剪枝

简介

关于剪枝这个概念，有的同学对机器学习有一定了解的肯定能脱口而出：“剪枝是为了解决 决策树过拟合，为了降低模型复杂度的一种手段。”

而我们这次要介绍的剪枝又何尝不是为了降低我们所写程序的时空复杂度呢?

我们在日常编程中或多或少都用到了剪枝，只不过大家没有系统去了解过这块的概念而已， 所以希望通过这个专题将大家对剪枝中模糊不清对概念有一个比较清晰的认识。

剪枝的概念

剪枝最常出现在搜索相关的问题上，我们常用的搜索算法，其实描绘出的搜索空间就是一个 树形结构，我们成为搜索树。

算法中的剪枝指的是**在搜索过程中，提前退出根本不可能是答案的决策分支，进而减少时 空复杂度**。由于我们提前退出了，因此**搜索树的规模自然就会变小**，形象地看像是一 棵树被我们减掉了。日常生活中剪枝剪的就是树的枝桠，在搜索树中剪枝剪掉的就是**必得 不到解的子树**来减小搜索空间。

算法中的剪枝和现实生活这工人剪枝是非常类似的，只不过目的不同。我们是为了提高算法 性能，而现实中则是为了植物更好生长，亦或是为了美观。

那如何剪枝？常见的剪枝策略又是什么?

剪枝遵循的三原则

- 正确性：这个很好理解，我们把这个树杈剪掉的前提是剪掉的这块一定不存在我们所要搜 寻的解，不然我们把正确结果都剪没了。这对应上文的**减掉的是必得不到解的子树**。
- 准确性：我们在保证正确性的前提下，**尽可能多**的剪掉不包含所搜寻解的枝叶，也就 是咱们剪，就要努力剪到最好。这里的关键是是**尽可能多**。
- 高效性：这个就是一个衡量我们剪枝是否必要的一个标准了，比如我们设计出了一个非常 优秀的剪枝策略，可以把搜索规模控制在非常小范围，很棒！但是我们去实现这个剪枝策 略的时候，又耗费了大量的时间和空间，是不是有点得不偿失呢？也就是我们需要在算法 的整体效率和剪枝策略之间 trade-off。

常用的剪枝策略

- 可行性剪枝：如果我们当前的状态已经不合法了，我们也没有必要继续搜索了，直接把这 块搜索空间剪掉，也就是 return。比如题目让我们求某个数组的子集，要求子集个数不 大于 5。那么当我们暴力回溯的时候，如果已经选择的数字

大于 5 了，那么就没必须继 续选择了，可以提前退出。实际中的可行性剪枝则更加复杂，需要根据题目进行分析 。**这是搜索回溯问题的难点**

eg:

```
def dfs(pos, path):
    # 从题目中挖掘非法状态，在这里判断，提前退出以达到剪枝的目的
    if 不合法: return
    # go deeper
```

- 记忆化：常做 dp 题的同学应该也知道，我们把已经计算出来的问题答案保存下来，下次 遇到该问题就可以直接取答案而不用重复计算。这为什么叫剪枝呢？试想，如果你不使用 记忆化，那么搜索树的规模则很大（通常是指数），而使用了记忆化则可最好可优化到多 项式，这时搜索树的规模是不是变小了？这就是剪枝。

eg:

```
memo = {}
def dp(i):
    if i <= 0: return 0
    ans = max(dp(i-1), dp(i-2))
    memo[i] = ans
    return ans
```

- 搜索顺序剪枝：在我们已知一些有用的先验信息的前提下，定义我们的搜索顺序。举个最 简单例子，有时候我们正序遍历数组遇到答案返回，这种解法会 TLE，但是，我们倒着遍 历却过了，这就是对搜索顺序进行剪枝。再比如回溯法解决背包问题，当我们先在背包中 放入一个较大的物品，那么剩余的搜索空间就更小，选择也更少。这比先放小的，最后放 大的（发现放不下）在很多情况下都快。**力扣中的很多回溯题都使用了这个思路，值得 大家重点关注**。

> 背包问题应该使用 dp 来做，这里只是举个例子。

eg:

```
nums.sort(reverse=True)

def knapsack(pos):
    pass
knapsack(0)
```

再比如 **5302. 加密解密字符串** 的解密等价：先将 dictionary 加密统计频率 fraq，然后返回 word2 在 fraq 的频次即可 。 这种方法之所以可以达到剪枝的目的，核心点在于**从起点有很多分叉，然后某一个叶 子分叉节点才能判断是否合法，而如果从终点搜索，那么只有一条链路**，复杂度大大降低 。 具体到这道题就是一个字符串解密后对应的是多个字符串，这么多字符串要

一一验证是 否在 dictionary 中很复杂。反向思考，我们可以直接从 dictionary 出发（这是搜索终点 ），加密（反向搜索）后看是否和 word2 相等，如果相等，那么就是我们要找的字符串。

- 最优性剪枝：也叫上下边界剪枝，Alpha-Beta 剪枝，常用于对抗类游戏。当算法评估出 某策略的后续走法比之前策略的还差时，就会剪掉该策略的后续发展。
- 等等。

用好剪枝，会让我们的算法事半功倍，所以大家一定要掌握剪枝这一强力的思想。

总结

练习剪枝最有利的方式就是通过回溯法。 由于回溯本质就是暴力穷举， 因此如果不剪枝很 可能会超时。 毫不夸张地说，剪枝剪地好，回溯 AC 少不了。

比如 N 皇后问题，如果暴力穷举就是 N^N，如果使用剪枝则可大大减少时间，可以减 少到 $N!$。推荐一篇 N 皇后的文章给大家 https://old-panda.com/2020/12/12/eight-queens-puzzle/?hmsr=toutiao.io&utm_medium=toutiao.io&utm_source=toutiao.io

上一页 下一页

© 2020 lucifer. 保留所有权利