codeforces.com 解题报告

张昆玮

2019 年 6 月 6 日

20C Dijkstra?

使用堆优化的 Dijkstra 算法。为了使用系统内置的 priority_queue 模板库，在堆里存放的应该是二元组 {顶点, 临时标号}，同时又因为修改临时标号只能越改越小，所以可以在修改时不删除旧的二元组，只添加新的二元组。如果从优先队列中拿出的顶点已经处理过，就忽略。

提示 Dijkstra 算法的本质就是不停地使用贪心算法，把临时标号固定成永久标号的过程。

26A Almost Prime

筛法求出所有质数，然后递归生成所有符合条件的数。

提示 欧拉筛法和埃氏筛法都可以。

380C Sereja and Brackets

处理区间查询问题的有力武器：线段树。观察到满足区间加法，因此可以使用。每个点上要记录未匹配的左括号，右括号，和已经成对的括号。

提示 使用 zkw 线段树速度较快。

26B Regular Bracket Sequence

贪心算法。每个右括号都和最临近的左括号相匹配，如果不能匹配的就丢弃。

提示 从左到右扫描，累计当前嵌套的括号层数，是括号序列的常见思路。

86D Powerful array

莫队算法。基本想法是把查询分为 $\sqrt{n}$ 的网格状，然后蛇形计算。同一个网格里面计算顺序是任意的。

提示 莫队算法有很复杂的变形，但是基本如此题，应该熟练掌握。

2B The least round way

动态规划。需要算 3 次，一次是假设 2 的幂限制了末尾的零，一次是假设 5 的幂限制了末尾的零，一次是假设答案就是零。取最终答案是三个答案中最小的。

提示 让两个值中较小的最小，分类讨论。如果让两个值中较大的最小，需要增加一维状态，较难。

547B Mike and Feet

考虑输入中的每个值对输出的贡献。每个点作为区间最小值，所能贡献的最大区间长度，是从它向左右尽量延伸且不小于它的数构成的。算出这个最大区间长度，然后用当前点的值去更新答案。但是直接去更新需要更新很多答案，不如先更新最大的区间长度，然后对答案求一个后缀 max.

提示 要求的答案比较多时，考虑每个输入对答案的贡献是可行的方法。

607B Zuma

如果首尾相同，那么去掉首尾答案不变。如果首尾不同，那么首和尾属于不同的回文串，枚举分割点可以转化成更小的子问题。这样就变成了一个动态规划问题。

提示 在做题时，不要忽视一些显然的性质。往往利用这些性质把特殊情况排除以后，就能简化问题或者看出思路。

484B Maximum Value

先把除数相同的放在一起考虑。排序一下，模的最大值总是一段一段的，因此枚举段数可能是一个好办法。用二分检索或者预处理可以找到每一段里面最大的，更新结果。

提示 与顺序无关的问题都要先排序为好。

269B Greenhouse Effect

由于移动一棵树总可以移动到正确的位置上，相当于删除这棵树。因此就是删除尽可能少的树，即最长不下降子序列。

提示 经典题目经常会稍稍改头换面出现，需要加强思维来做出这些题目。

463D Gargari and Permutations

考虑最终答案，一个数在另一个数前面，就是要求在每一个输入中，它都在前面。如果把边定义为“在前面”，就是求一个最长路，是一个经典的记忆化搜索。

提示 本题目的 DP 解法本质和这个一样。

482B Interesting Array

按位分开后，如果 AND 结果为 1，其实是要求这段数全都为 1。令没有要求的都为 0。这样就转化为区间查询 AND ，同时允许修改：某区间同时 OR 一个数。使用带标记的线段树即可。

提示 使用自顶向下的 zkw 线段树可以处理带标记的题。

505C Mr. Kitayuta, the Treasure Hunter

容易想到令 $f(\text{start}, d) =$ 从 start 点出发，初始跳跃距离为 d 的最多宝石数。但是有很多状态是浪费的，因此用记忆化搜索。可惜内存不够用。注意到 d 是连续变化的，因此有些 d 是不会被用到的。因此交换两维顺序，d 的一维现用现开即可。

提示 好不容易得到的思路别放弃，一点点修补，就能得到答案。

1C Ancient Berland Circus

计算几何。先设法算出外接圆直径，再依次试出最小的边数。最后计算面积。

提示 复数在平面计算几何中很好用。还有好用的比如向量的点积叉积等。

159D Palindrome pairs

计算出每个点结尾的回文串数，然后求前缀和。计算出每个点开始的回文串数，分别乘以上一个点的前缀和就是答案。

提示 有 $O(n)$ 的算法求出最长回文串，可以学习一下。另外，从中间开始枚举所有回文串也是个不错的方法。

479E Riding in a Lift

动态规划。令 $f(\text{start}, \text{length}) =$ 从 start 层开始，走 length 这么多步，总方案数。我们有

$$f(x, \text{length}) = \sum_{x-|x-b|<i<x+|x-b|} f(i, \text{length} - 1) \ .$$

令 $g(x, y) = \sum_{i=1}^{x} f(i, y)$，则

$$\begin{aligned}
&g(x,y) - g(x-1,y) \\
=&f(x,y) + \left(\sum_{x-|x-b|<i<x+|x-b|} f(i, \text{length} - 1) \right) - f(x, y-1) \\
=&g(x+|x-b|-1, y-1) - g(x-|x-b|, y-1) \\
&- g(x, y-1) + g(x-1, y-1)
\end{aligned}$$

这样能计算出 $g(x, y)$，从而计算出 $f(x, y)$。

提示 动态规划前缀和是经典优化。

455B A Lot of Games

容易想到是字典树 Trie。博弈论最基本的方法就是动态规划计算哪些状态有必胜，这个题目中有时可能要双方争败，因此还要计算哪些状态有必败策略，方法是把叶子标成必胜，然后套用求必胜策略的求法。最后按照是否有必胜策略，是否有必败策略分类讨论就可以了。

提示 博弈论最基本的方法就是动态规划计算哪些状态有必胜，即一个状态先手必胜，等价于它的后继状态中，存在一个后手必胜。

455B A Lot of Games

容易想到是字典树 Trie。博弈论最基本的方法就是动态规划计算哪些状态有必胜，这个题目中有时可能要双方争败，因此还要计算哪些状态有必败策略，方法是把叶子标成必胜，然后套用求必胜策略的求法。最后按照是否有必胜策略，是否有必败策略分类讨论就可以了。

提示 博弈论最基本的方法就是动态规划计算哪些状态有必胜，即一个状态先手必胜，等价于它的后继状态中，存在一个后手必胜。

61E Enemy is weak

三个数之间的关系一般从中间一个入手。也就是求以 i 结尾的逆序对的数量和以 i 开始的逆序对的数量，使用线段树就可以了。

提示 尽量不要在程序中复制粘贴同样的代码块。仔细想想能不能重用。

449B Jzzhu and Cities

其实就是判断最短路径最后一跳的入度是否唯一。修改 Dijkstra 算法，如果更新时遇到相等的情况，就标记成非唯一，反之标记成唯一。

提示 1 非负权图上的 SPFA 已死。

提示 2 注意重边的情况。