《石中剑的考验》命题报告

June 12, 2014

题目大意

题目大意

给出 $1 \sim n$ 的一个排列的一个最长上升子序列（长度为 k），求原排列可能的种类数。

题目大意

给出 $1 \sim n$ 的一个排列的一个最长上升子序列（长度为 k），求原排列可能的种类数。

- 对于 30% 的数据，$1 \leq n \leq 11$。

题目大意

给出 $1 \sim n$ 的一个排列的一个最长上升子序列（长度为 k），求原排列可能的种类数。

- 对于 30% 的数据，$1 \leq n \leq 11$。
- 对于 70% 的数据，$1 \leq n \leq 14$。

题目大意

给出 $1 \sim n$ 的一个排列的一个最长上升子序列（长度为 k），求原排列可能的种类数。

- 对于 30% 的数据，$1 \leq n \leq 11$。
- 对于 70% 的数据，$1 \leq n \leq 14$。
- 对于 100% 的数据，$1 \leq n \leq 15$，答案小于 2^{31}。

算法一

算法一

经过简单思考可以发现，原排列合法的充要条件为：

算法一

经过简单思考可以发现，原排列合法的充要条件为：

- 给出的子序列是原排列的子序列；

算法一

经过简单思考可以发现，原排列合法的充要条件为：

- 给出的子序列是原排列的子序列；
- 原排列的最长上升子序列长度为 k。

算法一

算法一

由此，直接的想法就是用 DFS 枚举排列，要求给出的子序列中的数依次出现，同时在枚举过程中通过简单的动态规划求出最长上升子序列长度来进行合法性判断。

算法一

由此，直接的想法就是用 DFS 枚举排列，要求给出的子序列中的数依次出现，同时在枚举过程中通过简单的动态规划求出最长上升子序列长度来进行合法性判断。

可以适当加一些剪枝进行加速。

算法一

由此，直接的想法就是用 DFS 枚举排列，要求给出的子序列中的数依次出现，同时在枚举过程中通过简单的动态规划求出最长上升子序列长度来进行合法性判断。

可以适当加一些剪枝进行加速。

时间复杂度 $O(n!n)$，空间复杂度 $O(n)$，期望得分 $15 \sim 35$ 分。

算法二

算法二

仍然从算法一中提到的条件出发，考虑状态压缩动态规划。

算法二

仍然从算法一中提到的条件出发，考虑状态压缩动态规划。

第一个条件是比较容易满足的，可以用 n 位二进制数表示每个数是否出现过。考虑从当前状态出发进行转移，枚举序列的下一个数 x，有以下两种情况：

算法二

仍然从算法一中提到的条件出发，考虑状态压缩动态规划。

第一个条件是比较容易满足的，可以用 n 位二进制数表示每个数是否出现过。考虑从当前状态出发进行转移，枚举序列的下一个数 x，有以下两种情况：

- 若 x 不在给出的子序列中，则不受影响；

算法二

仍然从算法一中提到的条件出发，考虑状态压缩动态规划。

第一个条件是比较容易满足的，可以用 n 位二进制数表示每个数是否出现过。考虑从当前状态出发进行转移，枚举序列的下一个数 x，有以下两种情况：

- 若 x 不在给出的子序列中，则不受影响；
- 若 x 在给出的子序列中，则要求其在子序列中的前一个数已经在当前状态中出现过。

算法二

算法二

第二个条件是本题的瓶颈所在，必须要能够通过状态表示在枚举序列的下一个数 x 后求出以 x 结尾的最长上升子序列的长度。

算法二

第二个条件是本题的瓶颈所在，必须要能够通过状态表示在枚举序列的下一个数 x 后求出以 x 结尾的最长上升子序列的长度。

显然，如果把以当前出现过的每个数结尾的最长上升子序列长度都记录下来是无法承受的。而要想不记录以当前出现过的每个数结尾的最长上升子序列长度就能转移，这就使我们联想到经典的 $O(nlogn)$ 求长度为 n 的序列的最长上升子序列的算法。

算法二

算法二

在这个算法中，我们需要实时维护长度为 i 的上升子序列结尾的最小值，这是一个单调增的序列，在这里称其为最小结尾序列。

算法二

在这个算法中，我们需要实时维护长度为 i 的上升子序列结尾的最小值，这是一个单调增的序列，在这里称其为最小结尾序列。

那么，我们同样可以用 n 位二进制数表示每个数是否在最小结尾序列中，这样利用其单调增的性质就能恢复出最小结尾序列，从而在枚举序列的下一个数 x 后以 $O(n)$ 的时间复杂度求出以 x 结尾的最长上升子序列的长度。

算法二

算法二

由于在最小结尾序列中的数一定是已经出现过的数，我们把上述两个 n 位二进制数合并为一个 n 位三进制数即可。

算法二

由于在最小结尾序列中的数一定是已经出现过的数，我们把上述两个 n 位二进制数合并为一个 n 位三进制数即可。

时间复杂度 $O(3^n n^2)$，空间复杂度 $O(3^n)$，期望得分 $50 \sim 70$ 分。

算法三

算法三

为了进一步提升速度，我们需要在算法二的基础上进行一些改进，考虑从状态和转移两个方面进行优化。

算法三

为了进一步提升速度，我们需要在算法二的基础上进行一些改进，考虑从状态和转移两个方面进行优化。

转移的优化是比较显然的，由于我们从小到大枚举序列的下一个数 x，所求得的以 x 结尾的最长上升子序列的长度必然是单调不减的，因此整个转移过程中最小结尾序列中的数最多被比较 $2n$ 次，故转移的总时间复杂度可以优化到 $O(n)$。

算法三

算法三

遗憾的是，上述优化效果并不明显，我们必须从状态着手。可以想象的是，在算法二中，真正可能形成合法原排列的状态其实很少，因此去除冗余状态成为了优化的关键。

算法三

遗憾的是，上述优化效果并不明显，我们必须从状态着手。可以想象的是，在算法二中，真正可能形成合法原排列的状态其实很少，因此去除冗余状态成为了优化的关键。

为了判断一个状态是否真正可能形成合法原排列，我们尝试寻找一个原排列合法的必要条件：

算法三

遗憾的是，上述优化效果并不明显，我们必须从状态着手。可以想象的是，在算法二中，真正可能形成合法原排列的状态其实很少，因此去除冗余状态成为了优化的关键。

为了判断一个状态是否真正可能形成合法原排列，我们尝试寻找一个原排列合法的必要条件：

- 以最长上升子序列中第 i 个数结尾的最长上升子序列长度为 i。

算法三

算法三

由此，在枚举序列的下一个数 x 后，对当前状态中未出现过的最长上升子序列中的数进行逐一判断，若最长上升子序列中第 i 个数大于 x 且以 x 结尾的最长上升子序列的长度不小于 i，则当前状态不能通过 x 转移到新状态。

算法三

由此，在枚举序列的下一个数 x 后，对当前状态中未出现过的最长上升子序列中的数进行逐一判断，若最长上升子序列中第 i 个数大于 x 且以 x 结尾的最长上升子序列的长度不小于 i，则当前状态不能通过 x 转移到新状态。

虽然上述判断使得转移的时间复杂度又回到了 $O(n^2)$，但其对冗余状态的减少量是相当可观的（令合法状态总数为 s，在 $n = 15$ 时最多为 10^5 级别），至此我们比较精巧地实现了该状态压缩动态规划。

算法三

由此，在枚举序列的下一个数 x 后，对当前状态中未出现过的最长上升子序列中的数进行逐一判断，若最长上升子序列中第 i 个数大于 x 且以 x 结尾的最长上升子序列的长度不小于 i，则当前状态不能通过 x 转移到新状态。

虽然上述判断使得转移的时间复杂度又回到了 $O(n^2)$，但其对冗余状态的减少量是相当可观的（令合法状态总数为 s，在 $n = 15$ 时最多为 10^5 级别），至此我们比较精巧地实现了该状态压缩动态规划。

时间复杂度 $O(sn^2)$，空间复杂度 $O(3^n)$，期望得分 $85 \sim 100$ 分。