一、 基础算法（枚举、模拟、贪心、分治）

【简介】

枚举、模拟、贪心、分治是最基础也是最需要打好基础的几个算法，这些题目普遍分布在 NOIP 的前两题，代码难度较低，学好这些算法不但是拿到 NOIP 保底分的保障，同时也能够帮助你更好的分析部分综合题。

不能因为这一块的相对简单而忽视了他们的重要性，相反的，选手的目标并不局限于掌握算法与会做题目上，更要同时借这个机会提升自己的思考问题的速度、对题目的敏感程度、写代码的速度、调试程序的能力等多维代码能力，为赛场上的代码效率打下牢固的基础。

【学习要点】

1. 对于枚举和模拟，一定要多写多练，锻炼自己写代码的速度和正确性，保证用最快的时间完成正确的代码。
2. 贪心题要靠一定题量的积累，选手通过掌握一些贪心的定式达到举一反三的效果。
3. 需要掌握基本分治思想、同时准备自己的二分和三分模板，做到写二分这种题时形成一种套路和定式，减少自己的出错概率。
4. 养成一个良好的读题习惯，题目中的每句话都可能十分关键，尤其是题目结尾样例数据和数据范围约定以及可能存在的注释部分，更是尤为重要。
5. 养成一个良好的代码习惯，首先确认算法的正确性，再理清楚代码的逻辑结构，最后再写代码，急于动手一般会导致你浪费更多的时间在写代码中途的迷茫和写完后永无止境的调试中。

【学习难点】

1. 这些题虽然简单，但这类题经常有一个很糟糕的共性——题目很难完全表述清楚，所以题目文字叙述会很长很长，题目本身的长度往往干扰到选手本身的做题心态，所以一定要锻炼自己的读题能力，不惧怕读长题，学会从长题中逐渐提炼出题目最根本的意思。
2. 贪心题的难点在于贪心算法本身的正确性，即大胆猜想，小心求证，在寻找规律时一定要足够大胆，但是找到的规律本身一定要符合逻辑，在面对自己提出的算法时，应从两个角度去考虑：i) 尝试为自己的算法找一个合逻辑的解释；ii) 反复寻找可能存在的反例去反向验证。而掌握大量的套路，可以快速看出题目的套路来源，就能很快找到一个合理逻辑。
3. 对分治题的要求体现在两点：i) 能正确的写出分治代码，对分治算法的框架有一个清晰的掌握并有一个的漂亮实现；ii) 分治题一般都有很明显的特征，通过题目的积累能慢慢对分治的题型有个大致了解，在遇到新的分治题的时候能迅速意识到此题的算法应该是一个分治，帮助你有一个正确的思考导向，题目就已经完成了一半（在二分题上尤为明显）

【例题】

1. 蛇形矩阵（CODEVS 1160）
2. 均分纸牌（2002 年 NOIP 提高组）（CODEVS 1098）
3. 国王游戏（2013 年 NOIP 提高组）

4. 跳石头（2015 年 NOIP 提高组）
5. Balanced Lineup（POJ 3264）
6. KD 之死（BZOJ 1555）
7. Kieszonkowe（BZOJ 4291）
8. Well （BZOJ 2792）

二、 数据结构

【简介】

这一章要进一步学习更丰富的数据结构知识，和以前的学习一样，这一章也是为了以后的学习作为基础，学习的数据结构也是为了其他算法的需求而服务，所以对各种数据结构都要有一个清楚的了解。

【知识点一览】

1. 二叉堆
2. 二叉搜索树
3. 并查集
4. RMQ（st-表）
5. LCA（tarjan、树上倍增）
6. 树状数组、线段树

【需要掌握的部分】

1. 每种数据结构的原理。
2. 每种数据结构的功能（每种数据结构维护数据的形式，支持的操作，每种操作的复杂度）。
3. 每种数据结构的具体实现（与功能一一对应）。
4. LCA 与 RMQ 两种算法之间的相互转化。

【难点一览】

1. 对并查集的两种优化方式（按秩合并、路径压缩）都理解和掌握。
2. 通过 LCA 的两种算法初步理解离线算法和在线算法的区别，同时比较他们的复杂度。
3. 掌握筛选法建堆和 O(n) 建树状数组的算法。
4. 笛卡尔树的实现。
5. 如何利用并查集维护额外信息。
6. 线段树的标记下传问题。
7. 线段树的双标记下传问题。

【例题】

1. 表达式运算 Strikers（CODEVS 2177）
2. 最小的 N 个和（CODEVS 1245）
3. NOIP 2012 借教室
4. NOIP 2010 关押罪犯（CODEVS 1069）
5. NOI2011 食物链（CODEVS 1074）
6. NOI2002 银河英雄传说（CODEVS 1540）
7. 苹果树（CODEVS 1228）
8. 等差子序列（CODEVS 1283）

【思考问题】

1. 二叉搜索树和笛卡尔树的区别？
2. 树上倍增的二维数组应该把哪一维作为第一维，为什么？

3. 在不使用 malloc 和 new 的情况下如何使用指针实现数据结构？
4. 如何实现一个在需要时才会分配节点，否则就用哨兵节点代替的线段树？
5. 如何用树状数组实现一个最基本的区间加减和区间求和？

三、 基本数论算法

【简介】

数学在 NOIP 的问题中一般分成三个部分：1.高中课本的数学；2.简单数论；3.组合数学数学题的知识点分布很细很“杂”。除了上面整理的大块、成系统的的知识点以外，还有很多各式的小结论需要在平时学习的过程中去学习和记忆。但和单纯的数学不同的是，NOIP 中的数学并不是非常在意结论的正确性，而更在意如何去使用一个结论。就像做数学题时，并不关心数学公式如何被证明，更关心一个正确的数学公式如何通过正确的变形去解题。所以可以不去死磕一个结论的正确性证明。

【知识点清单】

1. 高精度
2. 排列组合数，进制转换，Lucas 定理
3. 容斥原理
4. gcd 算法、exgcd 算法以及裴蜀定理
5. 逆元、解同余方程组（中国剩余定理）
6. 线性筛求素数
7. 欧拉函数、欧拉定理、费马小定理、线性筛求欧拉函数
8. 卡塔兰数、斯特林数
9. 高斯消元
10. 矩阵乘法

【学习重点】

1. 通过题目去不断学习以上各种算法的变形与实际运用。
2. 掌握每个定理的性质，便于在做题的时候通过联想得到解。
3. 准备属于自己的数论模板（尤其是高精度和 exgcd），将每个数论算法都熟背。
4. 注意 long long 和取模的问题。
5. 高精度除法（取模）的正确实现，高精度压位操作的实现。
6. 组合数的各种计算方式（杨辉三角，Lucas 定理，以及计算 n!的逆元）
7. 利用中国剩余定理解决模数不是素数时的同余方程
8. 理解线性筛求积性函数的过程
9. 利用矩阵快速幂求递推函数（常用于优化 DP）
10. 异或高斯消元与高斯消元求行列式及矩阵逆

【例题】

1. 基因变异（CODEVS 3194）
2. 同余方程（NOIP2012 提高组）（CODEVS 1200）
3. 青蛙的约会（POJ1061）
4. 礼物（CODEVS 1321）
5. 麦森数（CODEVS 1087）
6. 统计公共子序列个数（CODEVS 1778）
7. Xn 数列（CODEVS1281）
8. 组合数问题（BZOJ 4870）
9. 树（BZOJ 2466）

四、 字符串算法

【简介】

字符串算法主要针对 NOIP 中可能出现的字符串题，而字符串题在 NOIP 中算是一块独立的大题，这类题的特点就是没有掌握对应的字符串算法，虽然能完全理解题目的意思，但依旧寸步难行，完全没有任何思路。字符串的每一个算法几乎都是高度凝练的智慧结晶，十分值得我们去理解，当然从解题的角度去看，我们依旧需要背诵所有的字符串算法代码，毕竟很难仅通过自己的理解去重现这些算法。

【知识点一览】

1. KMP（字符串匹配）
2. TRIE（字典树）
3. HASH
4. MANACHER（回文串相关算法）

【重难点一览】

1. 最最基本的，找到适合自己的字符串读入输出处理方式。
2. 难点集中在对 KMP 整个算法的理解上，KMP 算法有很多巧妙的应用，所以不能只是单纯去背诵其代码，要对整个算法有清晰的理解，以及了解对 next 数组的多角度解释。
3. 一定要掌握 HASH 算法，有很多字符串的题目都能用字符串 HASH 解决，学会在题目中积累 HASH 的经验，背诵一个 HASH 模板，以及学会双 HASH 值、HASH 挂链等最基本的技巧。
4. MANACHER 算法专用于回文串相关的问题，OI 中所有回文串题目几乎都离不开 MANACHER，不但要理解 MANACHER 算法本身，还要注意如何利用 MANACHER 算法维护其他数据域（类似 MANACHER 和 DP 的结合）

【例题】

1. Poj 3080 Blue Jeans
2. Poj 2406 Power Strings
3. Poj 2752 Seek the Name, Seek the Frame
4. Poj 1743 Musical Theme
5. Hdu 4821 String
6. NOI2014 动物园
7. BZOJ 2160 拉拉队训练
8. BZOJ2342 双倍回文

五、 图论（图、树）

【简介】

这一章主要介绍一些进阶的图论算法，而每个图论算法都是一类图论题的核心，这就要求选手掌握每一个图论算法的实现。同时，由于图论题的灵活性，经常会对算法进行一定程度的变形和利用，这就要求选手对每个图论算法的思路有着清晰的掌握。

大体上 NOIP 的图论题目多可以分为三类:

1. 考察图论算法本身性质的（如利用三角形不等式的差分约束，负权回路，最小长度环等）
2. 难点集中在如何对原图进行转化的，或如何建模形成一个图
3. 综合题，图论算法和其他算法的结合，图论算法只是为了更好的引出其他算法

【知识点一览】

1. 图的基本存储

2. 最小生成树
3. 最短路算法
4. 有向图的拓扑排序
5. 树上倍增
6. 外向树
7. 强连通分量
8. 双连通分量

【难点一览】

1. dijkstra 的堆优化，SPFA 的 SLF、LLL 优化
2. 差分约束问题
3. 如何改进最短路算法来得到一个 K 短路算法
4. 如何求一个次小生成树
5. 如何利用 Floyd 算法求一个最小长度环
6. 如何利用 SPFA 判定一个负环
7. 外向树是一类题目，其难度在与如何找环，以及环外侧树上的 DP 以及环上 DP 的处理
8. 强连通分量 tarjan 算法的理解，以及常见的在强连通之后的缩点与拓扑排序操作
9. 分辨点双连通和边双连通，理解桥和顶点的概念
10. 和联通分量有关的经典 2-sat 问题

【例题】

1. Car 的旅行线路（NOIP2001）（CODEVS 1041）
2. BZOJ 1977 次小生成树
3. Poj 1201 Intervals
4. Poj 3621 Sightseeing Cows
5. 货车运输（NOIP 2013）
6. 骑士（BZOJ 1040）
7. BZOJ 1512
8. BZOJ 1093 最大半联通子图
9. BZOJ 2199 奶牛议会

六、动态规划

【简介】

动态规划可以说是在 NOIP 最灵活的一类题目，故也是 NOIP 中常见的难题。

动态规划最重要的两环就是想清楚状态的划分和状态的转移过程（状态转移方程）而不同类型的动态规划往往有不同的状态划分和状态转移方程，所以想在这两个点上提高自己的能力，必须提升自己的做题量，找到同类问题的套路，学会例题的思考方式，以后在面对新题时，就可以类比相似的题目的思考方式，为自己打开思路，找出正确的状态划分和转移方程。

故动态规划的学习要领就是多写题，写不同类型的题，多学习题解思路，最后总结出同一个套路题的共有解题思路，并运用到新题上去。

同时还要学习动态规划一些优化手段，如单调队列。

单调队列是一种类似于决策单调性优化的、但比决策单调性更强的优化方法，它更类似于一种数据结构，主动舍弃了更差的决策，从而保证更快地得到更好地子决策。

学习动归的过程不要害怕自己不会，也不要觉得不会做去看代码就是不对的。相反，

在动归打基础的时间中，直接尝试消化题解往往比自己傻想发呆要来的有效率，只要不对题解产生依赖心理即可。

在学习的过程中，可以通过递推等概念来帮助自己理解动归的状态划分等，动态规划的核心就是，通过合理的划分状态，来减少重复的搜索，利用合理的划分，通过几个子问题的最优解来计算当前问题的最优解，形成一种步步递推的过程。

【经典题目类型一览】

序列类：1. 最长上升子序列；2. 最长公共子序列；

背包类：1. 0-1 背包问题；2. 完全背包问题；

区间类：1. 石子归并类问题；（NOIP2006 能量项链）

其他：1. 数字三角形问题；2. 有向无环图最短路问题；

树形 DP：思路和序列类或背包类类似，只是结构从序列变成了树（CODEVS1380 没有上司的舞会）

棋盘 DP: 同上，只是从一维的序列变成了一个二维的棋盘（NOIP2007 矩阵取数游戏）

记忆化搜索：dfs 的优化，通过防止 dfs 的重复计算来加快速度，本质就是 DP（poj 1088 滑雪）

基本优化：1. 决策单调性优化；2. 四边形不等式优化；

优化：单调队列优化（BZOJ 1855、POJ 1821、POJ 3926）

【复杂 DP 类型一览】

这一类的 DP 较上面的更加复杂，但仍具有通性

状压 DP：状态划分时状态本身是一串 0 ~ (k-1) 的数字，代表当前第 i 个节点的状态，故可以用一个 N 位的 k 进制数表示，通过位运算来进行状态转移，故叫状压 DP（例题：poj 3254、BZOJ 4057、BZOJ 3195）

数位 DP：数位 DP 是一种计数形式的 DP，其状态是每一个数位，一般用于求一个区间内满足条件的数的个数，通过枚举每一位放的数来进行状态转移（例题：HDU 2089、BZOJ 1026、HDU 4734、POJ 3252）

七、STL

【简介】

STL——即标准模板库，并不是一种数据结构或者是算法，而是很多算法已经实现，如果我们掌握了 STL，那么很多算法和数据结构我们只要利用 STL 中现成的即可，不需要去自己实现了，从某种程度上大大方便了我们实现程序（仅限 C++）。

【注意事项】

1. 并不是 STL 实现了意味着你就可以不学习，必须要在理解算法本身的基础上去再去使用 STL，才能更好的理解 STL 中每个容器的构成和功能。
2. STL 可以说是博大精深，而里面只有一部分是适合 OI 选手使用的，想学习 STL 中常用的用法，多看别人的代码，看别人在 STL 的使用上是否有值得学习的地方。
3. 对 STL 的使用不清楚的时候可以查阅文档（www.cplusplus.com）
4. 过于简单的数据结构和算法尽量不使用 STL，因为可能会导致效率下降引发超时问题
5. 了解指针的概念对理解 STL 中很关键的一环——迭代器有很大的帮助
6. 了解二叉搜索树和平衡树有助于理解 set 和 map
7. 了解位运算有助于理解 bitset

【基本知识点清单】

1. stack——<stack>

2. queue、priority_queue——<queue>
3. deque——<deque>
4. map——<map>
5. set——<set>
6. bitset——<bitset>
7. <algorithm>