数据结构

目录

Treap

```
const int maxn=100010;
template<class T>
struct Treap {
    int C, Top, bin[maxn];
    int L[maxn], R[maxn], P[maxn], pri[maxn], size[maxn];
    T key[maxn];
    int root;
    void init() {
        srand(time(0));
        root = 0;
        C = 0;
        Top = 0;
        memset(L, 0, sizeof(L));
        memset(R, 0, sizeof(R));
        memset(P, 0, sizeof(P));
    }
    void rightRotate(int x) {
        int y = L[x];
        L[x] = R[y];
        if (R[y])
            P[R[y]] = x;
        P[y] = P[x];
        if (!P[x])
            root = y;
        else if (x == L[P[x]])
            L[P[x]] = y;
        else
            R[P[x]] = y;
        R[y] = x;
        P[x] = y;
    }
    void leftRotate(int x) {
        int y = R[x];
        R[x] = L[y];
        if (L[y])
            P[L[y]] = x;
        P[y] = P[x];
        if (!P[x])
            root = y;
        else if (x == L[P[x]])
            L[P[x]] = y;
        else
            R[P[x]] = y;
        L[y] = x;
        P[x] = y;
    }
    int min(int x) {
        while (L[x])
            x = L[x];
        return x;
    }
    int max(int x) {
        while (R[x])
            x = R[x];
        return x;
    }
    int next(int x) {
        if (R[x])
            return min(R[x]);
```

```
    int y = P[x];
    while (y && x == R[y]) {
        x = y;
        y = P[y];
    }
    return y;
}
int pre(int x) {
    if (L[x])
        return max(L[x]);
    int y = P[x];
    while (y && x == L[y]) {
        x = y;
        y = P[y];
    }
    return y;
}
void makeSize(int x) {
    if (!L[x] && !R[x])
        size[x] = 1;
    else {
        if (L[x])
            makeSize(L[x]);
        if (R[x])
            makeSize(R[x]);
        size[x] = 1;
        if (L[x])
            size[x] += size[L[x]];
        if (R[x])
            size[x] += size[R[x]];
    }
}
void keepSize(int x) {
    while (x) {
        size[x] = 1;
        if (L[x])
            size[x] += size[L[x]];
        if (R[x])
            size[x] += size[R[x]];
        x = P[x];
    }
}
int insert(T k) {
    int z;
    if (Top) {
        z = bin[--Top];
    } else {
        z = ++C;
        pri[z] = rand();
    }
    L[z] = R[z] = P[z] = 0;
    key[z] = k;
    int x = root, y = 0;
    while (x) {
        y = x;
        if (k < key[x])
            x = L[x];
        else
            x = R[x];
    }
    P[z] = y;
    if (!y)
        root = z;
```

```
    else if (k < key[y])
        L[y] = z;
    else
        R[y] = z;

    while (P[z] && pri[z] < pri[P[z]]) {
        if (z == L[P[z]])
            rightRotate(P[z]);
        else
            leftRotate(P[z]);
    }
    if (!P[z])
        root = z;

    makeSize(z);
    keepSize(P[z]);
    return z;
}
void del(int z) {
    int x, y;
    if (!L[z] || !R[z])
        y = z;
    else
        y = next(z);
    if (L[y])
        x = L[y];
    else
        x = R[y];
    if (x)
        P[x] = P[y];
    if (!P[y])
        root = x;
    else if (y == L[P[y]])
        L[P[y]] = x;
    else
        R[P[y]] = x;
    if (y != z)
        key[z] = key[y];

    keepSize(P[y]);

    bin[Top++] = y;
}
int search(T& k) {
    int x = root;
    while (x && !(key[x] == k)) {
        if (k < key[x])
            x = L[x];
        else
            x = R[x];
    }
    return x;
}
int select(int x, int i) {
    if (!x)
        return 0;
    int r = size[L[x]] + 1;
    if (i == r)
        return x;
    else if (i < r)
        return select(L[x], i);
    else
        return select(R[x], i - r);
```

```
    }
    int select(int i) {return select(root, i);}
    int rank(int x) {
        int r = size[L[x]] + 1;
        int y = x;
        while (y != root) {
            if (y == R[P[y]])
                r += size[L[P[y]]] + 1;
            y = P[y];
        }
        return r;
    }
    T& operator[](int i) {return key[i];}
};
```

线段树

```
const int maxn = 100010;
template<class T>
struct SegNode {
    T key;
    int flag;
    int left, right;

    int mid() {
        return (left + right) >> 1;
    }
};
template<class T>
struct SegTree {
    SegNode<T> tree[5 * maxn];
    void init(int left, int right, int idx, T value[]) {
        tree[idx].left = left;
        tree[idx].right = right;
        tree[idx].flag = 0;
        if (left == right) {
            tree[idx].key = value[left];
            return;
        }
        int mid = tree[idx].mid();
        init(left, mid, idx << 1, value);
        init(mid + 1, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    void update(int left, int right, int idx, T value) {
        //It's a sub-interval, update it here.
        /*if (left<=tree[idx].left && right>=tree[idx].right)
         {
         tree[idx].key=value;
         return;
         }*/
        push_down(idx);
        int mid = tree[idx].mid();
        if (left <= mid)
            update(left, right, idx << 1, value);
        if (mid < right)
            update(left, right, (idx << 1) + 1, value);
        push_up(idx);
    }
    T query(int left, int right, int idx) {
        //Query result here.
        /*if (left==tree[idx].left && right==tree[idx].right)
         {
```

```
         return tree[idx].key;
         }*/
        push_down(idx);
        int mid = tree[idx].mid();
        if (right <= mid)
            return query(left, right, idx << 1);
        else if (left > mid)
            return query(left, right, (idx << 1) + 1);
        else {
            return OPE(query(left, mid, idx << 1), query(mid + 1,
right, (idx
                    << 1) + 1));
        }
    }
    void push_down(int idx) {
        if (tree[idx].flag) {
            tree[idx].flag = 0;
            //left, right, respectively.
        }
    }
    void push_up(int idx) {
    }
};
```

区间线段树

```
const int maxn = 140010;
template<class T>
struct SegNode {
    T key;
    int flag;
    int left, right;
    int flip;
    int add;
    int mid() {
        return (left + right) >> 1;
    }
    void update() {
        if (key != -1) {
            key ^= flip;
            add = key;//!
            flip = 0;
        }
    }
};
int result[140010 * 2];
template<class T>
struct SegTree {
    SegNode<T> tree[8 * maxn];
    void init(int left, int right, int idx, T value[]) {
        tree[idx].left = left;
        tree[idx].right = right;
        tree[idx].flag = 0;
        tree[idx].flip = 0;
        tree[idx].add = -1;
        tree[idx].key = 0;
        if (left == right) {
            tree[idx].key = value[left];
            return;
        }
        int mid = tree[idx].mid();
        init(left, mid, idx << 1, value);
        init(mid + 1, right, (idx << 1) + 1, value);
        push_up(idx);
    }
```

```
void update(int left, int right, int idx, int ope) {
    //It's a sub-interval, update it here.
    if (left <= tree[idx].left && right >= tree[idx].right) {
        tree[idx].flag = 1;
        if (ope == 1) {
            tree[idx].key = 1;
            tree[idx].add = 1;
            tree[idx].flip = 0;
        } else if (ope == 0) {
            tree[idx].key = 0;
            tree[idx].add = 0;
            tree[idx].flip = 0;
        } else {
            if (tree[idx].key != -1) {
                tree[idx].flip ^= 1;
                tree[idx].key ^= tree[idx].flip;
                tree[idx].flip = 0;
                tree[idx].add = tree[idx].key;
            } else {
                //tree[idx].add=-1;
                tree[idx].flip ^= 1;
            }
        }
        return;
    }
    push_down(idx);
    int mid = tree[idx].mid();
    if (left <= mid)
        update(left, right, idx << 1, ope);
    if (mid < right)
        update(left, right, (idx << 1) + 1, ope);
    push_up(idx);
}
void query(int left, int right, int idx) {
    if (left == right) {
        result[left] = tree[idx].key;
        return;
    }
    push_down(idx);
    int mid = tree[idx].mid();
    query(left, mid, idx << 1);
    query(mid + 1, right, (idx << 1) + 1);
    push_up(idx);
}
void push_down(int idx) {
    if (tree[idx].flag) {
        tree[idx].flag = 0;
        tree[idx << 1].flag = 1;
        tree[(idx << 1) + 1].flag = 1;

        if (tree[idx].add != -1) {
            int add = tree[idx].add;
            tree[idx].add = -1;
            tree[idx].flip = 0;
            int t = idx << 1;
            tree[t].key = add;
            tree[t].add = add;
            tree[t].flip = 0;
            tree[t].flag = 1;

            t++;
            tree[t].add = add;
            tree[t].key = add;
```

```
                tree[t].flip = 0;
                tree[t].flag = 1;
            }
            if (tree[idx].flip) {
                int flip = tree[idx].flip;
                tree[idx].flip = 0;
                int t = idx << 1;

                tree[t].flip ^= flip;
                tree[t].update();
                t++;
                tree[t].flip ^= flip;
                tree[t].update();
            }
        }
    }
    void push_up(int idx) {
        int p = idx << 1;
        int q = p + 1;
        if (tree[p].key != tree[q].key || tree[p].key == -1 ||
tree[q].key
                == -1) {
            tree[idx].key = -1;
        }
    }
};
//处理区间
void build(char *str, int &rx, int &ry) {
    int left, right;
    int len = strlen(str);
    if (str[0] == '(')
        left = 1;
    else
        left = 0;
    if (str[len - 1] == ')')
        right = -1;
    else
        right = 0;
    str[len - 1] = 0;
    int t, s;
    sscanf(str + 1, "%d,%d", &t, &s);
    t++;
    s++;
    rx = 2 * t + left;
    ry = 2 * s + right;
}
void print(int p, int q) {
    char left, right;
    if (p % 2)
        left = '(';
    else
        left = '[';
    if (q % 2)
        right = ')';
    else
        right = ']';
    int lv = p / 2, rv = (q + 1) / 2;
    lv--;
    rv--;
    printf("%c%d,%d%c", left, lv, rv, right);
}
```

查找满足条件最左

//查找满足条件的最左空间．加入左起最大连续，右起最大连续和该区间最大连续这几个统计域

并查集_差统计量 Ural 1701

```
const int maxn = 50010;
int p[maxn], value[maxn], result[maxn], beginValue[maxn];
int find(int x) {
    if (p[x] != x) {
        int t = p[x];
        p[x] = find(p[x]);
        value[x] += value[t];
    }
    return p[x];
}
int N, M;
void init() {
    scanf("%d%d", &N, &M);
    for (int i = 0; i <= N; i++)
        p[i] = i;
}
void buildBegin() {
    beginValue[0] = 0;
    for (int i = 1; i < N; i++) {
        if (find(i) == 0)
            continue;
        int delta = value[i] - value[find(i)];
        if (delta < 0) {
            if (beginValue[find(i)] + delta < 0)
                beginValue[find(i)] = -delta;
        }
    }
}
void work() {
    for (int i = 1; i <= M; i++) {
        int m, n, d;
        scanf("%d%d%d", &m, &n, &d);
        if (find(m) == find(n)) {
            if (value[m] - value[n] != d) {
                printf("Impossible after %d statements\n", i);
                return;
            }
        } else {
            int x = find(m);
            int y = find(n);
            if (x < y) {
                int t = m;
                m = n;
                n = t;
                d = -d;
                x = find(m);
                y = find(n);
            }
            p[x] = y;
            //value[x]+value[m]-value[n]=d
            value[x] = value[n] - value[m] + d;
        }
    }
    buildBegin();
    for (int i = 0; i < N; i++) {
```

```
        result[i] = beginValue[find(i)] + value[i];
    }
    for (int i = 0; i < N; i++) {
        if (result[i] > 1000000000 || result[i] < 0) {
            printf("Impossible after %d statements\n", M);
            return;
        }
    }
    printf("Possible\n");
    for (int i = 0; i < N; i++) {
        printf("%d\n", result[i]);
    }
}
```

移点操作并查集

```
int p[200010], value[200010];
long long sum[200010];
int trans[200010];
int num[200010];
int current;
int find(int x) {
    if (x != p[x]) {
        p[x] = find(p[x]);
    }
    return p[x];
}
void uni(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y)
        return;
    p[x] = y;
    value[y] += value[x];
    sum[y] += sum[x];
    value[x] = 0;
    sum[x] = 0;
}
void move(int X, int Y) {
    int x = trans[X];
    int y = trans[Y];
    if (find(x) == find(y))
        return;
    int t = find(x), s = find(y);
    value[t]--;
    sum[t] -= num[x];
    value[s]++;
    sum[s] += num[x];
    trans[X] = ++current;
    p[current] = s;
    num[current] = X;
}
int N, M;
void init() {
    current = N;
    for (int i = 1; i <= N; i++) {
        p[i] = i;
        value[i] = 1;
        sum[i] = i;
        trans[i] = i;
        num[i] = i;
    }
}
```

```
void work() {
    int s;
    for (int i = 1; i <= M; i++) {
        scanf("%d", &s);
        if (s == 1) {
            int t, v;
            scanf("%d%d", &t, &v);
            uni(trans[t], trans[v]);
            find(trans[t]);
            find(trans[v]);
        } else if (s == 2) {
            int t, v;
            scanf("%d%d", &t, &v);
            move(t, v);
            t = trans[t];
            v = trans[v];
            find(t);
            find(v);
        } else {
            int t;
            scanf("%d", &t);
            t = trans[t];
            cout << value[find(t)] << " " << sum[find(t)] << endl;
        }
    }
}
```

并查集_普通

```
int p[maxn], rank[maxn];
void init() {
    for (int i = 0; i < maxn; i++) {
        p[i] = i;
        rank[i] = 0;
    }
}
int find(int x) {
    if (x != p[x]) p[x] = find(p[x]);
    return p[x];
}
void uni(int x, int y) {
    if (rank[x] > rank[y])
        p[y] = x;
    else {
        p[x] = y;
        if (rank[x] == rank[y])
            rank[y]++;
    }
}
```

Giant For

```
const int maxn = 200010;
#define keyTree (ch[ ch[root][1] ][0])
struct Point {
    int x, y;
    bool operator<(Point &p) {
        if (x != p.x) return x < p.x;
        return y < p.y;
    }
    bool operator==(Point &p) {return x == p.x && y == p.y;}
};
template<class T>
```

```
struct SplayTree {
    int size[maxn], ch[maxn][2], P[maxn], stack[maxn], queue[maxn];
    int C, Top, root, *L, *R;
    T key[maxn], mx[maxn];
    int count;
    void init() {
        C = root = 0;
        count = 0;
        Top = 0;
        memset(ch, 0, sizeof(ch));
        memset(P, 0, sizeof(P));
        Point t;
        t.x = -0x7FFFFFFF;
        t.y = -0x7FFFFFFF;
        newNode(root, t);
        t.x = 0x7FFFFFFF;
        t.y = 0x7FFFFFFF;
        newNode(ch[root][1], t);
        P[C] = root;
        size[root] = 2;
        //makeTree( keyTree ,1,N,ch[root][1]);
        push_up(ch[root][1]);
        push_up(root);
        count = 2;
    }
    void rotate(int x, int f) {
        int y = P[x];
        //push_down(y);
        //push_down(x);
        ch[y][!f] = ch[x][f];
        P[ch[x][f]] = y;
        P[x] = P[y];
        if (P[x]) ch[P[y]][ch[P[y]][1] == y] = x;
        ch[x][f] = y;
        P[y] = x;//
        push_up(y);
    }
    void splay(int x, int goal) {
        if (!x) return;
        //push_down(x);
        while (P[x] != goal) {
            if (P[P[x]] == goal) rotate(x, ch[P[x]][0] == x);
            else {
                int y = P[x], z = P[y];
                int f = (ch[z][0] == y);
                if (ch[y][f] == x) rotate(x, !f), rotate(x, f);
                else rotate(y, f), rotate(x, f);
            }
        }//
        push_up(x);
        if (goal == 0) root = x;
    }
    int select(int x, int i) {
        if (!x) return 0;
        int r = size[ch[x][0]] + 1;
        if (i == r) return x;
        else if (i < r) return select(ch[x][0], i);
        else return select(ch[x][1], i - r);
    }
    void rotateTo(int k, int goal) {
        int x = select(root, k);
        if (x) splay(x, goal);
    }
```

```
    void erase(int x) {
        int head = 0, tail = 0;
        for (queue[tail++] = x; head < tail; head++) {
            stack[Top++] = queue[head];
            if (ch[queue[head]][0]) queue[tail++] = ch[queue[head]]
[0];
            if (ch[queue[head]][1]) queue[tail++] = ch[queue[head]]
[1];
        }
    }

    void newNode(int &x, Point c) {
        if (Top) x = stack[--Top];
        else x = ++C;
        ch[x][0] = ch[x][1] = P[x] = 0;
        size[x] = 1;
        key[x] = mx[x] = c;
    }

    void push_up(int x) {
        key[0].x = -1;
        mx[0].x = -1;
        size[x] = 1 + size[ch[x][0]] + size[ch[x][1]];
        mx[x] = key[x];
        if (mx[ch[x][0]].y > mx[x].y) mx[x] = mx[ch[x][0]];
        if (mx[ch[x][1]].y > mx[x].y) mx[x] = mx[ch[x][1]];
    }
    /*void push_down(int x)
     {
     if (add[x])
     {
     key[x].value-=add[x];
     add[ch[x][0]]+=add[x];
     add[ch[x][1]]+=add[x];
     add[x]=0;
     }
     }*/

    void insert(T k) {
        int z;
        newNode(z, k);
        /*if (Top) z=stack[--Top];
         else z=++C;
         ch[z][0]=ch[z][1]=P[z]=0;
         key[z]=k;
         size[z]=1;
         mx[z]=k;*/
        int x = root, y = 0;
        while (x) {
            y = x;
            if (k < key[x]) x = ch[x][0];
            else x = ch[x][1];
        }
        P[z] = y;
        if (!y) root = z;
        else if (k < key[y]) ch[y][0] = z;
        else ch[y][1] = z;
        splay(z, 0);
        count++;
    }
    void del(int z) {
        count--;
        int x, y, dest;
```

```
    if (!ch[z][0] || !ch[z][1]) { y = z;
    } else { y = next(z);
    }
    if (ch[y][0]) x = ch[y][0];
    else x = ch[y][1];
    if (x) P[x] = P[y];
    if (!P[y]) root = x;
    else if (y == ch[P[y]][0]) ch[P[y]][0] = x;
    else ch[P[y]][1] = x;
    if (y != z) key[z] = key[y];
    splay(P[y], 0);
    stack[Top++] = y;
}
int min(int x) {
    while (ch[x][0]) x = ch[x][0];
    return x;
}
int max(int x) {
    while (ch[x][1]) x = ch[x][1];
    return x;
}
int next(int x) {
    if (ch[x][1]) return min(ch[x][1]);
    int y = P[x];
    while (y && x == ch[y][1]) {
        x = y;
        y = P[y];
    }
    return y;
}
int pre(int x) {
    if (ch[x][0]) return max(ch[x][0]);
    int y = P[x];
    while (y && x == ch[y][0]) {
        x = y;
        y = P[y];
    }
    return y;
}
int search(T& k) {
    int x = root;
    while (x && !(k == key[x])) {
        if (k < key[x]) x = ch[x][0];
        else x = ch[x][1];
    }
    if (x) splay(x, 0);
    return x;
}
T& operator[](int i) {return key[i];}
void add(int x, int y) {
    Point t;
    t.x = x;
    t.y = y;
    insert(t);
}
void remove(int x, int y) {
    Point t;
    t.x = x;
    t.y = y;
    del(search(t));
}
void query(int x, int y) {
    Point t;
```

```
        t.x = x + 1;
        t.y = -0x7FFFFFFF;
        insert(t);
        rotateTo(count, root);
        int idx = keyTree;
        if (!idx) {
            printf("-1\n");
        } else {
            if (mx[idx].y <= y) {
                printf("-1\n");
            } else {
                int ans = -0x77FF;
                int current = idx;
                while (current && mx[current].y > y) {
                    if (key[current].y > y) {
                        if (ans < 0 || key[current] < key[ans]) {
                            ans = current;
                        }
                    }
                    if (mx[ch[current][0]].y > y)
                        current = ch[current][0];
                    else
                        current = ch[current][1];
                }
                printf("%d %d\n", key[ans].x, key[ans].y);
            }
        }
        t.x = x + 1;
        t.y = -0x7FFFFFFF;
        del(search(t));
    }
};
SplayTree<Point> tree;
int N;
void work() {
    tree.init();
    char temp[100];
    int p, q;
    for (int i = 1; i <= N; i++) {
        scanf("%s%d%d", temp, &p, &q);
        if (temp[0] == 'a') tree.add(p, q);
        else if (temp[0] == 'r') tree.remove(p, q);
        else tree.query(p, q);
    }
}
```

三维树状数组_三维求和

```
int c[130][130][130], N;
inline int lowbit(int t) {
    return t & (t ^ (t - 1));
}
int sum(int x, int y, int z) {
    int result = 0;
    for (int i = x; i > 0; i -= lowbit(i)) {
        for (int j = y; j > 0; j -= lowbit(j)) {
            for (int k = z; k > 0; k -= lowbit(k))
                result += c[i][j][k];
        }
    }
    return result;
}
void plus(int x, int y, int z, int value) {
```

```
    for (int i = x; i <= N; i += lowbit(i)) {
        for (int j = y; j <= N; j += lowbit(j)) {
            for (int k = z; k <= N; k += lowbit(k)) {
                c[i][j][k] += value;
            }
        }
    }
}
void work() {
    scanf("%d", &N);
    N++;
    int t;
    scanf("%d", &t);
    while (t != 3) {
        if (t == 1) {
            int x, y, z, K;
            scanf("%d%d%d%d", &x, &y, &z, &K);
            x++;y++;z++;
            plus(x, y, z, K);
        } else {
            int x1, y1, z1, x2, y2, z2;
            scanf("%d%d%d%d%d%d", &x1, &y1, &z1, &x2, &y2, &z2);
            x1++;x2++;y1++;
            y2++;z1++;z2++;
            int result = sum(x2, y2, z2) - sum(x2, y2, z1 - 1) -
sum(x2,
                    y1 - 1, z2) - sum(x1 - 1, y2, z2) + sum(x1 - 1, y1
- 1, z2)
                    + sum(x1 - 1, y2, z1 - 1) + sum(x2, y1 - 1, z1 - 1)
- sum(
                    x1 - 1, y1 - 1, z1 - 1);
            printf("%d\n", result);
        }
        scanf("%d", &t);
    }
}
```

Treap 论文版

```
const int maxn = 100010;
template<class T>
struct TreeNode
{
    T key;
    int left, right;
    int size;
    int pri;
    void init()
    {
        left = right = 0;
        size = 1;
        pri = rand();
    }
};
template<class T>
struct Treap
{
    TreeNode<T> nodes[maxn];
    int root;
    int C, Top;
```

```
int stack[maxn];
void init()
{
    C = 0;
    Top = 0;
    root = 0;
    nodes[0].pri = -0x7FFFFFFF;
}
int newNode()
{
    int ret;
    if (Top)
        ret = stack[--Top];
    else
        ret = ++C;
    nodes[ret].init();
    return ret;
}
void push_up(int idx)
{
    nodes[idx].size = nodes[nodes[idx].left].size
           + nodes[nodes[idx].right].size + 1;
}
void leftRotate(int &root)
{
    int tmp = nodes[root].right;
    nodes[root].right = nodes[nodes[root].right].left;
    nodes[tmp].left = root;
    push_up(root);
    push_up(tmp);
    root = tmp;
}
void rightRotate(int &root)
{
    int tmp = nodes[root].left;
    nodes[root].left = nodes[nodes[root].left].right;
    nodes[tmp].right = root;
    push_up(root);
    push_up(tmp);
    root = tmp;
}
void insert(const T& k, int& root)
{
    if (!root)
    {
        root = newNode();
        nodes[root].key = k;
        return;
    }
    else if (k < nodes[root].key)
    {
        insert(k, nodes[root].left);
        if (nodes[nodes[root].left].pri > nodes[root].pri)
            rightRotate(root);
    }
    else
    {
        insert(k, nodes[root].right);
        if (nodes[nodes[root].right].pri > nodes[root].pri)
            leftRotate(root);
    }
    push_up(root);
}
```

```
    void del(int &root, const T &k)
    {
        if (nodes[root].key == k)
        {
            if (!nodes[root].left && !nodes[root].right)
            {
                stack[Top++] = root;
                root = 0;
                return;
            }
            if (nodes[nodes[root].left].pri >
nodes[nodes[root].right].pri)
            {
                rightRotate(root);
                del(nodes[root].right, k);
            }
            else
            {
                leftRotate(root);
                del(nodes[root].left, k);
            }
            push_up(root);
            return;
        }
        if (k < nodes[root].key)
            del(nodes[root].left, k);
        else
            del(nodes[root].right, k);
        push_up(root);
    }
    int search(const T& k)
    {
        int rt = root;
        while (rt && !(k == nodes[rt].key))
        {
            if (k < nodes[rt].key)
                rt = nodes[rt].left;
            else
                rt = nodes[rt].right;
        }
        return rt;
    }
    int select(int root, int k)
    {
        int s = nodes[nodes[root].left].size;
        if (s >= k)
            return select(nodes[root].left, k);
        if (s + 1 == k)
            return root;
        return select(nodes[root].right, k - 1 - s);
    }
    T& operator[](int k)
    {
        return nodes[k].key;
    }
};
```

伸展树

```
const int maxn = 2000010;
```

```
#define keyTree (ch[ ch[root][1] ][0])
template<class T>
struct SplayTree
{
    int size[maxn], ch[maxn][2], P[maxn], stack[maxn], queue[maxn];
    int flag[maxn];
    int C, Top, root;
    T key[maxn];
    int count;
    void makeTree(int &x, int l, int r, int p, T value[])
    {
        if (l > r)
            return;
        int m = (l + r) >> 1;
        newNode(x, value[m]);
        makeTree(ch[x][0], l, m - 1, x, value);
        makeTree(ch[x][1], m + 1, r, x, value);
        P[x] = p;
        push_up(x);
    }
    void init(int l, int r, T value[])
    {
        C = root = 0;
        count = 0;
        Top = 0;
        newNode(root, value[0]);
        newNode(ch[root][1], value[0]);
        P[C] = root;

        makeTree(keyTree, l, r, ch[root][1], value);

        push_up(ch[root][1]);
        push_up(root);
    }
    void rotate(int x, int f)
    {
        int y = P[x];
        push_down(y);
        push_down(x);
        ch[y][!f] = ch[x][f];
        P[ch[x][f]] = y;
        P[x] = P[y];
        if (P[x])
            ch[P[y]][ch[P[y]][1] == y] = x;
        ch[x][f] = y;
        P[y] = x;
        push_up(y);
    }
    void splay(int x, int goal)
    {
        if (!x)
            return;
        push_down(x);
        while (P[x] != goal)
        {
            if (P[P[x]] == goal)
                rotate(x, ch[P[x]][0] == x);
            else
            {
                int y = P[x], z = P[y];
                int f = (ch[z][0] == y);
                if (ch[y][f] == x)
                    rotate(x, !f), rotate(x, f);
```

```
            else
                rotate(y, f), rotate(x, f);
        }
    }
    push_up(x);
    if (goal == 0)
        root = x;
}
int select(int x, int i)
{
    if (!x)
        return 0;
    push_down(x);
    int r = size[ch[x][0]] + 1;
    if (i == r)
        return x;
    else if (i < r)
        return select(ch[x][0], i);
    else
        return select(ch[x][1], i - r);
}
void rotateTo(int k, int goal)
{
    int x = select(root, k);
    if (x)
        splay(x, goal);
}
void delTree(int x)
{
    int head = 0, tail = 0;
    for (queue[tail++] = x; head < tail; head++)
    {
        stack[Top++] = queue[head];
        count--;
        if (ch[queue[head]][0])
            queue[tail++] = ch[queue[head]][0];
        if (ch[queue[head]][1])
            queue[tail++] = ch[queue[head]][1];
    }
}
void newNode(int &x, const T& c)
{
    if (Top)
        x = stack[--Top];
    else
        x = ++C;
    ch[x][0] = ch[x][1] = P[x] = 0;
    size[x] = 1;
    key[x] = c;
    count++;
}
void push_up(int x)
{
    size[x] = size[ch[x][0]] + size[ch[x][1]] + 1;
}
void push_down(int x)
{
    if (flag[x])
    {
    }
}
void insert(const T& k)
{
```

```
    int z;
    newNode(z, k);
    int x = root, y = 0;
    while (x)
    {
        y = x;
        if (k < key[x])
            x = ch[x][0];
        else
            x = ch[x][1];
    }
    P[z] = y;
    if (!y)
        root = z;
    else if (k < key[y])
        ch[y][0] = z;
    else
        ch[y][1] = z;
    splay(z, 0);
}
void del(int z)
{
    int x, y, dest;
    if (!ch[z][0] || !ch[z][1])
    {
        y = z;
    }
    else
    {
        y = next(z);
    }
    if (ch[y][0])
        x = ch[y][0];
    else
        x = ch[y][1];
    if (x)
        P[x] = P[y];
    if (!P[y])
        root = x;
    else if (y == ch[P[y]][0])
        ch[P[y]][0] = x;
    else
        ch[P[y]][1] = x;
    if (y != z)
        key[z] = key[y];
    splay(P[y], 0);

    stack[Top++] = y;
    count--;
}
int min(int x)
{
    while (ch[x][0])
        x = ch[x][0];
    return x;
}
int max(int x)
{
    while (ch[x][1])
        x = ch[x][1];
    return x;
}
int next(int x)
```

```
    {
        if (ch[x][1])
            return min(ch[x][1]);
        int y = P[x];
        while (y && x == ch[y][1])
        {
            x = y;
            y = P[y];
        }
        return y;
    }
    int pre(int x)
    {
        if (ch[x][0])
            return max(ch[x][0]);
        int y = P[x];
        while (y && x == ch[y][0])
        {
            x = y;
            y = P[y];
        }
        return y;
    }
    int search(T& k)
    {
        int x = root;
        while (x && !(k == key[x]))
        {
            if (k < key[x])
                x = ch[x][0];
            else
                x = ch[x][1];
        }
        if (x)
            splay(x, 0);
        return x;
    }
    T& operator[](int i)
    {
        return key[i];
    }
    /*T query(int x,int y)
     {
     rotateTo(x-1,0);
     rotateTo(y+1,root);
     return key[keyTree];
     }*/
};
```

动态树

```
const int maxn = 300010;
int dparent[maxn], dcost[maxn];
int N;
template<class T>
struct SplayTree
{
    int size[maxn], ch[maxn][2], P[maxn];
    T cost[maxn], mx[maxn], add[maxn];
    int C;
    bool rev[maxn];
```

```
int count;
void init(int N)
{
    C = 0;
    for (int i = 1; i <= N; i++)
    {
        int x;
        newNode(x, 0);
    }
}
void rotate(int x, int f)
{
    int y = P[x];
    dparent[x] = dparent[y];
    //COST!
    push_down(y);
    push_down(x);
    ch[y][!f] = ch[x][f];
    P[ch[x][f]] = y;
    P[x] = P[y];
    if (P[x])
        ch[P[y]][ch[P[y]][1] == y] = x;
    ch[x][f] = y;
    P[y] = x;
    push_up(y);
}
void splay(int x, int goal)
{
    if (!x)
        return;
    push_down(x);
    while (P[x] != goal)
    {
        if (P[P[x]] == goal)
            rotate(x, ch[P[x]][0] == x);
        else
        {
            int y = P[x], z = P[y];
            int f = (ch[z][0] == y);
            if (ch[y][f] == x)
                rotate(x, !f), rotate(x, f);
            else
                rotate(y, f), rotate(x, f);
        }
    }
    push_up(x);
}
void newNode(int &x, const T& c)
{
    x = ++C;
    ch[x][0] = ch[x][1] = P[x] = 0;
    size[x] = 1;
    add[x] = cost[x] = mx[x] = rev[x] = 0;
}
void access0(int v)
{
    if (P[v])
        access0(P[v]);
    else if (dparent[v])
        access0(dparent[v]);
    push_down(v);
}
void access(int x)
```

```
{
    access0(x);
    for (int v = 0, u = x; u; u = dparent[u])
    {
        splay(u, 0);
        if (ch[u][0])
        {
            dparent[ch[u][0]] = u;
            P[ch[u][0]] = 0;
            //COST!
        }
        ch[u][0] = v;
        P[v] = u;
        push_up(v = u);
    }
    splay(x, 0);
}
void updateLCA(int x, int y, T w)
{
    access(x);
    for (int v = 0, u = y; u; u = dparent[u])
    {
        splay(u, 0);
        if (dparent[u] == 0)
        {
            cost[u] += w;
            add[ch[u][0]] += w;
            cost[ch[u][0]] += w;
            mx[ch[u][0]] += w;

            add[v] += w;
            cost[v] += w;
            mx[v] += w;
        }
        if (ch[u][0])
        {
            dparent[ch[u][0]] = u;
            P[ch[u][0]] = 0;
        }
        ch[u][0] = v;
        P[v] = u;
        push_up(v = u);
    }
}
T queryLCA(int x, int y)
{
    access(x);
    int res;
    for (int v = 0, u = y; u; u = dparent[u])
    {
        splay(u, 0);
        if (dparent[u] == 0)
        {
            res = cost[u];
            if (ch[u][0])
                res = max(res, mx[ch[u][0]]);
            if (v)
                res = max(res, mx[v]);
        }
        if (ch[u][0])
        {
            dparent[ch[u][0]] = u;
            P[ch[u][0]] = 0;
```

```
        }
        ch[u][0] = v;
        P[v] = u;
        push_up(v = u);
    }
    return res;
}
void push_up(int x)
{
    size[x] = size[ch[x][0]] + size[ch[x][1]] + 1;
    mx[x] = cost[x];
    if (ch[x][0])
        mx[x] = max(mx[x], mx[ch[x][0]]);
    if (ch[x][1])
        mx[x] = max(mx[x], mx[ch[x][1]]);
}
void push_down(int x)
{
    if (add[x])
    {
        int l = ch[x][0];
        int r = ch[x][1];

        add[l] += add[x];
        cost[l] += add[x];
        mx[l] += add[x];

        add[r] += add[x];
        cost[r] += add[x];
        mx[r] += add[x];

        add[x] = 0;
    }
    if (rev[x])
    {
        int l = ch[x][0];
        int r = ch[x][1];
        swap(ch[x][0], ch[x][1]);
        rev[l] ^= 1;
        rev[r] ^= 1;

        rev[x] = 0;
    }
}

int head(int x)
{
    push_down(x);
    while (ch[x][0])
    {
        x = ch[x][0];
        push_down(x);
    }
    return x;
}
int tail(int x)
{
    push_down(x);
    while (ch[x][1])
    {
        x = ch[x][1];
        push_down(x);
    }
```

```
    return x;
}
int after(int x)
{
    if (ch[x][1])
        return head(ch[x][1]);
    int y = P[x];
    while (y && x == ch[y][1])
    {
        x = y;
        y = P[y];
    }
    return y;
}
int before(int x)
{
    if (ch[x][0])
        return tail(ch[x][0]);
    int y = P[x];
    while (y && x == ch[y][0])
    {
        x = y;
        y = P[y];
    }
    return y;
}
void update(int root, T v)
{
    add[root] += v;
    mx[root] += v;
    cost[root] += v;
    //push_down(root);
}

void reverse(int root)
{
    rev[root] ^= 1;
    push_down(root);
}

T getCost(int v)
{
    access(v);
    return cost[v];
}
int root(int v)
{
    access(v);
    return tail(v);
}
T findMax(int v)
{
    access(v);
    T res = cost[v];
    if (ch[v][1])
        res = max(res, mx[ch[v][1]]);
    return res;
}
void addCost(int v, T val)
{
    access(v);
    update(v, val);
    push_down(v);
```

```
        if (ch[v][0])
        {
            update(ch[v][0], -val);
            push_up(v);
        }
    }
    void link(int v, int w)
    {
        access(v);
        access(w);
        dparent[v] = w;
    }
    void cut(int v)
    {
        access(v);
        int r = ch[v][1];
        P[r] = 0;
        dparent[r] = 0;
        ch[v][1] = 0;
    }
    void changeRoot(int v)
    {
        access(v);
        push_down(v);
        if (ch[v][0])
        {
            P[ch[v][0]] = 0;
            dparent[ch[v][0]] = v;
            ch[v][0] = 0;
            push_up(v);
        }
        reverse(v);
        push_down(v);
    }
};
SplayTree<int> tree;
int parent[maxn];
int C;
struct ListNode
{
    ListNode *next;
    int index;
};
ListNode *list[maxn], nodes[3 * maxn];
void caseInit()
{
    C = 0;
    memset(nodes, 0, sizeof(nodes));
    for (int i = 1; i <= N; i++)
    {
        list[i] = &nodes[C++];
    }
    tree.init(N);
}
void addEdge(int p, int q)
{
    ListNode *t = &nodes[C++];
    t->index = q;
    t->next = list[p]->next;
    list[p]->next = t;
}
int vis[maxn], queue[maxn];
void build()
```

```
{
    memset(vis, 0, sizeof(vis));
    int head = 0, tail = 0;
    vis[1] = 1;
    queue[tail++] = 1;
    dparent[1] = 0;
    while (head < tail)
    {
        int current = queue[head++];
        for (ListNode *ite = list[current]->next; ite; ite = ite->next)
        {
            if (!vis[ite->index])
            {
                vis[ite->index] = 1;
                queue[tail++] = ite->index;
                dparent[ite->index] = current;
            }
        }
    }
}
void input()
{
    for (int i = 1; i <= N - 1; i++)
    {
        int p, q;
        scanf("%d%d", &p, &q);
        addEdge(p, q);
        addEdge(q, p);
    }
    for (int i = 1; i <= N; i++)
    {
        int p;
        scanf("%d", &p);
        tree.cost[i] = p;
        tree.mx[i] = p;
    }
    build();
}
```

ST算法求RMQ

```
template<class T>
struct RMQ
{
    int N;
    T val[maxn * 2];
    int rmq[20][maxn * 2];
    int qry[maxn * 2];

    void init(T arr[], int n)
    {
        N = n;
        for (int i = 1; i <= N; i++)
            val[i] = arr[i];
        for (int i = 1, cnt = 0; i <= N; i <<= 1, cnt++)
        {
            for (int j = i; j < (i << 1) && j <= N; j++)
            {
                qry[j] = cnt;
```

```
            }
        }
        val[0] = 0x7FFFFFFF;
        build();
    }
    void build()
    {
        for (int i = 1; i <= N; i++)
            rmq[0][i] = i;
        for (int j = 1; (1 << j) <= N; j++)
        {
            for (int i = 1; i <= N; i++)
            {
                int p = rmq[j - 1][i];
                int q = 0;
                if (i + (1 << (j - 1)) <= N)
                    q = rmq[j - 1][i + (1 << (j - 1))];
                if (val[p] < val[q])
                {
                    rmq[j][i] = p;
                }
                else
                    rmq[j][i] = q;
            }
        }
    }
    int query(int p, int q)
    {
        if (p > q)
            return query(q, p);
        int k = qry[q - p + 1];
        int i = p;
        int j = q - (1 << k) + 1;
        if (val[rmq[k][i]] < val[rmq[k][j]])
            return rmq[k][i];
        else
            return rmq[k][j];
    }
};
```

LCA_RMQ版本

```
struct LCA
{
    ListNode *list[maxn];

    int N;

    int seq[2 * maxn];
    int dseq[2 * maxn];
    int first[maxn];
    int time;
    int vis[maxn];

    int depth[maxn];

    RMQ<int> rmq;

    void init(int n, ListNode *lst[])
    {
        N = n;
```

```
        for (int i = 1; i <= N; i++)
            list[i] = lst[i];
        memset(vis, 0, sizeof(vis));
        time = 0;
        depth[1] = 0;
        vis[1] = 1;
        DFS(1);
        rmq.init(dseq, time);
    }
    void DFS(int current)
    {
        seq[++time] = current;
        dseq[time] = depth[current];
        first[current] = time;
        for (ListNode *ite = list[current]->next; ite; ite = ite-
>next)
        {
            if (!vis[ite->index])
            {
                vis[ite->index] = 1;
                depth[ite->index] = depth[current] + 1;
                DFS(ite->index);
                seq[++time] = current;
                dseq[time] = depth[current];
            }
        }
    }
    int query(int p, int q)
    {
        return seq[rmq.query(first[p], first[q])];
    }
};
```

树链剖分_点权 Ural 1553

```
#pragma comment(linker, "/STACK:50331648")
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 100010;

//------------------------------此处线段树------------------------------

//------------------------------此处RMQ------------------------------

struct ListNode
{
    ListNode *next;
    int index;
};

ListNode *list[maxn], nodes[maxn * 5];
int C;

int N, Q;

void initList()
{
    for (int i = 1; i <= N; i++)
```

```
    {
        list[i] = &nodes[C++];
    }
}

void addEdge(int p, int q)
{
    ListNode *t = &nodes[C++];
    t->index = q;
    t->next = list[p]->next;
    list[p]->next = t;
}

//--------------------------------此处LCA-----------------------------

LCA lca;

struct TreeNode
{
    int parent;
    int type;
    int value;
    int heavy;
};

int size[maxn];
int vis[maxn];
int leaf[maxn];
int queue[maxn];

TreeNode tree[maxn];

void stage1(int current)
{
    size[current] = 1;
    for (ListNode *ite = list[current]->next; ite; ite = ite->next)
    {
        if (!vis[ite->index])
        {
            leaf[current] = 0;
            vis[ite->index] = 1;
            stage1(ite->index);
            size[current] += size[ite->index];
            tree[ite->index].parent = current;
        }
    }
}

void stage1()
{
    memset(vis, 0, sizeof(vis));
    vis[1] = 1;
    for (int i = 1; i <= N; i++)
        leaf[i] = 1;
    stage1(1);
}

void stage2()
{
    memset(vis, 0, sizeof(vis));
    vis[1] = 1;
    int head = 0, tail = 0;
    queue[tail++] = 1;
```

```
    tree[1].type = 1;
    while (head != tail)
    {
        int current = queue[head++];
        int mx = -1, rx = 0;
        for (ListNode *ite = list[current]->next; ite; ite = ite-
>next)
        {
            if (!vis[ite->index])
            {
                vis[ite->index] = 1;
                if (mx < size[ite->index])
                {
                    mx = size[ite->index];
                    rx = ite->index;
                }
                queue[tail++] = ite->index;
            }
        }
        tree[current].heavy = rx;
        tree[rx].type = 1;
    }
}

int type[maxn];
int previous[maxn];
int treeIndex[maxn];
int top[maxn];
int treeId;
int typeId;

SegTree<int> segTree;

int isHead[maxn];

void stage3()
{
    memset(vis, 0, sizeof(vis));
    int head = 0, tail = 0;
    queue[tail++] = 1;
    vis[1] = 1;
    for (int i = 1; i <= N; i++)
        isHead[i] = 1;
    while (head != tail)
    {
        int current = queue[head++];
        if (tree[current].type == 1 && isHead[current])
        {
            typeId++;
            previous[current] = tree[current].parent;
            top[current] = current;
            type[current] = typeId;
            treeIndex[current] = ++treeId;

            for (int i = tree[current].heavy; i; i = tree[i].heavy)
            {
                treeIndex[i] = ++treeId;
                previous[i] = previous[current];
                type[i] = typeId;
                top[i] = current;
                isHead[i] = 0;
            }
        }
```

```
        if (tree[current].type == 0)
        {
            typeId++;
            type[current] = typeId;
            previous[current] = tree[current].parent;
        }
        for (ListNode *ite = list[current]->next; ite; ite = ite-
>next)
        {
            if (!vis[ite->index])
            {
                vis[ite->index] = 1;
                queue[tail++] = ite->index;
            }
        }
    }
    segTree.init(1, treeId, 1);
}
void input()
{
    scanf("%d", &N);
    initList();
    for (int i = 1; i <= N - 1; i++)
    {
        int p, q;
        scanf("%d%d", &p, &q);
        addEdge(p, q);
        addEdge(q, p);
    }
}
void init()
{
    lca.init(N, list);
    stage1();
    stage2();
    stage3();
}
void update(int idx, int value)
{
    tree[idx].value += value;
    if (tree[idx].type)
    {
        segTree.update(treeIndex[idx], treeIndex[idx], 1, value);
    }
}
int querySingle(int from, int to)
{
    int result = tree[to].value;
    for (int i = from; i != to; i = previous[i])
    {
        if (tree[i].type)
        {
            if (type[i] == type[to])
            {
                if (treeIndex[i] > treeIndex[to])
                    result = max(result, segTree.query(treeIndex[to],
                            treeIndex[i], 1));
                else
                    result = max(result, segTree.query(treeIndex[i],
                            treeIndex[to], 1));
                return result;
            }
            else
```

```
            {
                if (treeIndex[i] > treeIndex[top[i]])
                    result = max(result,
segTree.query(treeIndex[top[i]],
                            treeIndex[i], 1));
                else
                    result = max(result, segTree.query(treeIndex[i],
                            treeIndex[top[i]], 1));
            }
        }
        else
            result = max(result, tree[i].value);
    }
    return result;
}
int query(int p, int q)
{
    int anc = lca.query(p, q);
    return max(querySingle(p, anc), querySingle(q, anc));
}
void work()
{
    scanf("%d", &Q);
    char ope[10];
    int p, q;
    for (int i = 1; i <= Q; i++)
    {
        scanf("%s%d%d", ope, &p, &q);
        if (ope[0] == 'I')
        {
            update(p, q);
        }
        else
        {
            printf("%d\n", query(p, q));
        }
    }
}
int main()
{
    input();
    init();
    work();
    return 0;
}
```

树链剖分_边权 QTREE

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 10010;

//------------------------------此处线段树----------------------------

//------------------------------此处RMQ----------------------------

struct ListNode
{
```

```
    ListNode *next;
    int index;
    int number;
    int value;
};

ListNode *list[maxn], nodes[maxn * 5];
int C, edgeNumber[maxn];

int N;

void initList()
{
    memset(nodes, 0, sizeof(nodes));
    C = 0;
    for (int i = 1; i <= N; i++)
    {
        list[i] = &nodes[C++];
    }
}

void addEdge(int p, int q, int num, int val)
{
    ListNode *t = &nodes[C++];
    t->index = q;
    t->next = list[p]->next;
    t->number = num;
    t->value = val;
    list[p]->next = t;
}

//------------------------------此处LCA-----------------------------

LCA lca;

struct TreeNode
{
    int parent;
    int type;
    int value;
    int heavy;
};

int size[maxn];
int vis[maxn];
int leaf[maxn];
int queue[maxn];

TreeNode tree[maxn];

void stage1(int current)
{
    size[current] = 1;
    for (ListNode *ite = list[current]->next; ite; ite = ite->next)
    {
        if (!vis[ite->index])
        {
            leaf[current] = 0;
            vis[ite->index] = 1;
            stage1(ite->index);
            size[current] += size[ite->index];
            tree[ite->index].parent = current;
            edgeNumber[ite->number] = ite->index;
```

```
            tree[ite->index].value = ite->value;
        }
    }
}

void stage1()
{
    memset(vis, 0, sizeof(vis));
    vis[1] = 1;
    for (int i = 1; i <= N; i++)
        leaf[i] = 1;
    stage1(1);
}

void stage2()
{
    memset(vis, 0, sizeof(vis));
    vis[1] = 1;
    int head = 0, tail = 0;
    queue[tail++] = 1;
    tree[1].type = 1;
    while (head != tail)
    {
        int current = queue[head++];
        int mx = -1, rx = 0;
        for (ListNode *ite = list[current]->next; ite; ite = ite-
>next)
        {
            if (!vis[ite->index])
            {
                vis[ite->index] = 1;
                if (mx < size[ite->index])
                {
                    mx = size[ite->index];
                    rx = ite->index;
                }
                queue[tail++] = ite->index;
            }
        }
        tree[current].heavy = rx;
        tree[rx].type = 1;
    }
}

int type[maxn];
int previous[maxn];
int treeIndex[maxn];
int top[maxn];
int treeId;
int typeId;
int initValue[maxn];

SegTree<int> segTree;

int isHead[maxn];

void stage3()
{
    treeId = 0;
    typeId = 0;
    memset(vis, 0, sizeof(vis));
    int head = 0, tail = 0;
    queue[tail++] = 1;
```

```
    vis[1] = 1;
    for (int i = 1; i <= N; i++)
        isHead[i] = 1;
    while (head != tail)
    {
        int current = queue[head++];
        if (tree[current].type == 1 && isHead[current])
        {
            typeId++;
            previous[current] = tree[current].parent;
            top[current] = current;
            type[current] = typeId;
            treeIndex[current] = ++treeId;
            initValue[treeId] = tree[current].value;

            for (int i = tree[current].heavy; i; i = tree[i].heavy)
            {
                treeIndex[i] = ++treeId;
                initValue[treeId] = tree[i].value;
                previous[i] = previous[current];
                type[i] = typeId;
                top[i] = current;
                isHead[i] = 0;
            }
        }
        if (tree[current].type == 0)
        {
            typeId++;
            type[current] = typeId;
            previous[current] = tree[current].parent;
        }
        for (ListNode *ite = list[current]->next; ite; ite = ite-
>next)
        {
            if (!vis[ite->index])
            {
                vis[ite->index] = 1;
                queue[tail++] = ite->index;
            }
        }
    }
    segTree.init(1, treeId, 1, initValue);
}

void input()
{
    scanf("%d", &N);
    initList();
    for (int i = 1; i <= N - 1; i++)
    {
        int p, q, r;
        scanf("%d%d%d", &p, &q, &r);
        addEdge(p, q, i, r);
        addEdge(q, p, i, r);
    }
}

void init()
{
    memset(tree, 0, sizeof(tree));
    lca.init(N, list);
    stage1();
    stage2();
```

```
    stage3();
}

void update(int idx, int value)
{
    int index = edgeNumber[idx];
    tree[index].value = value;
    if (tree[index].type)
    {
        segTree.update(treeIndex[index], treeIndex[index], 1, value);
    }
}

int querySingle(int from, int to)
{
    int result = 0;
    for (int i = from; i != to; i = previous[i])
    {
        if (tree[i].type)
        {
            if (type[i] == type[to])
            {
                int p = i;
                int q = tree[to].heavy;
                if (treeIndex[p] > treeIndex[q])
                    result = max(result, segTree.query(treeIndex[q],
                           treeIndex[p], 1));
                else
                    result = max(result, segTree.query(treeIndex[p],
                           treeIndex[q], 1));
                return result;
            }
            else
            {
                if (treeIndex[i] > treeIndex[top[i]])
                    result = max(result,
segTree.query(treeIndex[top[i]],
                           treeIndex[i], 1));
                else
                    result = max(result, segTree.query(treeIndex[i],
                           treeIndex[top[i]], 1));
            }
        }
        else
            result = max(result, tree[i].value);
    }
    return result;
}

int query(int p, int q)
{
    int anc = lca.query(p, q);
    return max(querySingle(p, anc), querySingle(q, anc));
}

void work()
{
    char ope[10];
    int p, q;
    scanf("%s", ope);
    while (ope[0] != 'D')
    {
        scanf("%d%d", &p, &q);
```

```
        if (ope[0] == 'Q')
            printf("%d\n", query(p, q));
        else
            update(p, q);
        scanf("%s", ope);
    }
}

int main()
{
    int t;
    scanf("%d", &t);
    for (int i = 1; i <= t; i++)
    {
        input();
        init();
        work();
    }
    return 0;
}
```

树的点分治

```
import java.util.*;
class ListNode {
    ListNode next;
    int index;
    long value;
}
public class Main {
    Scanner scan = new Scanner(System.in);

    ListNode[] list = new ListNode[10010];
    long[] d = new long[10010];
    int[] size = new int[10010];
    int[] queue = new int[10010];

    void addEdge(int p, int q, long value) {
        ListNode t = new ListNode();
        t.next = list[p].next;
        t.value = value;
        t.index = q;
        list[p].next = t;
    }

    void initList() {
        for (int i = 1; i <= 10005; i++)
            list[i] = new ListNode();
    }

    void clearList() {
        for (int i = 1; i <= N; i++)
            list[i].next = null;
    }

    int[] vis = new int[10010];
    long[] depth = new long[10010];
    int N;
    long K;
    int ans, len;

    void init() {
```

```
        ans = 0;
        clearList();
        for (int i = 1; i <= N - 1; i++) {
            int p = scan.nextInt(), q = scan.nextInt();
            long l = scan.nextLong();
            addEdge(p, q, l);
            addEdge(q, p, l);
            vis[i] = 0;
        }
        vis[N] = 0;
    }

    void buildSize(int current) {
        size[current] = 1;
        for (ListNode ite = list[current].next; ite != null; ite =
ite.next) {
            if (vis[ite.index] == 0) {
                vis[ite.index] = 2;
                buildSize(ite.index);
                size[current] += size[ite.index];
            }
        }
    }

    int findRoot(int start) {
        vis[start] = 2;
        buildSize(start);
        int head = 0, tail = 0;
        queue[tail++] = start;
        vis[start] = 3;
        int result = 0x7FFFFFFF, rx = 0;
        while (head != tail) {
            int current = queue[head++];
            int t = 0;
            if (current != start) {
                t = size[start] - size[current];
            }
            for (ListNode ite = list[current].next; ite != null; ite =
ite.next) {
                if (vis[ite.index] == 2) {
                    vis[ite.index] = 3;
                    queue[tail++] = ite.index;
                    t = Math.max(t, size[ite.index]);
                }
            }
            if (result > t) {
                result = t;
                rx = current;
            }
        }
        for (int i = 0; i < tail; i++)
            vis[queue[i]] = 0;
        return rx;
    }

    int getAns(long[] a, int from, int to) {
        int res = 0;
        for (int i = from, j = to; i < j; i++) {
            while (i < j && a[i] + a[j] > K)
                j--;
            res += j - i;
        }
        return res;
```

```
    }

    void DFS(int start) {
        int root = findRoot(start);
        vis[root] = 1;
        len = 0;
        for (ListNode ite = list[root].next; ite != null; ite =
ite.next) {
            if (vis[ite.index] == 0) {
                int t = len;
                BFS(ite.index, ite.value);
                Arrays.sort(depth, t + 1, len + 1);
                ans -= getAns(depth, t + 1, len);
            }
        }
        Arrays.sort(depth, 1, len + 1);
        ans += getAns(depth, 1, len);
        for (int i = 1; i <= len; i++) {
            if (depth[i] <= K)
                ans++;
        }
        for (ListNode ite = list[root].next; ite != null; ite =
ite.next) {
            if (vis[ite.index] == 0) {
                DFS(ite.index);
            }
        }
    }

    void BFS(int start, long base) {
        int head = 0, tail = 0;
        queue[tail++] = start;
        d[start] = base;
        vis[start] = 1;
        while (head != tail) {
            int current = queue[head++];
            depth[++len] = d[current];
            for (ListNode ite = list[current].next; ite != null; ite =
ite.next) {
                if (vis[ite.index] == 0) {
                    vis[ite.index] = 1;
                    d[ite.index] = d[current] + ite.value;
                    queue[tail++] = ite.index;
                }
            }
        }
        for (int i = 0; i < tail; i++)
            vis[queue[i]] = 0;
    }
    void work() {
        DFS(1);
        System.out.println(ans);
    }
    void run() {
        initList();
        N = scan.nextInt();
        K = scan.nextLong();
        while (N != 0 || K != 0) {
            init();
            work();
            N = scan.nextInt();
            K = scan.nextLong();
        }
```

```
    }
    public static void main(String[] args) {
        new Main().run();
    }
}
```

块状链表

```
int LEN = 2050;
struct ListNode
{
    char arr[8000 + 10];
    int size;
    int rev;
    ListNode()
    {
        clear();
    }
    void clear()
    {
        size = 0;
        rev = 0;
    }
    void update()
    {
        if (rev)
        {
            reverse(arr + 1, arr + 1 + size);
            rev = 0;
        }
    }
    ListNode *pre, *next;
};
/*ListNode nodes[5000];
 ListNode *stack[5010];
 int Top,C;

 ListNode* newNode()
 {
 if (Top)
 {
 stack[Top-1]->clear();
 return stack[--Top];
 }
 nodes[C].clear();
 return &nodes[C++];
 }
 void delNode(ListNode *node)
 {
 stack[Top++]=node;
 }*/
struct BlockList
{
    ListNode *head, *tail;
    //int size;
    void init()
    {
        head = new ListNode;
        tail = new ListNode;
        //size=0;
        //head=newNode();
```

```
    //tail=newNode();
    head->next = tail;
    head->pre = tail;
    head->next->pre = head;
    head->pre->next = head;
}
void split(ListNode *node, int pivot)
{
    node->update();
    if (pivot == -1)
        pivot = LEN;
    ListNode *tmp = new ListNode;
    //ListNode *tmp=newNode();
    tmp->next = node->next;
    tmp->pre = node;
    tmp->pre->next = tmp;
    tmp->next->pre = tmp;

    for (int i = pivot + 1; i <= node->size; i++)
    {
        tmp->arr[++(tmp->size)] = node->arr[i];
    }
    node->size = pivot;
}
void merge(ListNode *node)
{
    ListNode *pre = node->pre;
    node->update();
    pre->update();
    for (int i = 1; i <= node->size; i++)
    {
        pre->arr[++(pre->size)] = node->arr[i];
    }
    node->next->pre = node->pre;
    node->pre->next = node->next;
}
void adjust()
{
    if (head->next)
    {
        for (ListNode *ite = head->next->next; ite != tail;)
        {
            if (ite->size + ite->pre->size <= LEN)
            {
                merge(ite);
                ListNode *tmp = ite;
                ite = ite->next;
                delete tmp;
                //delNode(tmp);
            }
            else
                ite = ite->next;
        }
    }
}
void adjust(ListNode *node)
{
    ListNode *pre = node->pre;
    if (pre->size + node->size <= LEN)
    {
        ListNode *tmp = node;
        merge(node);
        delete tmp;
```

```
        }
        else
        {
            balance(node);
        }
    }
    void balance(ListNode *node)
    {
        int tot = node->pre->size + node->size;
        int pivot = tot / 2;
        char tmp[10010];
        int cnt = 0;
        for (int i = 1; i <= node->pre->size; i++)
        {
            tmp[++cnt] = node->pre->arr[i];
        }
        for (int i = 1; i <= node->size; i++)
        {
            tmp[++cnt] = node->arr[i];
        }
        for (int i = 1; i <= pivot; i++)
            node->pre->arr[i] = tmp[i];
        node->pre->size = pivot;
        for (int i = pivot + 1; i <= cnt; i++)
            node->arr[i - pivot] = tmp[i];
        node->size = cnt - pivot;
    }
    BlockList() {}
    BlockList(int n, char* str)
    {
        init();
        for (int i = 0; i < n;)
        {
            ListNode *tmp = new ListNode;
            //ListNode *tmp=newNode();
            ListNode *current = tail->pre;
            tmp->size = 0;
            for (int j = 1; i < n && j <= LEN; j++, i++)
            {
                tmp->arr[++(tmp->size)] = str[i];
            }
            tmp->next = tail;
            tmp->pre = current;
            tmp->next->pre = tmp;
            tmp->pre->next = tmp;
        }
    }
};

BlockList list;
int currentIndex;
ListNode *currentNode;
int nodeIndex;

void fixCur()
{
    if (nodeIndex > currentNode->size)
    {
        nodeIndex -= currentNode->size;
        currentNode = currentNode->next;
    }
}
void insert(int n, char* str)
```

```
{
    BlockList blk(n, str);
    list.split(currentNode, nodeIndex);

    ListNode *from = blk.head->next;
    ListNode *to = blk.tail->pre;

    from->pre = currentNode;
    to->next = currentNode->next;
    from->pre->next = from;
    to->next->pre = to;

    delete blk.head;
    delete blk.tail;
    //delNode(blk.head);
    //delNode(blk.tail);

    list.adjust(to->next);
    list.adjust(currentNode->next);
    fixCur();
    //list.adjust();
}
ListNode *temp[5000];
void del(int n)
{
    list.split(currentNode, nodeIndex);
    ListNode *to = currentNode->next;

    while (n > to->size)
    {
        ListNode *tmp = to->next;
        n -= to->size;
        delete to;
        to = tmp;
    }
    list.split(to, n);

    currentNode->next = to->next;
    to->next->pre = currentNode;
    delete to;

    list.adjust(currentNode->next);
    fixCur();
    //list.adjust();
}
void move(int k)
{
    currentNode = list.head->next;
    while (k > currentNode->size)
    {
        k -= currentNode->size;
        currentNode = currentNode->next;
    }
    nodeIndex = k;
}
void get(int n)
{
    if (n <= currentNode->size - nodeIndex)
    {
        for (int i = 1; i <= n; i++)
            putchar(currentNode->arr[nodeIndex + i]);
        puts("");
        return;
```

```
    }
    for (int i = 1; i <= currentNode->size - nodeIndex; i++)
        putchar(currentNode->arr[nodeIndex + i]);
    n -= currentNode->size - nodeIndex;
    //list.split(currentNode,nodeIndex);
    ListNode *to = currentNode->next;

    while (n > to->size)
    {
        for (int j = 1; j <= to->size; j++)
            putchar(to->arr[j]);
        n -= to->size;
        to = to->next;
    }
    for (int i = 1; i <= n; i++)
        putchar(to->arr[i]);
    puts("");
}
void rotate(int n)
{
    list.split(currentNode, nodeIndex);
    ListNode *from = currentNode->next;
    ListNode *to = currentNode->next;

    for (; to != list.tail && n >= 0; to = to->next)
    {
        n -= to->size;
    }
    to = to->pre;
    n += to->size;
    list.split(to, n);

    int c = 0;
    for (ListNode *current = from; current != to->next; current =
current->next)
    {
        current->rev ^= 1;
        temp[c++] = current;
    }
    ListNode *t = currentNode;
    ListNode *s = to->next;

    t->next = to;
    s->pre = from;

    to->next = t;
    from->pre = s;

    for (int i = 0; i < c; i++)
    {
        swap(temp[i]->next, temp[i]->pre);
    }

    list.adjust();
}
void prev()
{
    if (nodeIndex == 1)
    {
        currentNode = currentNode->pre;
        nodeIndex = currentNode->size;
    }
    else
```

```
    {
        nodeIndex--;
    }
    if (currentNode == list.head)
    {
        currentNode = list.head->next;
        nodeIndex = 0;
    }
}
void next()
{
    if (nodeIndex == currentNode->size)
    {
        nodeIndex = 1;
        currentNode = currentNode->next;
    }
    else
        nodeIndex++;
}
void init()
{
    //  C=0;
    //  Top=0;
    list.init();
    ListNode *tmp = new ListNode;
    //ListNode *tmp=newNode();
    tmp->pre = list.head;
    tmp->next = list.tail;
    tmp->next->pre = tmp;
    tmp->pre->next = tmp;
    tmp->size = 0;
    currentNode = tmp;
    nodeIndex = 0;
}
char tmp[2200010];
void getString(char *str, int len)
{
    for (int i = 0; i < len;)
    {
        char c = getchar();
        if (c >= 32 && c <= 126)
            str[i++] = c;
    }
}
```