

实验二 分治策略

May 12, 2022

1 前言

分治 (divide and conquer)，字面上的解释是“分而治之”，就是把一个复杂的问题分解成两个或多个相同或相似的**子问题**，再把子问题分成更小的子问题，直到子问题的规模足够简单可以直接求解。在计算机科学汇总，分治法是很多高效算法的基础，如排序算法（快速排序，归并排序），傅里叶变换（快速傅里叶变换）等等。分治法在实际问题中具有很高的指导意义，例如全国人口普查过程中，可以将问题递归地拆分成省级、市级、区级、乡镇级、街道级、小区级，自下而上的统计人数。分治法的主要实现步骤如下：

1. 分治 (Divide)：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题
2. 解决 (Conquer)：若当前问题规模足够小，直接返回答案，否则递归地解决每个子问题
3. 合并 (Combine)：将各个问题的解合并为原问题的解

如果题目符合下面性质，或许可以用分治法来解决：

1. 该问题的规模缩小到一定程度就可以容易的解决
2. 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。
3. 利用该问题分解出的子问题的解可以合并为该问题的解
4. 该问题所分解出的各个子问题都是相互独立的，即子问题之间不包含公共的子问题。

递归 (recursion) 是分治中非常重要的步骤，在数学和计算机科学中，递归是指在函数调用函数本身。递归一词常用于描述以自身相似方法重复事物的过程。例如，当两面镜子相互之间近似平行时，镜中嵌套的图像是以无限递归的形式出现的。将时光往前推，在中国还流传着这样一个有趣的故事：从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？“从前



图 1: 《盗梦空间》(Inception, 2010) 中带有“德罗斯特效应”性质的电影镜头。

有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？
‘从前有座山，山里有座庙，庙里有个老和尚，正在给小和尚讲故事呢！故事是什么呢？’”

上面的故事似乎是没有终点的，如果将故事写成计算机程序然后运行，恐怕不到一会程序就会崩溃，因为递归调用函数会在系统栈中维护函数栈，无限递归下去会使得系统栈超出内存。图 1 所示的图形被称为**德罗斯特效应** (Droste effect)，是递归的一种视觉形式，是指一张照片的某个部分与整张图片相同，如此产生无限循环。这张照片是通过名为 Mathmp 的数学软件制作出来的，使用 PhotoShop 的 Droste Effect 滤镜也可以制作出这种效果。

在计算机程序中，递归实现并不是任何一个算法必须要有的部分。理论上说，任何的递归都可以使用分支语句和循环语句实现，只不过不使用递归的实现难度远远超过了递归实现。因此，有效的利用递归可以帮助我们更轻松的解决实际问题。本节实验课的基本内容是两个非常经典的题目，它们都可以优雅地使用分治法进行分析，并使用递归来求解。

2 实验项目结构

- find_maximum_subarray
 - include
 - util.hpp
 - Solution.hpp
 - data
 - main.cpp
- round_robin_schedule
- 略
- perfect_permutation

— 略

本实验包含三个独立的题目：find_maximum_subarray（最大子序列）和 round_robin_schedule（循环赛时间表），perfect_permutation（拓展题：完美排列）。

每个题目的代码结构是类似的，include 文件夹中包含了 util.hpp 和 Solution.hpp，data 文件夹中包含了测试数据，对于每个题目，你需要完成 Solution.hpp 的编写，然后编译运行 main.cpp 来进行校验。

请注意，每次对 Solution.hpp 修改完之后，需要重新编译运行 main.cpp，如何直接执行上次编译好的 main.exe 或 main，新的修改将不会生效。

3 实验内容

3.1 最大子序列

给定一个长度为 N 的数组 A ，其任意连续子序列可以表示为 A_i, A_{i+1}, \dots, A_j ，其中 $0 \leq i \leq j \leq N - 1$ 。最大子序列是指所有的子序列中和最大的那一个，注意子序列不能为空。你需要输出它的和，即：

$$result = \max_{0 \leq i \leq j \leq N-1} \sum_{k=i}^j A[k]$$

例如：[-2, 11, -4, 13, -5, -2] 的答案为 20。

请根据课件中的伪代码思想完成 Solution.hpp 的实现。

```
class Solution {
public:
    int find_maximum_crossing_subarray(vector<int> &A, int low, int
        mid, int high) {
        // 请在这里完成你的代码
    }
    int find_maximum_subarray(vector<int> &A, int low, int high) {
        // 请在这里完成你的代码
    }
    int find_maximum_subarray(vector<int> &A) {
        return find_maximum_subarray(A, 0, A.size() - 1);
    }
};
```

测试数据范围： $N \leq 10^5$, $|A[i]| \leq 10000$ 。

3.2 循环赛日程表

设有 $n = 2^k$ 个运动员要进行羽毛球循环赛，现要设计一个满足以下要求的比赛日程表：

- 每个选手必须与其他 $n - 1$ 个选手各比赛一次
- 每个选手一天只能比赛一次
- 循环赛一共需要进行 $n - 1$ 天

一些例子：

- $n = 2$ 时的日程表如下表所示：

运动员	第一天
1	2
2	1

- $n = 4$ 时的日程表如下表所示：

运动员	第一天	第二天	第三天
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

- $n = 8$ 时的日程表如下表所示：

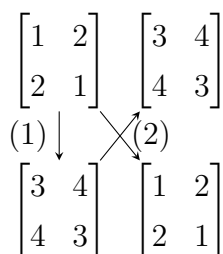
运动员	第一天	第二天	第三天	第四天	第五天	第六天	第七天
1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

你需要实现 Solution 类中的 round_robin_schedule 方法，对于传入的参数 n ，返回一个大小为 $n \times n$ 的二维数组。该二维数组满足：

- 所有元素都在 $[1, n]$ 范围内。
- 第一列从上到下依次为 $1 \sim n$ 。
- 同行或同列的元素不能相同。

```
class Solution {
public:
    vector<vector<int>> round_robin_schedule(int n) {
        // 请在这里完成你的代码
    }
};
```

提示：按照分治策略，将所有选手分成两部分。令 $m = n/2$ ，当得到了 m 名选手前 m 天的日程表后，通过平移、交换等方式分别拼凑出后 m 名选手前 m 天的日程表和 n 名选手后 m 天的日程表。下面的示意图中， $n = 4, m = 2$ 。



测试数据范围： $1 \leq k \leq 10, n = 2^k$ 。

4 实验思考

1. 对于序列 $[3, 2, -5, 3, -9, 9, -4, 6]$ ，画出求解最大子序列的递归调用树，并标出每个函数的返回值。
2. 分析两个实验的时空复杂度。
3. 如果最大子序列问题中允许选择空的子序列（和为 0），代码应该如何修改？

4. 【拓展题】完美排列（Google 面试题）

如果一个长度为 n ($1 \leq n \leq 10^3$) 的排列 a 满足对于每对 i, j ，都不存在 k 使得 $a[k] * 2 = a[i] + a[j]$ 成立，那么该排列就被称为完美排列。给定 n ，请你求出任意一个长度为 n 的完美排列。

注意：长度为 n 的排列是指由整数 $1, 2, \dots, n$ 构成的数组。

提示：分奇偶考虑

5 其他

5.1 vector 的基本使用

vector 可以被简单的看做是一个动态数组，可以像普通的数组一样使用 [] 运算符来访问其中的元素。与普通数组不同的是，它可以方便的进行创建、调整大小、添加或删除元素等等。

为了使用它，我们需要引入头文件：

```
#include <vector> // 引入头文件

// 1. 定义方法：
vector<int> a; // 定义一个元素类型为 int 的动态数组 a
vector<int> b(10); // 定义一个包含 10 个 int (默认为 0) 的动态数组 b
vector<int> c(10, 100); // 定义一个包含 10 个 100 的动态数组 c

// 2. 访问元素
int x = c[0]; // 访问 c 中下标为 0 的元素

// 3. 获取数组大小
int n = c.size();

// 4. 判断数组是否为空
if(a.empty()) {
    // 为空则 if 条件成立
}

// 5. 插入元素
a.push_back(x); // 把 x 插入到 a 的尾部

// 6. 删除元素
a.pop_back(); // 删除 a 的最后一个元素

// 7. 清空数组
c.clear();
```

有关更多内容，你可以参考：

- <https://www.runoob.com/w3cnote/cpp-vector-container-analysis.html>
- <https://en.cppreference.com/w/cpp/container/vector>
- <https://www.w3cschool.cn/cpp/cpp-i6da2pq0.html>
- <http://c.biancheng.net/view/6749.html>