

# 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 22 专业班 软工 2 班

学 生 姓 名 潘铷葳 学 号 20221982

开 课 时 间 2023 至 2024 学年第 2 学期

总 成 绩	
教师签名	付春雷

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

2024 年 6 月 11 日

学院	大数据与软件学院	年级、专业、班	22 软工 2 班	姓名	潘铷葳	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		付春雷	
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“矩阵最短路程和问题求解算法”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；  
Windows 操作系统，VS2015 编译环境（不限）；

三、实验步骤

问题一：

- 1.输入数据：读取 n 和 n 个正整数。
- 2.排序数组：对输入的数组进行排序。
- 3.统计出现次数：遍历排序后的数组，统计每个数的出现次数。
- 4.记录最大次数及对应数：在统计过程中，记录出现次数最多的数及其出现次数。如果有多个数出现次数相同，记录其中最小的一个。
- 5.输出结果：输出出现次数最多的数。

问题二：

- 1.定义状态数组：
  - (1) dp[i][j] 表示从起点到位置 (i, j) 的最小路径和。
  - (2) pre[i][j] 记录路径的方向，0 表示从上方过来，1 表示从左边过来。
- 2.初始化：
  - (1) 读取矩阵的行数 n 和列数 m，以及矩阵 a 的元素。
  - (2) 初始化 dp[0][0] 为 a[0][0]。
- 3.处理边界情况：
  - (1) 初始化第一列 dp[i][0]。
  - (2) 初始化第一行 dp[0][j]。
- 4.状态转移：  
对于一般位置 (i, j)，通过状态转移方程计算 dp[i][j]。
- 5.输出结果：
  - (1) 输出最短路径长度 dp[n-1][m-1]。
  - (2) 通过 pre 数组回溯路径，并输出路径。

#### 四、实验过程原始记录(数据、图表、计算等)

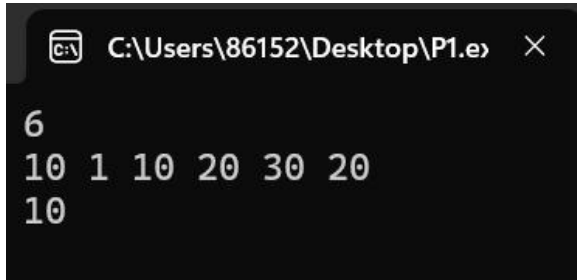
问题一：

求解一个序列中出现次数最多的元素问题”，实现求解序列中出现次数最多元素问题的算法。

代码：

```
1.  #include <iostream>
2.  #include <algorithm>
3.  using namespace std;
4.
5.  int main() {
6.      int n;
7.      cin >> n;
8.      int a[1000];
9.
10.     for (int i = 0; i < n; i++) {
11.         cin >> a[i];
12.     }
13.
14.     sort(a, a + n); // 先排序
15.
16.     int result = a[0]; // 出现次数最多的最小数
17.     int count = 1; // 出现最多的次数
18.     int currentCount = 1; // 当前数的出现次数
19.
20.     for (int i = 1; i < n; i++) {
21.         if (a[i] == a[i - 1]) {
22.             currentCount++;
23.         } else {
24.             if (currentCount > count) {
25.                 count = currentCount;
26.                 result = a[i - 1];
27.             }
28.             currentCount = 1;
29.         }
30.     }
31.
32.     // 检查最后一个数的出现次数
33.     if (currentCount > count) {
34.         result = a[n - 1];
35.     }
36.
37.     cout << result << endl;
38.     return 0;
39. }
```

运行结果:



```
C:\Users\86152\Desktop\P1.e
6
10 1 10 20 30 20
10
```

代码编写思路:

1.输入处理:

读取整数  $n$  和  $n$  个正整数, 存入数组  $a$ 。

2.排序数组:

使用 `sort` 函数对数组进行排序, 方便后续统计出现次数。

3.统计出现次数:

遍历排序后的数组, 统计每个数的出现次数。

如果当前数的出现次数大于记录的最大次数, 更新最大次数和对应的数。

在每次数值改变时, 重置当前数的出现次数。

4.输出结果:

输出出现次数最多的数。如果有多个数出现次数相同, 输出其中最小的一个。

问题二:

求解矩阵最小路径和问题”, 实现矩阵最小路径和问题的求解算法。

代码:

```
1.  #include <iostream>
2.  #include <stack>
3.  #include <algorithm>
4.  using namespace std;
5.
6.  int main() {
7.      int a[10][10];
8.      int dp[10][10]; // 最小路径值
9.      int pre[10][10]; // 记录具体路径, 0 表示垂直, 1 表示水平
10.     int n, m, i, j; // 存储行, 列
11.
12.     cin >> n >> m;
13.     for (i = 0; i < n; i++) {
14.         for (j = 0; j < m; j++)
15.             cin >> a[i][j];
16.     }
17.
18.     dp[0][0] = a[0][0];
19.
20.     // 处理第一列
21.     for (i = 1; i < n; i++) {
22.         dp[i][0] = dp[i - 1][0] + a[i][0];
23.         pre[i][0] = 0;
```

```

24.     }
25.
26.     // 处理第一行
27.     for (j = 1; j < m; j++) {
28.         dp[0][j] = dp[0][j - 1] + a[0][j];
29.         pre[0][j] = 1;
30.     }
31.
32.     // 处理一般情况
33.     // dp[i][j] = min(dp[i][j - 1], dp[i - 1][j]) + a[i][j]
34.     for (i = 1; i < n; i++) {
35.         for (j = 1; j < m; j++) {
36.             if (dp[i][j - 1] < dp[i - 1][j]) {
37.                 dp[i][j] = dp[i][j - 1] + a[i][j];
38.                 pre[i][j] = 1;
39.             } else {
40.                 dp[i][j] = dp[i - 1][j] + a[i][j];
41.                 pre[i][j] = 0;
42.             }
43.         }
44.     }
45.
46.     cout << "最短路径长度: " << dp[n - 1][m - 1] << endl;
47.     cout << "具体路径: " << endl;
48.
49.     // 回溯路径
50.     stack<int> res; // 设置一个栈, 因为路径是由末向前推的, 所以输出时要
    反转
51.     i = n - 1;
52.     j = m - 1;
53.     res.push(a[i][j]);
54.     while (i != 0 || j != 0) {
55.         if (pre[i][j] == 1) {
56.             j--;
57.         } else {
58.             i--;
59.         }
60.         res.push(a[i][j]);
61.     }
62.
63.     cout << res.top(); // 输出第一个元素
64.     res.pop();
65.     while (!res.empty()) {
66.         cout << " -> " << res.top();
67.         res.pop();
68.     }

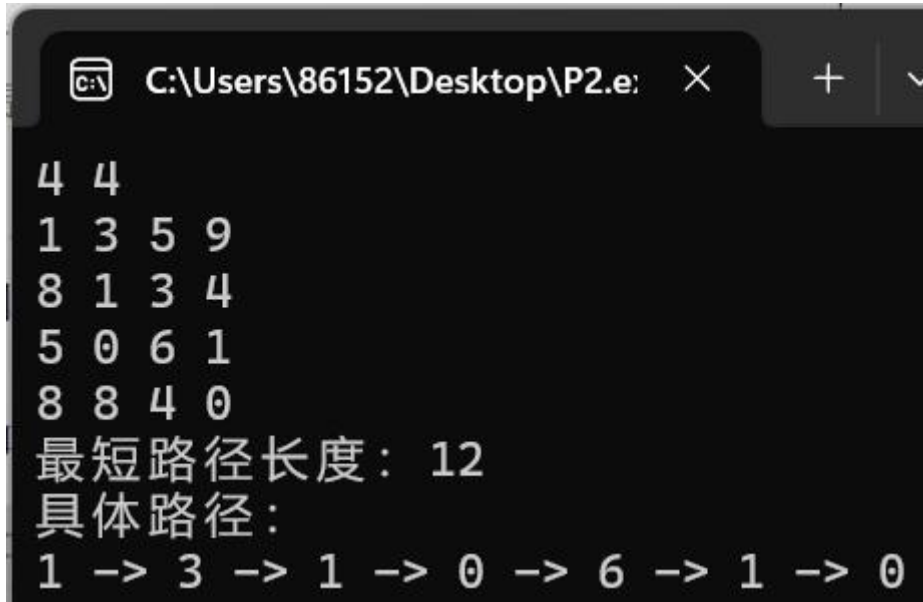
```

```

69.         cout << endl;
70.
71.         return 0;
72.     }

```

运行结果：



```

4 4
1 3 5 9
8 1 3 4
5 0 6 1
8 8 4 0
最短路径长度： 12
具体路径：
1 -> 3 -> 1 -> 0 -> 6 -> 1 -> 0

```

代码编写思路：

- 1.输入处理：  
读取矩阵的行数  $n$  和列数  $m$ ，以及矩阵  $a$  的元素。
- 2.初始化  $dp$  数组：  
 $dp[0][0]$  初始化为  $a[0][0]$ 。
- 3.处理边界情况：  
初始化第一列和第一行的  $dp$  数组。
- 4.状态转移：  
对于每个位置  $(i, j)$ ，根据状态转移方程计算最小路径和。
- 5.输出结果：  
输出最短路径长度。  
通过  $pre$  数组回溯路径，并输出路径。

总结贪心法的一般求解过程，动态规划法求解的基本步骤：

### 1.贪心法

基本思路是在对问题求解时总是做出在当前看来是最好的选择，也就是说贪心法不从整体最优上加以考虑，所做出的仅是在某种意义上的局部最优解。人们通常希望找到整体最优解，所以采用贪心法需要证明设计的算法确实是整体最优解或求解了它要解决的问题。

所谓贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。也就是说，贪心法仅在当前状态下做出最好选择，即局部最优选择，然后再去求解做出这个选择后产生的相应子问题的解。

### 2.动态规划

动态规划是一种解决多阶段决策问题的优化方法，把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解。

## 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。