重庆师范大学

实验报告

实验课程名称	算法设计与分析
实验序号	2
实验内容	递归
班级	23 计科 6 班
姓名	周子依
学号	2023051603162

2024 年 3 月 8 日

实验目的与要求

1. 学习目标：
   - **理解递归核心思想**：掌握分治与回溯的递归逻辑，能通过递归分解问题（如全排列固定元素、整数划分限制加数）。
   - **设计递归算法**：独立实现全排列和整数划分的递归代码，正确处理终止条件与回溯操作。
   - **复杂度分析**：能推导递归算法的时间与空间复杂度（如全排列的 (O(n!))、整数划分的 (O(2^n))）。
   - **实际应用能力**：通过实验案例（ABCD 全排列、8 的整数划分），提升递归在排列组合、数值分解中的编程实践能力。

实验内容

一、 实验原理

一个函数直接或间接地调用自己就是递归调用，递归调用是特殊的一种嵌套调用，递归即递推与回归，先有递推的过程，即把大事一步步化小，再有回归过程，递归需要有出口，即要有 if 的判断语句使递推结束，一步步接近出口，遇到出口再一步步回来，大部分题目写出分段函数的形式是设计递归的关键。

注意：递归层次太深，容易出现栈溢出的现象，所以虽然递归代码简单，也不要太迷恋。

二、 设计思路

1、 全排列

（1）什么是全排列

从 n 个不同元素中任取 m（m≤n）个元素，按照一定的顺序排列起来，叫做从 n 个不同元素中取出 m 个元素的一个排列。当 m=n 时所有的排列情况叫全排列。例如 ABC 的全排列有 ABC,ACB,BAC,BCA,CAB,CBA。在数学中学过，n 个数的全排列有 n!种排法。

（2）如何运用递归的思想解决全排列问题

- 问题分解

全排列问题可以分解为固定一个位置的元素，然后对剩余元素进行全排列。对于一个包含 n 个元素的数组，我们可以依次选择每个元素放在第一个位置，然后对剩

下的 n - 1 个元素进行全排列。

- 递归步骤

确定递归函数的参数：通常需要一个数组来存储元素，以及两个整数 k 和 m 分别表示当前要处理的起始位置和数组的最后一个位置。明确递归终止条件：当 k 等于 m 时，意味着已经固定了前面所有位置的元素，此时得到了一个完整的排列，将其输出。

- 递归过程

对于当前位置 k，遍历从 k 到 m 的所有元素，将它们依次与 k 位置的元素交换。

交换后，固定 k 位置的元素，递归调用函数处理 k + 1 到 m 的元素，生成剩余元素的全排列。递归调用返回后，将 k 位置的元素与之前交换的元素再交换回来（回溯操作），以便尝试下一个可能的元素放在 k 位置。如图 1 演示 ABCD 初步的排列演示。

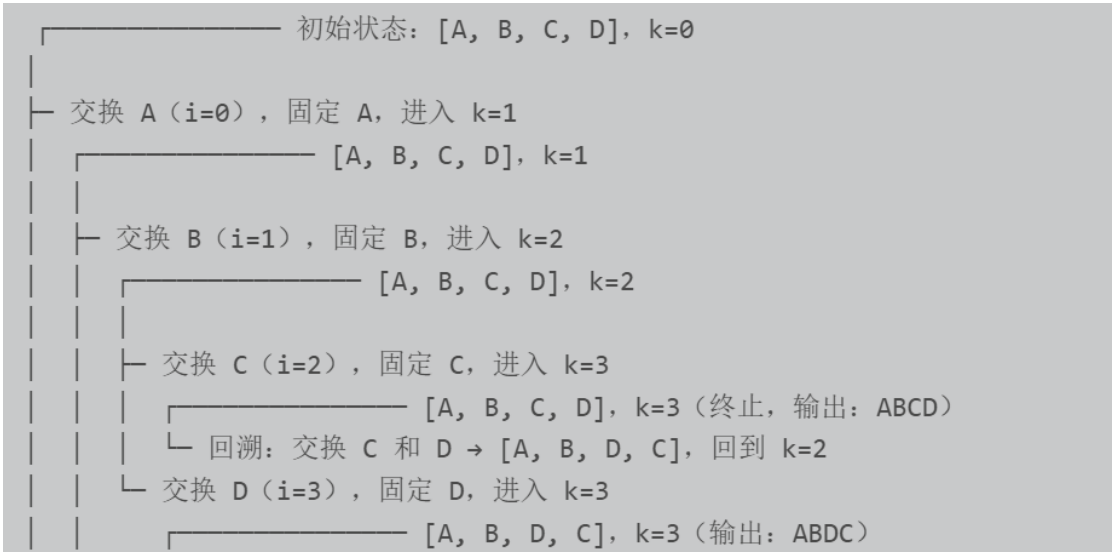


图 1

（3）复杂度分析

- **时间复杂度：**求 n 个元素的全排列，排列总数为(n!)。在代码里，生成每个排列需 O(n) 时间输出，所以整体时间复杂度为 (O(n n!)；若不考虑输出，仅生成排列，复杂度为 O(n!)。
- **空间复杂度：**主要由递归调用栈决定，递归深度最大为 n，所以空间复杂度是 O(n)。

2、 整数划分

（1）什么是整数划分

整数划分指将一个正整数 n 表示为一系列正整数之和。例如，整数 4 的划分有：4、(3+1)、(2+2)、(2+1+1)、(1+1+1+1)。

（2）如何运用递归思想解决整数划分问题

- 问题分解

整数划分问题可分解为：先确定第一个加数 i（(1 leq i leq n)），然后对剩余值 (n - i) 进行划分，且保证后续加数不大于 i（确保划分不重复）。例如，对 5 划分，先选 (i=2)，再对 (5-2=3) 划分，且后续加数不超过 2。

- 递归步骤

确定递归函数的参数： 通常需要表示待划分的整数 n，以及当前允许的最大加数 (max_num)（保证划分有序，避免重复）。

明确递归终止条件： 当 (n = 0) 时，说明已完成一种合法划分，输出结果；当 (n < 0) 时，跳过该划分路径。

- 递归过程：

遍历 i 从 1 到 (min(n, max_num))，将 i 作为当前划分的一个加数。对(n - i) 进行递归划分，此时允许的最大加数更新为 i（保证后续加数不大于 i）。递归返回后，回溯（无需显式操作，因递归参数已控制逻辑），继续尝试下一个 i。

如图 2，以 6 做演示。

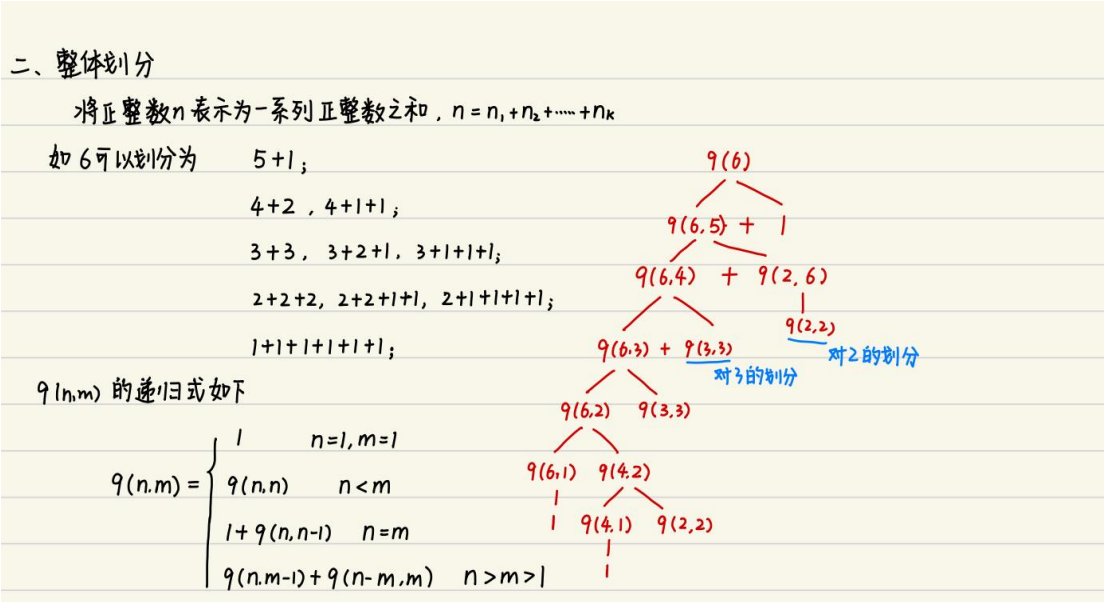


图 2

（3）复杂度分析

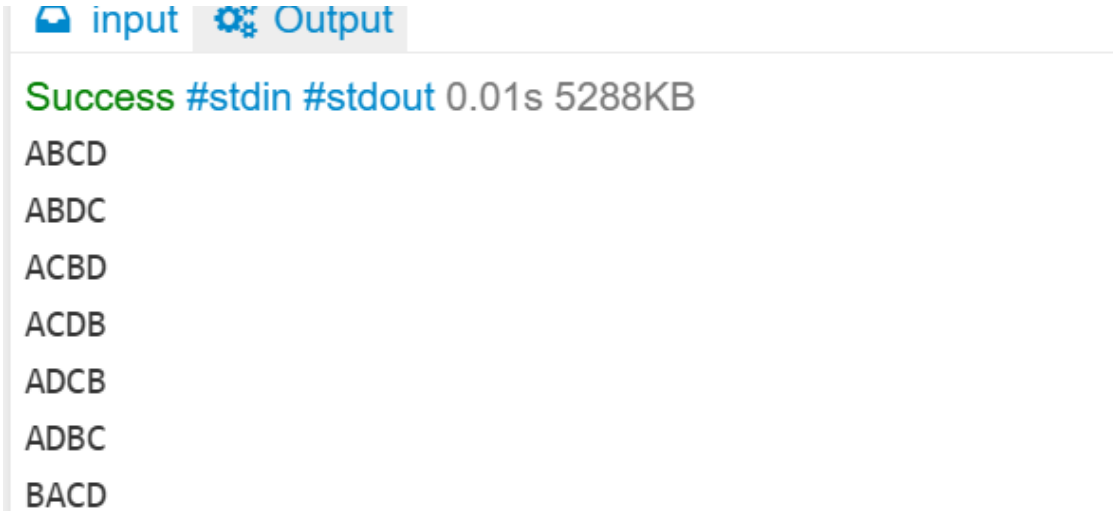
- **时间复杂度**：由于整数划分的方案数随着 n 的增大呈指数级增长，因此时间复杂度是指数级的，约为(0(2^n))。
- **空间复杂度**：递归调用栈的深度最大为 n，因此空间复杂度为 0(n)。

实验代码

代码附在文档最后

实验结果

一、 全排列 ABCD



二、 整数划分 8

Success #stdin #stdout 0.01s 5288KB comments (?)

stdin copy

```
Standard input is empty
```

stdout copy

```
整数 8 的所有划分结果如下:
8
7+1
6+2
6+1+1
5+3
5+2+1
5+1+1+1
4+4
4+3+1
4+2+2
4+2+1+1
4+1+1+1+1
3+3+2
3+3+1+1
3+2+2+1
3+2+1+1+1
3+1+1+1+1+1
2+2+2+2
2+2+2+1+1
2+2+1+1+1+1
2+1+1+1+1+1+1
1+1+1+1+1+1+1+1
整数 8 的划分一共有 22 种。
```

总结与体会

一、学习总结

1. **递归思想**：掌握分治与回溯核心，通过固定元素递归生成子问题解（如全排列交换元素、整数划分限制加数）。
2. **算法设计**：学会通过参数控制递归逻辑（如全排列的 k/m、整数划分的 maxPart），确保结果正确性。

二、实验体会

1. **调试经验**：递归终止条件需精准（如全排列 k==m），初始误判导致结果错误，通过单步调试解决。
2. **递归局限**：处理大规模数据易栈溢出（如整数划分 n=20），需权衡递归与迭代的适用场景。

全排列 C++代码：

```
1. #include <iostream>
2. #include <algorithm>
3. using namespace std;
```

```
4.
5.  // 函数用于生成数组元素的全排列
6.  void Perm(char list[], int k, int m, int& count) {
7.      if (k == m) {
8.          // 当 k 等于 m 时，表示已经生成一个排列，将其输出
9.          for (int i = 0; i <= m; i++) {
10.             cout << list[i];
11.         }
12.         cout << endl;
13.         // 每生成一个排列，计数器加 1
14.         count++;
15.     } else {
16.         // 对于当前位置 k，尝试与后续位置的元素交换
17.         for (int i = k; i <= m; i++) {
18.             // 交换 list[k] 和 list[i]
19.             swap(list[k], list[i]);
20.             // 递归生成剩余元素的全排列
21.             Perm(list, k + 1, m, count);
22.             // 回溯，恢复原来的顺序
23.             swap(list[k], list[i]);
24.         }
25.     }
26. }
27.
28. int main() {
29.     // 定义一个测试字符数组
30.     char list[] = {'A', 'B', 'C', 'D'};
31.     // 计算数组的长度
32.     int n = sizeof(list) / sizeof(list[0]);
33.     // 用于记录排列的总数
34.     int count = 0;
35.     // 调用 Perm 函数生成全排列
36.     Perm(list, 0, n - 1, count);
37.     // 输出排列的总数
38.     cout << "排列的总数为: " << count << endl;
39.     return 0;
40. }
```

整数划分 C++代码：

```
1.  #include <iostream>
2.  #include <vector>
```

```
using namespace std;

// 辅助函数，用于将存储在 vector 中的划分结果以 a+b+c 的格式输出到控制台
void printPartition(const vector<int>& partition) {
    // 遍历存储划分结果的向量
    for (size_t i = 0; i < partition.size(); ++i) {
        // 如果不是第一个元素，先输出一个加号
        if (i > 0) {
            cout << "+";
        }
        // 输出当前元素
        cout << partition[i];
    }
    // 输出完一个划分结果后换行
    cout << endl;
}

// 递归函数，用于生成整数 n 的所有划分结果
// n 表示当前需要进行划分的整数
// maxPart 表示当前划分中允许使用的最大数，确保划分结果是不递增的
// currentPartition 是一个向量，用于存储当前正在生成的划分结果
// count 是一个引用类型的变量，用于记录划分结果的总数
void partition(int n, int maxPart, vector<int>& currentPartition, int& count) {
    // 当 n 为 0 时，说明已经完成了一个有效的划分
    if (n == 0) {
        // 调用 printPartition 函数输出这个划分结果
        printPartition(currentPartition);
        // 划分结果总数加 1
        count++;
        return;
    }
    // 从 min(maxPart, n) 开始递减到 1，尝试将每个数作为当前划分的一部分
    for (int i = min(maxPart, n); i >= 1; --i) {
        // 将当前数 i 加入到当前划分结果中
        currentPartition.push_back(i);
        // 递归调用 partition 函数，处理剩余的数 n - i
        // 同时更新 maxPart 为 i，保证后续划分使用的数不大于当前数 i
        partition(n - i, i, currentPartition, count);
        // 回溯操作，移除刚刚添加的数 i，以便尝试下一个可能的划分
        currentPartition.pop_back();
    }
}

```

```
46.  // 包装函数，用于初始化整数划分的过程
47.  // n 是要进行划分的整数
48.  void partition(int n) {
49.      // 用于存储当前正在生成的划分结果
50.      vector<int> currentPartition;
51.      // 初始化划分结果的计数器
52.      int count = 0;
53.      // 调用递归函数开始生成整数 n 的划分结果
54.      // 初始时允许使用的最大数为 n
55.      partition(n, n, currentPartition, count);
56.      // 输出整数 n 的划分结果总数
57.      cout << "整数 " << n << " 的划分一共有 " << count << " 种。" << endl;
58.  }
59.
60.  int main() {
61.      // 要进行划分的整数
62.      int num = 8;
63.      // 提示信息，表明接下来将输出整数 8 的所有划分结果
64.      cout << "整数 " << num << " 的所有划分结果如下：" << endl;
65.      // 调用 partition 函数生成并输出整数 8 的划分结果及总数
66.      partition(num);
67.      return 0;
68.  }
69.
```