《落日珊瑚》解题报告

中国人民大学附属中学　黄洛天

October 2023

1　简要题意

给一个长度为 n、包含方括号和圆括号的括号串，定义一个串合法当且仅当存在一组完美匹配，使得任意两个匹配区间要么包含要么相离。

定义一个操作叫选择一个区间 $[l,r]$，并把所有在区间里的字符从方括号变圆括号，从圆括号变方括号。

定义一个括号串的权值 $val(S)$ 为：如果这个括号串能通过操作变成合法，就是最小的操作次数；否则是 0。

给出 q 次修改查询，有以下两种可能。

1. 修改，给出一个区间 $[l,r]$ 把所有在区间里的字符从方括号变圆括号，从圆括号变方括号。
2. 查询，给出一个区间 $[l,r]$，求 $\sum_{[l',r']\in[l,r]} val(s[l',r'])$。

强制在线，输入 L,R，真实的
$l=\min((L+T\cdot lastans) \bmod n+1,(R+T\cdot lastans) \bmod n+1)$
$r=\max((L+T\cdot lastans) \bmod n+1,(R+T\cdot lastans) \bmod n+1)$
其中 lastans 是上一次询问的答案，如果没有上次询问则为 0。

2　数据范围

子任务编号	$n,q\le$	特殊性质	分值
1	100	E	5
2	2000	E	5
3	10^5	AE	5
4	2×10^5	BE	5
5	2×10^5	CDE	5
6	2×10^5	C	10
7	2×10^5	D	10
8	2×10^5	E	20
9	2×10^5	无	10
10	5×10^5	无	25

A 性质：每个位置有 $\frac{1}{4}$ 的概率为方圆左右括号。

B 性质：保证没有修改。

C 性质：保证修改为单点修改。

D 性质：保证查询区间 $[l,r]$ 满足 $S[l,r]$ 经过若干次操作可以变成合法串，且不存在另一个 $k \in [l,r)$，使得 $S[l,k]$ 可以经过若干次操作变成合法串。

E 性质；保证 $T=0$，即可以离线。

对于所有数据，满足：$1 \le n,q \le 5 \times 10^5$ 。

3 解题过程

3.1 算法 1

首先，我们考虑如何计算一个串的权值，这可以使用区间 dp 进行计算，其时间复杂度为 $O(n^4)$。

由于这部分与后续算法无关，因此不作详细阐述。在后文中，我们假设所有的时间复杂度中 n 与 q 是同阶的。

期望通过 subtask 1。

3.2 算法 2

我们可以构造一个所谓的“括号树”。具体地，将左括号视为 1、右括号视为-1，然后基于前缀和数组建立一个广义笛卡尔树。在此树中，每个节点代表一对需要匹配的括号。

我们观察到以下性质：

1. 一个括号的左或右属性是不变的。这意味着一个左括号与哪个右括号匹配是固定的。这对需要匹配的括号在括号树上对应一个节点。如果没有与其匹配的另一个括号，我们可以认为序列的开头有无限个左括号，而序列的末尾有无限个右括号。
2. 一个合法的括号串在括号树上对应某个节点的子节点区间。
3. 进行的 flip 操作是不重叠的。因为一个位置被 flip 两次等同于该位置未被 flip。因此，可以进行优化得到一个更优的解。

基于上述性质，可以直接上面树形 $dp(u,0/1,0/1)$ 表示这个节点的左侧有没有可以向左延申的操作，向右有没有可以延申的操作。直接转移并统计答案，时间复杂度可以达到 $O(n^2)$。

期望通过 subtask 1，2。

3.3 算法 2.5

3.3.1 转化

接下来我们分析一些性质，总结出一个整体框架，并由这个框架导出各个部分分以及正解的做法。

由于 dp 带取 min 的操作，对求和难以维护，因此考虑做一些转化并计算贡献。

令 $a_i \in \{0,1\}$ 表示 i 这个位置是否被 flip 过，令 $b_i = a_i \oplus a_{i+1}$，那么每个节点 u，根据左右括号的种类是否相同，对 $\bigoplus_{i=l}^{r-1} b_i$ 的值有以下两种限制：限制其为 0 或 1。我们把这个限制称为 p_u，即若限制其为 0，则记 $p_u=0$，否则 $p_u=1$。

观察到以下性质：

1. 进行一次区间 flip 操作，可以使得 b_i 数组增加至多两个 1。

2. 对于任意的 k 个 1 的 b 数组，均可以用 $\lceil \frac{k}{2} \rceil$ 次操作做出来。

因此答案可以表示为 $\lceil \frac{\sum_{i=1}^{n-1} b_i}{2} \rceil$，因此考虑最小化 $\sum b_i$。

对于节点 u，考虑他子树内 $\sum b_i$ 最小是多少。那么一定先把儿子都考虑了，如果此时 $\bigoplus_v p_v = p_u$，那么不需要做任何修改，反之需要选择任意一个位置 b_i 变成 1。这个是否需要使得答案加一，我们称为 q_i，同时 $q_u = (\bigoplus_v p_v) \oplus p_u$

因为要求很多区间的权值，除以二上取整不好处理，因此我们可以归纳的证明：

一个合法括号串，假设对应了一段儿子序列 $v_1, v_2, \cdots, v_k$，那么答案为 $\frac{\sum_{x \in \bigcup subtree(v_i)} q_x + \bigoplus q_{v_i}}{2}$。

接下来我们分前后两个部分维护答案。

接下来我们分析一下修改和查询操作的性质。

3.3.2 修改

对于一次修改操作，我们不妨把他差分成 $[1,l),[1,r]$ 两次操作，接下来讨论我们修改为 $[1,r]$ 的情况。

首先讨论 p_i 的影响。

假设 r 是左括号，那么当前节点 x 到根所有节点的左端点会被反转，这个路径左侧的所有节点左右两个括号都会反转，子树及路径右侧节点两个都不会反转。也就是说对 p_i 的影响是 x 到根的链 $\oplus 1$。

假设 r 是右括号，那么当前节点 x 的父亲到根所有节点的左端点会被反转，这个路径左侧的所有节点包括子树内的点左右两个括号都会反转，路径右侧节点两个都不会反转。也就是说对 p_i 的影响是 fa_x 到根的链 $\oplus 1$。

发现对 x 到根的链的 $p_i \oplus 1$，对 q_i 的影响是 $q_x \oplus 1$，也就是个单点修改。

经过我们的重重转化，修改从一个子树修改的样子变成了链修改，这个性质将会为后续做法节省大量代码难度。

3.3.3 查询

考虑查询本质上在查询什么，首先我们不妨设 l 是左括号，r 是右括号，l 对应的节点是 u，r 对应的节点是 v, u,v 的最近公告祖先是 w。

我们只讨论 $w \neq u, w \neq v$，其余的情况基本是这个情况的子部分。

那么需要统计的部分是 u,v 的子树，$u \sim w$ 路径上右侧的子树，$v \sim w$ 路径左侧的子树,w 的介于 u,v 所在子树之间的那些子树，也就是说我们把整个查询划分成若干个儿子区间的子树查询。

对于第一类贡献我们需要维护一下每个点贡献。

假设查询的就是一个子树。

我们令 $lcnt_u, rcnt_u$ 分别表示 u 在 fa_u 的区间里左侧和右侧分别有多少空隙。

这里的空隙是指 v_i, v_{i+1} 之间有空隙，u, v_1 之间有空隙，v_k, u 之间有空隙，换句话就是说合法串的可行左右端点。

那么一个点的贡献就是 $\sum_{v \in subtree(u)} \sum_{x \in path(u,v)-\{u\}} (lcnt_x \cdot rcnt_x) q_v$，我们把贡献差分一下。令 $val_u = \sum_{x \in path(root,u)} lcnt_x \cdot rcnt_x$，那么就是 $\sum_{v \in subtree(u)} q_v (val_v - val_u)$。

如果是儿子区间的子树。

我们需要额外减去一些贡献，具体来说计算 x 的贡献。设 u, v, x' 分别表示查询节点 u, v 以及 x 所在的子树。

那么需要减去 $lcnt_u \cdot rcnt_{x'} + lcnt_{x'} \cdot rcnt_v - lcnt_u \cdot rcnt_v$。

因此我们需要维护子树的 $\sum q_x, \sum q_x \cdot val_x, \sum q_x \cdot rcnt_{x'} \sum q_x \cdot lcnt_{x'}$。

对于第二类贡献。

我们只需要查询一个子树和，而非第一类贡献中子树和的子树和状物，而代价是链修改。

考虑靠某个节点 x 以及他的儿子序列 $y_1, y_2, \cdots y_k$，也就是我们要计算 $\sum_{l \le r} \bigoplus_{i=l}^{r} p_{y_i}$。

然后我们要维护这个东西的子树和，当 x 为 u 或 v 的祖先时，儿子序列不完整需要特殊处理。

3.4 算法 3

注意到括号树的高度很小，是 $O(\sqrt{n})$ 级别的，并且实验表明这些点的度数之和可以接受，因此考虑做一些对于每个祖先及其儿子个数的修改和查询。

我们可以暴力的维护每个节点子树内的所有信息。这样修改时第一类贡献考虑贡献的变化量维护一下即可，第二类贡献可以暴力重构一遍所有儿子。

查询时同理，可以扫出每个子树，并使用维护号的子树信息。

最坏时间复杂度 $O(n^2)$，但期望可以通过 subtask 1,2,3.

3.5 算法 4

因为没有修改，可以使用差分的方法，拆出来 u, v, w 到根的答案，这部分只需要提前 dfs 一次预处理。查询的时候额外计算一下在 w 介于 u, v 所在儿子之间的儿子子树中处的贡献，这一部分可以对儿子序列差分，拆出来贡献。

可以使用一些线性 lca 的方法，时间复杂度 (n)，期望通过 subtask 4.

3.6 算法 5

C 性质保证了是单点修改，所以 p_i, q_i 都是单点修改.

D 性质保证了查询区间一定是树上某个节点。那么就不需要讨论 u, v 和 x 的 lca，一定是 $u = v$ 代表的节点。这样就只需要做子树查询。

单点修改子树查询，于是不需要使用链剖。

具体来说需要线段树维护一下每个点的儿子序列，用来处理 p_i 修改后的二类贡献。

子树查询可以拍到 dfn 序上，然后需要一个树状数组维护单点修改区间和，查询处理一下各类贡献即可。

时间复杂度 $O(n \log n)$，期望通过 subtask 5。

3.7 算法 6

只有 C 性质，修改很简洁，但是查询还是链查询，所以仍然需要写链剖。

但对于第二类贡献来说变成了单点修改，所以不需要下放链修改的标记，会方便一些，详情见算法 8，9。

时间复杂度 $O(n \log n)$ 或 $O(n \log^2 n)$，期望通过 subtask 4，5，6。

3.8 算法 7

只有 D 性质，需要做链修改子树查询，但修改是链修改。

查询只需要做一个子树查询，不需要跳链。能节省一部分代码，详情见算法 8，9。

时间复杂度 $O(n\log n)$ 或 $O(n\log^2 n)$，期望通过 subtask 5，7。

3.9 算法 8

考虑对括号树进行轻重链剖分，对每个重链和每个点的轻儿子序列，用线段树维护信息。

我们需要查询时需要查询一个重链左侧 or 右侧所有轻子树内的贡献，这个贡献可以在修改的时候确切的知道是多少。在跳轻边的时候要查询儿子序列一个前后缀的子树信息和，而且如果重儿子在前后缀里，应该把重儿子的贡献塞进去。

查询时还需要求出 u,v 子树的信息和，这个可以由重链上查询一个后缀的到。

这个维护轻儿子的线段树不仅要支持 $\sum q_x, \sum q_x\cdot val_x, \sum q_x\cdot rcnt_{x'} \sum q_x\cdot lcnt_{x'}$，还需要支持维护当前节点的第二类贡献 $\sum_{l\le r}\bigoplus_{i=l}^{r} p_{y_i}$，以及子树的第二类贡献之和。

同理维护重链的线段树可以省去第二部分，但需要维护重链左侧或右侧的 $\sum q_x, \sum q_x val_x, \sum q_x\cdot rcnt_{x'}\cdot lcnt_{y'}, \sum q_x\cdot lcnt_{x'}\cdot rcnt_{y'}$，其中 y' 表示 x 往上跳到重链第一个点的重儿子，以及需要维护第二类贡献的 $\sum_{v\in cluster}\sum_{l\le r}\bigoplus_{i=l}^{r} p_{y_i}$，这里的线段树需要支持区间 flip 也就是说要维护重儿子的状态有没有翻转两个版本的第二类贡献。

我们需要维护求出每个重链链顶的子树信息，这意味着修改的时跳轻边的时候需要结算当前重链目前的信息，并且更新轻边的线段树。类似的，我们还需要给整个重链打重边状态的标记，并且单点修改 q_i 的贡献，并向上更新。更新顺序是重链线段树，轻儿子线段树……循环 $O(\log n)$ 轮。

时间复杂度 $O(n\log^2 n)$，期望通过 subtask 1,2,3,4,5,6,7,8,9.

3.10 算法 9

考虑建立静态 TopTree。[1]

具体来说仍然对树进行重链剖分，核心思想的带权的建立上文提到的重链线段树以及轻儿子线段树。

对于每个点的轻儿子，我们建立若干辅助点，采用 leafy 的方法来合并他们。对于重链上的点我们不采用 leaf 的方式合并，而是对于每个点的两个儿子存重链上比他深和浅的两部分的子树的根。

其中我们把重链节点带权合并之后的结构叫做 compress tree，把轻儿子带权合并之后的结果叫 rake tree。事实上这正对应了 top tree 中的 compress 和 rake 两种操作。

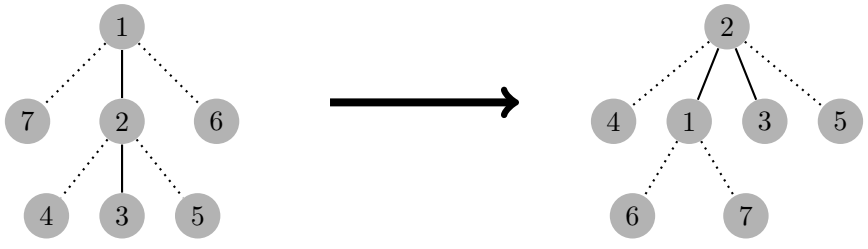

图 1: 左侧是原树，右侧是静态 TopTree

具体来说对于编号 $\leq n$ 的点，最多会有 4 个儿子。前两个儿子表示重链上的序列上的两个 compress tree 子树，后面两个儿子表示这个节点轻儿子中在重儿子前后的两个的 rake tree。

对于 rake tree 上的节点，最多会有两个儿子。这两个儿子有可能是辅助节点或者原树上的节点，分别表示靠左和靠右的轻儿子及这些轻儿子子树内部的信息。

我们需要合理的选取划分方式使得复杂度降低。具体来说，rake tree 需要按照轻儿子的子树大小每次选取带权中点；compress tree 需要按照每个点轻儿子 size 之和 + 1 作为权值，每次选取带权中点。

可以通过一些简单的分析得到，每跳常数层，size 至少乘二，因此每个点在静态 top tree 上的深度都是 $O(\log n)$ 级别。

有了上述结构并经过上文提到的若干节点在树上相对位置的讨论之后，就可以进行修改和查询了。

对于一条重链上的修改，需要做的事情是找到 u, v 两个点在 compress tree 上的 lca，并把他们中间的所有子树打上 $p \oplus 1$ 的标记。还需要更新子树内 q_x 的修改导致的 $\sum q_x, \cdots$ 等变化，并且 pushup 到根。

对于一个轻边上的修改，需要把 rake tree 上更新 $\sum q_x, \cdots$ 的信息，并且根据这个链顶的 p_i 重新计算第二类贡献，并依次 pushup 到根。

想要维护第二类贡献可以维护区间内 $\bigoplus_{i\in[l,r]} p_{v_i}$ ， $\sum_{i\in[l,r]} \bigoplus_{j\in[l,i]} p_{v_j}$ ，以及 $\sum_{i\leq j\in[l,r]} \bigoplus_{k\in[i,j]} p_{v_k}$ ，上述结构可以合并。

对于一个重链上的查询，我们仍然需要找到 compress tree 上的对应节点，并且对应侧的统计贡献。

对于一个轻边上的查询，我们有查询的是轻儿子区间其实有可能包含重儿子。令 u, v 分别为两个儿子，以如果 u, v 的 lca 为 fa_u，那么需要统计一下重儿子处的贡献，此贡献需要在 compress tree 上查询，否则只需要在 rake tree 上统计。

总时间复杂度 $O(n \log n)$，期望通过 subtask 1,2,3,4,5,6,7,8,9,10。

4 备注

本道题目在命制过程中与时庆钰同学进行了交流与合作。

楼宇辰同学、吴畅同学（集训队）和沈吉濒同学（集训队）参与了本题的验题工作。

参考文献

[1] negiizhao. Top tree 相关东西的理论、用法和实现. 2019.