算法总论

基本算法

1. 递推与递归
2. 前缀和与差分
   ① 前缀和
      A. 激光炸弹
      B. Max Add
   ② 差分
      A. 增减序列 最高的牛
      B.
3. 二分

二分法是一种非常精妙的算法，很多算法的优化就来源于此方法，这里这要讲述两种常用的二分适用场景

① 对于明显的有序序列，我们可以使用二分快速查找。

② 对于隐含的有序序列，我们可以将搜索问题转化为判定问题快速求解，转化的关键在于判定时，我们判定的时间复杂度至少要低于搜索的时间复杂度 log(n)级别的，否则将没有意义。

例题：

   A. 最佳牛围栏
   B. 向下取整整数对和 答案
   C.

4. 排序
   ① 离散化：离散化指的是将大数映射到小数上，并保持他们的大小关系不变的操作。
      A. 大数映射到小数是为了便于操作。
      B. 保持大小关系是为了等比压缩，可以按照小数操作。

      电影
   ② 排序：
      A. 选择、插入、冒泡
         a) 不常用，基于比较的 O(n^2)的排序算法
      B. 堆排序，快排，归并
         b) 快排：第 k 大的数
         c) 归并：求解逆序对数目的问题 超快速排序
         d) 堆排序：动态中位数
      C. 计数，基数，桶排序
5. 倍增
   ① 思想

   我们设当前扩充的区间长度为 p，区间的左右端点分别是 L，R，那么倍增的主要操作为以下三步：

      A. 初始化 p=1，R=L.
      B. 验证区间[L，R+p]是否满足条件，满足条件时更新 R=R+p，p*=2；不满足条件时 R 不变，p/=2；
      C. 当 p=0 时，一次倍增操作结束，此时 R 即为所求区间的右端点。
   ② 单调递增：

A. 快速幂+倍增：graph smoothing

B. ST 算法：数列区间的最大值

③ 单调递减：

A. LCA：

④ 先增后减

A. 倍增搜索：天才 ACM

B. 向下取整整数对和 答案

6. 贪心

① 介绍：贪心的思想具有很大的技巧性在里面，我们通常需要找到某一个角度，由局部最优推导出全局最优，这样我们才可以使用贪心的思想。

② 微扰（邻项交换）

证明在任意局面下，任意对局部最优策略的微小改变都会造成整体结果变差。经常用于以排序为贪心策略的证明。

A. 均分纸牌

③ 范围缩放

证明任何对局部最优策略作用范围的扩展都不会造成整体结果变差。

④ 决策包容

证明在任意局面下，做出局部最优决策后，在问题状态空间中的可达集合中包含了作出其他任何决策后的可达集合。换言之，这个局部最优策略提供的可能性包含其它所有策略提供的可能性。

A. 七夕祭

⑤ 反证法

⑥ 数学归纳法

A. 货仓选址（中位数）

其他例题：

A. 构成交替字符串的最少交换次数 答案

经典例题，其实也谈不上是贪心问题，者更偏向与一个脑筋急转弯。对于对于一个交替字符串来说，当我们确定奇数和偶数位置的数是什么后，奇数位置不满足条件的字符数量一定等于偶数位置不满足条件的字符数量，我们只需要统计记录较小的一种分配方案即可。

B.

7. 区间和的最值问题

给定一个包含负数的数组，求解区间和的最大值。

① 利用滑动窗口机制，及时剪枝的贪心做法

② 将问题转化为扫描并维持一个前缀和最小的的方法，具体方法对于每个扫描到的位置 i，以该点为右端点的区间和的最值，一定是 sum[i]与 sum[i]之前的前缀和的最小值之间的差值。

最大子序和 最佳牛围栏

8. 总结

任何问题都有其可行解的范围空间，每个可行解有一系列的状态组成，所有状态空间组成的空间就是问题搜索过程中的状态空间，我们实现算法的目的就是在可行

基本数据结构

1. 栈

① 括号匹配+后缀表达式计算

栈的最经典应用就是括号匹配以及表达式计算。

A. 有效的括号

② 单调栈问题

对于区间的一个端点不变，而另一个端点变化的区间最值问题，我们可以使用单调栈求解。

A. 包含 min 的栈 （哈希+单调栈）

B. 下标对中的最大距离(双指针 单调栈+双指针 单调栈+二分)

C. 子数组最小乘积的最大值 答案

D. 直方图中最大矩形面积 答案

③ 卡特兰数

A. 火车进出栈问题

方法一：枚举 卡特兰数-枚举法

方法二：递推 卡特兰数-递推法

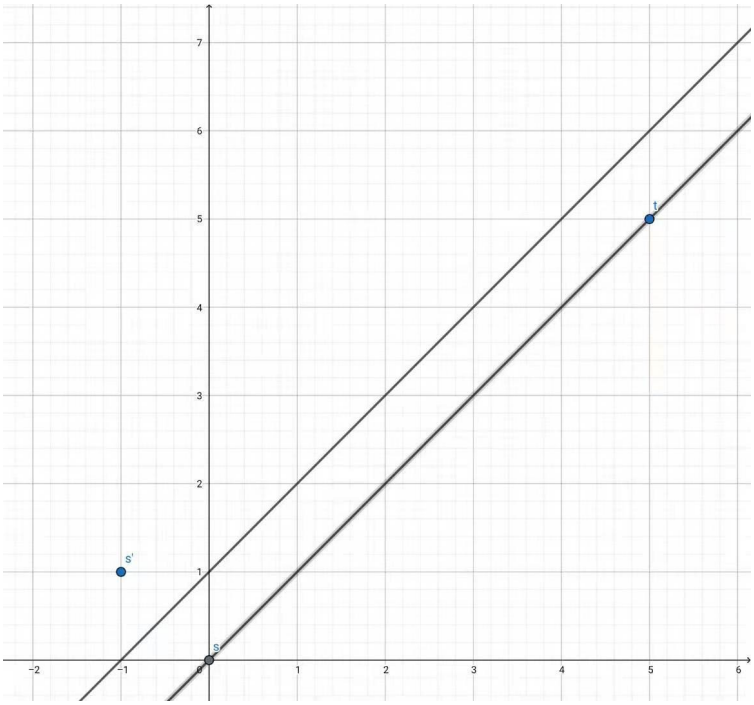
实际上满足卡特兰数性质的问题有很多，进栈出栈问题，二叉树子树问题，凸多边形的三角形划分等。值为 n 的卡特兰数我们定义为C_n，0 和 1 的卡特兰数为 1，通过以下递推公式我们可以由子问题推导出C_n。

$$C_n = \sum_{i=0}^{n-1} C_i \, C_{n-i-1}$$

方法三：DP 卡特兰数-dp 法

方法四：数学

一、非降路径问题。



对应于进栈出栈问题，实际上就是上图中点 s 到点 t 的所有路径中只位于 x=y 这条直线以下的路径总数。而不考虑路径限制的条件下，点 s 到点 t 的路径总数为组合数 C(2n,n)，我们要从这些路径里面剔除不满足条件的路径，而不满足条件的路径一定是经过直线 y=x+1 的路径，我们如何计算所有经过直线 y=x+1 的路径呢？这里的一个技巧，我们取 s 关于直线 y=x+1 的对称点 s'，s 到 t 经过直线 y=x+1 的路径总数与 s' 到 t 的路径总数相同，这样我们就得到了卡特兰数的数学解：

$$C_n = C(2n,n) - C(2n,n-1)$$

二、01 序列问题。

假设我们用 0 代表进站，1 代表出站，那么 01 序列问题中，满足进栈出栈限制的序列是从左往右数，数字 0 的数目总是大于等于 1 的序列数目。

证明：

显然我们只需要从所有可能的 01 序列中减去不满足条件的 01 序列即可。那么如何求解不满足条件的 01 序列呢？对于一个不满足条件的 01 序列一定存在某一个奇数前缀是满足条件的，而后半部分不满足条件，那么如果我们把后半部分的 01 序列中的 0 和 1 互换，那么每一个不满足条件的 01 序列就对应着一个由 n+1 个 1 和 n-1 个 0 对应的 01 序列。

另一种思路是，不合法的序列是某一个前缀中 1 的数目多于 0 的数目的序列，那么我们从 2n 个位置中选取 n-1 位置作为 0，那么剩余的 n+1 个位置都是 1，如果我们将剩余 n+1 个位置中的最后一个位置变成 0，那么此时该序列一定是不满足条件的。

2. 队列

队列是一种先进先出结构，常用来做层次遍历的结构。

① 队列例题。

小组队列 答案

② 单调队列问题

滑动窗口最大值 答案

最大子序和变形 答案

③ 优先级队列问题：

采用优先级队列的问题，往往与贪心的思想是相互配合的，例如非常经典的 Dijkstra 算法，

例题：

A. 包含每个查询的最小区间

方法一：

这道题的贪心策略相对简单一些，对于将要查询的点 i，我们要查找覆盖该点的最小区间的编号，所以我们只需要将所有区间按照左端点排序，将所有左端点小于查询点 i，右端点大于等于查询点 i 的区间加入优先级队列（优先级队列以区间长度为比较维度），即可求解出一个查询的最优值，而对于所有查询，我们可以将查询排序，在按照单个查询相同的思路，从左到右遍历每一个查询，同时将满足要求的区间加入，不满足要求的区间删除即可求解该问题。

方法二：

除了方法一以外，根据数据范围，我们还可以利用映射的关系进行求解，首先将所有的区间排序，然后建立一个 10^7 的数组用来记录每个位置对应的区间最小值，然后再将所有的区间覆盖的区域进行更新，最终得到每一个位置的最小值，然

后我们直接在处理后的数组上查询即可。

方法三：

线段树，采用线段树这一数据结构来完成方法二中对于区间的更新，然后直接返回要查询的值即可。

B. 批处理任务

3. 链表和邻接表

① 链表是一种非常重要的结构，我们要熟练掌握建立链表的细节。

② 链接链表是一种表示图的数据结构，通常我们建立领接链表的方式有以下两种：

A. 用 vector 动态数组模拟链表。优点是便于理解实现，缺点是时间复杂度稍高。

B. 用兄弟链表法，优点是速度快，缺点是不便于理解。

4. 哈希

① 哈希表是一种非常有用的数据结构，利用哈希表我们可以设计出很多巧妙的结构和算法。

② STL 里面的 unordered_map 以及 unordered_set 就是哈希表，这两者是利用开散列的方式解决冲突的，也即通过拉链法解决冲突。

③ 哈希表用于解决某些构造性问题时会有奇效。

④ 计数排序就是基于哈希思想的。

例题：

找出和为指定值得下标对 答案

字符串哈希：

① 字符串哈希是将字符串映射成一个 k 进制数的过程，不过映射后的数可能比较大，我们通常取这个字符串对应的数的 hash 值，k 我们通常取为 131 或者 13331，此时哈希冲突的概率极低。

② 此时我们可以用字符串的哈希值来比较两个字符串是否相等。这个思想其实就是 RK 字符串匹配的思想。

③ 最长回文子串问题可以中字符串哈希+二分的方式将时间复杂度维持在 O(nlogn) 级别。

5. 字符串

字符串处理是计算机里面一个非常重要的工作，本质上字符串问题没有特定的算法，我们只是利用已有的算法来解决字符串问题。最为经典的字符串问题应该是以下几个：

① 字符串匹配

A. KMP 算法：KMP 模板

周期 答案

B. RK 算法

② 最长公共子序列（后续 DP 部分介绍）

③ 最长回文子串（后续 DP 部分介绍）

④ 字符串的最小表示

A. 隐藏密码（模板题）答案

上述三个问题的求解方法实际上都是 dp，其中字符串匹配的 KMP 算法相对难以理解，但是理解好 dp 数组代表的状态的含义，KMP 算法也不会很难记住。

6. 字典树

① 字典树是另一种快速实现字符串匹配的数据结构，他是一个多叉树结构，这个多叉树结构是比较松散的，与我们平常理解的树结构是一致的。

② 字典树中保存字符串每个字符的实际上是多叉树中的边，每个节点链出的边就代表

了一个字符。

例题：模板

A. 前缀统计 答案

B. 数组中两个数的最大异或值 答案

C. 最长异或值路径 答案

7. 二叉堆

① 实际上就是堆，也被称作是优先级队列，前面在介绍优先级队列时已经提到过了。

例题：

A. 超市 答案 方法二

B. 增长的内存泄露 答案

② 霍夫曼树——霍夫曼树是对堆的一个应用，通常用来做字符串压缩的工作。霍夫曼树的构建过程我们可以用以下几个步骤来描述：

A. 首先将所有的字符权值加入到堆中。

B. 当堆中元素为 1 时，算法结束，返回该节点为霍夫曼树根节点。

C. 从堆中连续弹出两个最小权值的点，合并这两个点为根的子树，并生成一个新的节点作为两个子树根节点的父节点，新生成的节点的权值为两个子树权值和。将合并后的节点插入到堆中。

D. 重复步骤 B，C。

③ 多叉霍夫曼树的构建方式与二叉霍夫曼树稍稍不同，我们需要先补充一些权值为 0 的节点，使得最后一步时可以恰好只剩 1 个根节点。

8. 总结

学习算法总结不能少，这一章结束后我们算是稍微进了算法的大门，了解了一些基础的数据结构，本章知识点主要如下：

栈、队列、链表与邻接链表、哈希、字符串、字典树、二叉堆。

① 活学活用才是硬道理。

搜索

1. 树与图的遍历

① 树的遍历是处理图问题最为核心的一种手段，我们可以使用一些而外的标记数组来记录遍历过程中的数据。

② 除了显式的图结构、树形结构以外，实际上现实中的很多问题都可以抽象成对图的搜索，计算机解决问题的本质其实就是搜索可行解，如果我们把搜索过程中的每个中间状态表示为图中的一个节点，而将不同中间态之间的转化关系看作是图中的边，那么所有问题都可以抽象为使用搜索算法在可行空间里面搜索解的过程。

2. 深度优先搜索

① 深度优先遍历需要借助递归来实现，最基本的深度优先搜索问题我主要列出了四种，其中树的深度是基于自顶向下的思想实现的，数的重心是基于自底向上的思想实现的，无向图的连通分支则是最朴素的 dfs 搜索过程，拓扑排序是对 dfs 的扩展应用，指出了标记数组的作用。

例题：

A. 树的深度：模板

B. 树的重心：模板

C. 无向图的连通分支数：村村通 答案

D. 拓扑排序：

a) 模板题

b) 确定比赛名次 答案

② 搜索与剪枝往往是相辅相成的，合理的剪枝会显著地降低搜索的时间，但是剪枝不会改变问题的复杂度级别，原来的算法是什么样的的时间复杂度，剪枝后也依然是该时间复杂度。

A. 将字符串拆分成递减的连续值

3. 剪枝
4. 迭代加深
5. 广度优先搜索
6. 广搜变形
7. A*
8. IDA*
9. 总结

数学知识

1. 质数
2. 约数
3. 同余
4. 矩阵运算
   ① 矩阵转置
   ② 矩阵旋转
   旋转可以通过转置+交换的方式来实现。
   A. 旋转盒子 答案
5. 高斯消元与线性空间
6. 组合计数
7. 容斥原理与莫比乌斯函数
   ③ 概率与矩阵乘法
8. 数学期望
9. 0/1 分数规划
10. 博弈论

博弈论是起源于微观经济学，是以数学为基础，与经济学相结合的边缘学科。既然称之为博弈论，那么它研究的问题中设计的主体至少有两个。博弈论研究的是参与博弈的多方都站在各自最优的角度博弈时，最终可以达到一个均衡状态，达到这个状态所做的一系列决策就是博弈的结果。

我们来介绍一些关于博弈中的术语，首先加入参与博弈的只有两方，我们将先行动的人称之为先手，而将后行动的人称之为后手。如果存在某一种先手行动方式，使得后手不管怎么操作都面临必败局面，那么我们成先手必胜，反之我们也称之为后手必胜。

当然很多问题并不是询问博弈双方谁获胜，另一类问题的目标是询问达到博弈均衡状态是先手获得最大收益，我会在后续的例题中介绍。

① NIM 博弈

NIM 博弈是一个非常经典的博弈问题，问题是这样描述的，给定 n 堆物品，第 i 堆物品有A_i个。两名玩家轮流行动，每次可以任选一堆，取走任意多个物品，可以把一堆都取光，但是不能不取。取走最后一件物品者获胜。两人都采取最优策略，问先手是否能够必胜。

NIM 博弈有一个非常巧妙的结论——当$A_1 xor A_2 xor \dots xor A_n \neq 0$时，先手必胜。

反证法：

对于任意一个局面，如果$A_1 xor A_2 xor \dots xor A_n = x \neq 0$，设 x 的最高位 1 在第 k 位，那么必然存在一个A_i，它的第 k 位时 1，此时我们得到$A_i \ xor \ x \leq A_i$，那么我们可以从A_i中取掉若干个石子，使得剩余石子数目为$A_i \ xor \ x$个，此时就得到了一个各堆石子异或值等于 0 的局面。

对于一个各堆石子异或值为 0 的局面，我们按照 NIM 游戏的规则，不可能在一次取石子时，使得下一个局面中各堆石子的异或值为 0，所以这个局面一定是必输的局面。因为每次必须至少取一个石子，那么对于有限的石子来说，所有的石子一定会被取完，而各堆石子异或值为 0 的局面下，玩家不可能一次使得石子数变为 0，所以当前局面一定不是最后拿石子的人，所以当前局面一定是必输局面。

这样我们可以得到各堆石子异或值不为 0 的局面是必胜局面。反之则是必输局面。

例题：

A. NIM 博弈 答案

② 公平组合游戏 ICG

若一个游戏满足以下三个条件，那么它是一个公平组合游戏。

一、有两名玩家交替行动。

二、在游戏进程的任意时刻，可以执行的合法行动与轮到那名玩家无关。

三、不能行动的玩家判负。

③ 有向图游戏

对于一个有向无环图，图中有一个唯一的起点，在起点上有一枚棋子，两名玩家交替地将这枚棋子向前移动，每次可以移动一步，不能移动的玩家判负。

公平组合游戏都可以转化为有向图游戏。具体过程就是将博弈过程中的每个局面抽象成一个点，而将可能的行动抽象成每一条边。

④ SG 函数

我们首先定义 mex 运算，mex(S)定义为不属于集合 S 的最小非负整数。

$$mex(S) = \min_{x \in N, x \notin S} x$$

根据以上定义我们将 SG 函数定义为下面的表达式：

$$\mathrm{SG}(x) = \mathrm{mex}(\{\mathrm{SG}(y_1), \mathrm{SG}(y_2), \dots, \mathrm{SG}(y_k)\})$$

上式中，y_i表示从x可以到达的局面。特别的整个有向无环图的 SG 函数定义为：

$$\mathrm{SG}(G) = \mathrm{SG}(\mathrm{s})$$

⑤ 有向图游戏的和

设$G_1, G_2, \dots, G_m$是 m 个有向图游戏。定义有向图游戏 G，它的行动规则是任选某个有向图游戏G_i，并在G_i上行动一步。G被称为有向图游戏$G_1, G_2, \dots, G_m$的和。

有向图游戏的和的 SG 函数等于它包含的各个子游戏的 SG 函数的异或和。

$$\mathrm{SG}(G) = \mathrm{SG}(G_1) \ xor \ \mathrm{SG}(G_2) \ xor \dots xor \ \mathrm{SG}(G_m)$$

⑥ 定理

⑦ 最优状态博弈

例题：

A. 石子游戏 1 dp 法 脑筋急转弯
B. 石子游戏 2 答案
C. 石子游戏 3 答案
D. 石子游戏 4 答案
E. 石子游戏 5 正向 dp 记忆化搜索
F. 石子游戏 6 贪心
G. 石子游戏 7 答案
H. 石子游戏 8

总结：

通过上面一系列例题我们可以总结出来这样几个规律。

A. 博弈论的最优化问题通常是通过动态规划求解的，通常动态规划中的 dp 数组不具体指某一个人的收入，而是任何一个当前玩家在当前状态下可以获得的最优值，这样对于先手操作的人，我们只需要知晓其初始状态就可以可到解。

B. 想要求解此类博弈问题，我们要明白怎问题怎么定义，以及如何进行递推。石子游戏是一个典型的问题，直观地讲我们很难按照正向的方式去思考求解出最优解。因为最优博弈过程可能在正向思考时并不是局部最优的，相当于强化学习中，我们只是做出一些操作，甚至它的收益可能并不高，但是此时的操作会影响我们后面的选择，后面选择的收益会很大。也即我们需要做出一些折中的操作，来影响后面的操作，最终使得收益反馈最大。基于此，我们按照正向递推不可能得到最优解。

C. 基于以上讨论，我们要求解此类问题，最重要的是搞清楚什么是子问题，什么是父问题。此时我们就可以通过子问题递推父问题的方式来求解。回想以下石子问题 1，最大的父问题是长度为 n 的区间中当前玩家获得的最大收益，而最小的子问题可能是任意长度为 1 的区间的玩家获得的总收益。最开始时我们并不知道最优博弈会到哪一个终点，但是我们可以假设最开始时任意一个区间都可能是最优博弈的最后一步，那么我们就可以以此为基准，以及求解出更大区间的子问题对应的最优解。可以看出实际上博弈论中的子问题实际上就是更大的博弈问题已经操作了几步后的问题。

D. 确定父子问题更无脑一点的想法是，最大的父问题往往是博弈的初始状态对应的问题，例如石子游戏 2 中，博弈的最初始状态就是位于第一堆石子，M=1 时博弈的最优价值。假设其那面的部分已经被取完，那么从任意一堆石子开始，M=j 时的状态就是一个子问题，而子问题递推过程实际上是从后往前递推的。

E. 博弈论的问题我们用记忆化搜索往往比较容易实现，这是因为这本身就与博弈论的思路契合，但是从记忆化搜索我们也可以反推出递推方式的动态规划过程。

11. 总结

数据结构进阶

1. 并查集
2. 树状数组

3. 线段树
4. 分块
5. 二叉搜索树与平衡树初步
6. 离线分治算法
7. 可持久化数据结构
8. 总结

动态规划

1. 思想

在具体介绍动态规划之前我先介绍一下动态规划的思想，以及遇到一个问题我们该如何思考，得到动态规划的递推顺序以及状态转移方程。

动态规划实际上是对搜索问题的剪枝，我们在搜索那一节中提到的记忆化搜索就是动态规划的思想。首先介绍一些书上对于动态规划的抽象性描述。

能够使用动态规划的问题，具有三个重要的特点，①问题可以划分成等价的子问题，同时子问题必须有重叠性，否则搜索过程中用空间换时间就没有意义，因为对于每个问题来说，它的子问题都需要单独计算，因为没有重叠性。②递归过程无后效性。③最优子结构性质。

①问题可以划分成子问题，是可以使用动态规划技术的基本要求，因为动态规划的本质是将搜索过程中的中间态进行存储，用时间换空间的技术。如果问题不可划分，且子问题不重叠，那么 dp 数组也就无用了，这样动态规划实际上就是无效的。

②动态规划递推过程要求无后效性，所谓无后效性就是递推关系形成的图是一个有向无环图。怎么来看待递推关系形成的图呢？如果我们把搜索过程中的每一个数据稳定的中间状态看做是一个节点，那么这些节点之间的搜索关系的反向关系构成的图就是一个有向无环图。

举个例子，对于最为经典的斐波那契数列来说，如果我们用搜索的方式进行求解，那么我们是从 n 开始搜索的，而每个一个数据稳定的状态我们可以看做是某一个递归栈内保存的数据，也即每个递推体里的状态，对于斐波那契数列问题来说，每个递归体里面稳定的数据只有传入参数 n，所以我们可以用 dp[i]来记录 i=n 时这个递归体里的状态。

状态之间的关系其实就是递归调用的关系，我们发现对于 dp[n]这个状态来说，它在递归函数体内调用了 dp[n-1]以及 dp[n-2]，所以 dp[n]有对 dp[n-1]和 dp[n-2]的搜索关系，反过来将 dp[n-1]和 dp[n-2]到 dp[n]就有了子问题到父问题的递推关系，也即我们上面描述的搜索关系的反向关系，可以发现这个例子递推关系形成的图是一个有向无环图。

那么我们该如何递推呢？根据我们上面的描述，要递推出原始问题的解，就要从它的子问题进行倒推。这样实际上动态规划的过程，就是我们是从最初的子问题开始逐步递推出每个更大子问题的过程。而这个递推顺序实际上就是有向无环图的拓扑序。

动态规划的第二个条件是“最优子结构性质”，因为动态规划总是被用来求解最优解，那么我们在递推的过程中要保证可以从子问题的最优解地推出更大子问题的最优解，否则也不能使用动态规划。对于最优性问题来说，我们在表述其状态时要做到不漏即可，而对于计数类动态规划问题，我们要保证不重不漏。

动态规划我自己总结为三步：

中间态表示：所谓中间态表示，就是用一个 dp 数组来表示记录每一个数据稳定的状态的过程。值得注意的是这里的中间态不是针对结果数据规模的中间态，而是分析动态规划问题时，由子问题推导到结果问题时的各个中间态。例如在斐波那契数列这个例子中，中间态指的是有规模为 0,1 的子问题推导出规模为 n 的问题时的每一个中间态 k，其中 k∈[2,n]。

离散值决策：对于每一个状态，它可行的决策可能只有几种，也即从它导出的有向边有多少，我们就需要在这里进行决策。

确定递推顺序：得到了递推顺序以后，实际上我们通常可以根据一二两步得到递推顺序，但是有些问题可能需要先预处理数据，然后再做这三步。

当然上述三个步骤不一定完全按照描述的一二三步进行思考，但是大部分问题是符合这一特征的

2. 线性 DP
3. 背包问题
4. 区间 DP
5. 树形 DP
   ① 树形 DP 是指在树形结构上进行 DP 的问题，一定要注意是在树形结构中，而不是任意的图结构，所以我们也可以基于这一性质来判断是否可以用树形 DP 求解该题。
   ② 树形 DP 通常使用 DFS 遍历的方式进行 DP，DP 的思路通常是有子树推导出更大的树的 DP 值，最后得到整个树的 DP 值，当然我们也可以反其道而行，但是需要我们给子节点传递父节点的信息，不如自底向上的推导。
   ③ 例题：
      A. 有向图中的最大颜色值(与求解树的直径的问题类似)
6. 环形与后效性处理
7. 状态压缩 DP
8. 倍增优化 DP
9. 数据结构优化 DP
10. 单调队列优化 DP
11. 斜率优化 DP
12. 四边形不等式优化 DP
13. 计数类 DP
    ① 计数类 DP 是动态规划中特殊的一类，在求解最值的动态规划问题中，子问题之间是可以重叠的，这不会影响最终结果的最优性。但是在计数类 DP 问题中，我们要保证统计的过程不重不漏。
    ② 计数类 DP 与组合数学的关系比较紧密，我们要对组合数学中的组合数，阶乘非常熟悉。因为组合数之间有很多变换规则，同一个问题不同的思考方式也会带来截然不同的效果，我们用例题来统计分析出一些规律。

    例题：

    A. 恰有 K 根木棍可以被看到的排列总数 答案

    思路：

    本题的思路还是比较有技巧性的，对于没有接触过此类问题的人来说，还是很难在短时间内想到解法。从常规的动态规划思路出发，搜索的中间态是数量为 i，可以看到的木棍数为 j 时的方案数目，也即我们可以用 dp[i][j]表示搜索过程中的一个中间态。

    那么我们如何决策呢？首先考虑数量为 i 的木棍，可以看到 j 个木棍的状态

是什么样的。显然这种状态下，最高的木棍一定是最后一个可以被看到的，在它后面的木棍我们就无法看到了。那么 dp[i][j]可能由那些状态转化过来呢？根据前面的分析，我们需要按照最高的木棍的位置来分类讨论。

首先，如果最高的木棍在最后一个，那么此时 dp[i][j]可以由所有 dp[i-1][j-1]的可行方案最后面添加一个高度为 i 的木棍转化而来。

其次，如果最后一个木棍看不到，那么它的高度只能是[1,i-1]，此时，除了最后一个木棍外，剩余的木棍是一个木棍数量为 i-1 的子问题，只要该子问题中可以看到的木棍数量是 j，那么按照最高木棍一定是最后一个被看到的，而最高的木棍又在木棍数量为 i-1 的排列中，那么只需要把最后一个木棍加入木棍数量为 i-1 的子问题的可行解的后面，即可得到一个木棍数量为 i 的子问题的一个可行解。按照组合数原理我们有 dp[i][j]+=dp[i-1][j]*(i-1)。

综上，根据我们分析得到的两个决策，我们可以得到状态转移方程 dp[i][j]=(dp[i-1][j-1]+dp[i-1][j]*(i-1))。

可以看出以上思路与我们求解组合数时考虑的种种情况相似，所以对于计数类 DP 问题，我们需要与组合计数的思想结合，才能较快地解决问题。

除了上述思想外，其实还有另一种思路，我们之前的思考中子问题是前 i 个木棍可以看到 j 个的方案数的问题，但是实际上我们可以不用定序，而将子问题描述为，i 个木棍可以看到 j 个木棍的方案数，那么对于 dp[i][j]来说，任意从 i 个木棍中挑出 i-1 个木棍的问题都可以作为更小规模的子问题。根据我们之前的分析，最高的木棍一定是最后一个被看到，那么如果从 i 个木棍中挑选的木棍包括 i，那么剩余的木棍一定不能被看到，那么此时 i-1 个木棍中必须要有 j 个被看到，如果高度为 i 的木棍没有别挑选，那么挑选的 i-1 个木棍里面只有 j-1 可以被看到才能满足 dp[i][j]的要求。

B.

14. 数位统计类 DP
15. 总结

图论

1. 最短路
2. 最小生成树
3. 树的直径与最近公共祖先
4. 基环树
5. 负环与差分约数
6. Tarjan 算法与无向图的连通性
7. Tarjan 算法与有向图的连通性
8. 二分图的匹配
9. 二分图的覆盖与独立集
10. 网络流
11. 总结

C++ STL 用法总结

一、 容器：

序列式容器：

1. Vector
2. Array
3. List

关联式容器：

1. Set:底层结构式红黑树
   红黑树是一种特殊的二叉搜索树，具有以下特点：
   ① 根节点为黑色。
   ② 不能存在连续两个点都为红色。
   ③ 任意一条从根节点到叶子节点的路径上黑色节点的数目相等。
   ④ 所有叶子节点都是黑色的。
   ⑤ 树的高度查不超过两倍，这样查询次数为 log n 级别的。
   这里再额外介绍一下 set 的非精确查找的两种方式：
   ① Lower_bound()方法，查找的是大于等于目标值的第一个元素。
   ② Upper_bound()方法，查找的是大于目标值的第一个元素。
   ③ 在使用这两个方法时，我们需要注意判断边界条件，避免超出范围。
   最近的房间
   对于使用 set 的问题，除了离散化以外，如果单纯地想要使用平衡二叉树的结构来实现快速插入和删除，并维持数据有序的问题，我们可以用 set。
2. Multiset: 底层结构是红黑树
3. Map
4. multimap
5. Unordered_set
6. Unordered_map

二、 算法：

三、 迭代器：

四、 仿函数：

五、 适配器：

六、 配置器：