「SCOI2016」幸运数字 - 线性基 + 树链剖分 / 倍增

2017-02-27 | OI

A 国共有 n 座城市，这些城市由 $n-1$ 条道路相连，使得任意两座城市可以互达，且路径唯一。每座城市都有一个幸运数字，以纪念碑的形式矗立在这座城市的正中心，作为城市的象征。一些旅行者希望游览 A 国。旅行者计划乘飞机降落在 x 号城市，沿着 x 号城市到 y 号城市之间那条唯一的路径游览，最终从 y 城市起飞离开 A 国。

在经过每一座城市时，游览者就会有机会与这座城市的幸运数字拍照，从而将这份幸运保存到自己身上。然而，幸运是不能简单叠加的，这一点游览者也十分清楚。他们迷信着幸运数字是以异或的方式保留在自己身上的。例如，游览者拍了 3 张照片，幸运值分别是 5、7、11，那么最终保留在自己身上的幸运值就是 9（$5 \operatorname{xor} 7 \operatorname{xor} 11$）。

有些聪明的游览者发现，只要选择性地进行拍照，便能获得更大的幸运值。例如在上述三个幸运值中，只选择 5 和 11，可以保留的幸运值为 14。现在，一些游览者找到了聪明的你，希望你帮他们计算出在他们的行程安排中可以保留的最大幸运值是多少。

链接

BZOJ 4568

题解

树链剖分或倍增维护链上的线性基，查询时合并线性基即可。

复杂度 $O(n \log^4 n)$ 或 $O(n \log^3 n)$。

代码

```
#include <cstdio>
#include <ctime>
#include <queue>
#include <algorithm>
```

```cpp
#include <vector>
#include <stack>

const int MAXN = 20000;
const int MAXN_LOG = 14;
const int MAXL = 60;

struct Node {
    std::vector<Node *> adj;
    Node *fa, *ch, *top;
    bool vis;
    int dep, size, dfn;
    long long w;
} N[MAXN + 1];

inline void addEdge(int u, int v) {
    N[u].adj.push_back(&N[v]);
    N[v].adj.push_back(&N[u]);
}

struct LinearBasis {
    // std::vector<long long> v;
    long long a[MAXL + 1];

    LinearBasis() {
        std::fill(a, a + MAXL + 1, 0);
    }

    LinearBasis(long long *x, int n) {
        build(x, n);
    }

    void insert(long long t) {
        for (int j = MAXL; j >= 0; j--) {
            if (!t) return;
            if (!(t & (1ll << j))) continue;

            if (a[j]) t ^= a[j];
            else {
                for (int k = 0; k < j; k++) if (t & (1ll << k)) t ^= a[k];
                for (int k = j + 1; k <= MAXL; k++) if (a[k] & (1ll << j)) a[k] ^= t;
                a[j] = t;
                break;
            }
        }
    }

    void build(long long *x, int n) {
        std::fill(a, a + MAXL + 1, 0);

        for (int i = 1; i <= n; i++) {
            insert(x[i]);
```

```
            }
        }

        long long queryMax() {
            long long res = 0;
            for (int i = 0; i <= MAXL; i++) res ^= a[i];
            return res;
        }

        void mergeFrom(const LinearBasis &other) {
            for (int i = 0; i <= MAXL; i++) insert(other.a[i]);
        }

        static LinearBasis merge(const LinearBasis &a, const LinearBasis &b) {
            LinearBasis res = a;
            for (int i = 0; i <= MAXL; i++) res.insert(b.a[i]);
            return res;
        }
    };

    struct SegT {
        int l, r, mid;
        SegT *lc, *rc;
        LinearBasis lb;

        SegT(int l, int r, SegT *lc, SegT *rc) : l(l), r(r), mid(l + (r - l) / 2), lc(lc), r
        SegT(int l, int r, SegT *lc, SegT *rc, long long x) : l(l), r(r), mid(l + (r - l) /

        LinearBasis query(int l, int r) {
            if (l <= this->l && r >= this->r) return lb;
            else if (r <= mid) return lc->query(l, r);
            else if (l > mid) return rc->query(l, r);
            else return LinearBasis::merge(lc->query(l, r), rc->query(l, r));
        }

        static SegT *build(int l, int r, long long *a) {
            if (l == r) return new SegT(l, r, NULL, NULL, a[l]);
            else {
                int mid = l + (r - l) / 2;
                return new SegT(l, r, build(l, mid, a), build(mid + 1, r, a));
            }
        }
    } *seg;

    int n, root;

    inline void split() {
        std::stack<Node *> s;
        s.push(&N[root]);
        N[root].dep = 1;

        while (!s.empty()) {
```

```
            Node *v = s.top();
            if (!v->vis) {
                v->vis = true;
                for (Node **p = &v->adj.front(), *u = v->adj.front(); p <= &v->adj.back(); u
                    if (!u->dep) {
                        u->fa = v;
                        u->dep = v->dep + 1;
                        s.push(u);
                    }
                }
            } else {
                v->size = 1;
                for (Node **p = &v->adj.front(), *u = v->adj.front(); p <= &v->adj.back(); u
                    v->size += u->size;
                    if (!v->ch || v->ch->size < u->size) v->ch = u;
                }
                s.pop();
            }
        }

        for (int i = 1; i <= n; i++) N[i].vis = false;

        int ts = 0;
        static Node *seq[MAXN + 1];
        s.push(&N[root]);
        while (!s.empty()) {
            Node *v = s.top();
            if (!v->vis) {
                v->vis = true;

                v->top = (!v->fa || v != v->fa->ch) ? v : v->fa->top;
                seq[v->dfn = ++ts] = v;
                for (Node **p = &v->adj.front(), *u = v->adj.front(); p <= &v->adj.back(); u
                    if (u != v->fa && u != v->ch) {
                        s.push(u);
                    }
                }
                if (v->ch) s.push(v->ch);
            } else {
                s.pop();
            }
        }

        static long long x[MAXN + 1];
        for (int i = 1; i <= n; i++) x[i] = seq[i]->w;

        seg = SegT::build(1, n, x);
    }

    inline long long query(int u, int v) {
        Node *a = &N[u], *b = &N[v];
        LinearBasis lb;
```

```
        while (a->top != b->top) {
            if (a->top->dep < b->top->dep) std::swap(a, b);
            lb.mergeFrom(seg->query(a->top->dfn, a->dfn));
            a = a->top->fa;
        }
        if (a->dep > b->dep) std::swap(a, b);
        lb.mergeFrom(seg->query(a->dfn, b->dfn));
        return lb.queryMax();
    }

    /*
    int f[MAXN + 1][MAXN_LOG + 1], logn;
    LinearBasis g[MAXN + 1][MAXN_LOG + 1];

    inline void prepare() {
        std::queue<Node *> q;
        q.push(&N[root]);
        N[root].dep = 1;

        f[root][0] = root;
        g[root][0] = LinearBasis(&N[root].w - 1, 1);

        while (!q.empty()) {
            Node *v = q.front();
            q.pop();

            for (Node **p = &v->adj.front(), *u = v->adj.front(); p <= &v->adj.back(); u = *
                if (!u->dep) {
                    u->dep = v->dep + 1;
                    f[u - N][0] = v - N;

                    long long a[2];
                    a[0] = u->w, a[1] = v->w;
                    g[u - N][0] = LinearBasis(a - 1, 2);

                    q.push(u);
                }
            }
        }

        while ((1 << (logn + 1)) <= n) logn++;

        for (int j = 1; j <= logn; j++) {
            for (int i = 1; i <= n; i++) {
                f[i][j] = f[f[i][j - 1]][j - 1];
                g[i][j] = LinearBasis::merge(g[i][j - 1], g[f[i][j - 1]][j - 1]);
            }
        }
    }

    inline long long query(int u, int v) {
        if (N[u].dep < N[v].dep) std::swap(u, v);
```

```
    LinearBasis lb;

    if (N[u].dep > N[v].dep) {
        for (int i = logn; i >= 0; i--) {
            if (N[f[u][i]].dep >= N[v].dep) {
                lb.mergeFrom(g[u][i]);
                u = f[u][i];
            }
        }
    }

    if (u != v) {
        for (int i = logn; i >= 0; i--) {
            if (f[u][i] != f[v][i]) {
                lb.mergeFrom(g[u][i]);
                lb.mergeFrom(g[v][i]);
                u = f[u][i];
                v = f[v][i];
            }
        }

        lb.mergeFrom(g[u][0]);
        lb.mergeFrom(g[v][0]);

        u = f[u][0];
        v = f[v][0];
    }

    lb.insert(N[u].w);
    return lb.queryMax();
}
*/

int main() {
    int q;
    scanf("%d %d", &n, &q);

    // srand((n * q) ^ 20000528);
    // root = rand() % n + 1;
    root = 1;

    for (int i = 1; i <= n; i++) scanf("%lld", &N[i].w);

    for (int i = 1; i <= n - 1; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    // for (int i = 1; i <= n; i++) std::random_shuffle(N[i].adj.begin(), N[i].adj.end()
```

BZOJ # 最近公共祖先 # 线性基 # SCOI

「BZOJ 2844」albus 就是要第一个出场 - 线性基

「BZOJ 2588」Count on a tree - 主席树