《剪枝》解题报告

杭州学军中学 李恺威

目录

题目简述

题目描述的是一棵点带权的有根树，每个点的孩子是从左到右有序的。一棵树的价值在题目上已明确定义。剪枝的本质是将某些结点的子树全部删去，使自己作为新的叶结点。为了方便起见我们把新的叶结点称为标记点。标记点满足如下性质：

1、 两个标记点互不为祖孙关系；

2、 从根到任意一个叶结点有且仅有一个标记点。

算法 1

有 20%的数据，$n\leq 20$。可以考虑用搜索解决。搜索的方法有很多种。比较方便的是用 $O(2^n)$枚举每个点是否标记，再从根开始按题意遍历每个点，遇到标记的点就回朔。一边遍历，一边统计路径上的最大点权。忽略那些未遍历到的标记点。对于每种枚举的方案，就可以在 $O(n)$时间内就出总价值，用它去更新最优答案。总复杂度为 $O(2^n \times n)$。虽然这个算法有些冗余的部分，但写起来非常简便，无论是用来做骗分还是对拍，都是很值得去写的。

寻找突破口

通过对一些例子进行分析，我们可以发现一些性质。在这之前，为了方便叙述，先对一些术语下定义：

对于结点 i，它的孩子从左到右依次为 Pi_1，Pi_2，……，Pi_{Ti}。

定义结点 i 为结点 $Pi_j(1\leq j\leq Ti)$的父亲，记 Fa $[Pi_j]$＝i；

定义结点 i 的大儿子为结点 Pi_1，记 Lson[i]＝Pi_1；

定义结点 i 的小儿子为结点 Pi_{Ti}；记 Rson[i]＝ Pi_{Ti}；

定义结点 Pi_j (j>1)左兄弟为结点 Pi_{j-1}；记 Lbro[Pi_j]＝ Pi_{j-1}；

定义结点 Pi_j (j＜Ti)右兄弟为结点 Pi_{j+1} 记 Rbro[Pi_j]＝ Pi_{j+1}；

定义结点 i 的左链集合为 Lchain[i]，满足 i∈Lchain[i]，对于任意非叶子结点 j∈Lchain[i]，Lson[j]∈Lchain[i]；

定义结点 i 的右链集合为 Rchain[i]，满足 i∈Rchain[i]对于任意非叶子结点 j∈Rchain[i]，Rson[j]∈Rchain[i]；

定义 Next[i]为 DFS 序列中，在 i 之后的第一个非 i 后代的结点 j，Next[i]＝j。当然，如果 i∈Rchain[1]，那么 Next[i]不存在；

定义一个合法的有序标记点集合 S＝{S_1,S_2,…,S_m}。序列 S_1,S_2,…,S_m 必定是 DFS 序列的子序列。

定义有点复杂，但理解了就很简单。接下来是一些性质：

性质 1：S_1∈{Lchain[1]}。

性质 2：S_m∈{Rchain[1]}。

性质 3：Next[i]的一种求法是，从 i 开始不停访问它的父亲，直到遇到拥有右兄弟的结点 j，Next[i]=Rbro[j]。

性质 4：对于 S_i(i＜m)，S_{i+1}∈{Lchain[Next[i]]}。

性质 4 的证明：假设它不成立，根据性质 3 的求法，那么从根开始到 Fa[j]的路径上均没有标记点，{Lchain[Next[i]]}也没有标记点。因此存在路径从根到 Fa[j]，再到 Next[i]，再不停访问大儿子，直到叶子结点，均没有标记点，与题意矛盾。所以性质 3 成立。

算法 2

经过上述分析，一个动态规划的思想便清晰了。F[i]表示将 i 作为标记点，考虑 DFS 序在 i 之后（含 i）的标记点，它们权值和减去相邻的影响值所能达到的最大值。于是有

$$\mathrm{F[i]} = \begin{cases} \mathrm{Max}\big(\mathrm{F[j]} + \mathrm{G[i][j]} \mid \mathrm{j} \in \{\mathrm{Lchain}\big[\mathrm{Next[i]}\big]\}\big) + \mathrm{w[i]} & \mathrm{i} \notin \mathrm{Rchain[1]} \\ \mathrm{w[i]} & \mathrm{i} \in \mathrm{Rchain[1]} \end{cases}$$

其中 G[i][j]表示 i 到 j 的路径上除 i、j 结点外最大的点权值。

通过预处理 next 和 G，将 i 按照逆 DFS 序做，整个算法可在 $O(n^2)$的时间内求出结果。其实 G[i][j]不需要用预处理求。只要先求出从 Fa[i]到 Fa[next[i]]路径上的最大点权值 d，当 j 从 next[i]开始不断访问大儿子时，同时用 w[j]去更新 d 就可以了。这样时间复杂度仍然为 $O(n^2)$。但每个状态的决策个数(也就是枚举 j 的个数)与树的结构有关，因此在某些情况下效率将远远小于 $O(n^2)$，而更接近于 O(n)。

更深的思考

题目上要求 n ≤100000，对于算法 2，是不能令人满意的。那么这个算法能再进行优化吗？

继续分析一些例子，可以发现更有用的信息。

性质 5：对于任意非叶子结点 i，有 next[i]＝next[Rson[i]]。

性质 5 的证明：令 j＝Rson[i]，因为 j 为 i 的小儿子，所以 j 没有右兄弟，即 Rbro[j]不存在。根据性质 3 对于 next 的求法，求 next[j]时，由于它没有右兄弟，因此便向上访问 Fa[j]＝i，接下来就与 next[i]的求法完全一致。所以 next[i]＝next[j]＝next[Rson[i]]。

性质 6：这是性质 5 的推广。对于任意结点 i，next[i]＝next[j] (j∈{Rchain[i]})。

性质 6 的证明：对{Rchain[i]}中的每个结点按生成的次序排好，形成一个序列 p(p[1]＝i,p[2]＝Rson[i],…)。首先，p[1]＝i，所以 next[i]＝next[p[1]]，满足性质。假设 p[k]满足性质，且 p[k]不是序列的末尾，对于 p[k+1]，因为 Rson[p[k]]＝p[k+1]，所以 next[p[k+1]]＝next[p[k]]＝next[i]。根据数学归纳法，性质 6 成立。

想必你已经明白我要说明什么了，但是我仍然要写。

对于结点 i，如果 i 有右兄弟或者，即存在 Rson[i]，或者 i=1，那么称结点 i 具有完整右链，即 Rchain[i]是完整的。

性质 7：如果结点 i 具有完整右链，那么对于结点 j，j∈{Rchain[i]}与 next[j]＝Rson[i]互为充要条件。

性质 7 的证明：

充分性。因为 i 有右兄弟，所以 next[i]＝Rson[i]，根据性质 6，有 next[j]＝next[i]，所以 next[j]＝Rson[i]。

必要性。如果 next[j]＝Rson[i]，根据性质 3，从 j 开始访问，必然在 i 处停下。路径上的任意非 i 的结点(含 j)均没有右兄弟，因此每一个点都是它父亲的小儿子，所以 j∈{Rchain[i]}。

所以性质 7 成立。

算法 3

稍微改动下算法 2，让 F[i]表示将 i 作为标记点，考虑 DFS 序在 i 之前（含 i）的标记点，它们权值和减去相邻的影响值所能达到的最大值。这并不影响算法的可行性，只要原来的算法反个方向就行了。更一般的，将整棵树左右反转，求出来的结果和方案都是完全等价的。

考虑这个新算法的顺序，是按 DFS 序做的。你可能本来就是这样想的。

根据观察转移的步骤可以发现，每次总是一个状态集合转移到另一个状态集合。实际上性质 7 就是为了说明这一点。我们考虑算法 2 的转移，对于任意兄弟结点 x,y 对，x＝Lbro[y]，y＝Rbro[x]，总是 Rchain[x]中的状态转移到 Lchain[y]的状态。那么，怎样利用这个特性呢？

首先，结点 x，y 拥有一个共同的父亲，记它为 r，r＝fa[x]＝fa[y]。记 Rchain[x]序列中第 i 个结点为 seq[i]，一共有 L 个结点。记 h[i]为的它的 f[seq[i]]值，m[i]表示从 fa[seq[i]]到 r 的结点中最大的权值(w 值)。

m[i]可以递推逐个求得：$m[i]=\begin{cases}\max(m[i-1],w[seq[i-1]]) & 1<i<L\\ w[r] & i=1\end{cases}$

让我们开始试着进行状态转移。让 i 依次访问 Lchain[y]序列中的每个元素，即 i 从 y 开始，不断访问它的大儿子。记 d 表示从 i 的父亲到 r 的结点中的最大权值（与 m[i]类似）。d 随着 i 的变化而不断变化。相比于算法 2 的转移，一条路径上的最大权值就变成了两条到 r 的路径的最大权值中的最大值，其实就是 max(d,m[j])。

状态转移方程稍微改一下，就是

$$F[i]=\max\{h[j]+\max(d,m[j])\}+w[i]\ (1\le j\le L)$$

将 max(d,m[j])分类讨论，状态转移方程就是：

$$F[i]=\max\begin{cases}\max(h[j]+d) & m[j]\le d\\ \max(h[j]+m[j]) & m[j]>d\end{cases}+w[i]$$

根据 m 的求法，我们可以知道 m[i]≤m[i+1]，即 m 是递增的。d 则可以看做是 m 序列中的分割线，令分割位置为 k，使其满足 m[k] ≤d 且 d<m[k+1]。为了不失一般性，我们规定 m[0]=0，m[L+1]=∞。

状态转移方程继续变化：

$$F[i]=\max\begin{cases}\max(h[j])+d & j\in[1..k]\\ \max(h[j]+m[j]) & j\in[k+1..L]\end{cases}+w[i]$$

注意 j 所取的区间，我们就可以在定义 h1 和 h2 来方便求解 max()。

定义 h1[i]=max(h[j]) j∈1..i；

定义 h2[i]=max(h[j]+m[j]) j∈i..L。

为了不失一般性，令 h1[0] =h2[L+1]=-∞。

新的方程产生了：

$$F[i]=\mathrm{Max}(h1[k],h2[k+1])+w[i]$$

每当求解 F[i]时，只要找到对应的 k 就能在 O(1)的时间求出结果。

k 可以通过二分找出。

如果你想出前面所有的步骤，但 k 却用二分来寻找，那就太假了。

考虑 i 的变化，每当访问新的 i 时，d 的值总是不减的（回过去看 d 的定义）。因此 k 总是随着 i 的变化而增加（至少不减少）。当 i 变化时，k 只要递增寻找，找到对应位置停下来就行。所以 k 总共变化不超过 L。

分析下时间复杂度。有 n 个结点，每个结点对应一种状态。每个结点只属于一个完整左链和一个完整右链。两条完整左链（右链）不会相交。每次转移时，都是将一条完整左链的状态转移到一条完整右链上。每次转移时的时间复杂度为 O(左链长度+右链长度)。因此总时间复杂度为 O(n)。

总结

出这道题,我的想法是提出一种新的的树形动态规划模式。它的做法不是一般的从叶子，向上到根，或者从根向下到叶子的顺序，而是以 DFS 序作为阶段，从左到右地做动态规划。希望这道题可以开拓树形问题解决的思路，对树的认识更深一层。