test

SOJ Round #12 题解

2020-02-21 22:27:18 **By test**

事实证明 SR #12 还是比某 skip1978 的 STR #13 良心的 

这次 SR 的题面有那么一点点长，一方面让大家体验一下出题人是如何将当前的新型冠状病毒时事与题目背景相关联，另一方面也算是让大家锻炼了一下阅读长题面的能力。

此外，SR #12 是首创的具有每道题具有二十多个子任务的比赛，让选手们体验一下部分分丰富但严格的比赛风格：降低骗分几率，但能提高区分度，整场比赛几乎就没有同分选手 ~~(看看你那 STR #13，所有人的分数都是 25 的倍数)~~。

下面就给出 SR #12 的题解啦。

冠状病毒

(首先注意到碱基互补配对原则是一个**双射**，因此可以直接忽略它，得到：)

简要题意：对于一个串 s，及一个整数 $l\in[0,|s|]$，定义 $E(s,l)$ 如下：

设当前你有一个串 $t=s[1..l]$，每次在 t 末尾随机增加一个字符 (字符集大小为 $2n$)，直到 s 作为子串出现即可。**增加字符**的个数的期望即为 $E(s,l)$。

现在，你有一个串集合 $S=\{s_0\}$ 和 q 次操作，每次操作要么是选择 S 中的一个串并在末尾添加字符，再插回 S 中，或者给定 S 中一个串 s 及一个整数 l，询问 $E(s,l)$ 的值，强制在线。

算法零

注意到前两个子任务没有 Q 事件，这意味着不需要输出，因此写一个**能在给定时间内正常运行且不产生输出**的程序即可，无论使用什么语言。

期望得分 **2** 分，实际得分 **2** 分。

这里声明一下：不要小看这 **2** 分，很(you)多(xie)选手因为写的不是标算，写了很多部(bao)分(li)分，导致在这个点 TLE (因为这个点的 n,q 是很大的)。于是，通过这个数据也希望大家无论你写的是不是正解，写的多少分，都加一句这个特判：因为这个特判也花不了你 3 秒钟时间：

```
if (id <= 2) return 0;
```

算法一

考虑暴力求 $E(s,l)$。

不难发现这个加串的过程可以看成一个模式匹配的过程，于是我们可以求出串 s 的 KMP 自动机/AC 自动机，然后每次在 t 尾部加字符相当于在 $2n$ 条转移边中随机取一条，问多少次后回到终点。

于是这就转化为了一个 KMP 自动机上的随机游走问题，可以列出变量后 Gauss 消元求解。注意到变量个数和字符串长度同阶，于是这个做法的复杂度约为 $O\left(q(n+q)^3\right)$，可以通过第 3 ~ 6 个子任务。

结合算法零，期望得分 **16** 分，参考提交：#72779。

算法二

考虑使用正经的方法求出 $E(s,l)$。

如果你做题百万，你能做到一个叫做 [CTSC2006]歌唱王国 的题，然后发现那个题的 $E(s,l)$ 的模型和这道题是相同的。

~~(ps: 其实我在出这题之前也没发现那道题，比赛现场有人就直接爆出来了)~~

所以这一部分我就直接给出结论了，具体的证明及分析可以参见那道题的题解。~~其实我只是懒得再写一遍过程了~~

最终的结论是：$E(s,l)$ 一定是一个整数，且

$$E(s,l)=\sum_{i=1}^{|s|}[s[1..i]=s[n-i+1..n]](2n)^i$$

于是通过该定义暴力计算 (可以 Hash 也可以不 Hash)，可以通过第 3 ~ 10 个子任务，期望得分 **27** 分，参考提交：#72435。

算法三

每次直接按照定义 (**(1)** 式) 去计算显得太暴力了，我们来分析一下集合的性质。

首先，注意到 $s[1..i]=s[n-i+1..n]$ 实际上就是说 $s[1..i]$ 是 s 的 border (含平凡 border)，于是我们考虑使用 KMP 代替暴力求解。

而一个串 (长度为 l) 的所有 border 的长度由下式给出：$\{l,next[l],next[next[l]],\cdots,\}$，其中 $next$ 是在 KMP 模式匹配中得到的数组，即 l 的最长非平凡 border 的长度 (**本质就是 border 的传递性**)。

于是可以使用 KMP 来替代上述过程。

其次，不难发现，我们在一个串的**尾部**添加字符，它的 $next$ 数组的变化，就是 KMP 算法所维护的。

即，依次检验原串的第 $l+1,next[l]+1,next[next[l]]+1,\cdots$ 个字符是否等于新加的字符，找到最长者，作为 $next[l+1]$。

于是可以通过 23, 24 两个子任务，期望得分 **35** 分。

算法四

对于离线的部分，我们可以预先建出整棵操作树，然后在树上 dfs 实现 KMP。

这样就可以将操作结构从链转移到树，不过注意到 KMP 复杂度是均摊的，转移到树后时间复杂度无法保证，因此只能通过随机数据。

~~不难注意到~~ (大胆猜想)，**一般生物体的碱基序列中不会有过多过长的 border**，于是可以近似看成随机，从而通过第 11, 19 个子任务。

算法五

其实这个算法变成在线也不难。

由于每次增加一个字符，我们可以看成是在原来的结构上**添加一个新节点**，从而动态构建操作树。

然后我们需要时刻寻找根到某个顶点的链上的深度为 d 的顶点相关信息，这实质上就是一个 **k 级祖先问题**，可以使用*倍增算法*来动态处理。

事实上，我们可以对每一个点记录 $next\,[i]$ 的深度之后，同时记录一下对应链上第 $next\,[i]+1$ 个字符。

这样，忽略 KMP 复杂度部分，时间复杂度就是 $O\,(N\log N)$ **(以下用 N 表示原题中的 $L+q$)** 而不是 $O\left(N\log^2 N\right)$ 了。

可以额外通过第 12, 20 个子任务，在常数较小的情况下可以获得 **77** 分，参考提交：#72436。

算法六 (std 算法)

现在问题的瓶颈是如何优化 KMP 的均摊复杂度。下面给出两种思路：

考虑为什么 KMP「可持久化」后时间复杂度是错的，比如这个例子：

$$s=\mathtt{a}^{10000}\mathtt{b}$$

那么，对于 s 的前 **10000** 个字符，均有 $next\,[i]=i-1$，而在最后一个字符时，它会对 $s\,[1..10000]$ 的所有 border (共 **10000** 个分别进行检验)。

于是，这一个字符就会耗费 $O\,(|s|)$ 的时间。

在一般的 KMP 中，我们会维护一个当前的 $next$ 指针，此时在这个 `b` 统计完毕后，它会在 **1** 的位置，从而下一个串接上来的时候又重新统计。

而在可持久化 KMP 中，这个位置可以被无数次地插入：比如现在插入 `b`，然后还是对于老串插入 `c`，插入 `d`，使劲插入这种慢吞吞的字符，于是就被 Hack 掉了。

那怎样才能可持久化呢？

首先，注意到如果 $next\,[i]$ 很大，它才会出现这种问题。而 $next\,[i]$ 大意味着什么？i 有很长的 border —— **i 有很短的周期**！

对，事实上就是利用这个简单的定理：

若 s 的最长 border 长度 $>\dfrac{|s|}{2}$，则 s 的所有长度大于 $\dfrac{|s|}{2}$ 的 border 的长度构成一个公差为 $|s|-L$ 的等差数列。

此时，不妨假设 $next\,[l]>\dfrac{l}{2}$，那么 l 就是一个周期串。设它的**最短周期**为 t，则 $l=t^k\cdot p$，其中 $p\sqsubset t$。

考虑 l 的 border 集合：不难发现 $t^k\cdot p,t^{k-1}\cdot p,\cdots,t\cdot p,p$ 均为它的 border，**且 $t^k\cdot p,t^{k-1}\cdot p,\cdots,t\cdot p$ 之间不存在其它 border**。

> 证明
>
> 设串 s 的周期长度 $|t|=T$，后缀长度 $|p|=P$。
>
> 于是显然有 $T,2T,\cdots,kT$ 为 s 的周期。
>
> 若存在额外的长度 $>P+T$ 的 border，说明存在一个额外的长度 $<(k-1)\,T$ 的周期 τ，且 $T\nmid\tau$。
>
> 因为 $T+\tau<kT\le|s|$，所以由**弱周期引理** (Weak periodicity lemma) 知，$\gcd(T,\tau)<T$ 也是原串的周期，与 T 是最小周期矛盾。

考虑 $t^k\cdot p$ 后面加入一个字符 c，我们需要依次判断每个 border 后是不是这个字符 c。如果此时 $t^{k-1}\cdot p$ 后面不是 c，由周期性立得 $t^{k-2}\cdot p$ 后面也不是 c，……，p 后面也不是 c。

由刚才的结论可知，一直到 $t\cdot p$ 为止**不存在其它** border，于是我们就可以直接跳到串 $t\cdot p$ 进行分析。即，令 $l\leftarrow l\bmod(l-next\,[l])+(l-next\,[l])$。

分析一下它的时间复杂度，若 $next\,[l]\le\dfrac{l}{2}$ 时，直接暴力转移，$l\leftarrow next\,[l]$ (上面的公式也适用)，否则，容易证明 $l'\le\dfrac{2}{3}l$。

因而，下一步的长度**不超过原来的三分之二**，从而只需要跳 $O\,(\log l)$ 次即可找到 `next`。

类似地，总复杂度还是 $O\,(N\log N)$，因为对于这个 $l\leftarrow l\bmod(l-next\,[l])+(l-next\,[l])$ (下简称 "$grandnext$") 也是可以再一个串找到 $next$ 后计算得出，从而在跳 $O\,(\log l)$ 次时可以 $O\,(1)$ 得到。

这里注意几个实现细节：

1. 跳转时一定是 $l\leftarrow l\bmod(l-next\,[l])+(l-next\,[l])$ 而不是其它乱七八糟的式子，否则~~如果 l 是**整周期串**会把最后一个周期都忽略掉~~由于 $t\cdot p$ 和 p 之间可能存在其它 border 而导致漏考虑。
   对比提交：#72820、#72891 和 #72892。
2. 在比较的时候，一定要拿当前最长的含周期串进行比较：即对于上面那个例子，我们判断 $t^{k-1}\cdot p$ 后面的字符是否为 c，后，如果答案是肯定的，则**结束于 $t^{k-1}\cdot p$ 后的位置** (即 $next$)，而不是 $grandnext$ 处。而且，在 $\log$ 轮中的每一轮都需要这么干，因为 $next\,[l]>\dfrac{l}{2}$ 可能会出现多次。
   对比提交：#72434、#72437 和 #72890，即下面这一段：

```
for (j = pa; j > 0; j = g[j])
    if (ch == fc[j]) {D = dep[ f[j] ] + 1; break;}
```

或者，也可以使用 Swiftlink 的方式来安全地跳过这些周期，参考提交：#72838。

总时间复杂度 $O\,(N\log N)$ (常数较小)。~~如果细节写挂了，则实际可得 40 ~ 97 分不等。~~如果注意到了所有细节并正确实现，则可以获得 **100** 分，参考提交 (std)：**#72892**。

算法七 (非 std 算法)

注：比赛中三份 AC 代码有一份 (#72838) 使用算法六 (std 算法及其变种)，另外两份 (#72775, #72836) 使用了下面将要介绍的非 std 算法。

考虑换一种方式来优化 KMP 的均摊复杂度。

回顾我们寻找 $next$ 指针的实质：

> 依次判断第 $l+1, next[l]+1, next[next[l]]+1, \cdots$ 个字符是否为给定字符 c，找到最前面的一个，或返回 0 (找不到)。

因此，如果字符集较小的话，我们可以对于操作树上的每个顶点，记录下如果它后面接上每个字符 c，上面问题的答案 (即至少跳几层 $next$ 后后面跟着给定字符 c)。

不难发现，它的实质就是构建出每个点的 KMP 自动机。

那么，和 AC 自动机的构造非常相像，当添加一个新字符时，先通过原串的 KMP 自动机找到 $next$ 节点 ($fail$ 指针)，然后考虑构建新的自动机。

找到 $next$ 后，首先，将 $next$ 节点的所有转移边拷贝过来，但是需要更改一条转移边 —— 即对 $next$ 节点本身的转移边 ($next \to next+1$)。

由于每添加一个字符，需要拷贝一个大小为 $2n$ 的数组，故时间复杂度 $O(N(\log N+n))$，期望得分 60 分。

但是，事实上注意到我们每次只需要更改一条转移边，即数组中一个元素的值。因此可以使用**可持久化线段树**来代替暴力的数组拷贝，从而单次修改 $O(\log n)$。

总时间复杂度 $O(N(\log N+\log n))$ (常数略大)，可以获得 100 分。

传播者

简要题意：给定一个具有 n 层的分层图，每一层由 k 个点，第 i 层和第 $i+1$ 层之间连了若干条有向边。

设 S,T 为两个点集，满足 S 中所有点在第 l 层，T 中所有点在第 r 层，且 $l<r$。

记 $cut(S,T)$ 为 S 和 T 的最小**点割** —— 即至少要去掉多少个点 (及其关联的边)，才能保证不存在一条从 s 到 t 的有向路径，其中 $s\in S, t\in T$，且 s,t 均未被去掉。

现在给定这张分层图，有 q 次操作，每次操作为改变一条边的存在状态或询问某两个点集的最小点割。

算法一

对于 $k=1$ 的情形，不难发现，当 $S=\varnothing \vee T=\varnothing$ 时，答案为 0，否则如果 l,r 两层所有边的都相连则答案为 1，否则为 0。

于是~~使用线段树~~**暴力**实现即可，期望得分 3 分。

算法二

考虑如何刻画两点间的最小点割。

由点的最大流最小割定理 ~~闫格定理~~ 可知它等于 S 集合到 T 集合的点不相交的路径的最大条数**点容量为 1 的最大流**。

于是我们只需要求出这个最大流是多少。

暴力使用 Dinic，可以通过 5, 6, 9, 14, 16, 20, 22 七个子任务，期望得分 33 分，结合算法一可得 36 分。参考提交：#72751。

算法三

使用牛逼的网络流板子或使用位运算优化网络流，可以额外通过第 7, 10 个子任务，期望得分 41 分，结合前述算法可得 44 分。参考提交：#72790。

算法四

注意到每一层的点数非常少，因此可以使用状态压缩来枚举每一层选择的可行点集，使用状态压缩 DP 完成，期望得分 $30\sim 40$ 分不等。

算法五 (std)

由于前述算法可标算没有关系，下面直接讲正解。

考虑我们所要求的量 —— 两个集合中点不相交路径的最大条数。

那么先考虑一个简化版的问题：给定两个满的点集，判断答案是否为 k (即是否能**满流**通过)。

这等价于，存在一个顶点的排列，使得 l 层的每个点能到达 r 层与之匹配的点，且经过路径**不相交**。

对于路径不相交，又注意到 k 的范围是 24，不难想到使用 Lindström-Gessel-Viennot 引理：**两组点的点不交匹配路径的带权带符号和，等于两两之间路径条数构成的行列式。**

那么，如果不存在这样的点不交匹配路径，则这个行列式的值一定是 0。

于是，我们只需要求两两之间的路径条数 —— 而这恰恰就是矩阵乘法所干的事情。

因此我们只需要求区间矩阵乘积的行列式是否为 0 即可。

慢着！这里行列式的结果是**带符号和**，因此万一路径不唯一，然后消成 0 了怎么办？

~~由熟知的技巧~~，对每一条边随机一个权值，那么在最终所得的行列式 ($\pmod P$) 中误判的概率就不超过 $\frac{1}{P}$ 了，实践知取 P 为 10^9 左右已经足够。

现在我们已经能判定答案是否为 k 了，那么怎么求具体答案呢？

~~枚举大法好~~ 其实不用那么暴力，我们来思考一下答案 (不妨为 c) 的实际意义：

一方面，既然答案可以为 c，那么存在源中 c 个点和汇中 c 个点，所得的导出子图存在点不交匹配路径，因而存在一个 $c\times c$ 的**子矩阵**，行列式不为 0。

另一方面，既然答案不能为 $c+1$，那么**不存在**源中 $c+1$ 个点和汇中 $c+1$ 个点，所得的导出子图中存在点不交匹配路径，因而任意一个 $(c+1)\times(c+1)$ 的**子矩阵**，行列式均为 0。

观察上面两条粉色性质，不难发现，这其实就是**矩阵的秩**的定义 (存在非零 r 阶子式，任意 $r+1$ 阶子式为 0)！

于是直接通过 Gauss 消元求矩阵的秩就可以了。

对于 S,T 不是全集的情况也比较容易，相当于求最终矩阵的一个**子矩阵的秩**。

对于修改的情况，每改一条边相当于修改一个矩阵，因此我们可以使用线段树等数据结构维护区间矩阵乘积 (因为有结合律)，每次询问只需要求区间矩阵积即可。

时间复杂度 $O\left((n+q\log n)k^3\right)$，可以通过所有子任务。

注：如果你的实现常数较大，则可以参考 #72749 的实现，选择一个满足 $kP^2 < 2^{64}$ 的素数 P 作为模数，然后循~~环展开~~ (使用宏定义) 将乘积的 k 项展开，最后再取一次模，可以获得较小的常数。

注：std 没有卡任何模数 (而且在正常的数据组数下也卡不掉，只要你是随机的权值，每次成功的概率至少是 $1-\frac{1}{P}$，于是 q 次成功的概率至少是 $1-\frac{q}{P}$，取 $P \approx 10^9$ 远远足够)。

医院

简要题意：给定一棵带点权 a_v 和带边权 $b_{(u,v)}$ 的树 $T=(V,E)$，你需要为每个点分配一个整数权值，满足 $\sum_{v\in V} w_v = 10^{10^{10}}$，求

$$\frac{\sum_{(u,v)\in E} b_{(u,v)} \cdot w_u \cdot w_v}{\sum_{v\in V} a_v \cdot w_v^2}$$

的最大值。

算法零

```
print(1)
```

~~期望得分 0 分~~，实际得分 **12** 分 (**证明见算法四**)，参考提交：#72763。

算法一

注意到 $\sum w_i$ 巨大无比，这暗示我们实际上可以求 (2) 式在正实数 $\mathbb{R}^+$ 中的最大值，而最终的精度误差可以忽略不计 (远远小于 10^{-11})。

而在实数范围内求最值，不难发现 (2) 式实际上是一个齐次式 (即所有变量同乘一个正实数结果不变)，因此我们可以**忽略 $\sum w_i$ 这个条件**，直接求最大值。

对于 $n=2$ 的情形，就是给定 a,b,c 求 $\frac{cxy}{ax^2+by^2}$ 的最大值，由~~众所周知的~~均值不等式，$\frac{cxy}{ax^2+by^2} \le \frac{cxy}{2\sqrt{ab}xy} = \frac{c}{2\sqrt{ab}}$，等号成立当且仅当 $\sqrt{a}x = \sqrt{b}y$，而这显然可以办到。

于是输出 $\frac{c}{2\sqrt{ab}}$ 即可，期望得分 **4** 分。

~~吐槽：比赛现场怎么那么多分子分母还能看反的啊，太不细心了！~~

算法二

对于 $a_i = b_i = 1$，且原树是星图 (菊花) 的部分分，注意到所有叶子的地位是相等的，于是大胆猜想在最优解中所有叶子的取值均相等。

于是，设根的取值为 x，每个叶子的取值为 y，则 $(2) = \frac{(n-1)xy}{x^2+(n-1)y^2} \le \frac{(n-1)xy}{2\sqrt{n-1}xy} = \frac{\sqrt{n-1}}{2}$，等号显然可以取到。

所以答案就是 $\frac{\sqrt{n-1}}{2}$，可以通过 15, 20, 26 三个子任务，期望得分 **12** 分，结合算法一可得 **16** 分，参考提交：#72780、#72859。

算法三

考虑一般的星图，设根为 x，叶子为 $y_1, y_2, \cdots, y_{n-1}$，则 $(2) = \frac{x \cdot \sum_{i=1}^{n-1} b_i y_i}{c \cdot x^2 + \sum_{i=1}^{n-1} a_i y_i^2}$

设答案为 λ，则不等式转化成

$$x \cdot \sum_{i=1}^{n-1} b_i y_i \le \lambda \cdot \left(c \cdot x^2 + \sum_{i=1}^{n-1} a_i y_i^2 \right)$$

我们还是想使用之前算法都使用过的**均值不等式**，那怎么使用呢？~~由初中数学老师教过的技巧~~，我们来**待定系数**！

考虑 y_i。等式右端为 $\lambda \cdot a_i \cdot y_i^2$，而等式左端为 $b_i \cdot x \cdot y_i$。

于是，我们需要 $\frac{b_i^2}{4\lambda \cdot a_i}$ 倍的 x^2 与它结合，则刚好能通过均值放到左端。即 $\frac{b_i^2}{4\lambda \cdot a_i} x^2 + \lambda \cdot a_i \cdot y_i^2 \ge b_i \cdot x \cdot y_i$。

那么，对比两次算得的系数，有 $\sum_{i=1}^{n-1} \frac{b_i^2}{4\lambda \cdot a_i} = c \cdot \lambda$，故 $\lambda = \frac{1}{2}\sqrt{\frac{1}{a_i \cdot c} \cdot \sum_{i=1}^{n-1} b_i^2}$，则在取 λ 时，可以恰好让 $\lambda \cdot c$ 倍的 x^2 恰好放到左边，不多不少。

因此等号也是容易取到的，因为每个子不等式相当于限制了 x 和某个 y_i 的比例。

可以额外通过 9, 12, 18, 23, 29 五个子任务，期望得分 **31** 分，结合算法一可得 **35** 分 (事实上 $n=2$ 一定是链，因此写对了自动就结合了算法一)，参考提交：#72802。

算法 ???

发现这是一个二次规划问题，于是首先二分答案，转判定性问题后开始乱搞，比如退火等。期望得分 **30 ~ 70** 分不等，详见 #72825，这里不展开。

(ps: 不过事实上可以通过把 checker 调成 10^{-9} 来安排这些算法。因为 std 实际上可以在挺快的时间内达到 10^{-11} 以内的精度)

算法四

我会打表找规律！如果你的数感比较好，你可以发现对于 $a_i = b_i = 1$，原树是链的部分分，答案就等于 $\cos\frac{\pi}{n+1}$，于是输出这个答案也能获得 **12** 分。

此外，由于 checker 精度 (10^{-5}) 比较松，所以算法零的 **1** 也能被水过去  ~~(所以这是个 feature 而不是 bug)~~

参考提交：#72867。

下面讲一下等号成立的条件，具体的不等式放缩直接通过相邻两项配比放缩即可：

设链为 $a_1, a_2, \cdots, a_n$，则取 $a_k = \frac{\sin \frac{k\pi}{n+1}}{\sin \frac{\pi}{n+1}}$。

算法五 (std)

我们延续算法三的思想，考虑对整棵树进行均值不等式的放缩。

同样先待定最终的答案 λ，得到

$$\sum_{(u,v)\in E} b_{(u,v)} \cdot w_u \cdot w_v \le \lambda \cdot \sum_{v\in V} a_v \cdot w_v^2$$

由于 $a_i, b_i \ge 1$，因此不需要考虑极端情况 (比如有 0 的情形)。

此时思路还是比较清晰的：考虑对于每条边 $b_{(u,v)} \cdot w_u \cdot w_v$，将它的两个端点在右端对应的表达式中取两项，使它们的乘积恰好为 $\frac{b_{(u,v)}^2}{4}$，然后可以通过均值传递。

因此我们的目标其实就很明确了：对于每个点 v，它现在有 $\lambda \cdot a_v$ 的系数，你需要将这些系数合理分配给**与它关联的每条边**，使得最终每条边的两个端点分配给它的系数乘积恰好为 $\frac{b_{(u,v)}^2}{4}$，即一个只与该边有关的正常数。

一旦如果做到了这点，λ 其实就是答案。

首先，这个小于等于肯定没问题，因为不等式 (3) 必须成立；其次，等号在所有情况下都能取到：

注意到每个不等式实际上就是限制两个 w_i 之间的**比例关系**，而树是没有圈的图，因此随便给定一个根节点的权值，则可以顺水推舟地 (dfs) 得到所有点相应的权值。

因此下面的任务就是求这个 "恰到好处" 的 λ。

首先，~~退一万步~~，假设我们已经得到了 λ，求每个点到每条边分配的系数。

不难发现这可以通过树形 DP：首先叶节点没有选择权，必须老老实实把它自己所有的系数分配给唯一一条边，而对于非叶节点 v，在 DP 的过程中，所有连向子节点的边的一端的系数已确定，从而可以推算出另一端 (即 v) 要分配的系数，从而由和 $\lambda \cdot a_v$ 可推出它分配给父节点的系数。

形式化地，设点 v 分配给父节点的系数为 f_v，则有转移方程

$$f_v = \lambda \cdot a_v - \sum_{c\in child(v)} \frac{b_{(c,v)}^2}{4 f_c}$$

那么，最终必有 $f_1 = 0$：因为根节点 (1) 没有父节点，因此它的所有系数都分配给子节点，从而不可能有多出来的系数去分配给不存在的东西。

那现在考虑不知道 λ 时该如何处理。

既然我们不知道 λ，那~~当然就~~先猜一个，设我们猜的值为 λ'。

然后考虑通过 (4) 式去算 DP 值 (分配系数)。如果 $\lambda' = \lambda$ (猜准了)，那么显然没有问题，否则，有两种情况：

1. $\lambda' > \lambda$ (猜大了)。
   此时，对于叶节点 l，显然由 f_l 会偏大，然后考虑逐级往上传递，由 (4) 式及单调性，不难发现每个 f_v 都会偏大。
   (事实上，一条边两端分配的系数的乘积固定，那么子节点大了，父节点必然小，从而一个节点向下分配的系数变小，从而向上分配的系数又会变大)
   于是，最终走到根节点时，一定有 $f_1 > 0$。
2. $\lambda' < \lambda$ (猜小了)。 此时，类似地，每个点的 f_i 都会变小。不过此时要注意的一点是，**一旦某个 $f_i \le 0$，则应该立即退出**，因为这显然能说明答案偏小 (因为在正常的 λ-分法中不可能出现非正值)，而避免它取倒数后出现减一个负数的问题。 那么，即使最后没有退出，也一定有 $f_1 < 0$。

于是，不难发现，我们在猜 λ' 的过程中，f_1 的值有着**明显的可二分性质**，因此只需把猜 λ 的过程换成**二分 λ 的过程**即可。

时间复杂度 $O(n|\log eps|)$，可以通过所有子任务，期望得分 100 分，参考提交：#72648。

要注意的一点是，最终答案的最大值的一个上界是 $\frac{\max b_i}{\min a_i} \cdot \frac{\sqrt{n-1}}{2}$，因此二分时上界至少需要开 40000，不要太小气了。否则可能会少过一些点，如 #72755。

当然，在实际操作的时候，你可以大方一点，比如直接 $100n, 10^9, \cdots$ 等等。反正这题时限非常松 (只要代码实现正常即可)，不会因为二分次数过多而 TLE。

修改 删除 ☒好评 ☒差评 [+7] ☒好评 ☒差评 [+7]

评论 ☒

暂无评论

发表评论

可以用 @mike 来提到 mike 这个用户，mike 会被高亮显示。如果你真的想打 “@” 这个字符，请用 “@@”。

想使用表情？参见表情指南。

内容

提交