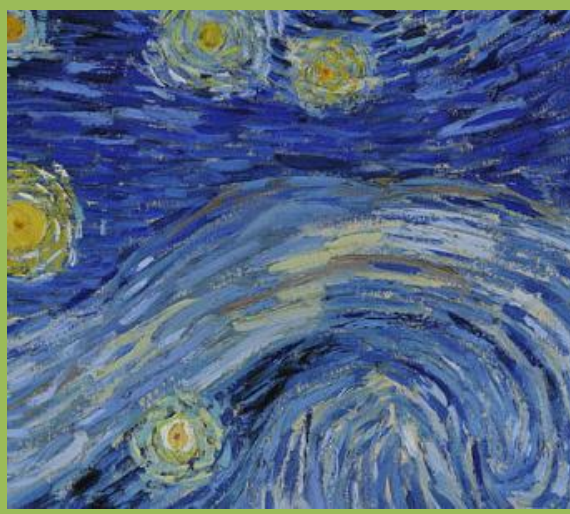
华中科技大学

2022

算法设计与分析实践报告



专　　业：计算机科学与技术

班　　级：CS2104 班

学　　号：U202115424

姓　　名：张森磊

完成日期：2022/12/20

华 中 科 技 大 学 课 程 设 计 报 告

目 录

1. 完成情况

oj 中做题列表如图 1 所示。

Compare 15670561689 and 15670561689 GO

Rank:	36111	Solved Problems List
Solved:	25	1000 1005 1042 1050 1062 1084 1088 1185 1201 1324 1328 1475 1700 1753 1860 2228 2366 2387 2449 2503 3040 3233 3295 3660 3714
Submissions:	70	
School:	华中科技大学	
Email:	1833842212@qq.com	

图 1 poj 做题列表

一共完成了 25 题。

2. 3233 解题报告

2.1 题目分析

题目大意：给出一个$n\times n$矩阵A和正整数k，求矩阵$S=\sum_{i=1}^{k}A^i$，要求矩阵的每一项对正整数m取模。

简要分析：本题题意比较直白，没有什么背景。如果直接计算和式中的每一项$A^i(1\le i\le k)$，显然做了很多重复工作，因为在计算过程中我们相当于重复求解了子问题，因此要考虑另外的方法，事实上，对于这个等比的矩阵和式，比较容易就能够看出当前要计算的S可以通过一些k值更小的S（下文中用S_k来表示$\sum_{i=1}^{k}A^i$）通过合适的组合来表示出来。此外，在做这道题的过程中，我感觉直接使用朴素的矩阵乘法去计算矩阵A的幂次也不是很好，所以顺便去学习了之前听说过的矩阵快速幂算法。

2.2 算法设计

首先我们可以尝试对原本的和式进行变换，可以先从规模较小的情况考虑，比如$S_4 = S_2 + A^2 S_2$，$S_5 = S_2 + A^3 + S_2 A^3$，类似的，当$k$为偶数时，$S_k = S_{\frac{k}{2}} + A^{\frac{k}{2}} S_{\frac{k}{2}}$，而当$k$为奇数时，$S_k = S_{\frac{k}{2}} + A^{\frac{k}{2}+1} + A^{\frac{k}{2}+1} S_{\frac{k}{2}}$，显然，经过这样的转化，就可以递归的通过二分法去求解这个问题。

另外，这里求解的时候显然还是要计算一个矩阵A的幂次的，这里采用了矩阵快速幂进行计算，矩阵快速幂就是把正常快速幂中的运算变成矩阵运算即可，主要是利用二进制表示幂次然后去进行运算。

最后，由于运算结果最终要求取模，需要对输入数据也进行一次取模运算，然后再求解问题，不然无法正常求解。

2.3 性能分析

首先矩阵的加法运算的时间复杂度为$O(n^2)$，矩阵的乘法运算的时间复杂度为$O(n^3)$，对于矩阵快速幂的时间复杂度，由于本身的快速幂算法的时间复杂度为$O(\log k)$，进而矩阵快速幂的时间复杂度为$O(n^3 \log k)$，这里$k \le 10^9$，个人感觉可以近似把$O(\log k)$看作一个常数的复杂度。然后在递归进行计算的时候，假设算法时间复杂度用$T(n)$表示，那么显然有$T(n) = T(n/2) + O(n^3)$，用主方法求解可知$T(n) = O(n^3)$，故最终时间复杂度为$O(n^3)$。

2.4 运行测试

将代码提交到 poj 上，运行结果如图 2 所示。

15670561689	3233	Accepted	1072K	610MS

图 2 3233 运行结果

3. 1185 解题报告

3.1 题目分析

题目大意：给出一个$n*m$的地图，地图上有平地也有山丘，平地上可以放置炮兵，炮兵攻击范围为上下左右两格，要求放置的炮兵不能相互攻击，求可放置炮兵的最大数目。

简要分析：根据经验，这里要求解最优化问题，而且也没有什么明显的如贪心选择性质这样的条件，那就考虑设计一个动态规划算法来求解这个问题。

3.2 算法设计

首先，我们可以先分析一下这道题中的状态以及状态之间的转移关系。假如我们现在要在第i行去放置炮兵，那我们需要考虑第$i-1$行和$i-2$行放置炮兵的情况。然后考虑如何表示状态，由于这里老师提示了要用状压 dp 去解这道题，所以我也去了解了一下状压 dp 的思想，即可以用一个 01 串去表示状态（当然这种理解本身还是很狭隘的，也可以用三进制串等等去表示状态，不过这里用二进制串就足够了），我们可以看出，炮兵在每个位置放与不放正好可以和 01 相对应，另外题目中的山丘和平地也可以和 01 对应，所以我们就可以用 01 串去表示每一行的状态，其中 1 表示这个地方是山丘或者已经放置了炮兵，0 表示这个地方没有炮兵。通过这种表示行状态的方法，我们就可以简单地用位运算来判断行与行之间是否发生冲突，而每一行状态的计算只需要考虑行内炮兵是否冲突以及放置点是否是平地即可。接下来考虑状态转移方程，如上文所说，这里的第i行仅仅与第$i-1$行和第$i-2$行的状态有关，所以我们应当用三维数组来刻画转移关系，即$dp[i][j][k]$表示第i行状态为j且第$i-1$行状态为k时的最大数量。

接下来我们就可以考虑具体实现，首先我们应该用一个一维数组g来存储地图，其中的每个元素都是一个二进制数，二进制位为 1 表示这个地方是山丘，这里只需要对输入进行处理即可。然后我们要找出所有可能的行状态，这里用一个数组$state$来存储所有可能的行状态，然后用数组$sodiers$来存储对应状态下这一行的士兵数量，后者主要是为了避免重复计算，所以在最开始计算状态的时候可

以顺便计算出士兵数量，只需要计算这一个行状态下的 1 的个数即可，当然，在统计的过程中需要记录全部的状态数$nums$。进而状态转移方程如下

$$dp[r][i][j]=\max_{1\le k\le nums}(dp[r-1][j][k]+soldier[i])$$

此外，这里的行状态仅仅考虑每一行的士兵之间是否相互冲突，也就是说可以用$(i\&i<<1)||(i\&i<<2)$来判断行间士兵是否有冲突。因为炮兵的攻击范围是两格。在得到所有行状态之后，由于这里当前行仅与前两行有关，所以我们需要单独初始化第一行和第二行。在初始化第一行时，用$state[i]\&g[0]$来表示当前状态的放置方案是否和地图发生冲突，这是因为我们在刚刚把放置炮兵和地图上为山丘的二进制位设置成了 1。在初始化第二行时，除了要考虑地图的影响，也需要考虑第一行和第二行状态之间的关系，可以直接用$state[i]\&state[j]$来表示。然后就可以依次去计算下面的行，检查冲突的方法就完全一致了。

3.3 性能分析

我们可以先考虑一下所有可能的状态数大致为多少，根据题中的数据范围，行总数不超过 100，列总数不超过 10，显然在没有限制的时候状态上限就是$1<<10$，而对于每一行的状态，本身炮兵一次影响左右两格，这里可以自己大致估算一下，也可以写一个程序枚举一下所有的可行状态，可以得到最终行状态最多为 60，也就是说$nums$最多为 60，在递推过程中，可以看到我们写了四重循环，因为内部有针对地图和行与行之间冲突的剪枝，所以可以大致给出一个上界，即$O(n*nums^3)$，而行数最多也只有 100，所以这个复杂度是可以接受的。

3.4 运行测试

将代码提交到 poj 上，运行结果如图 3 所示。

15670561689	1185	Accepted	3008K	313MS

图 3 1185 运行结果

4. 1042 解题报告

4.1 题目分析

题目大意：一个人打算在编号1~n的湖里钓鱼，钓鱼是单向走的，不能往回走。给你n个湖，每个湖初始鱼的数量p_i，每个湖钓鱼一次后鱼的减少量d_i，第i个湖到第$i+1$湖的距离时间t_i（单位是 5min），可以在任何湖停止钓鱼。求如何钓鱼才能在h小时内钓鱼量最多，输出在每个湖钓鱼的时间。相同钓鱼量情况下，输出湖编号小的用时多的时间。

简要分析：最开始也就是在往贪心这块想，但是由于这里有在湖之间转移消耗的时间，所以没有想清楚具体应该怎么去做，后来在网上查了题解，发现可以把在湖之间转移的时间和在湖中钓鱼的时间分开，也就是在最开始处理的时候把在湖之间转移的时间全部减去，然后剩余的时间就是钓鱼的时间，刚开始没有想明白这样为什么是对的，后来自己琢磨了一下，如果不考虑在湖之间转移的时间，那么最优策略肯定是每次都在当前这个时刻可钓鱼最大的湖里钓鱼，如果考虑在湖之间转移需要的时间的话，要想钓更多的鱼，他肯定不能往回走，这样就花费了多余的时间，也就是说，我们可以在每次先减去在湖间转移花费的时间，然后将问题转换成湖间转移没有代价的情况下来进行求解，这样最终我们是可以组合出一个满足条件的最优解的，只是调整一下钓鱼的顺序，但是每个湖钓鱼的量都是不变的。

4.2 算法设计

根据上面的分析，我们现在的主要问题变成了确定每次都要在哪几个湖钓鱼，这里就考虑直接枚举最后一个湖了，因为这个人必须从第一个湖开始钓鱼。我们用一个结构体$pond$来保存每个湖的信息，然后枚举湖的终点，在每一次枚举的时候都用一个优先队列来保存当前可选的鱼塘，优先级为f，即可以钓鱼的数量，然后在循环刚开始的时候减去当前从第一个湖转移到第i个湖所需要的时间，剩余时间就是可供钓鱼的时间，每次从优先队列中弹出一个元素，表示在这个湖里钓鱼，此外在枚举过程中要用数组b求记录在各个鱼塘钓鱼的时间。最终更新答案时，如果此次枚举的钓鱼数大于之前的最大值的时候，直接更新即可，

而如果此次枚举的钓鱼数等于之前的钓鱼数的时候，根据题目中的要求，我们需要把标号小而用时多的一组方案作为答案。

在这个题目中也有一些别的问题，比如在钓鱼的时候某个鱼塘的钓鱼数可能为负数，这个时候要及时将它更新成 0。

4.3 性能分析

在这个算法中，我们采用优先队列来找出每次钓鱼的地方，需要从第一个鱼塘开始一直枚举到最后一个鱼塘，每一次枚举过程中都要根据可钓鱼数量建立优先队列，其时间花费为$O(n)$，在枚举过程中剩余的钓鱼时间决定了对优先队列进行插入和弹出操作的次数，而对优先队列进行插入和弹出操作的时间复杂度均为$O(\log n)$，可知最终时间复杂度应为$O(n(n+h\log n))$

4.4 运行测试

将代码提交到 poj 上，运行结果如图 4 所示。

15670561689	1042	Accepted	252K	125MS

图 4 1042 运行结果

5. 1324 解题报告

5.1 题目分析

题目大意：在$n*m$的地图上，给出长度为L的蛇身体各个节位置，以及有k个点是墙，问蛇从初始位置走到$(1,1)$点的最小步数。蛇不能撞墙，不能撞自己的身体。

简要分析：这道题和平常遇到的简单的搜索问题的差别就在于这里的蛇是二维的，而不是想之前做的题一样，仅仅只用表示蛇头的位置，使用一次简单 bfs 即可。所以这里就是要考虑一种方法能够表示整个蛇的状态，由于本身蛇是一个整体，所以可以只记录蛇头的位置，然后记录蛇身每一个位置和前一个位置的偏移量，这个偏移量直接用一个整数去表示，然后正常的用 bfs 即可。

5.2 算法设计

我们这里首先采用一个结构体$node$来记录蛇的状态，其中包含了蛇头的坐标、蛇身的状态信息和当前走的距离，对于这个蛇身的状态，由于本身需要用数组dx和dy来记录可以转移的方向，可以用偏移位置在这个数组中的索引来表示这个位置，因为本身有四种方向，所以要用两位二进制位来保存。然后就用数组$vis[x][y][st]$来记录当前的状态是否被访问过，前两维表示蛇头的位置，第三维表示蛇身的状态，另外这里由于本题中有多个样例，所以每次都将该数组设置成当前的样例数，这样可以减少开销。然后在 bfs 的过程中需要把当前蛇身的状态解码出来，然后转移到下一个状态即可。

此外，由于这里的状态数量太多，所以需要考虑剪枝，这里的剪枝类似于LC检索中的剪枝，可以考虑通过一种方法找到上界，只要在 bfs 过程中超过这个上界就直接停止检索。注意到，在蛇前进的过程中，它的身体会逐步的到它的后面，也就是说它的身体最多一次称为他前进道路上的障碍，当然也有可能更少，因为每移动一步它的身体都会有向它身后移动的趋势，原来是障碍的身体就可能变成了空地。因此，我们只需要把它的身体当成一个永久的障碍然后仅仅让蛇头 bfs 一遍，看看到达终点需要多少步，就求出了一个上界。

5.3 性能分析

如果不考虑剪枝的话，由于本身数据中蛇身体的长度至多为 7，因此蛇的状态数最多为$20 * 20 * (1 << 14) = 6553600$个状态，最终 bfs 的时候，上界就是这些状态数，如果不考虑剪枝的话，运行时间的上界就是$O(nm2^{2*l})$，当然，由于计算过程中进行了剪枝，所以实际的运行时间要远小于这个结果。

5.4 运行测试

将代码提交到 poj 上，运行结果如图 5 所示。

15670561689	1324	Accepted	32968K	2000MS

图 5 1324 运行结果

6. 1062 解题报告

6.1 题目分析

题目大意：共有n个物品，每个物品有对应价格P，物品的主人有一个地位L，每个物品有一系列替代品，和一个对应的优惠价格V_i，也就是说如果想要换取这个物品，可以支付P，也可以找到其替代品并再支付相应的V_i来换取这个物品，物品主人等级差距在超过一个值M的时候不能发生交易，现在有一个外地人想要获得1号物品，求这个人要花费的最小的金额。

简要分析：从题意中我们可以看出一个很明显的图结构，即顶点是一系列物品，边是物品之间的替代关系，边的权重是优惠价格，当然，每个顶点上也有一个他自身的价格。很显然，如果没有地位差距的制约的话，我们只需要用一次Dijkstra算法求出1号物品到所有顶点的最短距离，然后计算这个最短距离与对应顶点上的物品价格的和的最小值即可。但现在由于多了一个地位制约，所以需要考虑做一些处理。事实上，由于我们必然要和1号商品的主人做交易，所以他的地位（不妨设为L_1）很关键，也就是说我们能够交易的所有人的地位必须都在$[L_1-M,L_1+M]$区间内，另外我们也需要中间交易的所有人的地位差距都在M以内，因此我们可以依次枚举区间$[L_1-M,L_1]$，$[L_1-M+1,L_1+1]$，……，$[L_1,L_1+M]$，每次枚举依次要求交易对象的地位在相应区间中。

6.2 算法设计

经过上面的分析，除了实现Dijkstra算法所需要的数据结构之外，这里使用数组rk来存储每个人的地位，在每一次枚举的过程中，首先遍历rk，找出这一次不能交易的物品，设置其$flag$为1，这里的数组$flag$就是Dijkstra算法中的vis数组，设置为1表示这个顶点已经被访问过。在处理过之后直接使用Dijkstra算法即可。

6.3 性能分析

这里的 Dijkstra 算法的时间复杂度为$O(n^2)$，共需枚举m次，故总时间复杂度为$O(m*n^2)$

6.4 运行测试

将代码提交到 poj 上，运行结果如图 6 所示。

图 6 1062 运行结果

7. 总结

7.1 实验总结

通过本次实验，我对课堂上学到的算法有了更深一步的了解，同时也学习到了程序设计过程中的一些技巧，具体有下面几点：

1. 了解了状态压缩这一思想。这一思想主要在做 1185 和 1324 时了解到的。1185 就是状压 dp 的模板题，而 1324 的难点主要就在于如何表示每次移动的时候的蛇的状态，它也是把蛇根据相对位置压缩成了一个二进制数。通过求解这两个题目，我总结了状压 dp 的模板，这类题是很有特点的，另外 1324 存储蛇的状态的思想也很值得借鉴，即通过把一系列坐标根据相对位置压缩成一个二进制数来减少空间使用。

2. 了解了剪枝的一些方法。这一点主要是在 1324 的时候了解到的，因为之前使用的大部分剪枝其实都是在情况明显不合理的时候进行的剪枝，而本题目中的剪枝是自己先预处理一个上界，然后将不满足这个界限的排除，感觉这个思路还是很不错的。

3. 了解了双向 bfs 的使用。主要是对于 1475 这道题目，要使用双向 bfs，从两个地点开始搜索，这种思想之前也没有遇到过。

4. 对贪心算法和动态规划有了更深刻的理解。这两种算法都用来求解最优化问题，而使用条件是有区别的，在本次实验的贪心算法部分，针对部分题目我也尝试了使用动态规划去求解问题，感觉对这两种算法的设计思路有了更深的了解。

7.2 心得体会和建议

心得体会：

1. 对各种算法的使用有了更加深刻的体会，同时也学习了一些算法设计技巧。从算法课上学的东西还是比较理论，在求解具有实际背景的问题时有时候不知道如何去抽象建模，有时候仅仅只有一些简单地改动我就不知道如何将它转化成算法中的限制条件。通过本次课程，我对课上学习的如动态规划、贪心算法等经典算法有了更深的理解，

2. 通过本次课程，我个人的细心与耐心有了更大的提升。对于很多算法题而言，其边界条件非常重要，有的时候如果考虑错误了一点最终都无法通过，这也要求我应当更加用心的考虑这些边界条件。

建议：

1. 个人感觉本次课程选取的部分题目感觉偏难，可以适当增加一些样板题或者降低题目难度，不然在遇到自己不会写的题目的时候在网上找题解看代码也需要花费大量的时间，而且感觉在网上也很难找到一些高质量题解。

2. 可以考虑更换一个 oj，比如洛谷，后者有专门的题解区，很多题解质量比较高，更加方便学习提升。

3. 感觉这次课程最难的就是后面搜索专题的题目，感觉难度有点过大了，有的时候看题解也看不懂是在做什么，希望能够降低一下搜索题目的难度，因为像我这样的只上过基础的数据结构的算法课的同学还是占大多数的，对于我们这样的同学而言太过吃力了。