Standard Code Library

ONGLU

North Eastern University

August 2021

Contents

初始化

数据结构

轻重链剖分

```
void dfs1(int x, int pre) {
    siz[x] = 1; mson[x] = 0;
    dth[x] = dth[pre] + 1;
    fa[x] = pre;
    for(auto y : son[x]) if(y != pre) {
        dfs1(y, x);
        siz[x] += siz[y];
        if(!mson[x] || siz[y] > siz[mson[x]])
            mson[x] = y;
    }
}
void dfs2(int x, int pre, int ntp) {
    id[x] = ++idcnt;
    ltp[x] = ntp;
    if(mson[x]) dfs2(mson[x], x, ntp);
    for(auto y : son[x]) {
        if(y == mson[x] || y == pre) continue;
        dfs2(y, x, y);
    }
}
void link_modify(int x, int y, int z) {
    z %= mod;
    while(ltp[x] != ltp[y]) {
        dth[ltp[x]] < dth[ltp[y]] && (x ^= y ^= x ^= y);
        modify(1, n, id[ltp[x]], id[x], 1, z);
        x = fa[ltp[x]];

    }
    dth[x] < dth[y] && (x ^= y ^= x ^= y);
    modify(1, n, id[y], id[x], 1, z);
}
int link_query(int x, int y) {
    int ans = 0;
    while(ltp[x] != ltp[y]) {
        dth[ltp[x]] < dth[ltp[y]] && (x ^= y ^= x ^= y);
        ans = (1ll * ans + query(1, n, id[ltp[x]], id[x], 1)) % mod;
        x = fa[ltp[x]];
    }
    dth[x] < dth[y] && (x ^= y ^= x ^= y);
    ans = (1ll * ans + query(1, n, id[y], id[x], 1)) % mod;
    return ans;
}
```

二维树状数组

- 矩阵修改，矩阵查询

  查询前缀和公式：

  令 $d[i][j]$ 为差分数组，定义 $d[i][j] = a[i][j] - (a[i-1][j] - a[i][j-1] - a[i-1][j])$

  $\sum_{i=1}^{x}\sum_{j-1}^{y} a[i][j] = (x+1)*(y+1)*d[i][j] - (y+1)*i*d[i][j] + d[i][j]*i*j$

```
void modify(int x, int y, int v) {
    for(int rx = x; rx <= n; rx += rx & -rx) {
        for(int ry = y; ry <= m; ry += ry & -ry) {
            tree[rx][ry][0] += v;
            tree[rx][ry][1] += v * x;
            tree[rx][ry][2] += v * y;
            tree[rx][ry][3] += v * x * y;
        }
    }
}
```

```
void range_modify(int x, int y, int xx, int yy, int v) {
    modify(xx + 1, yy + 1, v);
    modify(x, yy + 1, -v);
    modify(xx + 1, y, -v);
    modify(x, y, v);
}
int query(int x, int y) {
    int ans = 0;
    for(int rx = x; rx; rx -= rx & -rx) {
        for(int ry = y; ry; ry -= ry & -ry) {
            ans += (x + 1) * (y + 1) * tree[rx][ry][0]
            - tree[rx][ry][1] * (y + 1) - tree[rx][ry][2] * (x + 1)
            + tree[rx][ry][3];
        }
    }
    return ans;
}
int range_query(int x, int y, int xx, int yy) {
    return query(xx, yy) + query(x - 1, y - 1)
        - query(x - 1, yy) - query(xx, y - 1);
}
```

平衡树

- luogu P3369 【模板】普通平衡树

```
#define val(x) tree[x].val
#define cnt(x) tree[x].cnt
#define siz(x) tree[x].siz
#define fa(x) tree[x].fa
#define son(x, k) tree[x].ch[k]
struct Tree {
    struct node {
        int val, cnt, siz, fa, ch[2];
    } tree[N];
    int root, tot;
    int chk(int x) {
        return son(fa(x), 1) == x;
    }
    void update(int x) {
        siz(x) = siz(son(x, 0)) + siz(son(x, 1)) + cnt(x);
    }
    void rotate(int x) {
        int y = fa(x), z = fa(y), k = chk(x), w = son(x, k ^ 1);
        son(y, k) = w; fa(w) = y;
        son(z, chk(y)) = x; fa(x) = z;
        son(x, k ^ 1) = y; fa(y) = x;
        update(y); update(x);
    }
    void splay(int x, int goal = 0) {
        while(fa(x) != goal) {
            int y = fa(x), z = fa(y);
            if(z != goal) {
                //双旋
                if(chk(y) == chk(x)) rotate(y);
                else rotate(x);
            }
            rotate(x);
        }
        if(!goal) root = x;
    }
    int New(int x, int pre) {
        tot++;
        if(pre) son(pre, x > val(pre)) = tot;
        val(tot) = x; fa(tot) = pre;
        siz(tot) = cnt(tot) = 1;
        son(tot, 0) = son(tot, 1) = 0;
        return tot;
    }
```

```
    void Insert(int x) {
        int cur = root, p = 0;
        while(cur && val(cur) != x) {
            p = cur;
            cur = son(cur, x > val(cur));
        }
        if(cur) cnt(cur)++;
        else cur = New(x, p);
        splay(cur);
    }
    void Find(int x) {
        if(!root) return ;
        int cur = root;
        while(val(cur) != x && son(cur, x > val(cur)))
            cur = son(cur, x > val(cur));
        splay(cur);
    }
    int Pre(int x) {
        Find(x);
        if(val(root) < x) return root;
        int cur = son(root, 0);
        while(son(cur, 1))
            cur = son(cur, 1);
        return cur;
    }
    int Succ(int x) {
        Find(x);
        if(val(root) > x) return root;
        int cur = son(root, 1);
        while(son(cur, 0))
            cur = son(cur, 0);
        return cur;
    }
    void Del(int x) {
        int lst = Pre(x), nxt = Succ(x);
        splay(lst); splay(nxt, lst);
        int cur = son(nxt, 0);
        if(cnt(cur) > 1) cnt(cur)--, splay(cur);
        else son(nxt, 0) = 0, splay(nxt);
    }
    int Kth(int k) {
        int cur = root;
        while(1) {
            if(son(cur, 0) && siz(son(cur, 0)) >= k) cur = son(cur, 0);
            else if(siz(son(cur, 0)) + cnt(cur) >= k) return cur;
            else k -= siz(son(cur, 0)) + cnt(cur), cur = son(cur, 1);
        }
    }
} T;
```

可持久化数据结构

可持久化 Trie

```
namespace Trie {
    struct node {
        int ch[2], ed, siz;
    } tree[N * 40];
    int tot = 0;
    int _new() {
        tot++;
        tree[tot].ch[0] = 0;
        tree[tot].ch[1] = 0;
        tree[tot].ed = tree[tot].siz = 0;
        return tot;
    }
    void init() {
        tot = 0;
        rt[0] = _new();
    }
    int Insert(int x, int t, int i = 15) {
```

```
        int u = _new(), f = (x >> i) & 1;
        tree[u] = tree[t];
        if(i == -1) {
            ed(u)++;
            siz(u)++;
            return u;
        }
        son(u, f) = Insert(x, son(t, f), i - 1);
        siz(u) = siz(son(u, 0)) + siz(son(u, 1));
        return u;
    }
    void print(int u, int now) {
        if(u == 0) return ;
        for(int i = 1; i <= ed(u); i++) printf("%d ", now);
        if(son(u, 0)) print(son(u, 0), now * 2);
        if(son(u, 1)) print(son(u, 1), now * 2 + 1);
    }
    int query(int u1, int u2, int x, int i = 15, int now = 0) {
        if(i == -1) return now;
        int f = (x >> i) & 1;
        if(siz(son(u1, f ^ 1)) - siz(son(u2, f ^ 1)) > 0)
            return query(son(u1, f ^ 1), son(u2, f ^ 1), x, i - 1, now * 2 + (f ^ 1));
        else return query(son(u1, f), son(u2, f), x, i - 1, now * 2 + (f));
    }
}
```

主席树（静态第 k 小）

建立权值树，那么 $[l,r]$ 的区间权值树就是第 r 个版本减去第 $l-1$ 个版本的树。

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <assert.h>
#define Mid ((l + r) / 2)
#define lson (rt << 1)
#define rson (rt << 1 | 1)
using namespace std;
int read() {
    char c; int num, f = 1;
    while(c = getchar(),!isdigit(c)) if(c == '-') f = -1; num = c - '0';
    while(c = getchar(), isdigit(c)) num = num * 10 + c - '0';
    return f * num;
}
const int N = 1e7 + 1009;
const int M = 2e5 + 1009;
struct node {
    int ls, rs, v;
} tree[N];
int tb;
int n, m, tot, a[M], b[M], rt[M];
int _new(int ls, int rs, int v) {
    tree[++tot].ls = ls;
    tree[tot].rs = rs;
    tree[tot].v = v;
    return tot;
}
void update(int rt) {
    tree[rt].v = tree[tree[rt].ls].v + tree[tree[rt].rs].v;
}
int build(int l, int r) {
    if(l == r) return _new(0, 0, 0);
    int x = _new(build(l, Mid), build(Mid + 1, r), 0);
    update(x);
    return x;
}
int add(int l, int r, int p, int rt, int v) {
    int x = ++tot;
    tree[x] = tree[rt];
    if(l == r) {
```

```
        tree[x].v += v;
        return x;
    }
    if(p <= Mid) tree[x].ls = add(l, Mid, p, tree[x].ls, v);
    else tree[x].rs = add(Mid + 1, r, p, tree[x].rs, v);
    update(x);
    return x;
}
int query(int l, int r, int rt1, int rt2, int k) {
    if(l == r) return l;
    if(k <= tree[tree[rt1].ls].v - tree[tree[rt2].ls].v) return query(l, Mid, tree[rt1].ls, tree[rt2].ls, k);
    else return query(Mid + 1, r, tree[rt1].rs, tree[rt2].rs, k - (tree[tree[rt1].ls].v - tree[tree[rt2].ls].v));
}
void Debug(int l, int r, int rt) {
    printf("%d %d %d\n", l, r, tree[rt].v);
    if(l == r) return ;
    Debug(l, Mid, tree[rt].ls);
    Debug(Mid + 1, r, tree[rt].rs);
}
signed main()
{
    n = read(); m = read();
    for(int i = 1; i <= n; i++) a[i] = b[i] = read();
    sort(b + 1, b + 1 + n);
    tb = unique(b + 1, b + 1 + n) - b - 1;
    rt[0] = build(1, tb);
    for(int i = 1; i <= n; i++) {
        rt[i] = add(1, tb, lower_bound(b + 1, b + 1 + tb, a[i]) - b, rt[i - 1], 1);
    }
    for(int i = 1; i <= m; i++) {
        int l, r, k;
        l = read(); r = read(); k = read();
        assert(r - l + 1 >= k);
        printf("%d\n", b[query(1, tb, rt[r], rt[l - 1], k)]);
    }
    return 0;
}
```

数学

图论

树上问题

树的直径

模板：POJ - 1985

- 两遍 DFS

```
void dfs(int x, int fa) {
    for(int i = 0; i < E[x].size(); i++) {
        int y = E[x][i].ver;
        int w = E[x][i].val;
        if(y == fa) continue;
        d[y] = d[x] + w;
        if(d[y] > d[c]) c = y;
        dfs(y, x);
    }
}
signed main()
{
    n = read();
    for(int i = 1; i < n; i++) {
        int x = read(), y = read(), w = read();
        E[x].push_back((Edge) {y, w});
        E[y].push_back((Edge) {x, w});
    }
```

```
    dfs(1, 0);
    d[c] = 0;
    dfs(c, 0);
    printf("%d\n", d[c]);
    return 0;
}
```

- 树形 DP

```
void dfs(int x, int fa) {
    d1[x] = d2[x] = 0;
    for(int i = 0; i < E[x].size(); i++) {
        int y = E[x][i].ver;
        int w = E[x][i].val;
        if(y == fa) continue;
        dfs(y, x);
        int t = d1[y] + w;
        if(t > d1[x]) {
            d2[x] = d1[x];
            d1[x] = t;
        } else if(t > d2[x]) {
            d2[x] = t;
        }
    }
    d = max(d, d1[x] + d2[x]);
}
signed main()
{
    n = read();
    for(int i = 1; i < n; i++) {
        int x = read(), y = read(), w = read();
        E[x].push_back((Edge) {y, w});
        E[y].push_back((Edge) {x, w});
    }
    dfs(1, 0);
    printf("%d\n", d);
    return 0;
}
```

求 LCA

- 树链剖分

```
namespace Tree {
    int siz[N], mson[N], ltp[N], fa[N], dth[N];
    vector<int> son[N];
    void dfs1(int x, int pre) {
        siz[x] = 1;
        mson[x] = 0;
        fa[x] = pre;
        dth[x] = dth[pre] + 1;
        for(auto y : son[x]) if(y != pre) {
            dfs1(y, x);
            if(mson[x] == 0 || siz[y] > siz[mson[x]]) mson[x] = y;
        }
    }
    void dfs2(int x, int pre, int tp) {
        ltp[x] = tp;
        if(mson[x]) dfs2(mson[x], x, tp);
        for(auto y : son[x]) if(y != pre && y != mson[x]) {
            dfs2(y, x, y);
        }
    }
    void init() {
        dfs1(1, 0);
        dfs2(1, 0, 1);
    }
    int LCA(int x, int y) {
        while(ltp[x] != ltp[y]) {
```

```
            if(dth[ltp[x]] > dth[ltp[y]]) x = fa[ltp[x]];
            else y = fa[ltp[y]];
        }
        return dth[y] > dth[x] ? x : y;
    }
}
```

- 倍增

```
namespace Tree {
    vector<int> son[N];
    int root, fa[N][31], dth[N];
    void dfs(int x, int pre) {
        fa[x][0] = pre;
        dth[x] = dth[pre] + 1;
        for(int i = 1; i <= 30; i++)
            fa[x][i] = fa[fa[x][i - 1]][i - 1];
        for(auto y : son[x]) if(y != pre)
            dfs(y, x);
    }
    void init() {
        dfs(root, 0);
    }
    int LCA(int x, int y) {
        if(dth[x] > dth[y]) swap(x, y);
        for(int i = 30; ~i; i--)
            if(dth[fa[y][i]] >= dth[x])
                y = fa[y][i];
        if(x == y) return x;
        for(int i = 30; ~i; i--)
            if(fa[y][i] != fa[x][i]) {
                x = fa[x][i];
                y = fa[y][i];
            }
        return fa[x][0];
    }
}
```

树上启发式合并

长春站的痛.jpg

- 先递归计算轻儿子的答案
- 计算重儿子的答案，并且保留重儿子的状态数组
- 把其他所有轻儿子的答案加到状态数组中，更新当前点的答案

```
void dfs1(int x, int pre) {
    siz[x] = 1;
    mson[x] = 0;
    for(auto y : son[x]) if(y != pre) {
        dfs1(y, x);
        siz[x] += siz[y];
        if(!mson[x] || siz[y] > siz[mson[x]]) mson[x] = y;
    }
}
void add(int x, int pre, int v) {
    cnt[col[x]] += v;
    if(cnt[col[x]] > Mx) Mx = cnt[col[x]], sum = col[x];
    else if(cnt[col[x]] == Mx) sum += col[x];
    for(auto y : son[x]) {
        if(y == pre || y == Son) continue;
        add(y, x, v);
    }
}
void dfs2(int x, int pre, int keep) {
    for(auto y : son[x]) {
        if(y == pre || y == mson[x]) continue;
        dfs2(y, x, 0);
    }
    if(mson[x]) dfs2(mson[x], x, 1), Son = mson[x];
    add(x, pre, 1); Son = 0;
```

```
    ans[x] = sum;
    if(!keep) add(x, pre, -1), sum = 0, Mx = 0;

}
```

图算法

第 k 短路

模板：HDU-6351

```
bool operator<(const node &a, const node &b) {
    return a.f + a.g > b.f + b.g;
}
priority_queue<node> q;
signed main()
{
    n = read(); m = read();
    for(int i = 1; i <= m; i++) {
        int x, y, w;
        x = read(); y = read(); w = read();
        E[x].push_back((Edge) {y, w});
        re[y].push_back((Edge) {x, w});
    }
    s = read(); t = read(); k = read();
    memset(dis, 0x3f, sizeof(dis)); dis[t] = 0;
    q.push((node) {t, 0, 0});
    while(q.size()) {
        int x = q.top().x, d = q.top().f;
        q.pop();
        if(dis[x] < d) continue;
        for(int i = 0; i < re[x].size(); i++) {
            int y = re[x][i].y, w = re[x][i].w;
            if(dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                q.push((node) {y, dis[y], 0});
            }
        }
    }
    for(int i = 1; i <= n; i++) cnt[i] = k;
    cnt[s]++;
    q.push((node) {s, 0, dis[s]});
    while(q.size()) {
        int x = q.top().x, f = q.top().f, g = q.top().g;
        q.pop();
        if(cnt[x] == 0) continue;
        cnt[x]--;
        if(x == t && cnt[x] == 0) {
            printf("%lld\n", f);
            return 0;
        }
        for(int i = 0; i < E[x].size(); i++) {
            int y = E[x][i].y, w = E[x][i].w;
            q.push((node) {y, f + w, dis[y]});
        }
    }
    printf("-1\n");
    return 0;
}
```

网络流

Dinic 算法

```
const int inf = 0x3f3f3f3f;
int bfs() {
    memset(d, 0, sizeof(int) * (t + 10)); d[s] = 1;
    while(q.size()) q.pop(); q.push(s);
    while(q.size()) {
        int x = q.front(); q.pop();
```

```
        for(int i = head[x]; i; i = nxt[i]) {
            if(d[ver[i]]) continue;
            if(edge[i] <= 0) continue;
            d[ver[i]] = d[x] + 1;
            q.push(ver[i]);
        }
    }
    return d[t];
}
int dinic(int x, int flow) {
    if(x == t) return flow;
    int k, res = flow;
    for(int i = head[x]; i && res; i = nxt[i]) {
        if(d[ver[i]] != d[x] + 1 || edge[i] <= 0) continue;
        k = dinic(ver[i], min(flow, edge[i]));
        if(k == 0) d[ver[i]] = 0;
        edge[i] -= k;
        edge[i ^ 1] += k;
        res -= k;
    }
    return flow - res;
}
```

EK 算法费用流

```
//反向边 cost 为负数, 容量为 0
int SPFA() {
    queue<int> q; q.push(s);
    memset(dis, 0x3f, sizeof(dis)); dis[s] = 0;
    memset(vis, 0, sizeof(vis)); vis[s] = 1;
    q.push(s); flow[s] = 0x3f3f3f3f;
    while(q.size()) {
        int x = q.front();
        vis[x] = 0; q.pop();
        for(int i = head[x]; i; i = nxt[i]) {
            if(edge[i] <= 0) continue;
            if(dis[ver[i]] > dis[x] + cost[i]) {
                dis[ver[i]] = dis[x] + cost[i];
                pre[ver[i]] = i;
                flow[ver[i]] = min(flow[x], edge[i]);
                if(!vis[ver[i]]) {
                    q.push(ver[i]);
                    vis[ver[i]] = 1;
                }
            }
        }
    }
    return dis[t] != 0x3f3f3f3f;
}
void update() {
    int x = t;
    while(x != s) {
        int i = pre[x];
        edge[i] -= flow[t];
        edge[i ^ 1] += flow[t];
        x = ver[i ^ 1];
    }
    maxflow += flow[t];
    minncost += dis[t] * flow[t];
}
```

连通性算法

Tarjan 强连通分量

```
void tarjan(int x) {
    low[x] = dfn[x] = ++dfncnt;
    s[++t] = x; vis[x] = 1;
    for(int i = head[x]; i; i = nxt[i]) {
        if(!dfn[ver[i]]) {
```

```
            tarjan(ver[i]);
            low[x] = min(low[x], low[ver[i]]);
        } else if(vis[x]) {
            low[x] = min(low[x], dfn[ver[i]]);
        }
    }
    if(dfn[x] == low[x]) {
        int pp = -1;
        ++sc;
        while(pp != x) {
            scc[s[t]] = sc;
            siz[sc]++;
            vis[s[t]] = 0;
            pp = s[t];
            t--;
        }
    }
}
//从任意点开始跑，但是注意如果图不连通，需要
for(int i = 1; i <= n; i++)
    if(!dfn[i])
        tarjan(i);
```

计算几何

字符串

字串哈希

```
namespace String {
    const int x = 135;
    const int p1 = 1e9 + 7, p2 = 1e9 + 9;
    ull xp1[N], xp2[N], xp[N];
    void init_xp() {
        xp1[0] = xp2[0] = xp[0] = 1;
        for(int i = 1; i < N; i++) {
            xp1[i] = xp1[i - 1] * x % p1;
            xp2[i] = xp2[i - 1] * x % p2;
            xp[i] = xp[i - 1] * x;
        }
    }
    struct HashString {
        char s[N];
        int length, subsize;
        bool sorted;
        ull h[N], hl[N];
        ull init(const char *t) {
            if(xp[0] != 1) init_xp();
            length = strlen(t);
            strcpy(s, t);
            ull res1 = 0, res2 = 0;
            h[length] = 0;
            for(int j = length - 1; j >= 0; j--) {
            #ifdef ENABLE_DOUBLE_HASH
                res1 = (res1 * x + s[j]) % p1;
                res2 = (res2 * x + s[j]) % p2;
                h[j] = (res1 << 32) | res2;
            #else
                res1 = res1 * x + s[j];
                h[j] = res1;
            #endif
            }
            return h[0];
        }
        //获取子串哈希，左闭右开
        ull get_substring_hash(int left, int right)  {
            int len = right - left;
        #ifdef ENABLE_DOUBLE_HASH
            unsigned int mask32 = ~(0u);
```

```
            ull left1 = h[left] >> 32, right1 = h[right] >> 32;
            ull left2 = h[left] & mask32, right2 = h[right] & mask32;
            return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
                   (((left2 - right2 * xp2[len] % p2 + p2) % p2));
        #else
            return h[left] - h[right] * xp[len];
        #endif
        }
        void get_all_subs_hash(int sublen) {
            subsize = length - sublen + 1;
            for (int i = 0; i < subsize; ++i)
                hl[i] = get_substring_hash(i, i + sublen);
            sorted = 0;
        }

        void sort_substring_hash() {
            sort(hl, hl + subsize);
            sorted = 1;
        }

        bool match(ull key) const {
            if (!sorted) assert (0);
            if (!subsize) return false;
            return binary_search(hl, hl + subsize, key);
        }
    };
}
```

Trie

```
namespace trie {
    int t[N][26], sz, ed[N];
    int _new() {
        sz++;
        memset(t[sz], 0, sizeof(t[sz]));
        return sz;
    }
    void init() {
        sz = 0;
        _new();
        memset(ed, 0, sizeof(ed));
    }
    void Insert(char *s, int n) {
        int u = 1;
        for(int i = 0; i < n; i++) {
            int c = s[i] - 'a';
            if(!t[u][c]) t[u][c] = _new();
            u = t[u][c];
        }
        ed[u]++;
    }
    int find(char *s, int n) {
        int u = 1;
        for(int i = 0; i < n; i++) {
            int c = s[i] - 'a';
            if(!t[u][c]) return -1;
            u = t[u][c];
        }
        return u;
    }
}
```

KMP 算法

```
namespace KMP {
    void get_next(char *t, int m, int *nxt) {
        int j = nxt[0] = 0;
        for(int i = 1; i < m; i++) {
            while(j && t[i] != t[j]) j = nxt[j - 1];
            nxt[i] = j += (t[i] == t[j]);
```

```
        }
    }
    vector<int> find(char *t, int m, int *nxt, char *s, int n) {
        vector<int> ans;
        int j = 0;
        for(int i = 0; i < n; i++) {
            while(j && s[i] != t[j]) j = nxt[j - 1];
            j += s[i] == t[j];
            if(j == m) {
                ans.push_back(i - m + 1);
                j = nxt[j - 1];
            }
        }
        return ans;
    }
}
```

manacher 算法

```
namespace manacher {
    char s[N];
    int p[N], len;
    void getp(string tmp) {
        len = 0;
        for(auto x : tmp) {
            s[len++] = '#';
            s[len++] = x;
        }
        s[len++] = '#';
        memset(p, 0, sizeof(int) * (len + 10));
        int c = 0, r = 0;
        for(int i = 0; i < len; i++) {
            if(i <= r) p[i] = min(p[2 * c - i], r - i);
            else p[i] = 1;
            while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
                p[i]++;
            if(i + p[i] - 1 > r) {
                r = i + p[i] - 1;
                c = i;
            }
        }
        for(int i = 0; i < len; i++) p[i]--;
    }
    void getp(char *tmp, int n) {
        len = 0;
        for(int i = 0; i < n; i++) {
            s[len++] = '#';
            s[len++] = tmp[i];
        }
        s[len++] = '#';
        memset(p, 0, sizeof(int) * (len + 10));
        int c = 0, r = 0;
        for(int i = 0; i < len; i++) {
            if(i <= r) p[i] = min(p[2 * c - i], r - i);
            else p[i] = 1;
            while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
                p[i]++;
            if(i + p[i] - 1 > r) {
                r = i + p[i] - 1;
                c = i;
            }
        }
        for(int i = 0; i < len; i++) p[i]--;
    }
    int getlen() {
        return *max_element(p, p + len);
    }
    int getlen(string s) {
        getp(s);
        return getlen();
    }
```

```
}
```

AC 自动机

```
struct ac_automaton {
    int t[N][26], danger[N], tot, fail[N];
    int dp[N][N];
    void init() {
        tot = -1;
        _new();
    }
    int _new() {
        tot++;
        memset(t[tot], 0, sizeof(t[tot]));
        danger[tot] = 0;
        fail[tot] = 0;
        return tot;
    }
    void Insert(const char *s) {
        int u = 0;
        for(int i = 0; s[i]; i++) {
            if(!t[u][mp[s[i]]]) t[u][s[i] - 'a'] = _new();
            u = t[u][mp[s[i]]];
        }
        danger[u] = 1;
    }
    void build() {
        queue<int> q;
        for(int i = 0; i < 26; i++) {
            if(t[0][i]) {
                fail[i] = 0;
                q.push(t[0][i]);
            }
        }
        while(q.size()) {
            int u = q.front(); q.pop();
            danger[u] |= danger[fail[u]];
            for(int i = 0; i < 26; i++) {
                if(t[u][i]) {
                    fail[t[u][i]] = t[fail[u]][i];
                    q.push(t[u][i]);
                } else t[u][i] = t[fail[u]][i];
            }
        }
    }
    int query(const char *s) {
        memset(dp, 0x3f, sizeof(dp));
        int n = strlen(s);
        dp[0][0] = 0;
        for(int i = 0; i < n; i++) {
            for(int j = 0; j <= tot; j++) if(!danger[j]) {
                for(int k = 0; k < 26; k++) if(!danger[t[j][k]]) {
                    dp[i + 1][t[j][k]] = min(dp[i + 1][t[j][k]], dp[i][j] + (s[i] - 'a' != k));
                }
            }
        }
        int ans = 0x3f3f3f3f;
        for(int i = 0; i <= tot; i++) if(!danger[i]) {
            ans = min(ans, dp[n][i]);
        }
        return ans == 0x3f3f3f3f ? -1 : ans;
    }
};
```

杂项

int128

```cpp
typedef __uint128_t u128;
inline u128 read() {
    static char buf[100];
    scanf("%s", buf);
    // std::cin >> buf;
    u128 res = 0;
    for(int i = 0;buf[i];++i) {
        res = res << 4 | (buf[i] <= '9' ? buf[i] - '0' : buf[i] - 'a' + 10);
    }
    return res;
}
inline void output(u128 res) {
    if(res >= 16)
        output(res / 16);
    putchar(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
    //std::cout.put(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
}
```

tips:

- 如果使用 sort 比较两个函数，不能出现 $a < b$ 和 $a > b$ 同时为真的情况，否则会运行错误。
- 多组数据清空线段树的时候，不要忘记清空全部数组（比如说 lazytag 数组）。
- 注意树的深度和节点到根的距离是两个不同的东西，深度是点数，距离是边长，如果求 LCA 时用距离算会出错。