

可持久化字典树



引入

可持久化 Trie 的方式和可持久化线段树的方式是相似的，即每次只修改被添加或值被修改的节点，而保留没有被改动的节点，在上一个版本的基础上连边，使最后每个版本的 Trie 树的根遍历所能分离出的 Trie 树都是完整且包含全部信息的。

大部分的可持久化 Trie 题中，Trie 都是以 **01-Trie** 的形式出现的。

例题 最大异或和

对一个长度为 n 的数组 a 维护以下操作：

1. 在数组的末尾添加一个数 x ，数组的长度 n 自增 1。
2. 给出查询区间 $[l, r]$ 和一个值 k ，求当 $l \leq p \leq r$ 时， $k \oplus \bigoplus_{i=p}^n a_i$ 的最大值。

过程

这个求的值可能有些麻烦，利用常用的处理连续异或的方法，记 $s_x = \bigoplus_{i=1}^x a_i$ ，则原式等价于 $s_{p-1} \oplus s_n \oplus k$ ，观察到 $s_n \oplus k$ 在查询的过程中是固定的，题目的查询变化为查询在区间 $[l-1, r-1]$ 中异或定值 $(s_n \oplus k)$ 的最大值。

继续按类似于可持久化线段树的思路，考虑每次的查询都查询整个区间。我们只需把这个区间建一棵 Trie 树，将这个区间中的每个数都加入这棵 Trie 中，查询的时候，尽量往与当前位不相同的地方跳。

查询区间，只需要利用前缀和和差分的思想，用两棵前缀 Trie 树（也就是按顺序添加数的两个历史版本）相减即得到该区间的 Trie 树。再利用动态开点的思想，不添加没有计算过的点，以减少空间占用。

```
1  #include <algorithm>
2  #include <cstring>
3  #include <iostream>
4  using namespace std;
5  constexpr int MAXN = 600010;
6  int n, q, a[MAXN], s[MAXN], l, r, x;
7  char op;
8
9  struct Trie {
10     int cnt, rt[MAXN], ch[MAXN * 33][2], val[MAXN * 33];
11 }
```

```

12 void insert(int o, int lst, int v) {
13     for (int i = 28; i >= 0; i--) {
14         val[o] = val[lst] + 1; // 在原版的基础上更新
15         if ((v & (1 << i)) == 0) {
16             if (!ch[o][0]) ch[o][0] = ++cnt;
17             ch[o][1] = ch[lst][1];
18             o = ch[o][0];
19             lst = ch[lst][0];
20         } else {
21             if (!ch[o][1]) ch[o][1] = ++cnt;
22             ch[o][0] = ch[lst][0];
23             o = ch[o][1];
24             lst = ch[lst][1];
25         }
26     }
27     val[o] = val[lst] + 1;
28 }
29
30 int query(int o1, int o2, int v) {
31     int ret = 0;
32     for (int i = 28; i >= 0; i--) {
33         int t = ((v & (1 << i)) ? 1 : 0);
34         if (val[ch[o1][!t]] - val[ch[o2][!t]])
35             ret += (1 << i), o1 = ch[o1][!t],
36                 o2 = ch[o2][!t]; // 尽量向不同的地方跳
37         else
38             o1 = ch[o1][t], o2 = ch[o2][t];
39     }
40     return ret;
41 }
42 } st;
43
44 int main() {
45     cin.tie(nullptr)->sync_with_stdio(false);
46     cin >> n >> q;
47     for (int i = 1; i <= n; i++) cin >> a[i], s[i] = s[i - 1] ^ a[i];
48     for (int i = 1; i <= n; i++)
49         st.rt[i] = ++st.cnt, st.insert(st.rt[i], st.rt[i - 1], s[i]);
50     while (q--) {
51         cin >> op;
52         if (op == 'A') {
53             n++;
54             cin >> a[n];
55             s[n] = s[n - 1] ^ a[n];
56             st.rt[n] = ++st.cnt;
57             st.insert(st.rt[n], st.rt[n - 1], s[n]);
58         }
59         if (op == 'Q') {
60             cin >> l >> r >> x;
61             l--;
62             r--;
63             if (l == 0)
64                 cout << max(s[n] ^ x, st.query(st.rt[r], st.rt[0], s[n] ^ x)) <<
65                 '\n';
66             else
67                 cout << st.query(st.rt[r], st.rt[l - 1], s[n] ^ x) << '\n';
68         }

```

```
69     }
70     return 0;
    }
```

🔧 本页面最近更新：2023/2/18 07:57:07，[更新历史](#)

✎ 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

👤 本页面贡献者：[Ir1d](#), [StudyingFather](#), [H-J-Granger](#), [countercurrent-time](#), [NachtgeistW](#), [CCXXxi](#), [Early0v0](#), [Enter-tainer](#), [AngelKitty](#), [cjsoft](#), [diauweb](#), [ezoixx130](#), [GekkaSaori](#), [Konano](#), [LovelyBuggies](#), [Makkiy](#), [mgt](#), [minghu6](#), [P-Y-Y](#), [PotassiumWings](#), [SamZhangQingChuan](#), [sshwy](#), [Suyun514](#), [weiyong1024](#), [Chrogeek](#), [GavinZhengOI](#), [Gesrua](#), [Henry-ZHR](#), [hsfzLZH1](#), [iamtwz](#), [kenlig](#), [ksyx](#), [kxccc](#), [lychees](#), [ouuan](#), [Peanut-Tang](#), [REYwmp](#), [shuzhouliu](#), [SukkaW](#), [代建杉](#)

© 本页面的全部内容 [在 CC BY-SA 4.0 和 SATA 协议之条款下](#) 提供，附加条款亦可能应用