重 庆 大 学

学 生 实 验 报 告

实验课程名称 算法设计与分析

开课实验室 DS1501

学 院 大数据与软件学院 年级 2021 专业班软件工程 X 班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2022 至 2023 学年第 2 学期

总 成 绩	
教师签名	XXX

大数据与软件学院制

《算法设计与分析》实验报告

开课实验室：DS1501 **2023 年 6 月 13 日**

学院	大数据与软件学院	年级、专业、班	2021 级软件工程 X 班	姓名	XXX	成绩	
课程名称	算法设计与分析	实验项目名　　称	贪心法实验和动态规划法实验	指导教师	XXX		
教师评语	教师签名： 年　月　日						

一、 实验目的

- 掌握贪心法的一般求解过程，掌握动态规划法求解的基本步骤。
- 熟练掌握“一个序列中出现次数最多元素问题的求解算法”的实现，熟练掌握“矩阵最小路径和问题求解算法”的实现。
- 主要任务：实现教材配套实验指导书“第 2 章 2.7.1 小节 求解一个序列中出现次数最多的元素问题”和“第 2 章 2.8.1 求解矩阵最小路径和问题”。

二、使用仪器、材料

PC 微机 Lenovo Legion R9000P2021H;
Windows11 操作系统;
Clion2023 编译环境;

三、实验步骤

2.7.1 实验 1 求解一个序列中出现次数最多的元素问题

给定 n 个正整数,编写一个实验程序找出它们中出现次数最多的数。如果这样的数有多个,请输出其中最小的一个。

输入描述:输人的第 1 行只有一个正整数 n（1≤n≤1000）,表示数字的个数；输入的第 2 行有 n 个整数 S2、S3、…、Sn（1≤Si≤10000,1≤i≤n)。相邻的数用空格分隔。

输出描述:输出这 n 个次数中出现次数最多的数。如果这样的数有多个,输出其中最小的一个。

输入样例：

```
6
10 1 10 20 30 20
```

输出样例：

```
10
```

解：

用数组 a 存放 n 个整数,bestd 存放出现次数最多的最小数,maxn 存放出现最多的次数。将 a 按

整数值递增排序,这样值相同的元素连续排列在一起,累计相邻元素相同的个数 num,pred 存放元素值,只有当满足 num > maxn 条件时才执行 maxn=num,bestd=pred;当 num≤maxn 时查找下一个相同子序列。

2.8.1 实验 1 求解矩阵最小路径和问题

给定一个 m 行 n 列的矩阵,从左上角开始每次只能向右或者向下移动,最后到达右下角的位置,路径上的所有数字累加起来作为这条路径的路径和。编写一个实验程序求所右路径和中的最小路径和。例如,以下矩阵中的路径 1→3→1→0→6→1→0 是所有路径中路径和最小的,返回结果是 12:

1	3	5	9
8	1	3	4
5	0	6	1
8	8	4	0

解:

将矩阵用二维数组 a 存放,查找从左上角到右下角的路径,每次只能向右或者向下移动,所以结点(i,j)的前驱结点只有(i,i-1)和(i-1,j)两个,前者是水平走向(用 1 表示),后者是垂直走向(用 0 表示),如图 2.34 所示。

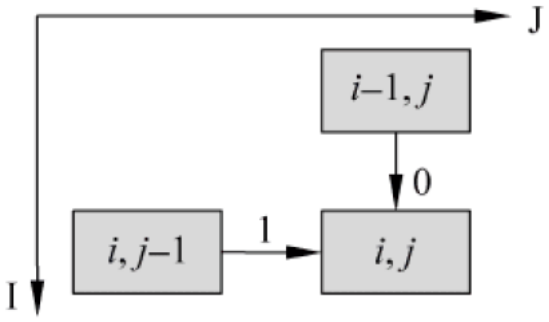

图 2.34 相邻结点到达(i,j)

用二维数组 dp 作为动态规划数组,dp[i][j]表示从顶部 a[0][0]查找到(i,j)结点时的最小路径和。显然这里有两个边界,即第 1 列和第 1 行,达到它们中结点的路径只有一条而不是常规的两条。

求出的 dp[m-1][n-1]就是最终结果 ans。为了求最小和路径,设计一个二维数组 pre,pre[i][j]表示查找到(i,i)结点时最小路径上的前驱结点,由于前驱结点只有水平(用 1 表示)和垂直(用 0 表示)走向两个,pre[i][j]根据路径走向取 1 或者 0。在求出 ans 后,通过 pre[m-1][n-1]反推求出反向路径,最后正向输出该路径。

四、实验过程原始记录(数据、图表、计算等)

2.7.1 实验 1 求解一个序列中出现次数最多的元素问题

给定 n 个正整数,编写一个实验程序找出它们中出现次数最多的数。如果这样的数有多个,请输出其中最小的一个。

源代码：

```
#include <iostream>     // 引入输入输出流库
#include <algorithm>    // 引入算法库

const int MAX = 1001;   // 定义常量最大值为1001
int test[MAX] = { [0]: 10, [1]: 1, [2]: 10, [3]: 20, [4]: 30, [5]: 20};   // 定义整型数组test，初始化为{10,1,10,20,30,20}
int number = 6;         // 数组中元素个数为6
int answer;             // 存储出现次数最多的数字
int mostcount = 0;      // 存储出现次数最多的数字的出现次数

void resolve(){
    std::sort( first: test, last: test+number);    // 使用算法库中的sort函数对数组进行升序排序
    int temp = test[0];             // 初始化temp为数组中第一个元素
    int count = 1;                  // 初始化count为1
    int i = 1;                      // 初始化i为1
    while(i<number){                // 当i小于数组元素个数时执行循环
        while(i<number&&test[i]==temp){    // 当i小于数组元素个数并且当前元素等于temp时执行循环
            count++;                // 统计temp出现的次数
            i++;                    // i加一
        }
        if(count>mostcount){        // 如果temp出现的次数大于存储的最大出现次数
            answer = temp;          // 更新存储的出现次数最多的数字
            mostcount = count;      // 更新存储的最大出现次数
        }
        temp = test[i];             // 更新temp为当前元素
        count = 1;                  // 将count重新初始化为1
        i++;                        // i加一
    }
}

int main() {
    resolve();                      // 调用函数resolve
    std::cout << "2.7.1 问题求解结果为：" << std::endl << answer << std::endl;   // 输出结果
    return 0;
}
```

运行结果：

```
2.7.1 问题求解结果为：
10


进程已结束,退出代码0
```

2.8.1 实验 1 求解矩阵最小路径和问题

给定一个 m 行 n 列的矩阵,从左上角开始每次只能向右或者向下移动,最后到达右下角的位置,路径上的所有数字累加起来作为这条路径的路径和。编写一个实验程序求所右路径和中的最小路径和。

源代码：

```
#include <iostream>
#include <vector>

const int MAXM = 100; // 定义二维数组的最大行数
const int MAXN = 100; // 定义二维数组的最大列数

int m = 4, n = 4;    // 定义二维数组的实际行数和列数
int answer;           // 最短路径和
int t[MAXM][MAXN] = { [0]: { [0]: 1, [1]: 3, [2]: 5, [3]: 9}, [1]: { [0]: 8, [1]: 1, [2]: 3, [3]: 4},
                      [2]: { [0]: 5, [1]: 0, [2]: 6, [3]: 1}, [3]: { [0]: 8, [1]: 8, [2]: 4, [3]: 0}}; // 存储每个位置的权重
int dp[MAXM][MAXN];   // 存储到达当前位置的最小路径和
int precursor[MAXM][MAXN]; // 存储到达当前位置的最短路径中上一个位置的信息
```

```
void ShortestPath() {
    int i,j;
    dp[0][0] = t[0][0]; // 初始位置的最短路为该位置的权重
    // 第一列位置只能由上面的位置到达，所以直接计算
    for (i = 1;i < m ; i++) {
        dp[i][0] = dp[i-1][0] + t[i][0];
        precursor[i][0] = 0; // 上一个位置在同一列，即 j=0
    }
    // 第一行位置只能由左边的位置到达，所以直接计算
    for (j = 1; j<n ; j++) {
        dp[0][j] = dp[0][j-1] + t[0][j];
        precursor[0][j] = 1; // 上一个位置在同一行，即 i=0
    }
    // 剩余位置可以由左边或上面的位置到达，分别计算最小值和记录上一个位置的信息
    for (i = 1; i < m ; i++) {
        for (j = 1; j < n; j++) {
            if (dp[i][j-1]<dp[i-1][j]){ // 左边的位置路径和更小
                dp[i][j] = dp[i][j-1]+t[i][j];
                precursor[i][j]=1; // 上一个位置在同一行
            } else{ // 上面的位置路径和更小或相等
                dp[i][j] = dp[i-1][j]+t[i][j];
                precursor[i][j]=0; // 上一个位置在同一列
            }
        }
    }
    answer = dp[m-1][n-1]; // 最终结果存储在右下角的位置上
}
```

```cpp
void ShowPath(){
    int i = m-1, j = n-1;
    std::vector<int>path; // 存储最短路径
    std::vector<int>::reverse_iterator the;
    while (true){
        path.push_back(t[i][j]);
        if (i == 0 && j == 0){ // 到达起始位置
            break;
        }
        if (precursor[i][j]==1)j--; // 上一个位置在同一行
        else i--; // 上一个位置在同一列
    }
    std::cout<<"最短路径为： "<<std::endl;
    for (the = path.rbegin(); the != path.rend(); ++the) { // 从终点到起点遍历路径
        std::cout<<*the<<" ";
    }
    std::cout<<std::endl<<"最短路径和为： "<<std::endl<<answer<<std::endl;
}

int main() {
    ShortestPath();
    std::cout<<"2.8.1 问题求解结果为： "<<std::endl;
    ShowPath();
    return 0;
}
```

运行结果：

```
2.8.1 问题求解结果为：
最短路径为：
1 3 1 0 6 1 0
最短路径和为：
12


进程已结束,退出代码0
```

五、实验结果及分析

结果都已对应显示在原始数据记录中，结果都与预期的分析符合。