, child := range children {
      tmp += idValue[child]
    }
    res = int(math.Max(float64(res), float64(tmp)))
  }

  fmt.Println(res)
}
```

华为OD机试 2025C卷 最富裕的小家庭

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

在一颗树中，每个节点代表一个家庭成员，节点的数字表示其个人的财富值，一个节点及其直接相连的子节点被定义为一个小家庭。
现给你一颗树，请计算出最富裕的小家庭的财富和。

输入描述

第一行为一个数 N，表示成员总数，成员编号 1~N。1 ⩽ N ⩽ 1000
第二行为 N 个空格分隔的数，表示编号 1~N 的成员的财富值。0 ⩽ 财富值 ⩽ 1000000
接下来 N -1 行，每行两个空格分隔的整数（N1, N2），表示 N1 是 N2 的父节点。

输出描述

最富裕的小家庭的财富和

用例1

输入

Plain Text

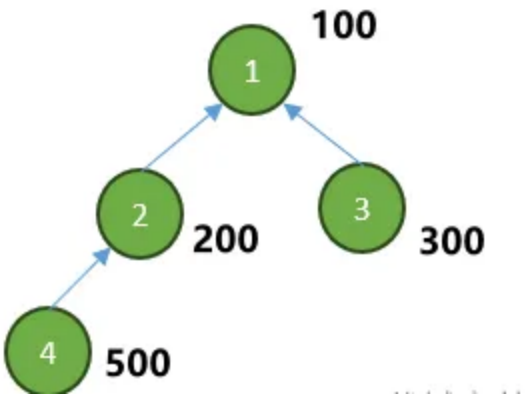
```
4
100 200 300 500
1 2
1 3
2 4
```

输出

Plain Text

```
700
```

说明



> 成员1，2，3 组成的小家庭财富值为600
> 成员2，4 组成的小家庭财富值为700

题解

思路：简单的 `数据结构` 练习题。

1. 使用哈希表 `idValue` (key 为编号 value 为财富值)映射id和财富的值，使用哈希表 `childrenMp` (key 为编号 value 为指定编号的子节点编号集合)映射id和子节点的关系。
2. 接下来就是遍历所有id，计算每个家庭财富(借助 `idValue` 和 `childrenMp` 实现) 自身的子节点财富和，记录其中的最大值，就是结果。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<set>
#include<map>
using namespace std;

int main() {
    int n;
    cin >> n;
    // 映射id 和财富
    map<int, int> idValue;
    for (int i = 1; i <= n; i++) {
        int value;
        cin >> value;
        idValue[i] = value;
    }
    // 映射 id和子节点关系
    map<int, set<int>> childrenMp;
    for (int i = 0; i < n - 1; i++) {
        int n1, n2;
        cin >> n1 >> n2;
        childrenMp[n1].insert(n2);
    }

    int res = 0;
    for (auto p : idValue) {
        int tmp = p.second;
        set<int> children = childrenMp[p.first];
        // 额外加上子节点的财富
        if (!children.empty()) {
            for (auto id : children) {
                tmp += idValue[id];
            }
        }
        res = max(res, tmp);
    }

    cout << res;
    return 0;
```

```
46    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        // 映射id 和财富
        Map<Integer, Integer> idValue = new HashMap<>();
        for (int i = 1; i <= n; i++) {
            int value = sc.nextInt();
            idValue.put(i, value);
        }

        // 映射 id和子节点关系
        Map<Integer, Set<Integer>> childrenMap = new HashMap<>();
        for (int i = 0; i < n - 1; i++) {
            int n1 = sc.nextInt();
            int n2 = sc.nextInt();
            childrenMap.putIfAbsent(n1, new HashSet<>());
            childrenMap.putIfAbsent(n2, new HashSet<>());
            childrenMap.get(n1).add(n2);
        }

        int res = 0;
        // 遍历每个节点计算财富
        for (Map.Entry<Integer, Integer> entry : idValue.entrySet()) {
            int tmp = entry.getValue();
            Set<Integer> children = childrenMap.get(entry.getKey());
            // 额外加上子节点的财富
            if (children != null) {
                for (int id : children) {
                    tmp += idValue.get(id);
                }
            }
            res = Math.max(res, tmp);
        }

        System.out.println(res);
    }
}
```

Python

```
def main():
    n = int(input())

    # 映射id 和财富
    id_value = {}
    wealth = list(map(int, input().split()))
    for i in range(1, n + 1):
        id_value[i] = wealth[i - 1]

    # 映射 id和子节点关系
    children_map = {i: [] for i in range(1, n + 1)}
    for _ in range(n - 1):
        n1, n2 = map(int, input().split())
        children_map[n1].append(n2)

    res = 0
    # 遍历每个节点计算财富
    for node, value in id_value.items():
        tmp = value
        children = children_map[node]
        # 额外加上子节点的财富
        for child in children:
            tmp += id_value[child]
        res = max(res, tmp)

    print(res)

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputs = [];

rl.on("line", (line) => {
    inputs.push(line.trim());
    if (inputs.length === Number(inputs[0]) + 1) {
        main(inputs);
        rl.close();
    }
});

function main(inputs) {
    const n = parseInt(inputs[0]);

    // 映射id 和财富
    const idValue = {};
    const wealth = inputs[1].split(" ").map(Number);
    for (let i = 1; i <= n; i++) {
        idValue[i] = wealth[i - 1];
    }

    // 映射 id和子节点关系
    const childrenMap = {};
    for (let i = 0; i < n - 1; i++) {
        const [n1, n2] = inputs[2 + i].split(" ").map(Number);
        if (!childrenMap[n1]) childrenMap[n1] = [];
        if (!childrenMap[n2]) childrenMap[n2] = [];
        childrenMap[n1].push(n2);
    }

    let res = 0;
    // 遍历每个节点计算财富
    for (let node in idValue) {
        let tmp = idValue[node];
        const children = childrenMap[node] || [];
        // 额外加上子节点的财富
        children.forEach(child => {
            tmp += idValue[child];
        });
```

```
            res = Math.max(res, tmp);
        }

        console.log(res);
    }
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math"
)

func main() {
  var n int
  fmt.Scan(&n)

  // 映射id 和财富
  idValue := make(map[int]int)
  var wealth []int
  for i := 1; i <= n; i++ {
    var value int
    fmt.Scan(&value)
    wealth = append(wealth, value)
    idValue[i] = wealth[i-1]
  }

  // 映射 id和子节点关系
  childrenMap := make(map[int][]int)
  for i := 0; i < n-1; i++ {
    var n1, n2 int
    fmt.Scan(&n1, &n2)
    childrenMap[n1] = append(childrenMap[n1], n2)
  }

  res := 0
  // 遍历每个节点计算财富
  for node, value := range idValue {
    tmp := value
    children := childrenMap[node]
    // 额外加上子节点的财富
    for _, child := range children {
      tmp += idValue[child]
    }
    res = int(math.Max(float64(res), float64(tmp)))
  }

  fmt.Println(res)
}
```

> 来自: 华为OD机考2025C卷 – 最富裕的小家庭 (C++ & Python & JAVA & JS & GO)-CSDN博客

> 来自: 华为OD机考2025C卷 – 最富裕的小家庭 (C++ & Python & JAVA & JS & GO)-CSDN博客