***2021***



**算法设计与分析实践报告**



|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： |  |
| 学 号： |  |
| 姓 名： |  |
| 完成日期： | 2021/12/21 |

目 录

[1 完成情况 2](#_Toc91261149)

[2 2366解题报告 3](#_Toc91261150)

[2.1 题目分析 3](#_Toc91261151)

[2.2 算法设计 3](#_Toc91261152)

[2.3 性能分析 4](#_Toc91261153)

[2.4 运行测试 4](#_Toc91261154)

[3 1088解题报告 4](#_Toc91261155)

[3.1 题目分析 4](#_Toc91261156)

[3.2 算法设计 4](#_Toc91261157)

[3.3 性能分析 5](#_Toc91261158)

[3.4 运行测试 5](#_Toc91261159)

[4 1328解题报告 6](#_Toc91261160)

[4.1 题目分析 6](#_Toc91261161)

[4.2 算法设计 6](#_Toc91261162)

[4.3 性能分析 7](#_Toc91261163)

[4.4 运行测试 8](#_Toc91261164)

[5 3660解题报告 8](#_Toc91261165)

[5.1 题目分析 8](#_Toc91261166)

[5.2 算法设计 8](#_Toc91261167)

[5.3 性能分析 9](#_Toc91261168)

[5.4 运行测试 10](#_Toc91261169)

[6 总结 10](#_Toc91261170)

[6.1 实验总结 10](#_Toc91261171)

[6.2 心得体会和建议 10](#_Toc91261172)

# 完成情况

本次实验总共解决了20道算法题，已解决问题列表如图1.1：

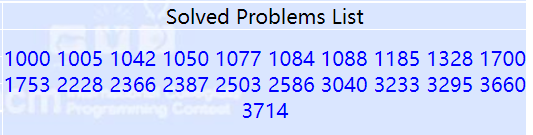


图 1.1 已解决问题截图

具体实验及问题详见图1.2：



图 1.2 具体解决(AC)问题列表

另外尝试题目：

二分+中位数：3269(Building A New Barn)

动态规划：1636(Prison rearrangement)

最短路径：1062(昂贵的聘礼)

最短路径：3169(Layout)

搜索：1475(Pushing Boxes)

搜索：1184(聪明的打字员)

# 2366解题报告

## 题目分析

给出两个数组，若两个数组中各取一个数，二者之和能为10000，则输出YES，否则输出NO。

这道题是一例很典型的利用分治思想的二分查找题，先求出数组A中各个数到10000的差值，再在B中查找有无这个数即可。

## 算法设计

输入的两个数组第一个数组A是升序，第二个数组B是降序。首先依次求出10000-A[i]的值，得到的这个新的数组其实就是降序排序了。在数组B中二分查找新数，如果查找到，则匹配成功，输出YES。



图 2.1 Sacrament of the sum流程图

## 性能分析

两次循环读入数据，之后在一次for循环中执行二分查找，所以总时间复杂度应为O(nlogn)。

所耗费空间仅两个数组的大小以及几个中间量。

## 运行测试

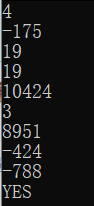


图 2.2 Sacrament of the sum运行结果



图 2.3 POJ运行结果

# 1088解题报告

## 题目分析

输入一个矩阵，每个数表示一块地区的高度，只能从大到小经过矩阵，求所经最长连续序列。

很典型的动态规划+搜索问题，对矩阵进行遍历，求出每个点作为起始点的最长序列，不断更新最长序列值，最后便可得到整个矩阵的最长序列。

## 算法设计

重点在于对递归函数的设计，遍历到某个点时，搜索它上下左右四个点是否小于该点（递归调用上下左右四个位置），持续递归；递归结束后返回的值为上下左右四个位置的最大值，然后将最大值+1便可记为当前点的最长序列值，记录到某个存储数组之中；在矩阵所有点都经历递归之后，便能在存储数组中找到最长序列，即为所求的“滑着最刺激的的滑坡”。



图 3.1 滑雪流程图

## 性能分析

算法所耗费的时间与空间主要都在于递归中，一次递归中需要再进行四次递归，同时每次递归都需要存储当前x、y值以及已有序列长度。但是由于测试例给出的矩阵并不大，所以运行时间并不长。

## 运行测试

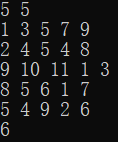


图 3.3 滑雪运行结果2

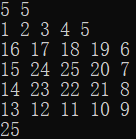


图 3.2 滑雪运行结果1



图 3.4 POJ运行结果

# 1328解题报告

## 题目分析

整个题目凝练下来，大意为，海岸线作为一条直线（x轴），陆地在y为负的一侧，海洋在y为正的一侧；有n个小岛，每一个岛屿占地一个点值；雷达位于海岸线（x轴）上，只能覆盖半径d距离。题目要求我们求解为了覆盖这些岛屿所需的最少的雷达数目。

## 算法设计

每个雷达的覆盖范围即是一个圆，要让雷达的总覆盖范围包括进所有的岛屿，那么反思路，每个岛屿以雷达半径作圆，便是雷达能够放置的区间。

对所有的岛作圆，与x轴的交区间便是雷达能够放置的区间，我们可以得到一个区间组。只要对这个组每个项的左起始值进行排序，重叠的部分是可以共用的，我们便能得到最少需要的雷达数。



图 4.1 Radar installation流程图

## 性能分析

该题目中算法运行时间最长的在于排序，需要两次循环，其他包括计算雷达范围、循环去重都只需要一次循环即可，所以整个代码的时间复杂度为O(n2)。

空间耗费不大，包括用于存储所有岛屿信息的数组、存储所有雷达范围的数组、以及几个中间变量。

给出的测试例的运行时间短，内存消耗也不大。

## 运行测试

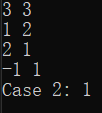


图 4.3 1328运行结果2

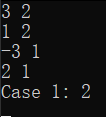


图 4.2 1328运行结果1



图 4.4 POJ运行结果

# 3660解题报告

## 题目分析

N头奶牛，参加编程比赛，我们可以知道部分牛编程能力的强弱，而且所有牛的编程能力都是不同的（没有并列）。给出M对关系(a,b)，表示a能力强于b，根据这些给出的关系，能确定多少牛的排名。

这道题是很典型的闭包传递，结合我们课上所学的知识，我们知道可以使用Floyd-Warshall算法，那么这道题的重点其实就在实现该算法。

## 算法设计

如果a强于b，就连接一条由a到b的边，表示a能走到b，而b无法走回a，这样排名就能根据走的顺序确定了。只要输入所有的强弱关系，连接完有向边，我们便能根据传递性，把所有可能的关系求出来。

为了确定一头牛的排名，我们只需要知道它被弱于多少头牛，强于多少头牛；例如一头牛强于x头牛，同时弱于y头牛，只要x+y=N-1，说明它与所有牛的关系都确定了，它的排名就能够确定了。

在这道题中，我们其实只需要知道牛A到牛B是否存在通路，所以可以对原本的Floyd-Warshall作出修改，只要判断出存在通路即可将A-B置1，以此简化我们的算法。



图 5.2 全流程图



图 5.1 算法流程图

## 性能分析

Floyd-Warshall算法包含三次循环，main函数中在整体的while循环判断中有一个双循环，所以整个算法的时间复杂度为O(n3)。

有N头牛，也即N个点，那么就需要大小为N2的二维数组来储存所有的边，还包括几个中间变量。

如果N的数量较大的话运行时间会较长，但是对于poj给出的测试例还是能很快运行出结果的。

## 运行测试



图 5.3 Cow Contest测试图



图 5.4 POJ运行结果

# 总结

## 实验总结

很多题一开始是完全没有思路的，但是只要静下心来仔细思考，根据课堂的内容，找到适合的知识框架（或者直接根据实验分类看考察的是什么知识），然后一步一步分析，就能慢慢找到算法的实现方法。

除此之外经过本次实验后对算法的选择和实现方式都有了更深的理解，不同的算法的优劣、时间和空间的占用都会对整个程序运行有影响。在学习众多算法之后，更加懂得如何对程序进行优化。

## 心得体会和建议

老师的选题选的很不错，每个板块的算法题都能够让学生对课堂的知识了解得更加透彻，其实选的很多题都不是很难，但是算法思想都很核心，是好题。

如果说美中不足的就是POJ，整个网页的UI和运行都不像这个时代的，而且在最后一周还出现了全waiting的情况，希望下一届的学生不用经受这样的痛苦。