t0729

华为OD机考2025C卷 - Excel单元格数值统计 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 网上商城优惠活动 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 无向图染色 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 数值同化 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - Excel单元格数值统计 (C++ & Python & JAVA & JS & GO)-CSDN博客

Excel单元格数值统计

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

Excel工作表中对选定区域的数值进行统计的功能非常实用。
仿照Excel的这个功能，请对给定表格中选中区域中的单元格进行求和统计，并输出统计结果。
为简化计算，假设当前输入中每个单元格内容仅为数字或公式两种。
如果为数字，则是一个非负整数，形如3、77
如果为公式，则固定以=开头，且仅包含下面三种情况：

- 等于某单元格的值，例如=B12
- 两个单元格的双目运算（仅为+或−），形如=C1−C2、C3+B2
- 单元格和数字的双目运算（仅为+或−），形如=B1+1、100−B2

注意：

1. 公式内容都是合法的，例如不存在，=C+1、=C1−C2+B3,=5、=3+5
2. 不存在循环引用，例如A1=B1+C1、C1=A1+B2
3. 内容中不存在空格、括号

输入描述

第一行两个整数rows cols，表示给定表格区域的行数和列数，1<=rows<=20，1<=cols<=26。
接下来rows行，每行cols个以空格分隔的字符串，表示给定表格values的单元格内容。
最后一行输入的字符串，表示给定的选中区域，形如A1:C2。

输出描述

一个整数，表示给定选中区域各单元格中数字的累加总和，范围−2,147,483,648 ~ 2,147,483,647

备注

- 表格的行号120，列号AZ，例如单元格B3对应values[2][1]。
- 输入的单元格内容（含公式）中数字均为十进制，值范围[0, 100]。
- 选中区域：冒号左侧单元格表示选中区域的左上角，右侧表示右下角，如可以为B2:C10、B2:B5、B2:Y2、B2:B2,无类似C2:B2、C2:A1的输入。

用例1

输入

```
1 3
1 =A1+C1 3
A1:C1
```

输出

```
8
```

用例2

输入

```
5 3
10 12 =C5
15 5 6
7 8 =3+C2
6 =B2-A1 =C2
7 5 3
B2:C4
```

输出

```
29
```

题解

思路：`字符串 + DFS`

1. 使用二维数组存储每个单元格输入值，接收选中区域字符串。
2. 接下选中区域字符串，确定需要累加的区域。`B2:C4` 这种可以先按 `:` 进行切割，每部分都转换为数组索引，解析过程中需要注意索引转换(excel中坐标都是1-base，数组索引是0-base)。
3. 确定好累加区域之后,使用循环累加区域内单元格的值，这里具体就涉及单元值解析，说一下不同类型

单元格值的处理过程：

a. 单元格值第一个不是 `=` ，那就说明就是一个正整数形式的字符串，直接转换整数即可。
b. 单元格第一个是 `=` 分为三种情况
    i. 等于另一个单元格的值，此时需要进行 `递归` 获取对应位置值。
    ii. 加法运算: 加法运算有两个操作数，其中操作数可能为 `另一个单元格值` 或者 `整数形式字符串` 。将两个操作数转换为具体的值。再进行相加返回
    iii. 减法运算：减法运算有两个操作数，其中操作数可能为 `另一个单元格值` 或者 `整数形式字符串` 。将两个操作数转换为具体的值。再进行相减返回。

4. 明白上面说，接下来就是进行编码实现即可。

C++

Plain Text

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


vector<int> getParseValue(vector<vector<string>> &table, vector<vector<in
t>> &tableValue, vector<string>& parts);

// 通用 切割函数 函数 将字符串str根据delimiter进行切割
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;
}

// 转换为横坐标和纵坐标
vector<int> getPos(string s) {
    vector<int> res(2);
    res[1] = s[0] - 'A';
    res[0] =  stoi(s.substr(1, s.size() -1)) - 1;
    return res;
}

// 判断字符串是不是数字形式
bool isNumber(const string& s) {
    if (s.empty()) return false;
    for (char c : s) {
        if (!isdigit(c)) return false;
    }
    return true;
}
```

```
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
```

```
// 获取指定网格位置的值
int getCellValue(vector<vector<string>> &table, int x, int y, vector<vect
or<int>> &tableValue)  {
    // 之前已经计算过了 缓存优化
    if (tableValue[x][y] != -1) {
        return tableValue[x][y];
    }
    int res = 0;

    string cell = table[x][y];
    // 涉及到计算
    if (cell[0] == '=') {
        cell = cell.substr(1, cell.size() - 1);
        // 加法运算
        if (cell.find('+') != string::npos) {
            vector<string> parts = split(cell, "+");
            vector<int> parseValue = getParseValue(table, tableValue, par
ts);
            res = parseValue[0] + parseValue[1];
        // 减法运算
        } else if (cell.find('-') != string::npos) {
            vector<string> parts = split(cell, "-");
            vector<int> parseValue = getParseValue(table, tableValue, par
ts);
            res = parseValue[0] - parseValue[1];
        // 等于另一个单元格
        } else {
          vector<int> nextPos = getPos(cell);
          res = getCellValue(table, nextPos[0], nextPos[1], tableValue);
        }
    // 正整数
    } else {
        res = stoi(cell);
    }
    tableValue[x][y] = res;
    return res;
}


// 解析两个操作数 如果某一个操作时单元格位置则进行递归获取 或者直接解析
vector<int> getParseValue(vector<vector<string>> &table, vector<vector<in
t>> &tableValue, vector<string>& parts) {
    string operatorNum1 = parts[0];
    string operatorNum2 = parts[1];
    vector<int> res(2);
```

```
    // 数字直接解析 operatorNum2处理类似
    if (isNumber(operatorNum1)) {
        res[0] = stoi(operatorNum1);
    } else {
        vector<int> rowAndCol = getPos(operatorNum1);
        int x = rowAndCol[0];
        int y = rowAndCol[1];
        // 递归获取值
        res[0] = getCellValue(table, x, y, tableValue);
        tableValue[x][y] = res[0];
    }

    if (isNumber(operatorNum2)) {
        res[1] = stoi(operatorNum2);
    } else {
        vector<int> rowAndCol = getPos(operatorNum2);
        int x = rowAndCol[0];
        int y = rowAndCol[1];
        res[1] = getCellValue(table, x, y, tableValue);
        tableValue[x][y] = res[1];
    }
    return res;
}


int main() {
    int m,n;
    cin >> m >> n;
    vector<vector<string>> table(m,vector<string>(n));
    vector<vector<int>> tableValue(m, vector<int>(n, -1));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> table[i][j];
        }
    }
    string ranges;
    cin >> ranges;
    vector<string> startAndEnd = split(ranges, ":");
    string start = startAndEnd[0];
    string end = startAndEnd[1];
    // 获取左上角坐标
    vector<int> startRowCol = getPos(start);
    // 获取右下角坐标
    vector<int> endRowCol = getPos(end);
    int sum = 0;
    // 对左上角和右下角的值累加求和
    for (int i = startRowCol[0]; i <= endRowCol[0]; i++) {
        for (int j = startRowCol[1]; j <= endRowCol[1]; j++) {
```

```
                sum += getCellValue(table, i, j, tableValue);
            }
        }
        cout << sum;
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    static int[][] tableValue;
    static String[][] table;

    // 获取某个位置单元格的值（含递归和缓存）
    static int getCellValue(int x, int y) {
        if (tableValue[x][y] != -1) return tableValue[x][y];
        String cell = table[x][y];
        int res = 0;
        // 公式
        if (cell.startsWith("=")) {
            cell = cell.substring(1);
            // 加法计算
            if (cell.contains("+")) {
                String[] parts = cell.split("\\+");
                int a = parseOperand(parts[0]);
                int b = parseOperand(parts[1]);
                res = a + b;
            // 减法计算
            } else if (cell.contains("-")) {
                String[] parts = cell.split("-");
                int a = parseOperand(parts[0]);
                int b = parseOperand(parts[1]);
                res = a - b;
            // 等于某个单元格的值
            } else {
                int[] pos = getPos(cell);
                res = getCellValue(pos[0], pos[1]);
            }
        // 数字
        } else {
            res = Integer.parseInt(cell);
        }
        tableValue[x][y] = res;
        return res;
    }

    // 解析数字或单元格引用
    static int parseOperand(String s) {
        if (s.matches("\\d+")) return Integer.parseInt(s);
        int[] pos = getPos(s);
        return getCellValue(pos[0], pos[1]);
    }
```

```
46
47      // A1 转换成坐标
48      static int[] getPos(String s) {
49          int col = s.charAt(0) - 'A';
50          int row = Integer.parseInt(s.substring(1)) - 1;
51          return new int[]{row, col};
52      }
53
54      public static void main(String[] args) {
55          Scanner sc = new Scanner(System.in);
56          int m = sc.nextInt(), n = sc.nextInt();
57          table = new String[m][n];
58          tableValue = new int[m][n];
59          for (int i = 0; i < m; i++)
60              for (int j = 0; j < n; j++) {
61                  table[i][j] = sc.next();
62                  tableValue[i][j] = -1;
63              }
64
65          String[] range = sc.next().split(":");
66          int[] start = getPos(range[0]), end = getPos(range[1]);
67          int sum = 0;
68          for (int i = start[0]; i <= end[0]; i++)
69              for (int j = start[1]; j <= end[1]; j++)
70                  sum += getCellValue(i, j);
71          System.out.println(sum);
72      }
73  }
```

Python

Plain Text

```
import sys
sys.setrecursionlimit(10000)

def get_pos(s):
    return [int(s[1:]) - 1, ord(s[0]) - ord('A')]

def is_number(s):
    return s.isdigit()
# 获取某个单元格的值
def get_cell_value(table, table_value, x, y):
    # 缓存使用
    if table_value[x][y] != -1:
        return table_value[x][y]
    cell = table[x][y]
    # 公式
    if cell.startswith("="):
        cell = cell[1:]
        # 加法计算
        if "+" in cell:
            a, b = cell.split("+")
            val1 = parse_operand(table, table_value, a)
            val2 = parse_operand(table, table_value, b)
            res = val1 + val2
        # 减法计算
        elif "-" in cell:
            a, b = cell.split("-")
            val1 = parse_operand(table, table_value, a)
            val2 = parse_operand(table, table_value, b)
            res = val1 - val2
        else:
            row, col = get_pos(cell)
            res = get_cell_value(table, table_value, row, col)
    # 单个值
    else:
        res = int(cell)
    # 缓存
    table_value[x][y] = res
    return res

# 解析运算中的操作数
def parse_operand(table, table_value, operand):
    if is_number(operand):
        return int(operand)
    row, col = get_pos(operand)
    return get_cell_value(table, table_value, row, col)
```

```
46
47  if __name__ == "__main__":
48      m, n = map(int, input().split())
49      table = [input().split() for _ in range(m)]
50      table_value = [[-1]*n for _ in range(m)]
51      start, end = input().split(":")
52      sr, sc = get_pos(start)
53      er, ec = get_pos(end)
54      total = 0
55      for i in range(sr, er + 1):
56          for j in range(sc, ec + 1):
57              total += get_cell_value(table, table_value, i, j)
58      print(total)
```

JavaScript

Plain Text

```
// 转换为数组索引
function getPos(s) {
    return [parseInt(s.slice(1)) - 1, s.charCodeAt(0) - 'A'.charCodeAt
(0)];
}

function isNumber(s) {
    return /^\d+$/.test(s);
}
// 获取指定位置值
function getCellValue(table, tableValue, x, y) {
    // 利用缓存
    if (tableValue[x][y] !== -1) return tableValue[x][y];
    let cell = table[x][y];
    let res = 0;
    // 公式
    if (cell.startsWith("=")) {
        cell = cell.slice(1);
        // 加法
        if (cell.includes("+")) {
            let [a, b] = cell.split("+");
            let val1 = parseOperand(table, tableValue, a);
            let val2 = parseOperand(table, tableValue, b);
            res = val1 + val2;
        // 减法
        } else if (cell.includes("-")) {
            let [a, b] = cell.split("-");
            let val1 = parseOperand(table, tableValue, a);
            let val2 = parseOperand(table, tableValue, b);
            res = val1 - val2;
        // =单元格
        } else {
            let [r, c] = getPos(cell);
            res = getCellValue(table, tableValue, r, c);
        }
    // 单个值
    } else {
        res = parseInt(cell);
    }
    tableValue[x][y] = res;
    return res;
}
// 解析操作数
function parseOperand(table, tableValue, s) {
    if (isNumber(s)) return parseInt(s);
```

```
    let [r, c] = getPos(s);
    return getCellValue(table, tableValue, r, c);
}

// ACM-style input
const readline = require('readline');
const rl = readline.createInterface({ input: process.stdin });
let input = [];

rl.on('line', line => {
    input.push(line.trim());
}).on('close', () => {
    let [m, n] = input[0].split(' ').map(Number);
    let table = [], tableValue = Array.from({ length: m }, () => Array(n).fill(-1));
    for (let i = 0; i < m; i++) table.push(input[i + 1].split(' '));
    let [start, end] = input[m + 1].split(':');
    let [sr, sc] = getPos(start);
    let [er, ec] = getPos(end);
    let sum = 0;
    // 累加左上角右下角区域值
    for (let i = sr; i <= er; i++) {
        for (let j = sc; j <= ec; j++) {
            sum += getCellValue(table, tableValue, i, j);
        }
    }
    console.log(sum);
});
```

Go

Plain Text

```
package main

import (
  "fmt"
  "strconv"
  "strings"
)
// 将坐标转换为数组索引
func getPos(s string) (int, int) {
  col := int(s[0] - 'A')
  row, _ := strconv.Atoi(s[1:])
  return row - 1, col
}
// 判断是否为数字
func isNumber(s string) bool {
  _, err := strconv.Atoi(s)
  return err == nil
}
// 获取指定表格的值
func getCellValue(table [][]string, tableValue [][]int, x, y int) int {
  // 利用缓存
  if tableValue[x][y] != -1 {
    return tableValue[x][y]
  }
  cell := table[x][y]
  res := 0
  // 公式
  if strings.HasPrefix(cell, "=") {
    cell = cell[1:]
    // 加法
    if strings.Contains(cell, "+") {
      parts := strings.Split(cell, "+")
      a := parseOperand(table, tableValue, parts[0])
      b := parseOperand(table, tableValue, parts[1])
      res = a + b
      // 减法
    } else if strings.Contains(cell, "-") {
      parts := strings.Split(cell, "-")
      a := parseOperand(table, tableValue, parts[0])
      b := parseOperand(table, tableValue, parts[1])
      res = a - b

    } else {
      r, c := getPos(cell)
      res = getCellValue(table, tableValue, r, c)
```

```
    }
  } else {
    res, _ = strconv.Atoi(cell)
  }
  // 缓存
  tableValue[x][y] = res
  return res
}

// 接下操作数
func parseOperand(table [][]string, tableValue [][]int, s string) int {
  if isNumber(s) {
    val, _ := strconv.Atoi(s)
    return val
  }
  r, c := getPos(s)
  return getCellValue(table, tableValue, r, c)
}

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

  table := make([][]string, m)
  tableValue := make([][]int, m)
  for i := range table {
    table[i] = make([]string, n)
    tableValue[i] = make([]int, n)
    for j := 0; j < n; j++ {
      fmt.Scan(&table[i][j])
      tableValue[i][j] = -1
    }
  }
  var rng string
  fmt.Scan(&rng)
  parts := strings.Split(rng, ":")
  sr, sc := getPos(parts[0])
  er, ec := getPos(parts[1])

  sum := 0
  // 累加选中区域值
  for i := sr; i <= er; i++ {
    for j := sc; j <= ec; j++ {
      sum += getCellValue(table, tableValue, i, j)
    }
  }
  fmt.Println(sum)
}
```

> 来自: 华为OD机考2025C卷 – Excel单元格数值统计 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 网上商城优惠活动 (C++ & Python & JAVA & JS & GO)-CSDN博客

网上商城优惠活动

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

某网上商场举办优惠活动，发布了满减、打折、无门槛3种优惠券。
分别为：每满100元优惠10元，无使用数限制，如100 ~ 199元可以使用1张减10元，200 ~ 299可使用2张减20元，以此类推； 92折券，1次限使用1张，如100元，则优惠后为92元；无门槛5元优惠券，无使用数限制，直接减5元。 优惠券使用限制每次最多使用2种优惠券，2种优惠可以叠加（优惠叠加时以优惠后的价格计算），以购物200元为例，可以先用92折券优惠到184元，再用1张满减券优惠10元，最终价格是174元，也可以用满减券2张优惠20元为180元，再使用92折券优惠到165（165.6向下取整），不同使用顺序的优惠价格不同，以最优惠价格为准。在一次购物种，同一类型优惠券使用多张时必须一次性使用，不能分多次拆开使用（不允许先使用1张满减券，再用打折券，再使用一张满减券）。
问题请设计实现一种解决方法，帮助购物者以最少的优惠券获得最优的优惠价格。优惠后价格越低越好，同等优惠价格，使用的优惠券越少越好，可以允许某次购物不使用优惠券。 约定优惠活动每人只能参加一次，每个人的优惠券种类和数量是一样的。

输入描述

- 第一行：每个人拥有的优惠券数量（数量取值范围为[0,10]），按满减、打折、无门槛的顺序输入
- 第二行：表示购物的人数n（1 ⩽ n ⩽ 10000）
- 最后n行：每一行表示某个人优惠前的购物总价格（价格取值范围(0, 1000]，都为整数）。
- 约定：输入都是符合题目设定的要求的

输出描述

- 每行输出每个人每次购物优惠后的最低价格以及使用的优惠券总数量 每行的输出顺序和输入的顺序保持一致
- 备注: 优惠券数量都为整数，取值范围为[0, 10] 购物人数为整数，取值范围为[1, 10000] 优惠券的购物总价为整数，取值范围为 (0, 1000] 优惠后价格如果是小数，则向下取整，输出都为整数

用例1

输入

```
3 2 5
3
100
200
400
```

输出

```
65 6
155 7
338 4
```

题解

思路： 模拟

1. 三种折扣方案，可以定义三个方法去实现。
2. 模拟三种折扣方案进行两两组合(注意顺序)，分别计算能够得到的最低折扣和使用优惠券数量。记录其中出现过的最低折扣以及使用优惠券的数量。
3. 输出 最低折扣值和 对应使用优惠券的数量。

Plain Text

```
#include <iostream>
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>

using namespace std;

// 满减
vector<int> manjian(int m, int price) {
    int res = price - min((price / 100), m) * 10;
    return {res, min((price / 100), m)};
}

// 打折
vector<int> dazhe(int m, int price) {
    int res = floor(price * pow(0.92, min(1, m)));
    return {res, min(1, m)};
}

// 无门槛
vector<int> wumenkan(int m, int price) {
    // 防止价格变为负数
    int used = min(m, price / 5);
    int res = price - 5 * used;

    return {res, used};
}

int main() {
    int manjianCount, dazheCount, wumenkanCount;
    cin >> manjianCount >> dazheCount >> wumenkanCount;

    int n;
    cin >> n;  // 读取商品个数
    vector<int> res;  // 存储每个商品的最优价格

    for (int i = 0; i < n; i++) {
        int tmp;
        cin >> tmp;
        int final_price = tmp;
        int min_count = 0;

        // **满减 + 打折**
        if (manjianCount > 0) {
```

```
            vector<int> res1 = manjian(manjianCount, tmp);
            vector<int> res2 = dazhe(dazheCount, res1[0]);
            if (res2[0] < final_price) {
                final_price = res2[0];
                min_count = res1[1] + res2[1];
            } else if (res2[0] == final_price) {
                min_count = min(min_count, res1[1] + res2[1]);
            }
        }

        // **打折 + 满减**
        if (dazheCount > 0 ) {
            vector<int> res1 = dazhe(dazheCount, tmp);
            vector<int> res2 = manjian(manjianCount, res1[0]);
            if (res2[0] < final_price) {
                final_price = res2[0];
                min_count = res1[1] + res2[1];
            } else if (res2[0] == final_price) {
                min_count = min(min_count, res1[1] + res2[1]);
            }
        }

        // **满减 + 无门槛**
        if (manjianCount > 0 ) {
            vector<int> res1 = manjian(manjianCount, tmp);
            vector<int> res2 = wumenkan(wumenkanCount, res1[0]);
            if (res2[0] < final_price) {
                final_price = res2[0];
                min_count = res1[1] + res2[1];
            } else if (res2[0] == final_price) {
                min_count = min(min_count, res1[1] + res2[1]);
            }
        }

        // **无门槛 + 满减**
        if (wumenkanCount > 0) {
            vector<int> res1 = wumenkan(wumenkanCount, tmp);
            vector<int> res2 = manjian(manjianCount, res1[0]);
            if (res2[0] < final_price) {
                final_price = res2[0];
                min_count = res1[1] + res2[1];
            } else if (res2[0] == final_price) {
                min_count = min(min_count, res1[1] + res2[1]);
            }
        }

        // **打折 + 无门槛**
        if (dazheCount > 0 ) {
```

```
            vector<int> res1 = dazhe(dazheCount, tmp);
            vector<int> res2 = wumenkan(wumenkanCount, res1[0]);
            if (res2[0] < final_price) {
                final_price = res2[0];
                min_count = res1[1] + res2[1];
            } else if (res2[0] == final_price) {
                min_count = min(min_count, res1[1] + res2[1]);
            }
        }

        // **无门槛 + 打折**
        if (wumenkanCount > 0 ) {
            vector<int> res1 = wumenkan(wumenkanCount, tmp);
            vector<int> res2 = dazhe(dazheCount, res1[0]);
            if (res2[0] < final_price) {
                final_price = res2[0];
                min_count = res1[1] + res2[1];
            } else if (res2[0] == final_price) {
                min_count = min(min_count, res1[1] + res2[1]);
            }
        }

        // **记录结果**
        cout << final_price << " " << min_count << endl;
    }

    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 满减
    static int[] manjian(int m, int price) {
        int used = Math.min(price / 100, m);
        return new int[]{price - used * 10, used};
    }

    // 打折
    static int[] dazhe(int m, int price) {
        int used = Math.min(1, m);
        return new int[]{(int) Math.floor(price * Math.pow(0.92, used)), u
sed};
    }

    // 无门槛
    static int[] wumenkan(int m, int price) {
        int used = Math.min(price / 5, m);
        return new int[]{price - used * 5, used};
    }

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

        // 读取满减、打折、无门槛优惠券的数量
        int manjianCount = scanner.nextInt();
        int dazheCount = scanner.nextInt();
        int wumenkanCount = scanner.nextInt();
        int n = scanner.nextInt(); // 读取商品个数

        for (int i = 0; i < n; i++) {
            int tmp = scanner.nextInt();
            int finalPrice = tmp;
            int minCount = 0; // 记录最少使用的优惠券数量

            // 遍历所有两两组合
            String[][] discounts = {
                    {"manjian", "dazhe"}, {"dazhe", "manjian"},
                    {"manjian", "wumenkan"}, {"wumenkan", "manjian"},
                    {"dazhe", "wumenkan"}, {"wumenkan", "dazhe"}
            };

            for (String[] d : discounts) {

```

```
                int[] res1 = d[0].equals("manjian") ? manjian(manjianCount, tmp) : d[0].equals("dazhe") ? dazhe(dazheCount, tmp) : wumenkan(wumenkanCount, tmp);
                int[] res2 = d[1].equals("dazhe") ? dazhe(dazheCount, res1[0]) : d[1].equals("manjian") ? manjian(manjianCount, res1[0]) : wumenkan(wumenkanCount, res1[0]);

                // 选择最优价格和最少使用的优惠券数量
                if (res2[0] < finalPrice) {
                    finalPrice = res2[0];
                    minCount = res1[1] + res2[1];
                } else if (res2[0] == finalPrice) {
                    minCount = Math.min(minCount, res1[1] + res2[1]);
                }
            }

            System.out.println(finalPrice + " " + minCount);
        }

        scanner.close();
    }
}
```

Python

Plain Text

```
import math
import sys

# 满减
def manjian(m, price):
    used = min(price // 100, m)
    return price - used * 10, used

# 打折
def dazhe(m, price):
    used = min(1, m)
    return math.floor(price * (0.92 ** used)), used

# 无门槛
def wumenkan(m, price):
    used = min(price // 5, m)
    return price - used * 5, used

def main():
    # 读取满减、打折、无门槛优惠券的数量
    manjian_count, dazhe_count, wumenkan_count = map(int, sys.stdin.readli
ne().split())
    n = int(sys.stdin.readline().strip())  # 读取商品个数

    for _ in range(n):
        tmp = int(sys.stdin.readline().strip())
        final_price = tmp
        min_count = 0  # 记录最少使用的优惠券数量

        # 遍历所有两两组合
        for first, second in [(manjian, dazhe), (dazhe, manjian),
                              (manjian, wumenkan), (wumenkan, manjian),
                              (dazhe, wumenkan), (wumenkan, dazhe)]:
            # 先使用第一种优惠
            res1, c1 = first(manjian_count if first == manjian else dazhe_
count if first == dazhe else wumenkan_count, tmp)
            # 再使用第二种优惠
            res2, c2 = second(dazhe_count if second == dazhe else manjian_
count if second == manjian else wumenkan_count, res1)

            # 选择最优价格和最少使用的优惠券数量
            if res2 < final_price:
                final_price, min_count = res2, c1 + c2
            elif res2 == final_price:
                min_count = min(min_count, c1 + c2)
```

```
43
44          print(final_price, min_count)
45
46
47  if __name__ == "__main__":
        main()
```

JavaScript

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);
}).on("close", () => {
    const [manjianCount, dazheCount, wumenkanCount] = inputLines[0].split(" ").map(Number);
    const n = Number(inputLines[1]);
    let items = inputLines.slice(2).map(Number);

    // 满减
    function manjian(m, price) {
        let used = Math.min(Math.floor(price / 100), m);
        return [price - used * 10, used];
    }

    // 打折
    function dazhe(m, price) {
        let used = Math.min(1, m);
        return [Math.floor(price * Math.pow(0.92, used)), used];
    }

    // 无门槛
    function wumenkan(m, price) {
        let used = Math.min(Math.floor(price / 5), m);
        return [price - used * 5, used];
    }

    items.forEach(tmp => {
        let finalPrice = tmp;
        let minCount = 0;

        // 遍历所有两两组合
        const discountFuncs = [
            [manjian, dazhe], [dazhe, manjian],
            [manjian, wumenkan], [wumenkan, manjian],
            [dazhe, wumenkan], [wumenkan, dazhe]
        ];

```

```
        discountFuncs.forEach(([first, second]) => {
            let [res1, c1] = first(first === manjian ? manjianCount : first === dazhe ? dazheCount : wumenkanCount, tmp);
            let [res2, c2] = second(second === dazhe ? dazheCount : second === manjian ? manjianCount : wumenkanCount, res1);

            if (res2 < finalPrice) {
                finalPrice = res2;
                minCount = c1 + c2;
            } else if (res2 === finalPrice) {
                minCount = Math.min(minCount, c1 + c2);
            }
        });

        console.log(finalPrice, minCount);
    });
});
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math"
)

// 满减
func manjian(m, price int) (int, int) {
  used := min(price/100, m)
  return price - used*10, used
}

// 打折
func dazhe(m, price int) (int, int) {
  used := min(1, m)
  return int(math.Floor(float64(price) * math.Pow(0.92, float64(used)))),
used
}

// 无门槛
func wumenkan(m, price int) (int, int) {
  used := min(price/5, m)
  return price - used*5, used
}

// 取最小值
func min(a, b int) int {
  if a < b {
    return a
  }
  return b
}

func main() {
  var manjianCount, dazheCount, wumenkanCount, n int
  fmt.Scan(&manjianCount, &dazheCount, &wumenkanCount)
  fmt.Scan(&n)

  for i := 0; i < n; i++ {
    var tmp int
    fmt.Scan(&tmp)
    finalPrice := tmp
    minCount := 0 // 记录最少使用的优惠券数量

```

```
    // 遍历所有两两组合
    pairs := []struct {
      first  func(int, int) (int, int)
      firstM int
      second func(int, int) (int, int)
      secondM int
    }{
      {manjian, manjianCount, dazhe, dazheCount},
      {dazhe, dazheCount, manjian, manjianCount},
      {manjian, manjianCount, wumenkan, wumenkanCount},
      {wumenkan, wumenkanCount, manjian, manjianCount},
      {dazhe, dazheCount, wumenkan, wumenkanCount},
      {wumenkan, wumenkanCount, dazhe, dazheCount},
    }

    for _, p := range pairs {
      res1, c1 := p.first(p.firstM, tmp)
      res2, c2 := p.second(p.secondM, res1)

      if res2 < finalPrice {
        finalPrice = res2
        minCount = c1 + c2
      } else if res2 == finalPrice {
        minCount = min(minCount, c1+c2)
      }
    }

    fmt.Println(finalPrice, minCount)
  }
}
```

来自: 华为OD机试2025C卷 – 网上商城优惠活动 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 无向图染色 (C++ & Python & JAVA & JS & GO)-CSDN博客

无向图染色

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

给一个无向图染色，可以填红黑两种颜色，必须保证相邻两个节点不能同时为红色，输出有多少种不同的染色方案?

输入描述

第一行输入M(图中节点数) N(边数)
后续N行格式为：V1 V2表示一个V1到V2的边。
数据范围：1 <= M <= 15,0 <= N <= M * 3，不能保证所有节点都是连通的。

输出描述

输出一个数字表示染色方案的个数。

用例1

输入

Plain Text

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

输出

Plain Text

```
7
```

说明

> 4个节点，4条边，1号节点和2号节点相连，
> 2号节点和4号节点相连，3号节点和4号节点相连，
> 1号节点和3号节点相连，
> 若想必须保证相邻两个节点不能同时为红色，总共7种方案。

用例2

输入

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

输出

```Plain Text
4
```

用例3

输入

```Plain Text
4 3
1 2
2 3
3 4
```

输出

```Plain Text
8
```

用例4

输入

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

输出

```
8
```

题解

思路：`暴力枚举`，题目数据量比较少直接使用 `m <= 15` 直接顶点所有颜色状态分配状态然后判断是否合法。

1. 通过顶点数量确定 `枚举范围 [0 ,2^m - 1]`
2. 枚举的值代表一种分配方案，举个例子假设当前 `m = 3 枚举值为3 对应二进制位 011`，我们设定二进制为1代表分配为红色，0代表分配为白色。
3. 对于每个枚举值，合法性判断就是判断当前方案是否将相邻边同时分配为红色。假设有一个边对应两个顶点分别为 `x y`,枚举值为 `i`,对应判断表达式为 `(i >> (x-1) & 1) == 1 && (i >> (y-1) & 1) == 1`. 如果当前所有边的合法性判断都通过则结果 + 1.
4. 输出枚举完所有方案的结果。

C++

Plain Text

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

int main() {
    int n,m;
    // 节点 边
    cin >> m >> n;
    vector<pair<int, int>> edges(n);
    for (int i = 0; i < n; i++) {
        int x , y;
        cin >> x >> y;
        edges[i] = {x, y};
    }
    // 没有边所以说明每个节点都能取红或黑
    if (n == 0) {
        cout << pow(2, m);
        return 0;
    }

    int res = 0;
    // 保留枚举所有情况判断是否合法 二进制方式枚举 二进制对应位 1 代表取红 0 代表为 黑
    for (int i = 0; i < (int)pow(2,m); i++) {
        // 判断是否合法
        bool flag = true;
        for (int j = 0; j < n; j++) {
            int x = edges[j].first;
            int y = edges[j].second;
            // 是否相邻边同为红
            if ((i >> (x-1) & 1) == 1 && (i >> (y-1) & 1) == 1) {
                flag = false;
                break;
            }
        }
        if (flag) {
            res++;
        }
    }

    cout << res;
```

```
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // 节点数 m，边数 n
        int m = sc.nextInt();
        int n = sc.nextInt();
        int[][] edges = new int[n][2];
        for (int i = 0; i < n; i++) {
            edges[i][0] = sc.nextInt();
            edges[i][1] = sc.nextInt();
        }

        // 没有边，说明每个节点可以独立染色为红或黑
        if (n == 0) {
            System.out.println((int)Math.pow(2, m));
            return;
        }

        int res = 0;
        // 枚举所有染色情况，用二进制方式表示染色状态
        for (int i = 0; i < (1 << m); i++) {
            boolean flag = true;
            for (int[] edge : edges) {
                int x = edge[0], y = edge[1];
                // 相邻节点不能同时为红色（即都为1）
                if (((i >> (x - 1)) & 1) == 1 && ((i >> (y - 1)) & 1) ==
1) {
                    flag = false;
                    break;
                }
            }
            if (flag) res++;
        }
        System.out.println(res);
    }
}
```

Python

```
# 节点数 m，边数 n
m, n = map(int, input().split())
edges = [tuple(map(int, input().split())) for _ in range(n)]

# 没有边，每个节点可红或黑，共2^m种方案
if n == 0:
    print(2 ** m)
    exit()

res = 0
# 枚举所有染色情况，1表示红色，0表示黑色
for i in range(1 << m):
    valid = True
    for x, y in edges:
        if (i >> (x - 1)) & 1 and (i >> (y - 1)) & 1:
            valid = False
            break
    if valid:
        res += 1

print(res)
```

JavaScript

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);
}).on('close', () => {
    const [m, n] = inputLines[0].split(' ').map(Number); // 读取节点数和边数
    const edges = inputLines.slice(1).map(line => line.split(' ').map(Numb
er));

    if (n === 0) {
        console.log(2 ** m);  // 没有边，2^m 种方案
        return;
    }

    let res = 0;
    // 用二进制枚举所有染色方案
    for (let i = 0; i < (1 << m); i++) {
        let valid = true;
        for (const [x, y] of edges) {
            // 判断相邻节点是否都为红色（用1表示红色）
            if (((i >> (x - 1)) & 1) && ((i >> (y - 1)) & 1)) {
                valid = false;
                break;
            }
        }
        if (valid) res++;
    }

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

Go

Plain Text

```
package main

import (
  "fmt"
)

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

  edges := make([][2]int, n)
  for i := 0; i < n; i++ {
    fmt.Scan(&edges[i][0], &edges[i][1])
  }

  // 没有边，每个节点可以红或黑，总共 2^m 种方案
  if n == 0 {
    fmt.Println(1 << m)
    return
  }

  res := 0
  // 枚举所有染色方案，1 表示红色
  for i := 0; i < (1 << m); i++ {
    valid := true
    for _, edge := range edges {
      x, y := edge[0], edge[1]
      if ((i>>(x-1))&1) == 1 && ((i>>(y-1))&1) == 1 {
        valid = false
        break
      }
    }
    if valid {
      res++
    }
  }

  fmt.Println(res)
}
```

来自: 华为OD机考2025C卷 – 无向图染色 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 数值同化 (C++ & Python & JAVA & JS & GO)-CSDN博客

数值同化

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

存在一个 m*n 的二维数组，其成员取值范围为 0，1，2。其中值为1的元素具备同化特性，每经过 1S，将上下左右值为0的元素同化为1，而值为2的元素，免疫同化。将数组所有成员随机初始化为0或2，再将矩阵的[0,0]元素修改为1，在经过足够长的时间后，求矩阵中有多少个元素是0或2（即0和2数量之和）。

输入描述

输入的前两个数字是矩阵大小。后面的数字是矩阵内容。

备注

- m和n不会超过30（包含30）。

输出描述

返回数字中非1的元素个数

用例1

输入

Plain Text

```
2 3
2 2 2
2 2 2
```

输出

Plain Text

```
5
```

说明

> 起始点 (0,0) 强制改为1，其余均为2且免疫同化，剩余非1元素为5。

用例2

输入

```
4 5
0 0 2 0 0
0 2 0 2 0
0 0 0 0 0
2 2 0 2 0
```

输出

```
6
```

说明

> 从 (0,0) 开始，同化所有可达的0，部分0因被2阻隔未同化，加上所有2元素，最终剩余6个非1元素。

题解

思路： `BFS` 模板题，可以参照下面对应代码，BFS模板题，可以背一下模板。

1. 使用 `队列` 模拟BFS实现过程。初始将 `(0,0)` 设置为1.然后进行迭代同化操作。
2. 迭代同化每一轮的基本逻辑为：取出队列队尾坐标,对这个坐标相邻的四个坐标判断是否能进行同化，如果可同化则将其设置为1，并将对应相邻坐标加入队列。 迭代过程中注意点： `数组越界判断、重复访问判断`
3. 循环迭代终止条件为 `队列为空` 。接下来可以使用循环统计矩阵中非1个数，对应数量就是结果。

C++

Plain Text

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

// BFS 同步
void BFS(vector<vector<int>>& grid, int m, int n) {
    int direct[4][2] = {{-1,0}, {1, 0}, {0, -1}, {0,1}};
    grid[0][0] = 1;
    queue<pair<int,int>> q;
    q.push({0,0});
    while (!q.empty()) {
        pair<int, int> tmp = q.front();
        q.pop();
        int x = tmp.first;
        int y = tmp.second;
        // 四个方向进行同化
        for (int i = 0; i < 4; i++) {
            int newX = x + direct[i][0];
            int newY = y + direct[i][1];
            if (newX < 0 || newX >= m || newY < 0 || newY >= n|| grid[new
X][newY] != 0) {
                continue;
            }
            grid[newX][newY] = 1;
            q.push({newX, newY});
        }
    }
}

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

```
45
46      BFS(grid, m, n);
47      int res = 0;
48      // 统计非1数量
49      for (int i = 0; i < m; i++) {
50          for (int j = 0; j < n; j++) {
51              if (grid[i][j] != 1) {
52                  res++;
53              }
54          }
55      }
56      cout << res;
57      return 0;
58  }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // BFS 扩展
    private static void bfs(int[][] grid, int m, int n) {
        int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
        Queue<int[]> queue = new LinkedList<>();
        grid[0][0] = 1;
        queue.offer(new int[]{0, 0});

        while (!queue.isEmpty()) {
            int[] cell = queue.poll();
            int x = cell[0], y = cell[1];

            for (int[] dir : directions) {
                int newX = x + dir[0], newY = y + dir[1];
                if (newX >= 0 && newX < m && newY >= 0 && newY < n && grid[newX][newY] == 0) {
                    grid[newX][newY] = 1;
                    queue.offer(new int[]{newX, newY});
                }
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int m = scanner.nextInt(), n = scanner.nextInt();
        int[][] grid = new int[m][n];

        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                grid[i][j] = scanner.nextInt();

        scanner.close();

        bfs(grid, m, n);

        int result = 0;
        for (int[] row : grid)
            for (int cell : row)
                if (cell != 1)
                    result++;

        System.out.println(result);
```

```
    }
}
```

Python

Plain Text

```
import sys
from collections import deque

def bfs(grid, m, n):
    """ 使用 BFS 从 (0,0) 开始进行同化 """
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    grid[0][0] = 1  # 标记起点
    queue = deque([(0, 0)])

    while queue:
        x, y = queue.popleft()
        # 遍历四个方向
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            # 确保坐标合法且当前点未被访问
            if 0 <= new_x < m and 0 <= new_y < n and grid[new_x][new_y] == 0:
                grid[new_x][new_y] = 1  # 标记访问
                queue.append((new_x, new_y))

def main():
    """ 读取输入、执行 BFS 并统计结果 """
    # 读取 m 和 n
    m, n = map(int, sys.stdin.readline().split())

    # 读取网格数据
    grid = [list(map(int, sys.stdin.readline().split())) for _ in range(m)]

    # 执行 BFS
    bfs(grid, m, n)

    # 统计 **所有非 1** 的格子数量
    result = sum(1 for row in grid for cell in row if cell != 1)
    print(result)

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

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", () => {
    let [m, n] = input[0].split(" ").map(Number);
    let grid = input.slice(1).map(row => row.split(" ").map(Number));

    bfs(grid, m, n);
    // 非1数量统计
    let result = grid.flat().filter(cell => cell !== 1).length;
    console.log(result);
});

function bfs(grid, m, n) {
    // 定义四个方向
    let directions = [[-1, 0], [1, 0], [0, -1], [0, 1]];
    let queue = [[0, 0]];
    grid[0][0] = 1;

    while (queue.length > 0) {
        let [x, y] = queue.shift();
        // 往四个方向同化
        for (let [dx, dy] of directions) {
            let newX = x + dx, newY = y + dy;
            if (newX >= 0 && newX < m && newY >= 0 && newY < n && grid[new
X][newY] === 0) {
                grid[newX][newY] = 1;
                queue.push([newX, newY]);
            }
        }
    }
}
```

Go

Plain Text

```
package main

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

// BFS 扩展
func bfs(grid [][]int, m, n int) {
  // 四个方向
  directions := [4][2]int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}
  queue := [][2]int{{0, 0}}
  grid[0][0] = 1
    // 遍历
  for len(queue) > 0 {
    x, y := queue[0][0], queue[0][1]
    queue = queue[1:]

    for _, dir := range directions {
      newX, newY := x+dir[0], y+dir[1]
      if newX >= 0 && newX < m && newY >= 0 && newY < n && grid[newX][new
Y] == 0 {
        grid[newX][newY] = 1
        queue = append(queue, [2]int{newX, newY})
      }
    }
  }
}

func main() {
  reader := bufio.NewReader(os.Stdin)
  line, _ := reader.ReadString('\n')
  fields := strings.Fields(line)
  m, _ := strconv.Atoi(fields[0])
  n, _ := strconv.Atoi(fields[1])

  grid := make([][]int, m)
  for i := 0; i < m; i++ {
    grid[i] = make([]int, n)
    line, _ := reader.ReadString('\n')
    values := strings.Fields(line)
    for j := 0; j < n; j++ {
```

```
        grid[i][j], _ = strconv.Atoi(values[j])
      }
    }

    bfs(grid, m, n)
      // 统计非1数量
    result := 0
    for i := 0; i < m; i++ {
      for j := 0; j < n; j++ {
        if grid[i][j] != 1 {
          result++
        }
      }
    }

    fmt.Println(result)
}
```

> 来自: 华为OD机考2025C卷 – 数值同化 (C++ & Python & JAVA & JS & GO)-CSDN博客