今日长缨在手，何时缚住苍龙?

: D

缘起

因为191的点子不错，很容易实现，于是在原来拆分的基础上，三下五除二搞定了。学是好学，但重码率不理想，但也比86好啦。

所以着手研究192，其实我最初思路就是做极致低重，所在在重启这个思路时不算是从零开始，反而是积累（zhe ten）了很多经验。比如贪婪算法。

开始我觉得退火没什么，怀着将信将疑的心态去自己试了下，没有按正统的公式去算，只是按自己的理解。

于是乎，好像效果还不错？但我发现个奇怪的现象，就是当我挂了3晚上退火后，后面都一无所获，并且奇异的是每次启动程序时能去重。

此中必有猫腻。

后面经过慢慢研究，发现，退后触发后，由于设置的条件让其很难再次回归到之前的状态，所以差值越积越多，导致后期根本没法再次超越之前的重码。

所以我就想，能不能在尝试退火后再次回到原来的状态？！

Yes!

答案是肯定的，但问题时，实际上实现不是那么容易的。

并且我发现，退火主要采用随机法，不是说这个方法不好，只是毕竟是随机，难免走重复路，不是那么有效率。

所以我想，能不能直接穷举完?

Yes!

经过我前期的退火实践，我观察到，如果采用每次随机一个字根 + 一个新键位的随

机法，这完全是可以穷举完的好不好，是字根数 * 25，因为自己不能随机到自己的键位，所以只有25，也就是说，这个数值并不大，基本上只需要几千到上万，而对于计算机或者说我使用的Rust语言来讲，很轻松啦。

所以这是一个优化点。

其次，我想，有一种方法swap，通过交换两个字根原本的键位是不是也可以?

Yes!

这个方法是从@秋风知道的。

并且，和普通随机一个字根不同的是，swap能一次性随机两个字根，但如果按贪婪来算，第一次失败是不会再接受第二次哪怕成功的组合的，所以理论上讲这个swap法和单一替换法不同（后面成single法）。

继而可以推理出的，还可以一次性换3个字根，换4个字根再一起结算等等，然而这种组合可以通过退火一定数量的字根实现，只是这个过程非常漫长，但我感觉基本上是可以涵盖的，并且当组合数越多，实际上去重效果也是越差的，所以不必死磕这种高维的组合随机法。

Ok，说完了“杀手锏”去重法，下面说说去重遇到瓶颈时的催化剂，退火🔥🔥。

而退火，才是突破局部最低值，超越更高层的法宝，普通的贪婪不行。

那么退火该怎么退呢？是否也有穷举的可能?

Yes!

比如我可以先退火一个字根，即无条件接受结果，如果能去重更好，不能也欣然接受，就好像以退为进，以守为攻，这次的退让是为了下次更好的冲锋，就是这个道理。

而实际上想想，假如存在一个最理想的字根布局，是否全程贪婪就能达到呢?

我想答案是否定的，因为实践证明，退火后稍微跑一跑就能超越原来的重码，所以纯贪婪是难以达到这个布局的。

那么我们可以先在所以字根中，按照一定顺序挨个选择一个字根，然后在挨个选择25个键位，分别进行退火的操作，如果退火后有收获，即重码降低了，那么可以再次从头开始退火。

那么只有这一种退火方案么？No，No。

这时候就要发挥排列组合功效啦，也就是我算法名字的来源，我们可以再在线选择两个字根的基础上，再次选择K为2的笛卡尔积，只是要过滤掉原本字根的键位，因为这种前面肯定算过啦。

那么以此类推。。我们可以再选择三个字根，再算K为3的笛卡尔积。。最后到字根总数的组合。

但实际上是几乎不可能达到的，因为这个组合退火法实际上第一步是选择一定数量的字根组合，而组合数计算公式我们都知道已经到达O(n!)的恐怖地步了，而在选择一个字根的组合时，还需要再选择对应数量的笛卡尔积，而笛卡尔积的时间复杂度更夸张，达到了O(x^n)，也就是最高的复杂度，指数级别的，我想，要跑完全部的情况，得到宇宙爆炸💥了。

并且由于组合数，笛卡尔积数字巨大，你也没法申请这么大内存（因为需要随机，所以要提前申请内存）。

所以在实际操作中，能达到K值得数量我相信不会太高，因为时间和空间双双都不允许。

但也没关系，只是这个思路是对的嘛，我们可以从K最小的开始跑，这样本身退火成功率也会高些。

好了，说完了大体思路，下面讲具体算法。

注意，这个算法我是Rust实现过了，代码放在https://github.com/wyhhh/19wubi上，速度非常快。无人能敌。感兴趣的可以去看下。

算法简单讲解

程序整体分为4个模块，分别是Single贪婪，Swap贪婪，Undo撤销和退火模块。

Single贪婪

对N个字根挨个确定，然后对25个键位挨个尝试，如果成功，则回到Single状态。失败进入下一状态。

Swap贪婪

大体和Single一致，只是需要一次确定两个字根，然后交换它们的键位。

成功回到初始Single，失败则进入下一个状态Undo。

Undo

这时算法灵魂所在，为了保证退火失败后能继续在之前的制高点继续前进，需要回退原先的尝试。

退火

这则是“点题”所在，是为了让程序不陷入局部最优，而慢慢尝试直到收敛到某个点。

无条件接受。

另外，所有以上提到的需要先随机shuffle一下数组，以保证下次遇到时不会走原来的路。

- 暮歌丶 2021.2.11.01:09