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 $l \le p \le r$ 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树都加入这棵 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。

```cpp
#include <algorithm>
#include <cstring>
#include <iostream>
using namespace std;
constexpr int MAXN = 600010;
int n, q, a[MAXN], s[MAXN], l, r, x;
char op;

struct Trie {
  int cnt, rt[MAXN], ch[MAXN * 33][2], val[MAXN * 33];

```

```
  void insert(int o, int lst, int v) {
    for (int i = 28; i >= 0; i--) {
      val[o] = val[lst] + 1;  // 在原版本的基础上更新
      if ((v & (1 << i)) == 0) {
        if (!ch[o][0]) ch[o][0] = ++cnt;
        ch[o][1] = ch[lst][1];
        o = ch[o][0];
        lst = ch[lst][0];
      } else {
        if (!ch[o][1]) ch[o][1] = ++cnt;
        ch[o][0] = ch[lst][0];
        o = ch[o][1];
        lst = ch[lst][1];
      }
    }
    val[o] = val[lst] + 1;
  }

  int query(int o1, int o2, int v) {
    int ret = 0;
    for (int i = 28; i >= 0; i--) {
      int t = ((v & (1 << i)) ? 1 : 0);
      if (val[ch[o1][!t]] - val[ch[o2][!t]])
        ret += (1 << i), o1 = ch[o1][!t],
                         o2 = ch[o2][!t];  // 尽量向不同的地方跳
      else
        o1 = ch[o1][t], o2 = ch[o2][t];
    }
    return ret;
  }
} st;

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n >> q;
  for (int i = 1; i <= n; i++) cin >> a[i], s[i] = s[i - 1] ^ a[i];
  for (int i = 1; i <= n; i++)
    st.rt[i] = ++st.cnt, st.insert(st.rt[i], st.rt[i - 1], s[i]);
  while (q--) {
    cin >> op;
    if (op == 'A') {
      n++;
      cin >> a[n];
      s[n] = s[n - 1] ^ a[n];
      st.rt[n] = ++st.cnt;
      st.insert(st.rt[n], st.rt[n - 1], s[n]);
    }
    if (op == 'Q') {
      cin >> l >> r >> x;
      l--;
      r--;
      if (l == 0)
        cout << max(s[n] ^ x, st.query(st.rt[r], st.rt[0], s[n] ^ x)) <<
'\n';
      else
        cout << st.query(st.rt[r], st.rt[l - 1], s[n] ^ x) << '\n';
    }
```

```
    }
    return 0;
}
```