水题走四方 解题报告

湖北省武汉市第二中学 吕凯风

1 试题来源

我和彭雨翔同学一起出的题。可以在这里找到：http://uoj.ac/problem/84。

2 试题大意

今天是世界水日，著名的水题资源专家蝈蝈大臣发起了水题走四方活动，向全世界发放成千上万的水题。

蝈蝈大臣是家里蹲大学的教授，当然不愿意出门发水题啦！所以他委托他的助手欧姆来发。

助手欧姆最近做 UR #6 被狗狗传染了懒癌，当然不愿意出门发水题啦！所以他请来了高手——地卜师。

全世界一共 n 个城市，编号分别为 $1,\ldots,n$。城市之间由双向道路相连，形成了一棵树。如果这棵树以 1 为根，则除 1 以外每个结点 v 的父亲结点的编号 p_v 满足 $p_v < v$。

由于地卜师掌握了克隆的核心科技，把自己完全复制了一份，他和他的分身一共两个地卜师一起出去执行任务。

现在，两个地卜师都站在 1 号城市。地卜师可以沿道路行走，从一个城市移动到另一个城市。走一条道路需要耗时 1 小时。当然，地卜师可以停留在某个城市不动。每到一个没有发放水题的城市，地卜师都会发水题。在一个城市里发水题的时间不计。

由于地卜师有强迫症，他沿道路移动时总是从一个城市移动到编号更大的城市。即总是从一个结点移动到它的儿子结点。地卜师和他的分身可以同时移动。

但是地卜师这样好像不一定能经过所有的城市？没关系！地卜师会瞬移。如果某个时刻两个地卜师都在某个城市里（而不是在去某个城市的道路上），那么其中一个地卜师可以瞬移到另一个地卜师所在的城市。瞬移所需的时间不计。

现在地卜师想知道，要想顺利给每个城市都发放水题，最短需要多少个小时。

欧姆当然知道怎么计算啦！但是他想考考你。

由于一些原因，本题使用捆绑测试。每个子任务有若干个测试点，分为 4 个子任务，你只有通过一个子任务的所有测试点才能得到这个子任务的分数。

子任务	分值	n 的规模
1	20	$n \le 12$
2	30	$n \le 1000$
3	30	$n \le 10^5$
4	20	$n \le 5 \times 10^6$

由于后一个子任务的数据范围总是包括了前一个子任务的数据范围，所以如果前一个子任务的测试点没有全部通过，后面的子任务都会被跳过并计后面的子任务 0 分。

时间限制：1s

空间限制：512MB

3 算法介绍

3.1 算法一

对于子任务 1， $n \le 12$。

我们可以记下当前两个地卜师在什么位置，以及每个结点是否被访问过，然后 DP 就可以了。

可以获得 20 分。

3.2 算法二

对于子任务 2， $n \le 1000$。

记 $f[v]$ 为两个地卜师现在都在 v，遍历以 v 为根的子树的最小代价。然后显然应该是一个人等着，另一个人出去到处逛，最后还剩一棵子树的时候，两个人一起往下走。

大错特错！有可能剩一棵子树和一个叶子没访问过，而如果往这棵子树走，将会经过很长一段只有一个儿子的结点的路。那么我们此时可以让这两个地卜师一起往下走，一个走到叶子停下，另一个走到那棵子树中往下第一个有不止一个儿子的位置停下。如果是先到达了分叉点，就等一等那个往叶子走的地卜师走到后瞬移过来。

显然，我们可以枚举最后剩下的子树和叶子分别是哪个，然后 DP 算算代价。当然你也可以注意到，如果已知选哪个子树，显然选最深的那个叶子更优。

我不知道你们有没有一开始想错了，至少我一开始想错了，还以为是水题。

然后你会发现这个算法他过不了样例三!

我觉得大家这时候应该写个算法一然后对拍，就能知道为什么了。但是为啥大家都弃疗了呢？（我好像是 $n = 15$ 的时候狂拍拍出来的，因为反例很稀少）

那么究竟是怎么回事呢？

我们考虑一下，假如有两根很长很长的筷子，我们把用于吃饭的那一头粘起来并作为根结点。那么答案显然是一根筷子的长度。但是如果这两根筷子是劣质一次性筷子，在两根筷子的侧边长了一点点毛刺，这些毛刺比较靠近吃饭的那一端。那么最优解是什么呢？

最优解是：先派个地卜师下去把毛刺都去掉，然后再两个地卜师一起下去分别遍历两根筷子的主干部分。这样大约就只需要一根筷子的长度。而如果用刚才那个算法，会算出来是大约两根筷子的长度。于是这个算法又是错的。

孩子算法老是错，一定是姿势不对。

我们来重新考虑问题。现在有两个地卜师，一个是本体一个是分身，在树上乱逛。注意如果两个地卜师站在一起，在我们眼前一晃其实我们是分不清楚也不用分清楚谁是谁的，我们可以根据自己的需要调换两人的身份。所以对于任意一组移动方案，我们一定可以把这组方案调整成本体从来没有瞬移过的方案。

然后我们考虑本体，他一定是往下走几步，停几步，再走几步，再停几步。如果一组方案最后本体没有停在某个叶子，我们肯定可以让本体追随分身的脚

步停在某个叶子。

好，那么现在本体是从根走到了某个叶子 v，我们现在考虑其它还没有被访问的结点。我们称根到 v 的链为主链，那么我们就是要让分身来访问主链上支出去的那些子树。我们可以断言一组最优方案一定是这样的：主链上有一些关键结点，分身和本体本来是在一起走的，走到关键结点处，分身会下去访问这个关键点到下一个关键点之间所有支出去的子树的叶子，访问一个，瞬移回来。当还剩下一个叶子的时候，本体和分身一起往下，本体去下一个关键点，分身去叶子。分身到了叶子处就瞬移到本体所在位置。

这是因为，每个支出去的子树的叶子都是要访问的，访问肯定对应着“溜出去”和“瞬移回来”这两个过程，溜出去的位置到该叶子的距离就是时间。显然如果已知你只能选择哪几个地方（即关键点）之一溜出去，那么选择最近的那个最好。（其实不太显然，从一个关键点到另一个关键点，最后一个访问的叶子也可能节约时间，但是也是好证的）

对于“最后一个访问的叶子”，显然取最深的叶子最优。

所以我们用 DP 求出 $f[v]$ 为从根到关键点 v 时的最短时间，然后在每个叶子的 f 中取最小值就可以了。

假设已经知道是哪条主链，设 $d[v]$ 为结点 v 的深度，$d_m[v]$ 为主链上结点 v 往外伸出去的子树中最深的叶子的深度，$d_m[u,v]$ 为主链上结点 u 到结点 v（不包括结点 v）往外伸出去的子树中最深的叶子的深度，$d_s[v]$ 为结点 v 的子树中叶子的深度之和，$n_l[v]$ 为结点 v 的子树中叶子的个数。那么我们就能用 DP 求出 $f[v]$ 为从根到关键点 v 时的最短时间。

$$f[v] = \min_u\{f[u] + d_s[u] - d_s[v] - (n_l[u] - n_l[v]) \cdot d[u] + \min\{d[v] - d_m[u,v], 0\}\} \quad (1)$$

其中 u 是 v 的祖先。

然后你 dfs 一遍边 dfs 边 DP 就可以了。

使用暴力就能做到 $O(n^2)$，可以获得 50 分。

3.3 算法三

我们仔细观察这个行为：“当还剩下一个叶子的时候，本体和分身一起往下，本体去下一个关键点，分身去叶子。分身到了叶子处就瞬移到本体所在位置。”这里有一种情况是：关键点到叶子的距离比到下一个关键点短，那么分身

瞬移到本体位置后，就会一起向下走。因为最后一次走的叶子一定两个关键点间最深的叶子，所以重逢的位置到下一个关键点一定是没有支出去的子树的。我们不妨把这个重逢的地方也设为关键点。

这样，我们 DP 的时候就可以考虑两种情况：

1. v 的父亲只有 v 这一个儿子，用父亲的 f 加 1 更新 $f[v]$。

2. 考虑一个 v 的祖先 u，且 $d_m[u,v] \geq d[v]$，用“一坨式子”更新。

考虑情况二的不等式：$d_m[u,v] \geq d[v]$。如果最后一个叶子不是 u 伸出去的子树中的，那么遍历最后一个叶子的时候两个地卜师可以一起向下走一段再分开。

于是为什么我们不把这个分开的位置作为关键点呢？答案显然不会更差——首先一起走的部分还是一起走了，而“叶子到关键点距离和”又减少了。所以，我们只用考虑那些 $d_m[u]$ 比后面所有 d_m 都大的 u 进行转移就行了。

注意到，一个 $d_m[u]$ 就意味着子树中至少 $d_m[u] - d[u]$ 个结点，于是我们在 dfs 的时候如果维护一个单调栈记录所有“$d_m[u]$ 比后面所有 d_m 都大的 u”，栈的长度就是 $O(\sqrt{n})$ 的。

时间复杂度 $O(n\sqrt{n})$，可以获得 80 分。

什么？你想起了 NOI2014 的购票去优化 DP？感觉 $O(n\log n) \sim O(n\log^3 n)$ 的算法。我并没有试过，得分不明，祝你好运。

3.4 算法四

其实你只要接下来注意到，情况二中你只用考虑在 v 上方离 v 最近的，且 $d_m[u] \geq d[v]$ 的 u 就行了。

因为假设 $d_m[u_1] \geq d[v]$，$d_m[u_2] \geq d[v]$，且 u_1 是 u_2 的祖先，u_2 是 v 的祖先。那么从关键点 u_1 到关键点 v 所花的时间一定是不如从关键点 u_1 到关键点 u_2 再到 v 的——因为在一起走的部分还是在一起，而“叶子到关键点距离和”减少了。

所以，其实只要你把暴力DP想清楚了，这题真的就顺流直下，沦为了一个大水题……但是你不多想想，就成了恶心斜率优化题。

你可以 dfs 一下，维护一个 d_m 的单调栈。弹栈的时候由于是基于均摊的，不能再用 while 循环弹了，你可以在栈上二分出新的栈顶指针位置，存下这个

地方原来的值，push 进去，然后递归下去。回溯回来的时候把栈顶指针和值复原。

时间复杂度 $O(n \log n)$。本来我想卡掉的，但是常数太小了。可以获得 100 分。

3.5 算法五

你只要黑科技弹栈就能线性了。

观察题目性质，你每次 push 一个 $d_m[u]$ 的时候，$d_m[u]$ 其实只有两种值——以 u 为根的子树的最大值或次大值。因为 $d_m[u]$ 总是去掉一个子树，然后取其它子树中最深的嘛。

所以你就干脆把最大值暴力 push 进去递归非最深子树，然后再把次大值暴力 push 进去递归最深子树。这样看上去只是个骗分，但是经过分析可以知道是线性的算法。（留给读者自己分析，我就不分析了！对比下算法六可以看出些端倪）

3.6 算法六

还有别的线性做法。我们考虑一种树的链剖分，每个儿子选一个自己最深的子树，把它和这个子树间的边染黑。

那么你在 dfs 的时候，沿着黑边走就要 push 次大值，否则 push 最大值。

考虑一条黑链，不管你怎么 push 最大值或次大值，总不可能把这条黑链上方那条黑链 push 进来的东西弹掉——因为那个更深嘛。

这时就很明显了——往黑链外走，push 最大值是什么行为？会把栈 pop 到 dfs 到这条黑链起始端的那个时候的栈的状态，因为之前 push 的次大值都比最大值小。我们 dfs 时记录下链起始端的那个时候的栈的栈顶指针就行了。

而 push 次小值呢？显然可以暴力嘛。沿着一条黑链 push 次小值之后，其实 pop 次数是不会超过 push 进来的次数，也就是黑链的长度。所有黑链的总长度为 $O(n)$，所以这里暴力就可以了～

3.7 算法七

楼上的算法都太复杂了怎么办？没关系！有个更 naive 的线性做法。

我们按逆 dfs 序，把子树信息一个个合并到父亲上去。子树信息就是假设整棵树就只有这一棵子树时，每个结点最近的祖先使得有伸出去的子树深度超过自己（叫做up祖先好了）。

现在这个子树跟它的父亲接了起来，就拿父亲下现在已经有的子树跟新加进来的子树互相更新一下还没找到up祖先的结点。这个我们只用记录一个链表存下每个子树中所有还没找到up祖先的结点，合并时搞搞就行了。

代码短，常数小，速度快，萌萌哒 $O(n)$，可惜我卡不掉其它算法。（需要开邻接表还要dfs的做法都有点被卡内存危险。）

4 数据生成方式

随机数据很弱，关键在于要让up祖先放置在不是那么显然的地方。所以可以先随机一条长链作为主链，然后分成若干块，每块放几条长链覆盖主链上该块内的深度，然后再在下面撒几条随机的小树。

至于把 $O(n\sqrt{n})$ 的算法卡超时，可以按他的最坏情况构造，即很多条长度 $O(\sqrt{n})$ 的链从主链上伸出来，再在某个结点下放很多个叶子。

5 参考程序

- `meepo.cpp` 是参考程序（算法七）。
- `meepo_stack.cpp` 是算法六。
- `meepo_log.cpp` 是算法四。
- `meepo_sqrt.cpp` 是算法三。
- `meepo_tle.cpp` 是算法二。
- `meepo_wa.cpp` 是算法二的分析中提到的错误算法。
- `meepo_orz.cpp` 是算法一。