# 模板

目录

[模板 1](#_Toc181182862)

[数据结构 4](#_Toc181182863)

[线段树 4](#_Toc181182864)

[势能线段树 6](#_Toc181182865)

[线段树合并 9](#_Toc181182866)

[线段树分裂 13](#_Toc181182867)

[扫描线 16](#_Toc181182868)

[树状数组 18](#_Toc181182869)

[并查集 20](#_Toc181182870)

[树链剖分 21](#_Toc181182871)

[普通平衡树 25](#_Toc181182872)

[文艺平衡树 31](#_Toc181182873)

[可并堆 33](#_Toc181182874)

[LCT 35](#_Toc181182875)

[主席树 39](#_Toc181182876)

[倍增RMQ 41](#_Toc181182877)

[莫队 41](#_Toc181182878)

[带修改莫队 43](#_Toc181182879)

[图论 46](#_Toc181182880)

[Tarjan缩点 46](#_Toc181182881)

[Tarjan割点 49](#_Toc181182882)

[边双连通分量 50](#_Toc181182883)

[点双连通分量 52](#_Toc181182884)

[点分治 54](#_Toc181182885)

[点分树 58](#_Toc181182886)

[2-SAT 63](#_Toc181182887)

[Dijkstra 65](#_Toc181182888)

[SPFA 67](#_Toc181182889)

[负环 68](#_Toc181182890)

[LCA 树剖 70](#_Toc181182891)

[LCA 倍增 72](#_Toc181182892)

[DSU On Tree 73](#_Toc181182893)

[树上背包 76](#_Toc181182894)

[最大流 79](#_Toc181182895)

[费用流 81](#_Toc181182896)

[数学 84](#_Toc181182897)

[三分 84](#_Toc181182898)

[欧拉筛 84](#_Toc181182899)

[欧拉函数 85](#_Toc181182900)

[Miller-Rabin素数检验和Pollard-Rho质因数分解 86](#_Toc181182901)

[快速幂 88](#_Toc181182902)

[拓展欧几里得 89](#_Toc181182903)

[逆元 89](#_Toc181182904)

[中国剩余定理 90](#_Toc181182905)

[线性基 91](#_Toc181182906)

[Simpson积分 94](#_Toc181182907)

[矩阵 94](#_Toc181182908)

[字符串 99](#_Toc181182909)

[Trie 99](#_Toc181182910)

[Manacher 100](#_Toc181182911)

[KMP 101](#_Toc181182912)

[字符串哈希 102](#_Toc181182913)

[Z函数（拓展KMP） 103](#_Toc181182914)

[后缀数组 105](#_Toc181182915)

[后缀自动机 107](#_Toc181182916)

[SAM例题 110](#_Toc181182917)

[AC自动机 119](#_Toc181182918)

[回文自动机 121](#_Toc181182919)

[其他 123](#_Toc181182920)

[多项式 123](#_Toc181182921)

[FFT 131](#_Toc181182922)

[计算几何 133](#_Toc181182923)

## 数据结构

### 线段树

#include <bits/stdc++.h>

using namespace std;

#define int long long

const int N = 1e5 + 10;

int tr[N << 2];

int lz[N << 2];

#define lson p << 1

#define rson p << 1 | 1

int a[N];

void mark(int l, int r, int v, int p)

{

    lz[p] += v;

    tr[p] += v \* (r - l + 1);

}

void pushdown(int l, int r, int p)

{

    int mid = (l + r) >> 1;

    mark(l, mid, lz[p], lson);

    mark(mid + 1, r, lz[p], rson);

    lz[p] = 0;

}

void build(int l, int r, int p)

{

    if (l == r)

    {

        tr[p] = a[l];

        return;

    }

    int mid = (l + r) >> 1;

    build(l, mid, lson);

    build(mid + 1, r, rson);

    tr[p] = tr[lson] + tr[rson];

}

int query(int l, int r, int x, int y, int p)

{

    if (x <= l && y >= r)

        return tr[p];

    int mid = (l + r) >> 1;

    int ret = 0;

    if (lz[p])

        pushdown(l, r, p);

    if (x <= mid)

        ret += query(l, mid, x, y, lson);

    if (y > mid)

        ret += query(mid + 1, r, x, y, rson);

    return ret;

}

void update(int l, int r, int x, int y, int v, int p)

{

    if (x <= l && y >= r)

    {

        mark(l, r, v, p);

        return;

    }

    if (lz[p])

        pushdown(l, r, p);

    int mid = (l + r) >> 1;

    if (x <= mid)

        update(l, mid, x, y, v, lson);

    if (y > mid)

        update(mid + 1, r, x, y, v, rson);

    tr[p] = tr[lson] + tr[rson];

}

signed main()

{

    int n, m;

    cin >> n >> m;

    for (int i = 1; i <= n; i++)

        cin >> a[i];

    build(1, n, 1);

    while (m--)

    {

        int op, x, y, k;

        cin >> op;

        switch (op)

        {

        case 1:

            cin >> x >> y >> k;

            update(1, n, x, y, k, 1);

            break;

        case 2:

            cin >> x >> y;

            printf("%lld\n", query(1, n, x, y, 1));

            break;

        }

    }

    return 0;

}

### 势能线段树

#include <algorithm>

#include <cctype>

#include <iostream>

using namespace std;

const int N = 1e6 + 6;

int t, n, m;

int a[N];

int mx[N << 2], se[N << 2], cn[N << 2], tag[N << 2];

long long sum[N << 2];

void pushup(int u)

{ // 向上更新标记

    const int ls = u << 1, rs = u << 1 | 1;

    sum[u] = sum[ls] + sum[rs];

    if (mx[ls] == mx[rs])

    {

        mx[u] = mx[rs];

        se[u] = max(se[ls], se[rs]);

        cn[u] = cn[ls] + cn[rs];

    }

    else if (mx[ls] > mx[rs])

    {

        mx[u] = mx[ls];

        se[u] = max(se[ls], mx[rs]);

        cn[u] = cn[ls];

    }

    else

    {

        mx[u] = mx[rs];

        se[u] = max(mx[ls], se[rs]);

        cn[u] = cn[rs];

    }

}

void pushtag(int u, int tg)

{ // 单纯地打标记，不暴搜

    if (mx[u] <= tg)

        return;

    sum[u] += (1ll \* tg - mx[u]) \* cn[u];

    mx[u] = tag[u] = tg;

}

void pushdown(int u)

{ // 下传标记

    if (tag[u] == -1)

        return;

    pushtag(u << 1, tag[u]), pushtag(u << 1 | 1, tag[u]);

    tag[u] = -1;

}

void build(int u = 1, int l = 1, int r = n)

{ // 建树

    tag[u] = -1;

    if (l == r)

    {

        sum[u] = mx[u] = a[l], se[u] = -1, cn[u] = 1;

        return;

    }

    int mid = (l + r) >> 1;

    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);

    pushup(u);

}

void modify\_min(int L, int R, int v, int u = 1, int l = 1, int r = n)

{

    if (mx[u] <= v)

        return;

    if (L <= l && r <= R && se[u] < v)

        return pushtag(u, v);

    int mid = (l + r) >> 1;

    pushdown(u);

    if (L <= mid)

        modify\_min(L, R, v, u << 1, l, mid);

    if (mid < R)

        modify\_min(L, R, v, u << 1 | 1, mid + 1, r);

    pushup(u);

}

int query\_max(int L, int R, int u = 1, int l = 1, int r = n)

{ // 查询最值

    if (L <= l && r <= R)

        return mx[u];

    int mid = (l + r) >> 1, r1 = -1, r2 = -1;

    pushdown(u);

    if (L <= mid)

        r1 = query\_max(L, R, u << 1, l, mid);

    if (mid < R)

        r2 = query\_max(L, R, u << 1 | 1, mid + 1, r);

    return max(r1, r2);

}

long long query\_sum(int L, int R, int u = 1, int l = 1, int r = n)

{ // 数值

    if (L <= l && r <= R)

        return sum[u];

    int mid = (l + r) >> 1;

    long long res = 0;

    pushdown(u);

    if (L <= mid)

        res += query\_sum(L, R, u << 1, l, mid);

    if (mid < R)

        res += query\_sum(L, R, u << 1 | 1, mid + 1, r);

    return res;

}

void go()

{ // 根据题意

    cin >> n >> m;

    for (int i = 1; i <= n; i++)

        cin >> a[i];

    build();

    for (int i = 1; i <= m; i++)

    {

        int op, x, y, z;

        cin >> op >> x >> y;

        if (op == 0)

            cin >> z, modify\_min(x, y, z);

        else if (op == 1)

            cout << query\_max(x, y) << '\n';

        else

            cout << query\_sum(x, y) << '\n';

    }

}

signed main()

{

    cin.tie(nullptr)->sync\_with\_stdio(false);

    cin >> t;

    while (t--)

        go();

    return 0;

}

/\*

维护一个序列 a，执行以下操作：

0 l r t [l,r]取和t取min

1 l r 输出max[l,r]

2 l r 输出sum[l,r]

\*/

### 线段树合并

#include <cstdio>

#include <algorithm>

using namespace std;

#define maxn 100001

int head[maxn], to[maxn << 1], nxt[maxn << 1], tot = 0;

int top[maxn], son[maxn], dep[maxn], sz[maxn], fa[maxn];

int root[maxn];

int n, m, maxx;

int sum[maxn \* 60], id[maxn \* 60], cnt = 0, lson[maxn \* 60], rson[maxn \* 60];

void add(int x, int y)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

}

void dfs1(int x, int f)

{

    fa[x] = f;

    sz[x] = 1;

    dep[x] = dep[f] + 1;

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (y == f)

            continue;

        dfs1(y, x);

        sz[x] += sz[y];

        if (sz[son[x]] < sz[y])

            son[x] = y;

    }

}

void dfs2(int p, int tp)

{

    top[p] = tp;

    if (son[p])

        dfs2(son[p], tp);

    for (int i = head[p]; i; i = nxt[i])

    {

        int y = to[i];

        if (y != fa[p] && y != son[p])

            dfs2(y, y);

    }

}

int lca(int x, int y)

{

    while (top[x] != top[y])

    {

        if (dep[top[x]] < dep[top[y]])

            swap(x, y);

        x = fa[top[x]];

    }

    return dep[x] < dep[y] ? x : y;

}

void pushup(int p)

{

    if (sum[lson[p]] >= sum[rson[p]])

    {

        sum[p] = sum[lson[p]];

        id[p] = id[lson[p]];

    }

    else

    {

        sum[p] = sum[rson[p]];

        id[p] = id[rson[p]];

    }

}

void update(int l, int r, int x, int v, int &p)

{

    if (!p)

        p = ++cnt;

    if (l == r)

    {

        sum[p] += v;

        id[p] = l;

        return;

    }

    int mid = (l + r) >> 1;

    if (x <= mid)

        update(l, mid, x, v, lson[p]);

    else

        update(mid + 1, r, x, v, rson[p]);

    pushup(p);

}

void merge(int &x, int y, int l, int r)

{

    if (!x || !y)

    {

        x = !x ? y : x;

        return;

    }

    if (l == r)

    {

        sum[x] += sum[y];

        return;

    }

    int mid = (l + r) >> 1;

    merge(lson[x], lson[y], l, mid);

    merge(rson[x], rson[y], mid + 1, r);

    pushup(x);

}

int ans[maxn];

void dfs3(int p)

{

    for (int i = head[p]; i; i = nxt[i])

    {

        int y = to[i];

        if (y == fa[p])

            continue;

        dfs3(y);

        merge(root[p], root[y], 1, m);

    }

    if (sum[root[p]])

        ans[p] = id[root[p]];

}

int main()

{

    scanf("%d%d", &n, &m);

    for (int i = 1; i < n; i++)

    {

        int a, b;

        scanf("%d%d", &a, &b);

        add(a, b);

        add(b, a);

    }

    dfs1(1, 0);

    dfs2(1, 1);

    int x, y, z;

    for (int i = 1; i <= m; i++)

    {

        scanf("%d%d%d", &x, &y, &z);

        maxx = max(maxx, z);

        int LCA = lca(x, y);

        update(1, maxx, z, 1, root[x]);

        update(1, maxx, z, 1, root[y]);

        update(1, maxx, z, -1, root[LCA]);

        if (fa[LCA])

            update(1, m, z, -1, root[fa[LCA]]);

    }

    dfs3(1);

    for (int i = 1; i <= n; i++)

        printf("%d\n", ans[i]);

    return 0;

}

/\*

每次给路径x到y上的村庄发z类型粮食，问最后每个村庄存放最多的救济粮

\*/

### 线段树分裂

#include <bits/stdc++.h>

#define ll long long

using namespace std;

const int MAXN = 200010;

int n, m, tot, cnt, seq = 1, op, x, y, z, bac[MAXN << 5], ch[MAXN << 5][2], rt[MAXN];

ll val[MAXN << 5];

int newnod() { return (cnt ? bac[cnt--] : ++tot); }

void del(int p)

{

    bac[++cnt] = p, ch[p][0] = ch[p][1] = val[p] = 0;

    return;

}

void modify(int &p, int l, int r, int pos, int v)

{

    if (!p)

    {

        p = newnod();

    }

    val[p] += v;

    if (l == r)

    {

        return;

    }

    int mid = (l + r) >> 1;

    if (pos <= mid)

    {

        modify(ch[p][0], l, mid, pos, v);

    }

    else

    {

        modify(ch[p][1], mid + 1, r, pos, v);

    }

    return;

}

ll query(int p, int l, int r, int xl, int xr)

{

    if (xr < l || r < xl)

    {

        return 0;

    }

    if (xl <= l && r <= xr)

    {

        return val[p];

    }

    int mid = (l + r) >> 1;

    return query(ch[p][0], l, mid, xl, xr) + query(ch[p][1], mid + 1, r, xl, xr);

}

int kth(int p, int l, int r, int k)

{

    if (l == r)

    {

        return l;

    }

    int mid = (l + r) >> 1;

    if (val[ch[p][0]] >= k)

    {

        return kth(ch[p][0], l, mid, k);

    }

    else

    {

        return kth(ch[p][1], mid + 1, r, k - val[ch[p][0]]);

    }

}

int merge(int x, int y)

{

    if (!x || !y)

    {

        return x + y;

    }

    val[x] += val[y];

    ch[x][0] = merge(ch[x][0], ch[y][0]);

    ch[x][1] = merge(ch[x][1], ch[y][1]);

    del(y);

    return x;

}

void split(int x, int &y, ll k)

{

    if (x == 0)

    {

        return;

    }

    y = newnod();

    ll v = val[ch[x][0]];

    if (k > v)

    {

        split(ch[x][1], ch[y][1], k - v);

    }

    else

    {

        swap(ch[x][1], ch[y][1]);

    }

    if (k < v)

    {

        split(ch[x][0], ch[y][0], k);

    }

    val[y] = val[x] - k;

    val[x] = k;

    return;

}

int main()

{

    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++)

    {

        scanf("%d", &x);

        modify(rt[1], 1, n, i, x);

    }

    for (int i = 1; i <= m; i++)

    {

        scanf("%d", &op);

        if (op == 0)

        {

            scanf("%d%d%d", &x, &y, &z);

            ll k1 = query(rt[x], 1, n, 1, z), k2 = query(rt[x], 1, n, y, z);

            int tmp = 0;

            split(rt[x], rt[++seq], k1 - k2);

            split(rt[seq], tmp, k2);

            rt[x] = merge(rt[x], tmp);

        }

        else if (op == 1)

        {

            scanf("%d%d", &x, &y);

            rt[x] = merge(rt[x], rt[y]);

        }

        else if (op == 2)

        {

            scanf("%d%d%d", &x, &y, &z);

            modify(rt[x], 1, n, z, y);

        }

        else if (op == 3)

        {

            scanf("%d%d%d", &x, &y, &z);

            printf("%lld\n", query(rt[x], 1, n, y, z));

        }

        else if (op == 4)

        {

            scanf("%d%d", &x, &y);

            if (val[rt[x]] < y)

            {

                printf("-1\n");

                continue;

            }

            printf("%d\n", kth(rt[x], 1, n, y));

        }

    }

    return 0;

}

/\*

0 p x y：将可重集 p 中大于等于 x 且小于等于 y 的值移动到一个新的可重集中（新可重集编号为从2 开始的正整数，是上一次产生的新可重集的编号+1）。

1 p t：将可重集 t 中的数放入可重集 p，且清空可重集 t（数据保证在此后的操作中不会出现可重集 t）。

2 p x q：在 p 这个可重集中加入 x 个数字 q。

3 p x y：查询可重集 p 中大于等于 x 且小于等于 y 的值的个数。

4 p k：查询在 p 这个可重集中第 k 小的数，不存在时输出 -1。

线段树分裂模板题，将 [x,y] 分裂出来。

将 t 树合并入 p 树：单次合并即可。

p 树中插入 x 个 q：单点修改。

查询 [x,y] 中数的个数：区间求和。

查询第 k 小。

\*/

### 扫描线

#include <bits/stdc++.h>

using namespace std;

#define int long long

typedef long long ll;

const int inf = INT\_MAX;

const ll Linf = LLONG\_MAX;

#define endl '\n'

const int N = 1e5 + 10;

int sy[N << 1];

int n;

struct node

{

    int y1, y2, x, val;

} line[N << 1];

bool cmp(node a, node b)

{

    return a.x < b.x;

}

int sum[N << 4], len[N << 4];

int ly[N << 4], ry[N << 4];

#define lson p << 1

#define rson p << 1 | 1

void build(int l, int r, int p)

{

    ly[p] = sy[l];

    ry[p] = sy[r];

    if (r - l <= 1)

        return;

    int mid = (l + r) >> 1;

    build(l, mid, lson);

    build(mid, r, rson);

}

void pushup(int p)

{

    if (sum[p])

        len[p] = ry[p] - ly[p];

    else

        len[p] = len[lson] + len[rson];

}

void update(int L, int R, int v, int p)

{

    if (L <= ly[p] && R >= ry[p])

    {

        sum[p] += v;

        pushup(p);

        return;

    }

    if (L < ry[lson])

        update(L, R, v, lson);

    if (R > ly[rson])

        update(L, R, v, rson);

    pushup(p);

}

signed main()

{

    ios::sync\_with\_stdio(0);

    cin.tie(0), cout.tie(0);

    cin >> n;

    for (int i = 1; i <= n; i++)

    {

        int a, b, c, d;

        cin >> a >> b >> c >> d;

        line[i] = node{b, d, a, 1};

        line[i + n] = node{b, d, c, -1};

        sy[i] = b;

        sy[i + n] = d;

    }

    sort(line + 1, line + 1 + 2 \* n, cmp);

    sort(sy + 1, sy + 1 + (n << 1));

    build(1, n << 1, 1);

    unsigned long long ans = 0;

    for (int i = 1; i <= 2 \* n; i++)

    {

        ans += len[1] \* (line[i].x - line[i - 1].x);

        update(line[i].y1, line[i].y2, line[i].val, 1);

    }

    cout << ans << endl;

    return 0;

}

### 树状数组

#include <bits/stdc++.h>

using namespace std;

#define lowbit(x) ((x) & (-x))

const int N = 5e5 + 10;

int tr[N];

int n, m;

void update(int x, int v)

{

    for (; x <= n; x += lowbit(x))

    {

        tr[x] += v;

    }

}

int query(int x)

{

    int ret = 0;

    for (; x; x -= lowbit(x))

    {

        ret += tr[x];

    }

    return ret;

}

int main()

{

    cin >> n >> m;

    for (int i = 1; i <= n; i++)

    {

        int x;

        cin >> x;

        update(i, x);

    }

    while (m--)

    {

        int op, x, y;

        cin >> op >> x >> y;

        if (op == 1)

        {

            update(x, y);

        }

        else

        {

            printf("%d\n", query(y) - query(x - 1));

        }

    }

    return 0;

}

### 并查集

struct DSU

{

    vector<int> f, siz;

    DSU() {}

    DSU(int n)

    {

        init(n);

    }

    void init(int n)

    {

        f.resize(n + 1);

        iotal(f.begin() + 1, f.end(), 1);

        siz.assign(n + 1, 1);

    }

    int find(int x)

    {

        while (x != f[x])

        {

            x = f[x] = f[f[x]];

        }

        return x;

    }

    bool same(int x, int y)

    {

        return find(x) == find(y);

    }

    bool merge(int x, int y)

    {

        x = find(x);

        y = find(y);

        if (x == y)

        {

            return false;

        }

        siz[x] += siz[y];

        f[y] = x;

        return true;

    }

    int size(int x)

    {

        return siz[find(x)];

    }

};

### 树链剖分

#include <bits/stdc++.h>

using namespace std;

int n, m, mod;

typedef long long ll;

const int N = 1e5 + 10;

int tr[N << 2], lz[N << 2];

#define lson p << 1

#define rson p << 1 | 1

void mark(int l, int r, int v, int p)

{

    lz[p] += v % mod;

    ll tmp = v \* (r - l + 1) % mod + tr[p] % mod;

    tmp %= mod;

    tr[p] = tmp;

}

void pushdown(int l, int r, int p)

{

    int mid = (l + r) >> 1;

    mark(l, mid, lz[p], lson);

    mark(mid + 1, r, lz[p], rson);

    lz[p] = 0;

}

int query(int l, int r, int x, int y, int p)

{

    if (x <= l && y >= r)

        return tr[p] % mod;

    int mid = (l + r) >> 1;

    if (lz[p])

        pushdown(l, r, p);

    ll ret = 0;

    if (x <= mid)

        ret += query(l, mid, x, y, lson) % mod;

    if (y > mid)

        ret += query(mid + 1, r, x, y, rson) % mod;

    ret %= mod;

    return ret;

}

void update(int l, int r, int x, int y, int v, int p)

{

    if (x <= l && y >= r)

    {

        mark(l, r, v, p);

        return;

    }

    if (lz[p])

        pushdown(l, r, p);

    int mid = (l + r) >> 1;

    if (x <= mid)

        update(l, mid, x, y, v, lson);

    if (y > mid)

        update(mid + 1, r, x, y, v, rson);

    ll tmp = tr[lson] % mod + tr[rson] % mod;

    tr[p] = tmp % mod;

}

int head[N], nxt[N << 1], to[N << 1], tot;

void add(int x, int y)

{

    nxt[++tot] = head[x];

    head[x] = tot;

    to[tot] = y;

}

int dep[N], fa[N], sz[N], son[N], idx[N], a[N], v[N], top[N], cnt;

void dfs1(int x, int f)

{

    sz[x] = 1;

    fa[x] = f;

    dep[x] = dep[f] + 1;

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (y == f)

            continue;

        dfs1(y, x);

        sz[x] += sz[y];

        if (sz[y] > sz[son[x]])

            son[x] = y;

    }

}

void dfs2(int x, int t)

{

    top[x] = t;

    idx[x] = ++cnt;

    a[idx[x]] = v[x];

    if (son[x])

        dfs2(son[x], t);

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (y == fa[x] || y == son[x])

            continue;

        dfs2(y, y);

    }

}

void add\_line(int x, int y, int z)

{

    while (top[x] != top[y])

    {

        if (dep[top[x]] < dep[top[y]])

            swap(x, y);

        update(1, n, idx[top[x]], idx[x], z, 1);

        x = fa[top[x]];

    }

    if (dep[x] < dep[y])

        swap(x, y);

    update(1, n, idx[y], idx[x], z, 1);

}

int query\_line(int x, int y)

{

    ll ret = 0;

    while (top[x] != top[y])

    {

        if (dep[top[x]] < dep[top[y]])

            swap(x, y);

        ret += query(1, n, idx[top[x]], idx[x], 1) % mod;

        x = fa[top[x]];

    }

    if (dep[x] < dep[y])

        swap(x, y);

    ret += query(1, n, idx[y], idx[x], 1) % mod;

    return ret;

}

void add\_tree(int x, int z)

{

    update(1, n, idx[x], idx[x] + sz[x] - 1, z, 1);

}

int query\_tree(int x)

{

    return query(1, n, idx[x], idx[x] + sz[x] - 1, 1) % mod;

}

void build(int l, int r, int p)

{

    if (l == r)

    {

        tr[p] = a[l];

        return;

    }

    int mid = (l + r) >> 1;

    build(l, mid, lson);

    build(mid + 1, r, rson);

    ll tmp = tr[lson] % mod + tr[rson] % mod;

    tmp %= mod;

    tr[p] = tmp;

}

int main()

{

    int rt;

    cin >> n >> m >> rt >> mod;

    for (int i = 1; i <= n; i++)

        cin >> v[i];

    for (int i = 1; i < n; i++)

    {

        int x, y;

        cin >> x >> y;

        add(x, y);

        add(y, x);

    }

    dfs1(rt, 0);

    dfs2(rt, rt);

    build(1, n, 1);

    while (m--)

    {

        int op, x, y, z;

        cin >> op;

        if (op == 1)

        {

            cin >> x >> y >> z;

            add\_line(x, y, z % mod);

        }

        else if (op == 2)

        {

            cin >> x >> y;

            printf("%d\n", query\_line(x, y) % mod);

        }

        else if (op == 3)

        {

            cin >> x >> z;

            add\_tree(x, z);

        }

        else

        {

            cin >> x;

            printf("%d\n", query\_tree(x));

        }

    }

    return 0;

}

/\*

1 x y z，表示将树从

x 到 y结点最短路径上所有节点的值都加上 z。

2 x y，表示求树从 x 到 y 结点最短路径上所有节点的值之和。

3 x z，表示将以 x 为根节点的子树内所有节点值都加上 z。

4 x 表示求以 x 为根节点的子树内所有节点值之和

\*/

### 普通平衡树

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

struct node

{

    int ch[2];

    int cnt;

    int val;

    int fa;

    int sz;

} spl[N];

int rt;

int tot = 0;

#define lson(x) spl[x].ch[0]

#define rson(x) spl[x].ch[1]

void inline update(int x)

{

    spl[x].sz = spl[lson(x)].sz + spl[rson(x)].sz + spl[x].cnt;

}

void connect(int x, int f, int s)

{

    spl[f].ch[s] = x;

    spl[x].fa = f;

}

#define ident(x, f) (x == spl[f].ch[1])

void rotate(int x)

{

    int f = spl[x].fa, ff = spl[f].fa, k = ident(x, f);

    connect(spl[x].ch[k ^ 1], f, k);

    connect(x, ff, ident(f, ff));

    connect(f, x, k ^ 1);

    update(f), update(x);

}

void splay(int x, int top)

{

    if (!top)

        rt = x;

    while (spl[x].fa != top)

    {

        int f = spl[x].fa, ff = spl[f].fa;

        if (ff != top)

            ident(x, f) ^ ident(f, ff) ? rotate(x) : rotate(f);

        rotate(x);

    }

    // 一样就不一样 不一样就一样

}

void newnode(int &now, int f, int v)

{

    now = ++tot;

    spl[now].val = v;

    spl[now].fa = f;

    spl[now].sz = spl[now].cnt = 1;

}

void delnode(int x)

{

    splay(x, 0);

    if (spl[x].cnt > 1)

        spl[x].cnt--, spl[x].sz--;

    else if (rson(x))

    {

        int p = rson(x);

        while (lson(p))

            p = lson(p);

        splay(p, x);

        connect(lson(x), p, 0);

        rt = p;

        spl[p].fa = 0;

        update(p);

    }

    else

        rt = lson(x), spl[rt].fa = 0;

}

void ins(int v, int &now = rt, int f = 0)

{

    if (!now)

        newnode(now, f, v), splay(now, 0);

    else if (v < spl[now].val)

        ins(v, lson(now), now);

    else if (v > spl[now].val)

        ins(v, rson(now), now);

    else

        spl[now].cnt++, splay(now, 0);

}

void del(int v, int now = rt)

{

    if (v == spl[now].val)

        delnode(now);

    else if (v < spl[now].val)

        del(v, lson(now));

    else

        del(v, rson(now));

}

int getrank(int v)

{

    int now = rt, rank = 1;

    while (now)

    {

        if (spl[now].val == v)

        {

            rank += spl[lson(now)].sz;

            splay(now, 0);

            break;

        }

        else if (v < spl[now].val)

            now = lson(now);

        else

        {

            rank += spl[lson(now)].sz + spl[now].cnt;

            now = rson(now);

        }

    }

    return rank;

}

int getnum(int rank)

{

    int now = rt;

    while (now)

    {

        int lsz = spl[lson(now)].sz;

        if (lsz + 1 <= rank && rank <= lsz + spl[now].cnt)

        {

            splay(now, 0);

            break;

        }

        else if (lsz >= rank)

        {

            now = lson(now);

        }

        else

        {

            rank -= lsz + spl[now].cnt;

            now = rson(now);

        }

    }

    return spl[now].val;

}

template <class T>

inline void read(T &x)

{

    int flag = 1;

    x = 0;

    char c = getchar();

    while (!isdigit(c))

    {

        if (c == '-')

            flag = -1;

        c = getchar();

    }

    while (isdigit(c))

    {

        x = (x << 3) + (x << 1) + (c ^ 48);

        c = getchar();

    }

    x \*= flag;

}

template <class T>

inline void write(T x)

{

    if (x == 0)

    {

        putchar('0');

        return;

    }

    if (x < 0)

    {

        putchar('-');

        x = -x;

    }

    char buf[20];

    int pos = 0;

    while (x)

    {

        buf[pos++] = x % 10 + '0';

        x /= 10;

    }

    while (pos > 0)

    {

        putchar(buf[--pos]);

    }

}

int main()

{

    int n;

    read(n);

    while (n--)

    {

        int op, x;

        read(op);

        read(x);

        if (op == 1)

        {

            ins(x);

        }

        else if (op == 2)

        {

            del(x);

        }

        else if (op == 3)

        {

            write(getrank(x));

            putchar('\n');

        }

        else if (op == 4)

        {

            write(getnum(x));

            putchar('\n');

        }

        else if (op == 5)

        {

            write(getnum(getrank(x) - 1));

            putchar('\n');

        }

        else

        {

            write(getnum(getrank(x + 1)));

            putchar('\n');

        }

    }

    return 0;

}

/\*

1 插入一个数 x。

2 删除一个数 x（若有多个相同的数，应只删除一个）。

定义排名为比当前数小的数的个数 +1。

3 查询 x 的排名。

4 查询数据结构中排名为 x 的数。

5 求 x 的前驱（前驱定义为小于 x，且最大的数）。

6 求 x 的后继（后继定义为大于 x，且最小的数）。

\*/

### 文艺平衡树

#include <bits/stdc++.h>

using namespace std;

const int inf = 0x3f3f3f3f;

const int maxn = 1e5 + 10;

int original[maxn];

int ch[maxn][2], fa[maxn], val[maxn], tag[maxn], sz[maxn];

int root;

#define connect(x, f, s) ch[fa[x] = f][s] = x

#define ident(x, f) (ch[f][1] == x)

#define lson(x) ch[x][0]

#define rson(x) ch[x][1]

#define reverse(x) swap(lson(x), rson(x)), tag[x] ^= 1

#define update(x) sz[x] = sz[lson(x)] + sz[rson(x)] + 1

void pushdown(int x)

{

    if (tag[x])

    {

        if (lson(x))

            reverse(lson(x));

        if (rson(x))

            reverse(rson(x));

    }

    tag[x] = 0;

}

void pushall(int x)

{

    if (x != root)

        pushall(fa[x]);

    pushdown(x);

}

void rotate(int x)

{

    int f = fa[x], ff = fa[f], k = ident(x, f);

    connect(ch[x][k ^ 1], f, k);

    connect(x, ff, ident(f, ff));

    connect(f, x, k ^ 1);

    update(f), update(x);

}

void splaying(int x, int top)

{

    pushall(x);

    while (fa[x] != top)

    {

        int f = fa[x], ff = fa[f];

        if (ff != top)

            ident(f, ff) ^ ident(x, f) ? rotate(x) : rotate(f);

        rotate(x);

    }

    if (top == 0)

        root = x;

}

void output(int x)

{

    pushdown(x);

    if (lson(x))

        output(lson(x));

    if (val[x] != inf && val[x] != -inf)

        printf("%d ", val[x]);

    if (rson(x))

        output(rson(x));

}

int find(int k)

{

    int now = root;

    while (1)

    {

        pushdown(now);

        if (k == (sz[lson(now)] + 1))

            break;

        if (k <= sz[lson(now)])

            now = lson(now);

        else if (k > sz[lson(now)])

        {

            k -= (sz[lson(now)] + 1);

            now = rson(now);

        }

    }

    return now;

}

void mark(int l, int r)

{

    l = l - 1, r = r + 1;

    l = find(l), r = find(r);

    splaying(l, 0);

    splaying(r, l);

    int pos = ch[root][1];

    pos = ch[pos][0];

    reverse(pos);

}

int cnt = 0;

int build(int l, int r, int f)

{

    if (l > r)

        return 0;

    int mid = (l + r) >> 1;

    int now = ++cnt;

    fa[now] = f;

    sz[now] = 1;

    val[now] = original[mid];

    lson(now) = build(l, mid - 1, now);

    rson(now) = build(mid + 1, r, now);

    update(now);

    return now;

}

int main()

{

    int n, m, x, y;

    scanf("%d%d", &n, &m);

    original[1] = -inf, original[n + 2] = inf;

    for (int i = 1; i <= n; i++)

    {

        original[i + 1] = i;

    }

    root = build(1, n + 2, 0);

    for (int i = 1; i <= m; i++)

    {

        scanf("%d%d", &x, &y);

        mark(x + 1, y + 1);

    }

    output(root);

    return 0;

}

// l r表示翻转的区间 最后输出反转后结果

### 可并堆

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e5 + 10;

int lson[maxn], rson[maxn], val[maxn], dis[maxn];

int rt[maxn];

int del[maxn];

int findrt(int x)

{

    return x == rt[x] ? x : rt[x] = findrt(rt[x]);

}

int merge(int x, int y)

{

    if (!x || !y)

        return x + y;

    if (val[x] > val[y])

        swap(x, y);

    rson[x] = merge(rson[x], y);

    if (dis[lson[x]] < dis[rson[x]])

        swap(lson[x], rson[x]);

    rt[lson[x]] = rt[rson[x]] = rt[x] = x;

    dis[x] = dis[rson[x]] + 1;

    return x;

}

void pop(int x)

{

    rt[lson[x]] = rt[rson[x]] = rt[x] = merge(lson[x], rson[x]);

}

int main()

{

    int n, m;

    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++)

        scanf("%d", &val[i]);

    for (int i = 1; i <= n; i++)

        rt[i] = i;

    while (m--)

    {

        int op, x, y;

        scanf("%d%d", &op, &x);

        if (op == 1)

        {

            scanf("%d", &y);

            if (del[x] || del[y])

                continue;

            if (findrt(x) == findrt(y))

                continue;

            rt[x] = rt[y] = merge(rt[x], rt[y]);

        }

        else

        {

            if (del[x])

            {

                printf("-1\n");

                continue;

            }

            printf("%d\n", val[findrt(x)]);

            del[rt[x]] = 1;

            pop(rt[x]);

        }

    }

    return 0;

}

/\*

1 x y：将第 x 个数和第 y 个数所在的小根堆合并（若第 x 或第 y 个数已经被删除或第 x 和第 y 个数在同一个堆内，则无视此操作）。

2 x：输出第 x 个数所在的堆最小数，并将这个最小数删除（若有多个最小数，优先删除先输入的；若第 x 个数已经被删除，则输出 −1 并无视删除操作）。

\*/

### LCT

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

template <int N, class Info>

class LCT

{

public:

    int fa[N];

    int ch[N][2];

    Info val[N];

    Info data[N];

    bool tag[N];

    int rt;

#define ls(x) (ch[x][0])

#define rs(x) (ch[x][1])

#define ident(x, f) (rs(f) == x)

#define connect(x, f, k) (ch[fa[x] = f][k] = x)

#define ntroot(x) (rs(fa[x]) == x || ls(fa[x]) == x)

#define reverse(x) swap(ls(x), rs(x)), tag[x] ^= 1

    void update(int x)

    {

        data[x] = val[x] ^ data[ls(x)] ^ data[rs(x)];

    }

    void pushdown(int x)

    {

        if (tag[x])

        {

            reverse(ls(x));

            reverse(rs(x));

        }

        tag[x] = 0;

    }

    void rotate(int x)

    {

        int f = fa[x], ff = fa[f], k = ident(x, f);

        connect(ch[x][k ^ 1], f, k);

        fa[x] = ff;

        if (ntroot(f))

            ch[ff][ident(f, ff)] = x;

        connect(f, x, k ^ 1);

        update(f), update(x);

    }

    void pushall(int x)

    {

        if (ntroot(x))

            pushall(fa[x]);

        pushdown(x);

    }

    void splaying(int x)

    {

        pushall(x);

        while (ntroot(x))

        {

            int f = fa[x], ff = fa[f];

            if (ntroot(f))

                ident(x, f) ^ ident(f, ff) ? rotate(x) : rotate(f);

            rotate(x);

        }

    }

    void access(int x)

    {

        for (int y = 0; x; x = fa[y = x])

        {

            splaying(x);

            rs(x) = y;

            update(x);

        }

    }

    void mkroot(int x)

    {

        access(x);

        splaying(x);

        reverse(x);

    }

    int findroot(int x)

    {

        access(x);

        splaying(x);

        while (ls(x))

        {

            pushdown(x);

            x = ls(x);

        }

        splaying(x);

        return x;

    }

    void link(int x, int y)

    {

        mkroot(x);

        if (findroot(y) == x)

            return;

        fa[x] = y; // 仅仅连了虚边，所以不用update

    }

    void cut(int x, int y)

    {

        mkroot(x);

        if (findroot(y) != x || fa[y] != x || ls(y))

            return;

        rs(x) = 0;

        fa[y] = 0;

        update(x); // 在一条splay中断开，需要更新

    }

    void split(int x, int y)

    {

        mkroot(x);

        access(y);

        splaying(y);

        // 此时y一定没有右儿子

    }

    void modify(int x, Info v)

    {

        splaying(x);

        val[x] = v;

        update(x);

    }

};

LCT<100010, int> L;

int main()

{

    int n, m;

    cin >> n >> m;

    for (int i = 1; i <= n; i++)

    {

        cin >> L.val[i];

        L.data[i] = L.val[i];

    }

    while (m--)

    {

        int op, x, y;

        cin >> op >> x >> y;

        if (op == 0)

        {

            L.split(x, y);

            cout << L.data[y] << endl;

        }

        else if (op == 1)

        {

            L.link(x, y);

        }

        else if (op == 2)

        {

            L.cut(x, y);

        }

        else

        {

            L.modify(x, y);

        }

    }

    return 0;

}

/\*

0 x y 代表询问从 x 到 y 的路径上的点的权值的 xor 和。保证 x 到 y是联通的。

1 x y 代表连接 x 到 y，若 x 到 y 已经联通则无需连接。

2 x y 代表删除边 (x,y)，不保证边 (x,y) 存在。

3 x y 代表将点 x上的权值变成 y。

\*/

### 主席树

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl '\n'

const int N = 2e5 + 10;

struct sg

{

    int l, r, sum;

} tr[N \* 40];

int cnt;

int a[N];

vector<int> v;

void pushup(int p)

{

    tr[p].sum = tr[tr[p].l].sum + tr[tr[p].r].sum;

}

void update(int l, int r, int pre, int &now, int v)

{

    tr[++cnt] = tr[pre];

    now = cnt;

    if (l == r)

    {

        tr[now].sum++;

        return;

    }

    int mid = (l + r) >> 1;

    if (v <= mid)

        update(l, mid, tr[pre].l, tr[now].l, v);

    else

        update(mid + 1, r, tr[pre].r, tr[now].r, v);

    pushup(now);

}

int query(int l, int r, int L, int R, int k)

{

    if (l == r)

        return l;

    int mid = (l + r) >> 1;

    int tmp = tr[tr[R].l].sum - tr[tr[L].l].sum;

    if (k <= tmp)

        return query(l, mid, tr[L].l, tr[R].l, k);

    else

        return query(mid + 1, r, tr[L].r, tr[R].r, k - tmp);

}

int rt[N];

int getid(int x)

{

    return lower\_bound(v.begin(), v.end(), x) - v.begin() + 1;

}

int n, m;

signed main()

{

    ios::sync\_with\_stdio(0);

    cin.tie(0), cout.tie(0);

    cin >> n >> m;

    for (int i = 1; i <= n; i++)

    {

        cin >> a[i];

        v.push\_back(a[i]);

    }

    sort(v.begin(), v.end());

    v.erase(unique(v.begin(), v.end()), v.end());

    for (int i = 1; i <= n; i++)

    {

        update(1, n, rt[i - 1], rt[i], getid(a[i]));

    }

    for (int i = 1; i <= m; i++)

    {

        int l, r, k;

        cin >> l >> r >> k;

        cout << v[query(1, n, rt[l - 1], rt[r], k) - 1] << endl;

    }

    return 0;

} // 记得root

### 倍增RMQ

#include <cstdio>

#define N 1000001

int min(int x, int y) { return x < y ? x : y; }

int f[N][21];

int sr[N];

int log[N];

int tot = 0;

int main()

{

    int n, q;

    scanf("%d%d", &n, &q);

    for (int i = 1; i <= n; i++)

        scanf("%d", &f[i][0]);

    for (int j = 1; (1 << j) <= n; j++)

    {

        for (int i = 1; i + (1 << j) - 1 <= n; i++)

            f[i][j] = min(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);

    }

    log[1] = 0;

    for (int i = 2; i <= n; i++)

        log[i] = log[i >> 1] + 1;

    while (q--)

    {

        int l, r, ans;

        scanf("%d%d", &l, &r);

        int d = r - l;

        int k = log[d + 1];

        ans = min(f[l][k], f[r - (1 << k) + 1][k]);

        sr[++tot] = ans;

    }

    for (int i = 1; i < tot; i++)

        printf("%d ", sr[i]);

    printf("%d", sr[tot]);

    return 0;

}

### 莫队

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl '\n'

const int N = 5e4 + 10;

int pos[N];

int c[N];

int cnt[N];

int ans = 0;

int ret[N];

int s[N];

struct Q

{

    int l, r, k;

} q[N];

void add(int p)

{

    int t = c[p];

    ans -= cnt[t] \* cnt[t] - cnt[t];

    cnt[t]++;

    ans += cnt[t] \* cnt[t] - cnt[t];

}

void de(int p)

{

    int t = c[p];

    ans -= cnt[t] \* cnt[t] - cnt[t];

    cnt[t]--;

    ans += cnt[t] \* cnt[t] - cnt[t];

}

int gcd(int x, int y)

{

    if (!y)

        return x;

    return gcd(y, x % y);

}

signed main()

{

    ios::sync\_with\_stdio(0);

    cin.tie(0), cout.tie(0);

    int n, m;

    cin >> n >> m;

    int sz = sqrt(n);

    for (int i = 1; i <= n; i++)

        cin >> c[i], pos[i] = i / sz;

    for (int i = 1; i <= m; i++)

    {

        cin >> q[i].l >> q[i].r;

        q[i].k = i;

    }

    sort(q + 1, q + 1 + m, [](Q a, Q b)

         { return pos[a.l] == pos[b.l] ? a.r < b.r : a.l < b.l; });

    int l = 1, r = 0;

    for (int i = 1; i <= m; i++)

    {

        while (q[i].l < l)

            add(--l);

        while (q[i].l > l)

            de(l++);

        while (q[i].r < r)

            de(r--);

        while (q[i].r > r)

            add(++r);

        ret[q[i].k] = ans;

        s[q[i].k] = (q[i].r - q[i].l + 1) \* (q[i].r - q[i].l);

    }

    for (int i = 1; i <= m; i++)

    {

        int g = gcd(ret[i], s[i]);

        if (s[i] == 0)

            ret[i] = 0, s[i] = g;

        ret[i] /= g, s[i] /= g;

        cout << ret[i] << "/" << s[i] << endl;

    }

    return 0;

}

/\*

求一个区间内每种颜色数目的平方和

\*/

### 带修改莫队

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl '\n'

const int N = 1e6 + 10;

int c[N];

int cc[N];

int cnt[N];

int pos[N];

int now = 0;

int ans[N];

int l = 1, r = 0;

void add(int x)

{

    if (!cnt[c[x]])

        now++;

    cnt[c[x]]++;

}

void del(int x)

{

    cnt[c[x]]--;

    if (!cnt[c[x]])

        now--;

}

void update(int v, int p)

{

    if (p >= l && p <= r)

    {

        del(p);

        c[p] = v;

        add(p);

    }

    else

    {

        c[p] = v;

    }

}

struct Q

{

    int l, r, k, t;

} q[N];

struct U

{

    int v, p, w;

} u[N];

signed main()

{

    ios::sync\_with\_stdio(0);

    cin.tie(0), cout.tie(0);

    int n, m;

    cin >> n >> m;

    int sz = pow(n, 0.6666666);

    for (int i = 1; i <= n; i++)

    {

        cin >> c[i];

        cc[i] = c[i];

        pos[i] = i / sz;

    }

    int num = 0, pre = 0;

    for (int i = 1; i <= m; i++)

    {

        char op;

        int a, b;

        cin >> op >> a >> b;

        if (op == 'Q')

        {

            num++;

            q[num] = Q{a, b, num, pre};

        }

        else

        {

            u[++pre] = U{b, a, c[a]};

            c[a] = b;

        }

    }

    memcpy(c, cc, sizeof(cc));

    sort(q + 1, q + 1 + num, [](Q a, Q b)

         {

        if(pos[a.l]==pos[b.l]){

            if(pos[a.r]==pos[b.r]) return a.t<b.t;

            return a.r<b.r;

        }

        return a.l<b.l; });

    pre = 0;

    for (int i = 1; i <= num; i++)

    {

        while (q[i].t > pre)

        {

            pre++;

            update(u[pre].v, u[pre].p);

        }

        while (q[i].t < pre)

        {

            update(u[pre].w, u[pre].p);

            pre--;

        }

        while (q[i].l < l)

            add(--l);

        while (q[i].l > l)

            del(l++);

        while (q[i].r < r)

            del(r--);

        while (q[i].r > r)

            add(++r);

        ans[q[i].k] = now;

    }

    for (int i = 1; i <= num; i++)

        cout << ans[i] << endl;

    return 0;

}

/\*

Q L R 代表询问你从第 L 支画笔到第

R 支画笔中共有几种不同颜色的画笔。

R P C 把第 P 支画笔替换为颜色 C

\*/

## 图论

### Tarjan缩点

#include <bits/stdc++.h>

using namespace std;

#define maxn 1000100

int head[maxn], nxt[maxn], to[maxn], v[maxn], tot = 0;

int n, m;

struct edge

{

    int l, r;

} e[maxn];

queue<int> q;

int dp[maxn];

void add(int x, int y)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

}

int low[maxn], dfn[maxn], tim = 0;

stack<int> s;

bool ins[maxn];

int scc[maxn]; // 缩点后编号

int V[maxn];

int cnt = 0;

void tarjan(int x)

{

    dfn[x] = low[x] = ++tim;

    s.push(x);

    ins[x] = 1;

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (!dfn[y])

        {

            tarjan(y);

            low[x] = min(low[x], low[y]);

        }

        else if (ins[y])

            low[x] = min(low[x], low[y]);

    }

    if (low[x] == dfn[x])

    {

        int cur;

        ++cnt;

        do

        {

            cur = s.top();

            s.pop();

            ins[cur] = 0;

            scc[cur] = cnt;

            V[cnt] += v[cur];

        } while (cur != x);

    }

}

int deg[maxn];

int topo()

{

    while (!q.empty())

    {

        int x = q.front();

        q.pop();

        for (int i = head[x]; i; i = nxt[i])

        {

            int y = to[i];

            dp[y] = max(dp[y], dp[x] + V[y]);

            deg[y]--;

            if (!deg[y])

                q.push(y);

        }

    }

    int ans = 0;

    for (int i = 1; i <= n; i++)

        ans = max(ans, dp[i]);

    return ans;

}

int main()

{

    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++)

        scanf("%d", &v[i]);

    for (int i = 1; i <= m; i++)

    {

        int x, y;

        scanf("%d%d", &x, &y);

        add(x, y);

        e[i].l = x, e[i].r = y;

    }

    for (int i = 1; i <= n; i++)

    {

        if (dfn[i] == 0)

            tarjan(i);

    }

    memset(head, 0, sizeof(head));

    memset(to, 0, sizeof(to));

    memset(nxt, 0, sizeof(nxt));

    tot = 0;

    for (int i = 1; i <= m; i++)

    {

        int x = e[i].l;

        int y = e[i].r;

        if (scc[x] == scc[y])

            continue;

        add(scc[x], scc[y]);

        deg[scc[y]]++;

    }

    for (int i = 1; i <= cnt; i++)

        if (!deg[i])

            q.push(i), dp[i] = V[i];

    int ans = topo();

    printf("%d", ans);

    return 0;

}

### Tarjan割点

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int inf = INT\_MAX;

const ll Linf = LLONG\_MAX;

#define endl '\n'

const int N = 4e5 + 10;

const int M = 2e5 + 10;

int n, m;

int head[N], nxt[M], to[M], tot = 0;

void add(int x, int y)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

}

int cut[N];

int low[N], dfn[N], tim;

int cnt = 0;

void tarjan(int x, int f)

{

    dfn[x] = low[x] = ++tim;

    int son = 0;

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (!dfn[y])

        {

            tarjan(y, f);

            low[x] = min(low[x], low[y]);

            son++;

            if (low[y] >= dfn[x] && x != f)

                cut[x] = 1;

        }

        else

            low[x] = min(low[x], dfn[y]);

    }

    if (x == f && son >= 2)

        cut[x] = 1;

    if (cut[x])

        cnt++;

}

signed main()

{

    ios::sync\_with\_stdio(false);

    cin.tie(0), cout.tie(0);

    cin >> n >> m;

    for (int i = 1; i <= m; i++)

    {

        int x, y;

        cin >> x >> y;

        add(x, y);

        add(y, x);

    }

    for (int i = 1; i <= n; i++)

    {

        if (!dfn[i])

            tarjan(i, i);

    }

    cout << cnt << endl;

    for (int i = 1; i <= n; i++)

        if (cut[i])

            cout << i << " ";

    return 0;

}

### 边双连通分量

#include <cstdio>

#include <algorithm>

#include <vector>

using namespace std;

const int maxn = 500010, maxm = 4000010;

int n, m, cnt = 1, ans, id;

int dfn[maxn], low[maxn], head[maxn], dcc[maxn];

struct edge

{

    int to, nxt;

} e[maxm];

bool b[maxm];

vector<vector<int>> Ans;

void add(int f, int t)

{

    e[++cnt].to = t;

    e[cnt].nxt = head[f];

    head[f] = cnt;

}

void tarjan(int node, int in\_edge)

{

    dfn[node] = low[node] = ++id;

    for (int i = head[node]; i; i = e[i].nxt)

    {

        const int to = e[i].to;

        if (dfn[to] == 0)

        {

            tarjan(to, i);

            if (dfn[node] < low[to])

                b[i] = b[i ^ 1] = 1;

            low[node] = min(low[node], low[to]);

        }

        else if (i != (in\_edge ^ 1))

            low[node] = min(low[node], dfn[to]);

    }

}

void dfs(int node, int ndcc)

{

    dcc[node] = ndcc;

    Ans[ndcc - 1].push\_back(node);

    for (int i = head[node]; i; i = e[i].nxt)

    {

        int to = e[i].to;

        if (dcc[to] || b[i])

            continue;

        dfs(to, ndcc);

    }

}

int main()

{

    scanf("%d %d", &n, &m);

    for (int i = 1; i <= m; ++i)

    {

        int f, t;

        scanf("%d %d", &f, &t);

        if (f == t)

            continue;

        add(f, t);

        add(t, f);

    }

    for (int i = 1; i <= n; ++i)

        if (dfn[i] == 0)

            tarjan(i, 0);

    for (int i = 1; i <= n; ++i)

        if (dcc[i] == 0)

        {

            Ans.push\_back(vector<int>());

            dfs(i, ++ans);

        }

    printf("%d\n", ans);

    for (int i = 0; i < ans; ++i)

    {

        printf("%d", Ans[i].size());

        for (int j = 0; j < Ans[i].size(); ++j)

            printf(" %d", Ans[i][j]);

        printf("\n");

    }

    return 0;

}

### 点双连通分量

#include <bits/stdc++.h>

using namespace std;

const int N = 5e5 + 5, M = 4e6 + 5;

int cnt = 1, fir[N], nxt[M], to[M];

int s[M], top, bcc, low[N], dfn[N], idx, n, m;

vector<int> ans[N];

inline void tarjan(int u, int fa)

{

    int son = 0;

    low[u] = dfn[u] = ++idx;

    s[++top] = u;

    for (int i = fir[u]; i; i = nxt[i])

    {

        int v = to[i];

        if (!dfn[v])

        {

            son++;

            tarjan(v, u);

            low[u] = min(low[u], low[v]);

            if (low[v] >= dfn[u])

            {

                bcc++;

                while (s[top + 1] != v)

                    ans[bcc].push\_back(s[top--]); // 将子树出栈

                ans[bcc].push\_back(u);            // 把割点/树根也丢到点双里

            }

        }

        else if (v != fa)

            low[u] = min(low[u], dfn[v]);

    }

    if (fa == 0 && son == 0)

        ans[++bcc].push\_back(u); // 特判独立点

}

inline void add(int u, int v)

{

    to[++cnt] = v;

    nxt[cnt] = fir[u];

    fir[u] = cnt;

}

int main()

{

    scanf("%d%d", &n, &m);

    for (int i = 1; i <= m; i++)

    {

        int u, v;

        scanf("%d%d", &u, &v);

        add(u, v), add(v, u);

    }

    for (int i = 1; i <= n; i++)

    {

        if (dfn[i])

            continue;

        top = 0;

        tarjan(i, 0);

    }

    printf("%d\n", bcc);

    for (int i = 1; i <= bcc; i++)

    {

        printf("%d ", ans[i].size());

        for (int j : ans[i])

            printf("%d ", j);

        printf("\n");

    }

    return 0;

}

### 点分治

#include <iostream>

#include <cstdio>

#include <algorithm>

using namespace std;

#define N 10001

#define re register

inline int read()

{

    int x = 0, f = 1;

    char c = getchar();

    while (c < '0' || c > '9')

    {

        if (c == '-')

            f = -1;

        c = getchar();

    }

    while (c >= '0' && c <= '9')

    {

        x = (x << 3) + (x << 1) + c - '0';

        c = getchar();

    }

    return x \* f;

}

int n, m, query[101];

int e\_cnt = 0, head[N], maxp[N], siz[N], root, tot = 0, d[N], b[N], a[N];

bool vis[N], ok[101];

struct Edge

{

    int to, nxt, val;

} edge[N << 1];

void add(int a, int b, int c)

{

    e\_cnt++;

    edge[e\_cnt].nxt = head[a];

    edge[e\_cnt].to = b;

    edge[e\_cnt].val = c;

    head[a] = e\_cnt;

}

void get\_root(int u, int fa, int total)

{

    siz[u] = 1;

    maxp[u] = 0;

    for (re int i = head[u]; i; i = edge[i].nxt)

    {

        int v = edge[i].to;

        if (v == fa || vis[v])

            continue;

        get\_root(v, u, total);

        siz[u] += siz[v];

        maxp[u] = max(siz[v], maxp[u]);

    }

    maxp[u] = max(maxp[u], total - siz[u]);

    if (!root || maxp[u] < maxp[root])

    {

        root = u;

    }

}

bool cmp(int x, int y)

{

    return d[x] < d[y];

}

void get\_dis(int u, int fa, int dis, int from)

{

    a[++tot] = u;

    d[u] = dis;

    b[u] = from;

    for (re int i = head[u]; i; i = edge[i].nxt)

    {

        int v = edge[i].to;

        if (v == fa || vis[v])

            continue;

        get\_dis(v, u, dis + edge[i].val, from);

    }

}

void calc(int u)

{

    tot = 0;

    a[++tot] = u;

    d[u] = 0;

    b[u] = u;

    for (int i = head[u]; i; i = edge[i].nxt)

    {

        int v = edge[i].to;

        if (vis[v])

            continue;

        get\_dis(v, u, edge[i].val, v);

    }

    sort(a + 1, a + tot + 1, cmp);

    for (int i = 1; i <= m; i++)

    {

        int l = 1, r = tot;

        if (ok[i])

            continue;

        while (l < r)

        {

            if (d[a[l]] + d[a[r]] > query[i])

            {

                r--;

            }

            else if (d[a[l]] + d[a[r]] < query[i])

            {

                l++;

            }

            else if (b[a[l]] == b[a[r]])

            {

                if (d[a[r]] == d[a[r - 1]])

                    r--;

                else

                    l++;

            }

            else

            {

                ok[i] = true;

                break;

            }

        }

    }

}

void solve(int u)

{

    vis[u] = true;

    calc(u);

    for (re int i = head[u]; i; i = edge[i].nxt)

    {

        int v = edge[i].to;

        if (vis[v])

            continue;

        root = 0;

        get\_root(v, 0, siz[v]);

        solve(root);

    }

}

int main()

{

    n = read(), m = read();

    for (re int i = 1; i <= n - 1; i++)

    {

        int u, v, w;

        u = read(), v = read(), w = read();

        add(u, v, w);

        add(v, u, w);

    }

    for (re int i = 1; i <= m; i++)

    {

        query[i] = read();

        if (!query[i])

            ok[i] = 1; // 这里,加个特判

    }

    maxp[0] = n;

    get\_root(1, 0, n);

    solve(root);

    for (re int i = 1; i <= m; i++)

    {

        if (ok[i])

        {

            cout << "AYE" << endl;

        }

        else

        {

            cout << "NAY" << endl;

        }

    }

    return 0;

}

/\*

给定一棵有

n 个点的树，询问树上距离为 k 的点对是否存在。

\*/

### 点分树

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e5 + 10;

int n, w[maxn], m;

int head[maxn], to[maxn << 1], nxt[maxn << 1], tot = 0;

void add(int x, int y)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

}

int cnt = 0;

struct BIT

{

    int tr[maxn \* 60];

    int rt[maxn];

    int lson[maxn \* 60], rson[maxn \* 60];

    void update(int l, int r, int x, int v, int &p)

    {

        if (!p)

            p = ++cnt;

        if (l == r)

        {

            tr[p] += v;

            return;

        }

        int mid = (l + r) >> 1;

        if (x <= mid)

            update(l, mid, x, v, lson[p]);

        else

            update(mid + 1, r, x, v, rson[p]);

        tr[p] = tr[lson[p]] + tr[rson[p]];

    }

    int query(int l, int r, int x, int y, int p)

    {

        if (y < x)

            return 0;

        if (x <= l && y >= r)

            return tr[p];

        int mid = (l + r) >> 1;

        int ret = 0;

        if (x <= mid)

            ret += query(l, mid, x, y, lson[p]);

        if (y > mid)

            ret += query(mid + 1, r, x, y, rson[p]);

        return ret;

    }

    // update(int x,int v)单点修改，把x+=v

    // query(int x)区间查询，查1～i的点权和

} S, SFA;

struct LCA

{

    int son[maxn], top[maxn], sz[maxn], fa[maxn], dep[maxn];

    void dfs1(int x, int f)

    {

        fa[x] = f;

        sz[x] = 1;

        dep[x] = dep[f] + 1;

        for (int i = head[x]; i; i = nxt[i])

        {

            if (to[i] == f)

                continue;

            dfs1(to[i], x);

            sz[x] += sz[to[i]];

            if (sz[to[i]] > sz[son[x]])

                son[x] = to[i];

        }

    }

    void dfs2(int x, int tp)

    {

        top[x] = tp;

        if (son[x])

            dfs2(son[x], tp);

        for (int i = head[x]; i; i = nxt[i])

        {

            if (to[i] == son[x] || to[i] == fa[x])

                continue;

            dfs2(to[i], to[i]);

        }

    }

    int lca(int x, int y)

    {

        while (top[x] != top[y])

        {

            if (dep[top[x]] < dep[top[y]])

                swap(x, y);

            x = fa[top[x]];

        }

        return dep[x] < dep[y] ? x : y;

    }

    int dis(int x, int y)

    {

        return dep[x] + dep[y] - 2 \* dep[lca(x, y)];

    }

    void init()

    {

        dfs1(1, 0);

        dfs2(1, 1);

    }

} G;

#define y to[i]

int cnt2 = 0;

struct PDV

{

    int tsz;

    int vis[maxn], fa[maxn], rt, sz[maxn], mxp[maxn];

    void getroot(int x, int f)

    {

        sz[x] = 1;

        mxp[x] = 0;

        for (int i = head[x]; i; i = nxt[i])

        {

            if (vis[y] || y == f)

                continue;

            getroot(y, x);

            mxp[x] = max(sz[y], mxp[x]);

            sz[x] += sz[y];

        }

        mxp[x] = max(mxp[x], tsz - sz[x]);

        if (mxp[rt] > mxp[x])

            rt = x;

    }

    void dfs\_s(int x, int f)

    {

        tsz++;

        for (int i = head[x]; i; i = nxt[i])

        {

            if (vis[y] || y == f)

                continue;

            dfs\_s(y, x);

        }

    }

    void build(int x)

    { // j建点分树的同时维护好所有的BIT

        vis[x] = 1;

        for (int i = head[x]; i; i = nxt[i])

        {

            if (vis[y])

                continue;

            tsz = 0;

            dfs\_s(y, x);

            rt = 0;

            getroot(y, x);

            fa[rt] = x;

            build(rt);

        }

    }

    void init()

    {

        mxp[0] = 0x7f7f7f7f;

        rt = 0;

        tsz = n;

        getroot(1, 0);

        fa[rt] = 0;

        build(rt);

    }

    int query(int x, int k)

    {

        int ret = 0;

        int cur = x, pre = 0;

        while (cur)

        {

            if (G.dis(cur, x) > k)

            {

                pre = cur;

                cur = fa[cur];

                continue;

            }

            ret += S.query(0, n, 0, k - G.dis(cur, x), S.rt[cur]);

            if (pre)

                ret -= SFA.query(0, n, 0, k - G.dis(cur, x), SFA.rt[pre]);

            pre = cur;

            cur = fa[cur];

        }

        return ret;

    }

    void update(int x, int d)

    {

        int cur = x;

        while (cur)

        {

            S.update(0, n, G.dis(cur, x), d, S.rt[cur]);

            if (fa[cur])

                SFA.update(0, n, G.dis(fa[cur], x), d, SFA.rt[cur]);

            cur = fa[cur];

        }

    }

} P;

#undef y

signed main()

{

    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++)

        scanf("%d", &w[i]);

    for (int i = 1, x, y; i < n; i++)

    {

        scanf("%d%d", &x, &y);

        add(x, y);

        add(y, x);

    }

    G.init();

    P.init();

    int lastans = 0;

    for (int i = 1; i <= n; i++)

        P.update(i, w[i]);

    while (m--)

    {

        int op, x, y;

        scanf("%d%d%d", &op, &x, &y);

        x ^= lastans;

        y ^= lastans;

        if (op == 0)

        {

            lastans = P.query(x, y);

            printf("%d\n", lastans);

        }

        else

        {

            P.update(x, y - w[x]);

            w[x] = y;

        }

    }

    return 0;

}

/\*

给定x，求与x距离<=k的点的权值和，支持单点修改

原理：

我们通过点分治每次找重心的方式来对原树进行重构。

将每次找到的重心与上一层的重心缔结父子关系，这样就可以形成一棵 \log n 层的树。

由于树是 \log n 层的，很多原来并不对劲的暴力在点分树上均有正确的复杂度。

\*/

### 2-SAT

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int inf = INT\_MAX;

const ll Linf = LLONG\_MAX;

#define endl '\n'

const int N = 2e6 + 10;

int n, m;

int head[N], nxt[N], to[N], tot = 0;

void add(int x, int y)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

}

int dfn[N], low[N], tim;

int scc[N];

bool ins[N];

int cnt;

stack<int> s;

void tarjan(int x)

{

    low[x] = dfn[x] = ++tim;

    ins[x] = 1;

    s.push(x);

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (!dfn[y])

        {

            tarjan(y);

            low[x] = min(low[x], low[y]);

        }

        else if (ins[y])

        {

            low[x] = min(low[x], low[y]);

        }

    }

    if (low[x] == dfn[x])

    {

        int cur;

        cnt++;

        do

        {

            cur = s.top();

            s.pop();

            ins[cur] = 0;

            scc[cur] = cnt;

        } while (cur != x);

    }

}

signed main()

{

    ios::sync\_with\_stdio(false);

    cin.tie(0), cout.tie(0);

    cin >> n >> m;

    for (int i = 1; i <= m; i++)

    {

        int a, va, b, vb;

        cin >> a >> va >> b >> vb;

        add(a + (va)\*n, b + (vb ^ 1) \* n);

        add(b + (vb)\*n, a + (va ^ 1) \* n);

    }

    for (int i = 1; i <= (n << 1); i++)

    {

        if (!dfn[i])

            tarjan(i);

    }

    int flag = 1;

    for (int i = 1; i <= n; i++)

    {

        if (scc[i] == scc[i + n])

        {

            flag = 0;

            break;

        }

    }

    if (!flag)

    {

        cout << "IMPOSSIBLE" << endl;

        exit(0);

    }

    cout << "POSSIBLE" << endl;

    for (int i = 1; i <= n; i++)

    {

        cout << (scc[i] < scc[i + n]) << " ";

    }

    return 0;

}

// a为va 或 b为vb为真

### Dijkstra

#include <bits/stdc++.h>

using namespace std;

const int M = 5e5 + 1;

const int N = 1e5 + 1;

int head[N], to[M], nxt[M], w[M];

int tot = 0;

int dist[N];

bool vis[N];

priority\_queue<pair<int, int>> q;

void add(int x, int y, int z)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

    w[tot] = z;

}

void dij(int s)

{

    dist[s] = 0;

    q.push(make\_pair(0, s));

    while (!q.empty())

    {

        int x = q.top().second;

        q.pop();

        if (vis[x])

            continue;

        vis[x] = 1;

        for (int i = head[x]; i; i = nxt[i])

        {

            int y = to[i];

            if (dist[y] > dist[x] + w[i])

            {

                dist[y] = dist[x] + w[i];

                q.push(make\_pair(-dist[y], y));

            }

        }

    }

}

int main()

{

    int n, m, s;

    scanf("%d%d%d", &n, &m, &s);

    int x, y, z;

    for (int i = 1; i <= m; i++)

    {

        scanf("%d%d%d", &x, &y, &z);

        add(x, y, z);

    }

    memset(dist, 0x3f, sizeof(dist));

    dij(s);

    for (int i = 1; i <= n; i++)

    {

        printf("%d ", dist[i]);

    }

    return 0;

}

### SPFA

#include <bits/stdc++.h>

using namespace std;

int n;

const int M = 5e6 + 1;

const int N = 1e6 + 1;

int head[N], to[M], nxt[M], w[M];

int tot = 0;

int dist[N];

bool inq[N];

queue<int> q;

void spfa(int s)

{

    for (int i = 0; i <= n; i++)

    {

        dist[i] = INT\_MAX;

    }

    memset(inq, 0, sizeof(inq));

    q.push(s);

    inq[s] = 1;

    dist[s] = 0;

    while (!q.empty())

    {

        int x = q.front();

        q.pop();

        inq[x] = 0;

        for (int i = head[x]; i; i = nxt[i])

        {

            int y = to[i];

            if (dist[x] != INT\_MAX && dist[y] > dist[x] + w[i])

            {

                dist[y] = dist[x] + w[i];

                if (!inq[y])

                    q.push(y), inq[y] = 1;

            }

        }

    }

}

void add(int x, int y, int z)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

    w[tot] = z;

}

int main()

{

    int m, s;

    scanf("%d%d%d", &n, &m, &s);

    int x, y, z;

    for (int i = 1; i <= m; i++)

    {

        scanf("%d%d%d", &x, &y, &z);

        add(x, y, z);

    }

    spfa(s);

    for (int i = 1; i <= n; i++)

    {

        printf("%d ", dist[i]);

    }

    return 0;

}

### 负环

#include <bits/stdc++.h>

using namespace std;

#define maxn 60010

int head[maxn], nxt[maxn], to[maxn], tot = 0, val[maxn];

void add(int x, int y, int z)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    val[tot] = z;

    head[x] = tot;

}

int dis[maxn], cnt[maxn], inq[maxn];

queue<int> q;

#define mem(x) memset(x, 0, sizeof(x))

void init()

{

    mem(head);

    mem(to);

    mem(nxt);

    mem(val);

    tot = 0;

}

int n, m;

bool spfa()

{

    q.push(1);

    inq[1] = 1;

    memset(dis, 0x3f, sizeof(dis));

    memset(inq, 0, sizeof(inq));

    memset(cnt, 0, sizeof(cnt));

    dis[1] = 0;

    while (!q.empty())

    {

        int x = q.front();

        q.pop();

        inq[x] = 0;

        for (int i = head[x]; i; i = nxt[i])

        {

            int y = to[i];

            if (dis[y] > dis[x] + val[i])

            {

                dis[y] = dis[x] + val[i];

                cnt[y] = cnt[x] + 1;

                if (cnt[y] > n + 1)

                    return 1;

                if (!inq[y])

                    q.push(y), inq[y];

            }

        }

    }

    return 0;

}

int main()

{

    int T;

    scanf("%d", &T);

    while (T--)

    {

        init();

        scanf("%d%d", &n, &m);

        int x, y, z;

        for (int i = 1; i <= m; i++)

        {

            scanf("%d%d%d", &x, &y, &z);

            if (z >= 0)

            {

                add(x, y, z);

                add(y, x, z);

            }

            else

                add(x, y, z);

        }

        if (spfa())

            printf("YES\n");

        else

            printf("NO\n");

    }

    return 0;

}

### LCA 树剖

#include <bits/stdc++.h>

using namespace std;

int n, m, rt;

#define maxn 500010

int head[maxn], nxt[maxn << 1], to[maxn << 1], tot = 0;

int top[maxn], sz[maxn], son[maxn], fa[maxn], dep[maxn];

void add(int x, int y)

{

    to[++tot] = y;

    nxt[tot] = head[x];

    head[x] = tot;

}

void dfs1(int x, int f)

{

    fa[x] = f;

    sz[x] = 1;

    dep[x] = dep[f] + 1;

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (y == f)

            continue;

        dfs1(y, x);

        sz[x] += sz[y];

        if (sz[y] > sz[son[x]])

            son[x] = y;

    }

}

void dfs2(int x, int tp)

{

    top[x] = tp;

    if (son[x])

        dfs2(son[x], tp);

    for (int i = head[x]; i; i = nxt[i])

    {

        int y = to[i];

        if (y == fa[x] || y == son[x])

            continue;

        dfs2(y, y);

    }

}

int lca(int x, int y)

{

    while (top[x] != top[y])

    {

        if (dep[top[x]] < dep[top[y]])

            swap(x, y);

        x = fa[top[x]];

    }

    if (dep[x] > dep[y])

        swap(x, y);

    return x;

}

int main()

{

    scanf("%d%d%d", &n, &m, &rt);

    for (int i = 1; i < n; i++)

    {

        int x, y;

        scanf("%d%d", &x, &y);

        add(x, y);

        add(y, x);

    }

    dfs1(rt, 0);

    dfs2(rt, rt);

    while (m--)

    {

        int x, y;

        scanf("%d%d", &x, &y);

        printf("%d\n", lca(x, y));

    }

    return 0;

}

### LCA 倍增

#include <iostream>

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

struct edge

{

    int t, nex;

} e[500010 << 1];

int head[500010], tot;

void add(int x, int y)

{

    e[++tot].t = y;

    e[tot].nex = head[x];

    head[x] = tot;

}

int depth[500001], fa[500001][22], lg[500001];

void dfs(int now, int fath)

{

    fa[now][0] = fath;

    depth[now] = depth[fath] + 1;

    for (int i = 1; i <= 21; ++i)

    {

        if (fa[now][i - 1])

            fa[now][i] = fa[fa[now][i - 1]][i - 1];

    }

    for (int i = head[now]; i; i = e[i].nex)

        if (e[i].t != fath)

            dfs(e[i].t, now);

}

int LCA(int x, int y)

{

    if (depth[x] < depth[y])

        swap(x, y);

    while (depth[x] > depth[y])

        x = fa[x][lg[depth[x] - depth[y]] - 1];

    if (x == y)

        return x;

    for (int k = lg[depth[x]] - 1; k >= 0; --k)

        if (fa[x][k] != fa[y][k])

            x = fa[x][k], y = fa[y][k];

    return fa[x][0];

}

int main()

{

    int n, m, s;

    scanf("%d%d%d", &n, &m, &s);

    for (int i = 1; i <= n - 1; ++i)

    {

        int x, y;

        scanf("%d%d", &x, &y);

        add(x, y);

        add(y, x);

    }

    for (int i = 1; i <= n; ++i)

        lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);

    dfs(s, 0);

    for (int i = 1; i <= m; ++i)

    {

        int x, y;

        scanf("%d%d", &x, &y);

        printf("%d\n", LCA(x, y));

    }

    return 0;

}

### DSU On Tree

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

#define endl '\n'

const int N = 1e5 + 10;

vector<int> g[N];

int c[N];

int cnt[N];

int ans[N];

int son[N];

int sz[N];

int counttot;

void getson(int x, int f)

{

    for (int y : g[x])

    {

        if (y == f)

            continue;

        getson(y, x);

        sz[x] += sz[y];

        if (sz[y] > sz[son[x]])

            son[x] = y;

    }

}

void calc(int x)

{

    if (cnt[c[x]] == 0)

        counttot++;

    cnt[c[x]]++;

}

void del(int x)

{

    cnt[c[x]]--;

    if (cnt[c[x]] == 0)

        counttot--;

}

void delt(int x, int f)

{

    del(x);

    for (int y : g[x])

    {

        if (y == f)

            continue;

        delt(y, x);

    }

}

void count(int x, int f)

{

    calc(x);

    for (int y : g[x])

    {

        if (y == f)

            continue;

        count(y, x);

    }

}

void dfs(int x, int f, int kep)

{

    for (int y : g[x])

    {

        if (y == son[x] || y == f)

            continue;

        dfs(y, x, 0);

    }

    if (son[x])

        dfs(son[x], x, 1);

    calc(x);

    for (int y : g[x])

    {

        if (y == son[x] || y == f)

            continue;

        count(y, x);

    }

    ans[x] = counttot;

    if (!kep)

    {

        delt(x, f);

    }

}

void solve()

{

    int n;

    cin >> n;

    for (int i = 1; i < n; i++)

    {

        int x, y;

        cin >> x >> y;

        g[x].push\_back(y);

        g[y].push\_back(x);

    }

    for (int i = 1; i <= n; i++)

    {

        cin >> c[i];

    }

    getson(1, 0);

    dfs(1, 0, 1);

    int m;

    cin >> m;

    for (int i = 1; i <= m; i++)

    {

        int x;

        cin >> x;

        cout << ans[x] << endl;

    }

}

signed main()

{

    ios::sync\_with\_stdio(false);

    cin.tie(0), cout.tie(0);

    int T = 1;

    // cin>>T;

    while (T--)

        solve();

    return 0;

}

/\*

修改时只需修改del  calc

\*/

### 树上背包

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

int f[305][305], s[305], n, m;

vector<int> e[305];

int dfs(int u)

{

    int p = 1;

    f[u][1] = s[u];

    for (auto v : e[u])

    {

        int siz = dfs(v);

        // 注意下面两重循环的上界和下界

        // 只考虑已经合并过的子树，以及选的课程数超过 m+1 的状态没有意义

        for (int i = min(p, m + 1); i; i--)

            for (int j = 1; j <= siz && i + j <= m + 1; j++)

                f[u][i + j] = max(f[u][i + j], f[u][i] + f[v][j]); // 转移方程

        p += siz;

    }

    return p;

}

int main()

{

    cin.tie(nullptr)->sync\_with\_stdio(false);

    cin >> n >> m;

    for (int i = 1; i <= n; i++)

    {

        int k;

        cin >> k >> s[i];

        e[k].push\_back(i);

    }

    dfs(0);

    cout << f[0][m + 1];

    return 0;

}

/\*

现在有 n 门课程，第 i 门课程的学分为 a\_i，每门课程有零门或一门先修课，有先修课的课程需要先学完其先修课，才能学习该课程。

一位学生要学习 m 门课程，求其能获得的最多学分数。

\*/

#include <bits/stdc++.h>

#define endl '\n'

using namespace std;

typedef long long ll;

const int N = 2e3 + 10;

ll dp[N][N][2];

vector<int> g[N]; //

ll h[N];

int sz[N];

int n;

const ll inf = 1e18;

void dfs(int x, int f)

{

    sz[x] = 1;

    dp[x][0][0] = h[x];

    dp[x][1][1] = 0;

    dp[x][0][1] = inf;

    dp[x][1][0] = inf;

    for (auto y : g[x])

    {

        if (y == f)

            continue;

        dfs(y, x);

        sz[x] += sz[y];

        for (int i = sz[x]; i >= 0; i--)

        {

            ll cur1 = inf, cur2 = inf;

            for (int j = min(i, sz[y]); j >= 0; j--)

            {

                if (i - j > sz[x] - sz[y])

                    break;

                cur1 = min(cur1, dp[x][i - j][0] + min(dp[y][j][0] + h[y], dp[y][j][1]));

                cur2 = min(cur2, dp[x][i - j][1] + min(dp[y][j][0], dp[y][j][1]));

            }

            dp[x][i][1] = cur2;

            dp[x][i][0] = cur1;

        }

    }

}

void solve()

{

    cin >> n;

    for (int i = 1; i <= n; i++)

    {

        g[i].clear();

    }

    for (int i = 2; i <= n; i++)

    {

        int x;

        cin >> x;

        g[x].push\_back(i);

        g[i].push\_back(x);

    }

    for (int i = 1; i <= n; i++)

    {

        cin >> h[i];

    }

    dfs(1, 0);

    for (int i = 0; i <= n; i++)

    {

        cout << min(dp[1][i][0], dp[1][i][1]) << " ";

    }

    cout << endl;

}

signed main()

{

    ios::sync\_with\_stdio(0);

    cin.tie(0), cout.tie(0);

    int T = 1;

    cin >> T;

    while (T--)

        solve();

    return 0;

}

/\*

求最小值

\*/

### 最大流

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl '\n'

int n, m, s, t;

const int N = 5010;

const int M = 5010;

int head[N << 1], nxt[M << 1], to[M << 1], val[M << 1], tot = 1;

void add(int x, int y, int z)

{

    nxt[++tot] = head[x];

    head[x] = tot;

    val[tot] = z;

    to[tot] = y;

}

int dep[N << 1];

int cur[N << 1];

int bfs()

{

    memset(dep, 0, sizeof(dep));

    queue<int> q;

    q.push(s);

    dep[s] = 1;

    while (!q.empty())

    {

        int x = q.front();

        q.pop();

        for (int i = head[x]; i; i = nxt[i])

        {

            int y = to[i];

            if (!val[i] || dep[y])

                continue;

            dep[y] = dep[x] + 1;

            q.push(y);

        }

    }

    return dep[t];

}

int dfs(int x, int flow)

{

    if (x == t)

        return flow;

    int ret = 0;

    for (int i = cur[x]; i && flow; i = nxt[i])

    {

        cur[x] = i;

        int y = to[i];

        if (val[i] > 0 && (dep[y] == dep[x] + 1))

        {

            int tmp = dfs(y, min(flow, val[i]));

            if (tmp == 0)

                dep[y] = 0;

            val[i] -= tmp;

            val[i ^ 1] += tmp;

            ret += tmp;

            flow -= tmp;

        }

    }

    return ret;

}

signed main()

{

    cin >> n >> m >> s >> t;

    for (int i = 1; i <= m; i++)

    {

        int x, y, z;

        cin >> x >> y >> z;

        add(x, y, z);

        add(y, x, 0);

    }

    int ans = 0;

    while (bfs())

    {

        memcpy(cur, head, sizeof(head));

        ans += dfs(s, 1e18);

    }

    cout << ans << endl;

    return 0;

}

### 费用流

#include <bits/stdc++.h>

using namespace std;

// 分为最小费用可行流（上下界网络流）和最小费用最大流，区别仅在addEdge()

struct MCFGraph

{

    struct Edge

    {

        int v, f, c;

        Edge(int v, int f, int c) : v(v), f(f), c(c) {}

    };

    const int n;

    vector<Edge> e;

    vector<vector<int>> g;

    vector<int> h, dis;

    vector<int> pre;

    bool dijkstra(int s, int t)

    {

        dis.assign(n + 1, numeric\_limits<int>::max());

        pre.assign(n + 1, -1);

        priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> que;

        dis[s] = 0;

        que.emplace(0, s);

        while (!que.empty())

        {

            auto [d, u] = que.top();

            que.pop();

            if (dis[u] < d)

                continue;

            for (int i : g[u])

            {

                auto [v, f, c] = e[i];

                if (f > 0 && dis[v] > d + h[u] - h[v] + c)

                {

                    dis[v] = d + h[u] - h[v] + c;

                    pre[v] = i;

                    que.emplace(dis[v], v);

                }

            }

        }

        return dis[t] != numeric\_limits<int>::max();

    }

    MCFGraph(int n) : n(n), g(n + 1) {}

    // 最小费用可行流

    void addEdge(int u, int v, int f, int c)

    { // 流量 费用

        if (f < 0)

        {

            g[u].push\_back(e.size());

            e.emplace\_back(v, 0, c);

            g[v].push\_back(e.size());

            e.emplace\_back(u, f, -c);

        }

        else

        {

            g[u].push\_back(e.size());

            e.emplace\_back(v, f, c);

            g[v].push\_back(e.size());

            e.emplace\_back(u, 0, -c);

        }

    }

    // 最小费用最大流

    void addEdge(int u, int v, int f, int c)

    {

        g[u].push\_back(e.size());

        e.emplace\_back(v, f, c);

        g[v].push\_back(e.size());

        e.emplace\_back(u, 0, -c);

    }

    pair<int, int> flow(int s, int t)

    {

        int flow = 0;

        int cost = 0;

        h.assign(n + 1, 0);

        while (dijkstra(s, t))

        {

            for (int i = 1; i <= n; i++)

                h[i] += dis[i];

            int aug = numeric\_limits<int>::max();

            for (int i = t; i != s; i = e[pre[i] ^ 1].v)

                aug = min(aug, e[pre[i]].f);

            for (int i = t; i != s; i = e[pre[i] ^ 1].v)

            {

                e[pre[i]].f -= aug;

                e[pre[i] ^ 1].f += aug;

            }

            flow += aug;

            cost += int(aug) \* h[t];

        }

        return make\_pair(flow, cost);

    }

};

int main()

{

    int n, m, s, t;

    cin >> n >> m >> s >> t;

    MCFGraph g(n);

    for (int i = 1; i <= m; i++)

    {

        int u, v, w, c;

        cin >> u >> v >> w >> c;

        g.addEdge(u, v, w, c);

    }

    auto ans = g.flow(s, t);

    cout << ans.first << " " << ans.second;

    return 0;

}

## 数学

### 三分

ll l = 0, r = 100000000;

// for(int i=1;i<=10;i++){

//     cout<<calc(i)<<" "<<i<<endl;

// }

while (r - l > 5)

{

    ll lmid = l + (r - l) / 3;

    ll rmid = r - (r - l) / 3;

    double lans = calc(lmid);

    double rans = calc(rmid);

    if (lans <= rans)

        r = rmid - 1;

    else

        l = lmid + 1;

}

double res = calc(l);

for (int i = l + 1; i <= r; i++)

{

    res = min(res, calc(i));

}

// 最小值

### 欧拉筛

#include <bits/stdc++.h>

using namespace std;

#define int long long

typedef long long ll;

const int inf = INT\_MAX;

const ll Linf = LLONG\_MAX;

#define endl '\n'

const int N = 1e8 + 10;

int prime[N], cnt = 0;

bool notprime[N];

void euler(int n)

{

    for (int i = 2; i <= n; i++)

    {

        if (!notprime[i])

            prime[++cnt] = i;

        for (int j = 1; j <= cnt && i \* prime[j] <= n; j++)

        {

            notprime[prime[j] \* i] = 1;

            if (i % prime[j] == 0)

                break;

        }

    }

}

int n, m;

signed main()

{

    ios::sync\_with\_stdio(0);

    cin.tie(0), cout.tie(0);

    cin >> n >> m;

    euler(n);

    for (int i = 1; i <= m; i++)

    {

        int x;

        cin >> x;

        cout << prime[x] << endl;

    }

    return 0;

}

### 欧拉函数

int phi(int n)

{

    int res = n;

    for (int i = 2; i \* i <= n; i++)

    {

        if (n % i == 0)

        {

            while (n % i == 0)

            {

                n /= i;

            }

            res = res / i \* (i - 1);

        }

    }

    if (n > 1)

    {

        res = res / n \* (n - 1);

    }

    return res;

}

### Miller-Rabin素数检验和Pollard-Rho质因数分解

int mul(int a, int b, int m)

{

    return static\_cast<\_\_int128>(a) \* b % m;

}

int power(int a, int b, int m)

{

    int res = 1 % m;

    for (; b; b >>= 1, a = mul(a, a, m))

        if (b & 1)

            res = mul(res, a, m);

    return res;

}

// 素数检测 O((logn)^3)

bool isprime(int n)

{

    if (n < 2)

        return false;

    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};

    int s = \_\_builtin\_ctzll(n - 1);

    int d = (n - 1) >> s;

    for (auto a : A)

    {

        if (a == n)

            return true;

        int x = power(a, d, n);

        if (x == 1 || x == n - 1)

            continue;

        bool ok = false;

        for (int i = 0; i < s - 1; ++i)

        {

            x = mul(x, x, n);

            if (x == n - 1)

            {

                ok = true;

                break;

            }

        }

        if (!ok)

            return false;

    }

    return true;

}

// 因式分解 O(n^(1/4))

vector<int> factorize(int n)

{

    vector<int> p;

    function<void(int)> f = [&](int n)

    {

        if (n <= 10000)

        {

            for (int i = 2; i \* i <= n; ++i)

                for (; n % i == 0; n /= i)

                    p.push\_back(i);

            if (n > 1)

                p.push\_back(n);

            return;

        }

        if (isprime(n))

        {

            p.push\_back(n);

            return;

        }

        auto g = [&](int x)

        {

            return (mul(x, x, n) + 1) % n;

        };

        int x0 = 2;

        while (true)

        {

            int x = x0;

            int y = x0;

            int d = 1;

            int power = 1, lam = 0;

            int v = 1;

            while (d == 1)

            {

                y = g(y);

                ++lam;

                v = mul(v, abs(x - y), n);

                if (lam % 127 == 0)

                {

                    d = gcd(v, n);

                    v = 1;

                }

                if (power == lam)

                {

                    x = y;

                    power \*= 2;

                    lam = 0;

                    d = gcd(v, n);

                    v = 1;

                }

            }

            if (d != n)

            {

                f(d);

                f(n / d);

                return;

            }

            ++x0;

        }

    };

    f(n);

    sort(p.begin(), p.end());

    return p;

}

### 快速幂

int qpow(int a, int b)

{

    int res = 1;

    for (; b; b /= 2, a = a \* a % mod)

    {

        if (b & 1)

        {

            res = res \* a % mod;

        }

    }

    return res;

}

### 拓展欧几里得

int exgcd(int a, int b, int &x, int &y)

{

    if (!b)

    {

        x = 1, y = 0;

        return a;

    }

    int g = exgcd(b, a % b, y, x);

    y -= a / b \* x;

    return g;

}

// ax + by = c, 可求解线性同余方程 ax === b (mod c)

bool liEu(int a, int b, int c, int &x, int &y)

{

    int d = exgcd(a, b, x, y);

    if (c % d != 0)

        return false;

    int k = c / d;

    x \*= k;

    y \*= k;

    return true;

}

### 逆元

const int mod = 1e9 + 7;

int qpow(int a, int b)

{

    int res = 1;

    for (; b; b /= 2, a = a \* a % mod)

    {

        if (b & 1)

        {

            res = res \* a % mod;

        }

    }

    return res;

}

int inv(int a)

{

    return qpow(a, mod - 2);

}

// 1-n的逆元

vector<int> linear\_inv(int n)

{

    vector<int> iv(n + 1);

    iv[1] = 1;

    for (int i = 2; i <= n; i++)

    {

        iv[i] = (mod - mod / i) \* iv[mod % i] % mod;

    }

    return iv;

}

// 任意n个数的逆元

vector<int> linear\_inv(const vector<int> &a)

{

    int n = a.size() - 1;

    vector<int> iv(n + 1), s(n + 1), sv(n + 1);

    s[0] = 1;

    for (int i = 1; i <= n; i++)

    {

        s[i] = s[i - 1] \* a[i] % mod;

    }

    sv[n] = inv(s[n]);

    for (int i = n; i >= 1; i--)

    {

        sv[i - 1] = sv[i] \* a[i] % mod;

    }

    for (int i = 1; i <= n; i++)

    {

        iv[i] = sv[i] \* s[i - 1] % mod;

    }

    return iv;

}

### 中国剩余定理

int exgcd(int a, int b, int &x, int &y)

{

    if (!b)

    {

        x = 1, y = 0;

        return a;

    }

    int g = exgcd(b, a % b, y, x);

    y -= a / b \* x;

    return g;

}

int inv(int a, int p)

{

    int x, y;

    exgcd(a, p, x, y);

    return x;

}

// x === a[i] (mod r[i]) 其中r[i]liang'liang

int CRT(const vector<int> &a, const vector<int> &r)

{

    int n = a.size() - 1;

    int s = 1;

    int ans = 0;

    for (int i = 1; i <= n; i++)

    {

        s \*= r[i];

    }

    for (int i = 1; i <= n; i++)

    {

        int m = s / r[i];

        int invm = inv(m, r[i]);

        ans = (ans + (\_\_int128)invm \* a[i] \* m % s) % s;

    }

    return (ans % s + s) % s;

}

### 线性基

// 1e9 : L = 30 | 1e18 : L = 60

constexpr int L = 60;

struct Basis

{

    int a[L + 1]{};

    bool zero = false;

    Basis() {}

    Basis(vector<int> v)

    {

        for (auto x : v)

        {

            add(x);

        }

    }

    void add(int x)

    {

        for (int i = L; i >= 0; i--)

        {

            if (!(x >> i & 1))

            {

                continue;

            }

            if (a[i] != 0)

            {

                x ^= a[i];

            }

            else

            {

                for (int j = 0; j < i; j++)

                {

                    if (x >> j & 1)

                    {

                        x ^= a[j];

                    }

                }

                for (int j = i + 1; j <= L; j++)

                {

                    if (a[j] >> i & 1)

                    {

                        a[j] ^= x;

                    }

                }

                a[i] = x;

                return;

            }

        }

        zero = true;

    }

    bool find(int x)

    {

        for (int i = 0; i <= L; i++)

        {

            if (x >> i & 1)

            {

                if (a[i] == 0)

                {

                    return false;

                }

                x ^= a[i];

            }

        }

        return true;

    }

    int queryMax()

    {

        int ret = 0;

        for (int i = 0; i <= L; i++)

        {

            ret ^= a[i];

        }

        return ret;

    }

    int queryMin()

    {

        if (zero)

        {

            return 0;

        }

        for (int i = 0; i <= L; i++)

        {

            if (a[i] != 0)

            {

                return a[i];

            }

        }

    }

    // 第k小

    int kth(int k)

    {

        k -= zero;

        vector<int> v;

        for (int i = 0; i <= L; i++)

        {

            if (a[i] != 0)

            {

                v.push\_back(a[i]);

            }

        }

        if (k >= (1ll << v.size()))

        {

            return -1;

        }

        int ret = 0;

        for (int i = 0; i < v.size(); i++)

        {

            if (k >> i & 1)

            {

                ret ^= v[i];

            }

        }

        return ret;

    }

};

### Simpson积分

double f(double x)

{

}

double simpson(double l, double r)

{

    double mid = (l + r) / 2;

    return (r - l) \* (f(l) + 4 \* f(mid) + f(r)) / 6;

}

double asr(double l, double r, double eps, double ans, int step)

{

    double mid = (l + r) / 2;

    double fl = simpson(l, mid), fr = simpson(mid, r);

    if (fabs(fl + fr - ans) <= 15 \* eps && step < 0)

        return fl + fr + (fl + fr - ans) / 15;

    return asr(l, mid, eps / 2, fl, step - 1) + asr(mid, r, eps / 2, fr, step - 1);

}

double calc(double l, double r, double eps)

{

    return asr(l, r, eps, simpson(l, r), 12);

}

### 矩阵

using db = double;

constexpr db eps = 1e-9;

struct Matrix : vector<vector<db>>

{

    int n, m;

    Matrix() {}

    Matrix(int n\_, int m\_)

    {

        init(n\_, m\_);

    }

    void init(int n\_, int m\_)

    {

        n = n\_, m = m\_;

        this->assign(n\_ + 1, vector<db>(m\_ + 1));

    }

    // 方阵

    void unit()

    {

        for (int i = 1; i <= n; i++)

        {

            (\*this)[i][i] = 1;

        }

    }

};

Matrix operator+(const Matrix &a, const Matrix &b)

{

    Matrix c = a;

    for (int i = 1; i <= a.n; i++)

    {

        for (int j = 1; j <= a.m; j++)

        {

            c[i][j] += b[i][j];

        }

    }

    return c;

}

Matrix operator-(const Matrix &a, const Matrix &b)

{

    Matrix c = a;

    for (int i = 1; i <= a.n; i++)

    {

        for (int j = 1; j <= a.m; j++)

        {

            c[i][j] -= b[i][j];

        }

    }

    return c;

}

Matrix operator\*(const Matrix &a, const Matrix &b)

{

    assert(a.m == b.n);

    int n = a.n, m = b.m;

    Matrix c(n, m);

    for (int i = 1; i <= n; i++)

    {

        for (int t = 1; t <= a.m; t++)

        {

            for (int j = 1; j <= m; j++)

            {

                c[i][j] += a[i][t] \* b[t][j];

            }

        }

    }

    return c;

}

Matrix qpow(Matrix a, int b)

{

    Matrix res(a.n, a.m);

    res.unit();

    for (; b; b /= 2, a = a \* a)

    {

        if (b & 1)

        {

            res = res \* a;

        }

    }

    return res;

}

struct Operation

{

    int type;

    int x, y;

    db k;

};

namespace Gauss

{

    // A 是系数矩阵

    Matrix A;

    vector<db> X;

    vector<int> pos;

    struct Operation

    {

        int type;

        int x, y;

        db k;

    };

    vector<Operation> op;

    // 将系数矩阵变换为行最简型

    int init(const Matrix &A\_)

    {

        A = A\_;

        int row = 0;

        for (int col = 1; col <= A.m && row <= A.n; ++col)

        {

            int pivot = row + 1;

            for (int i = pivot + 1; i <= A.n; ++i)

            {

                if (abs(A[i][col]) > abs(A[pivot][col]))

                    pivot = i;

            }

            if (abs(A[pivot][col]) <= eps)

            {

                pos[col] = -1;

                continue;

            }

            pos[col] = ++row;

            if (pivot != row)

            {

                op.push\_back({1, pivot, row, 0});

                for (int j = A.m; j >= col; --j)

                    swap(A[pivot][j], A[row][j]);

            }

            op.push\_back({2, row, 0, A[row][col]});

            for (int j = A.m; j >= col; --j)

                A[row][j] /= A[row][col];

            for (int i = 1; i <= A.n; ++i)

            {

                if (i == row || abs(A[i][col]) <= eps)

                    continue;

                op.push\_back({3, i, row, A[i][col]});

                for (int j = A.m; j >= col; --j)

                    A[i][j] -= A[row][j] \* A[i][col];

            }

        }

        return row;

    }

    // 解方程组 AX = b (-1 : 无解 | 0 : 唯一解 | 1 : 无穷解)

    int solve(vector<int> b)

    {

        assert(A.size() == b.size());

        for (auto [type, x, y, k] : op)

        {

            if (type == 1)

            {

                swap(b[x], b[y]);

                continue;

            }

            if (type == 2)

            {

                b[x] /= k;

                continue;

            }

            if (type == 3)

            {

                b[x] -= b[y] \* k;

                continue;

            }

        }

        for (int i = 1; i <= A.n; ++i)

        {

            if (abs(b[i]) <= eps)

                continue;

            bool ok = false;

            for (int j = 1; j <= A.m; ++j)

            {

                if (abs(A[i][j]) <= eps)

                    continue;

                ok = true;

                break;

            }

            if (!ok)

                return -1;

        }

        int ok = 0;

        for (int j = 1; j <= A.m; ++j)

        {

            if (pos[j] == -1)

            {

                X[j] = 0;

                ok = 1;

            }

            else

            {

                X[j] = b[pos[j]];

            }

        }

        return ok;

    }

}

## 字符串

### Trie

struct Trie

{

    int nxt[500010][26]{}, idx = 0;

    int cnt[500010]{}; // 该结点结尾的字符串的数量

    void insert(string s)

    { // 插入字符串

        int p = 0;

        for (int i = 0; i < (int)s.size(); i++)

        {

            int c = s[i] - 'a';

            if (!nxt[p][c])

                nxt[p][c] = ++idx; // 如果没有，就添加结点

            p = nxt[p][c];

        }

        cnt[p]++;

    }

    int find(string s)

    { // 查找字符串

        int p = 0;

        for (int i = 0; i < (int)s.size(); i++)

        {

            int c = s[i] - 'a';

            if (!nxt[p][c])

                return false;

            p = nxt[p][c];

        }

        return cnt[p];

    }

};

### Manacher

// s[l, r] 回文 => R[l + r + 1] >= r - l + 2 (下标从0开始)

vector<int> manacher(string s)

{

    string t = "#";

    for (auto c : s)

    {

        t += c;

        t += '#';

    }

    int n = t.size();

    vector<int> R(n);

    for (int i = 0, j = 0; i < n; i++)

    {

        if (2 \* j - i >= 0 && j + R[j] > i)

        {

            R[i] = min(R[2 \* j - i], j + R[j] - i);

        }

        while (i - R[i] >= 0 && i + R[i] < n && t[i - R[i]] == t[i + R[i]])

        {

            R[i] += 1;

        }

        if (i + R[i] > j + R[j])

        {

            j = i;

        }

    }

    return R;

}

### KMP

struct KMP

{

    string s;

    vector<int> nxt;

    KMP() {}

    KMP(string s)

        : s(s), nxt(s.size()) {}

    void getNext()

    {

        int j = 0;

        for (int i = 1; i < s.size(); i++)

        {

            while (j > 0 && s[i] != s[j])

            {

                j = nxt[j - 1];

            }

            if (s[i] == s[j])

            {

                j++;

            }

            nxt[i] = j;

        }

    }

    // s在t中第一次出现位置

    int find(string t)

    {

        getNext();

        int n = s.size(), m = t.size();

        int j = 0, p = -1;

        for (int i = 0; i < m; i++)

        {

            while (j > 0 && t[i] != s[j])

            {

                j = nxt[j - 1];

            }

            if (t[i] == s[j])

            {

                j++;

            }

            if (j == n)

            {

                p = i - n + 1;

                break;

            }

        }

        return p;

    }

    // s在t中出现的次数

    int count(string t)

    {

        getNext();

        int n = s.size(), m = t.size();

        int j = 0, ans = 0;

        for (int i = 0; i < m; i++)

        {

            while (j > 0 && t[i] != s[j])

            {

                j = nxt[j - 1];

            }

            if (t[i] == s[j])

            {

                j++;

            }

            if (j == n)

            {

                ans++;

                // 如果匹配的位置不可以重叠, 求匹配次数最大值，取消下行注释

                // j = 0;

            }

        }

        return ans;

    }

};

### 字符串哈希

constexpr int mod1 = 1e9 + 123, mod2 = 1e9 + 411;

vector<int> p1, p2;

void init(int N)

{

    p1.resize(N + 1), p2.resize(N + 1);

    p1[0] = p2[0] = 1;

    for (int i = 1; i <= N; i++)

    {

        p1[i] = p1[i - 1] \* 516 % mod1;

        p2[i] = p2[i - 1] \* 516 % mod2;

    }

}

struct Hash

{

    // 不要修改s := " " + s

    string s;

    vector<int> h1, h2;

    Hash() {}

    Hash(string s\_)

    {

        s = s\_;

        int n = s.size();

        h1.resize(n + 1), h2.resize(n + 1);

        for (int i = 1; i <= n; i++)

        {

            h1[i] = (h1[i - 1] \* 516 % mod1 + s[i - 1]) % mod1;

            h2[i] = (h2[i - 1] \* 516 % mod2 + s[i - 1]) % mod2;

        }

    }

    // l和r范围[1, n]

    pair<int, int> get(int l, int r)

    {

        return {(h1[r] + (mod1 - h1[l - 1]) \* p1[r - l + 1]) % mod1, (h2[r] + (mod2 - h2[l - 1]) \* p2[r - l + 1]) % mod2};

    }

};

### Z函数（拓展KMP）

vector<int> zFunction(string s) {

    int n = s.size();

    vector<int> z(n + 1);

    z[0] = n;

    for (int i = 1, j = 1; i < n; i++) {

        z[i] = max(0, min(j + z[j] - i, z[i - j]));

        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {

            z[i]++;

        }

        if (i + z[i] > j + z[j]) {

            j = i;

        }

    }

    return z;

}

/\*

对于一个长度为 n 的字符串 s，定义函数 z[i] 表示 s 和 s[i,n-1]（即以 s[i] 开头的后缀）的最长公共前缀（LCP）的长度，则 z 被称为 s 的 Z 函数。特别地，z[0] = 0。

\*/

inline void Z(char \*s, int n) {

    for (int i = 1; i <= n; i++) z[i] = 0;

    z[1] = n;

    for (int i = 2, l = 0, r = 0; i <= n; i++) {

        if (i <= r) z[i] = min(z[i-l+1], r - i + 1);

        while (i + z[i] <= n && s[i+z[i]] == s[z[i]+1]) ++z[i];

        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;

    }

}

inline void exkmp(char \*s, int n, char \*t, int m) {

    Z(t, m);

    for (int i = 1; i <= n; i++) p[i] = 0;

    for (int i = 1, l = 0, r = 0; i <= n; i++) {

        if (i <= r) p[i] = min(z[i-l+1], r - i + 1);

        while (i + p[i] <= n && s[i+p[i]] == t[p[i]+1]) ++p[i];

        if (i + p[i] - 1 > r) l = i, r = i + p[i] - 1;

    }

}

int main() {

    rds(a, n), rds(b, m);

    exkmp(a, n, b, m);

    ll ans = 0;

    for (int i = 1; i <= m; i++) ans ^= 1ll \* i \* (z[i] + 1);

    print(ans);

    ans = 0;

    for (int i = 1; i <= n; i++) ans ^= 1ll \* i \* (p[i] + 1);

    print(ans);

    return 0;

}

/\*

给定两个字符串 a,b，你要求出两个数组：

b 的 z 函数数组 z，即 b 与 b 的每一个后缀的 LCP 长度。

b 与 a 的每一个后缀的 LCP 长度数组 p。

\*/

### 后缀数组

struct SuffixArray

{

    int n;

    vector<int> sa, rk, lc;

    SuffixArray(const string &s)

    {

        n = s.length();

        sa.resize(n);

        lc.resize(n - 1);

        rk.resize(n);

        iota(sa.begin(), sa.end(), 0);

        sort(sa.begin(), sa.end(), [&](int a, int b)

             { return s[a] < s[b]; });

        rk[sa[0]] = 0;

        for (int i = 1; i < n; ++i)

            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);

        int k = 1;

        vector<int> tmp, cnt(n);

        tmp.reserve(n);

        while (rk[sa[n - 1]] < n - 1)

        {

            tmp.clear();

            for (int i = 0; i < k; ++i)

                tmp.push\_back(n - k + i);

            for (auto i : sa)

                if (i >= k)

                    tmp.push\_back(i - k);

            fill(cnt.begin(), cnt.end(), 0);

            for (int i = 0; i < n; ++i)

                ++cnt[rk[i]];

            for (int i = 1; i < n; ++i)

                cnt[i] += cnt[i - 1];

            for (int i = n - 1; i >= 0; --i)

                sa[--cnt[rk[tmp[i]]]] = tmp[i];

            swap(rk, tmp);

            rk[sa[0]] = 0;

            for (int i = 1; i < n; ++i)

                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);

            k \*= 2;

        }

        for (int i = 0, j = 0; i < n; ++i)

        {

            if (rk[i] == 0)

            {

                j = 0;

            }

            else

            {

                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j];)

                    ++j;

                lc[rk[i] - 1] = j;

            }

        }

    }

};

/\*

sa[i] 表示将所有后缀排序后第 i 小的后缀的编号，也是所说的后缀数组，后文也称编号数组 sa；

rk[i] 表示后缀 i 的排名，是重要的辅助数组，后文也称排名数组 rk。

这两个数组满足性质：sa[rk[i]]=rk[sa[i]]=i。

lc[i] = lcp(sa[i],sa[i-1])

lcp(sa[i],sa[j]) = min(lc[i],...,lc[j])

\*/

### 后缀自动机

#include <bits/stdc++.h>

using namespace std;

#define ll long long

int T;

struct SAM

{

    // edited by piaoyun from some other's code

    // 必须#define int long long

    static const int MAXN = 1000010, MAXS = 28;

    int tot = 1, last = 1, link[MAXN << 1], ch[MAXN << 1][MAXS], len[MAXN << 1], endpos[MAXN << 1];

    // 总点数tot，点的index属于[1-tot]，空串/根为1

    // last为上一次插入的点

    // link为点的parent树父节点 / 最长 出现位置与自己不同 的后缀

    // ch[n][s] 指节点n末尾加字符s所转移到的点

    // len指该节点的串的 最长长度，注意到 最短长度 等于 len[link[n]] + 1 即父节点最长 + 1

    // endpos[n] 参考get\_endpos()的注释

    void clear()

    {

        for (int i = 0; i <= tot; i++)

        {

            link[i] = len[i] = endpos[i] = 0;

            for (int k = 0; k < MAXS; k++)

                ch[i][k] = 0;

        }

        tot = 1;

        last = 1;

    }

    // 延长一个字符，通常为[1-26]

    void extend(int w)

    {

        int p = ++tot, x = last, r, q;

        // TODO

        endpos[p] = ;

        // END

        for (len[last = p] = len[x] + 1; x && !ch[x][w]; x = link[x])

            ch[x][w] = p;

        if (!x)

            link[p] = 1;

        else if (len[x] + 1 == len[q = ch[x][w]])

            link[p] = q;

        else

        {

            link[r = ++tot] = link[q];

            memcpy(ch[r], ch[q], sizeof ch[r]);

            len[r] = len[x] + 1;

            link[p] = link[q] = r;

            for (; x && ch[x][w] == q; x = link[x])

                ch[x][w] = r;

        }

    }

    //\*注意vector占用的空间

    vector<int> p[MAXN << 1]; // 建立parent树，以便从上到下dfs

    void dfs(int u)

    {

        int v;

        for (int i = 0; i < p[u].size(); i++)

        {

            v = p[u][i];

            dfs(v);

            endpos[u] += endpos[v];

        }

    }

    // 注意！在使用该方法前，endpos[]代表每个点作为“终结点”的次数

    // 使用该方法后，endpos[]指在串中出现总次数，即原数组的子树求和

    void get\_endpos()

    {

        for (int i = 1; i <= tot; i++)

            p[i].clear();

        for (int i = 2; i <= tot; i++)

        {

            p[link[i]].push\_back(i); // 建立parent树，以便从上到下dfs

        }

        dfs(1);

        for (int i = 1; i <= tot; i++)

            p[i].clear();

    }

    //\*在您不确定是否有抄写错误时再使用该方法

    //\*必须在输入任何数据前自检，此前的数据会被清空

    static const int STC = 998244353;

    void self\_test()

    {

        clear();

        for (int i = 1; i <= 1000; i++)

            extend(i \* i % 26 + 1);

        int tmp = 107 \* last + 301 \* tot;

        for (int i = 1; i <= tot; i++)

        {

            tmp = (tmp \* 33 + link[i] \* 101 + len[i] \* 97) % STC;

            for (int k = 1; k < MAXS; k++)

                tmp = (tmp + k \* ch[i][k]) % STC;

        }

        assert("stage 1" && tmp == 393281314); // stage1 : 检查建树是否正确

        tmp = 0;

        get\_endpos();

        for (int i = 1; i <= tot; i++)

            tmp = (tmp \* 33 + endpos[i]) % STC;

        assert("stage 2" && tmp == 178417668); // stage2 : 检查endpos计算是否正确，如果您修改了endpos[]的含义则会报错

        cout << "Self Test Passed.Remember to delete this function's use." << endl;

        clear();

    }

    // 调试时可调用

    void debug\_print()

    {

        for (int i = 1; i <= tot; i++)

        {

            cout << "node:" << i << " father:" << link[i] << " endpos:" << endpos[i] << " len:" << len[i] << endl;

        }

    }

    string str;

    void solve()

    {

        // TODO

        // END

    }

} sam;

string tmp;

void prepare()

{

    sam.solve();

}

signed main()

{

    T = 1;

    while (T--)

    {

        prepare();

    }

    return 0;

}

### SAM例题

#include <bits/stdc++.h>

using namespace std;

#define ll long long

int T;

struct SAM

{

    // edited by piaoyun from some other's code

    // 必须#define int long long

    static const int MAXN = 1000010, MAXS = 28;

    int tot = 1, last = 1, link[MAXN << 1], ch[MAXN << 1][MAXS], len[MAXN << 1], endpos[MAXN << 1];

    // 总点数tot，点的index属于[1-tot]，空串/根为1

    // last为上一次插入的点

    // link为点的parent树父节点 / 最长 出现位置与自己不同 的后缀

    // ch[n][s] 指节点n末尾加字符s所转移到的点

    // len指该节点的串的 最长长度，注意到 最短长度 等于 len[link[n]] + 1 即父节点最长 + 1

    // endpos[n] 参考get\_endpos()的注释

    void clear()

    {

        for (int i = 0; i <= tot; i++)

        {

            link[i] = len[i] = endpos[i] = 0;

            for (int k = 0; k < MAXS; k++)

                ch[i][k] = 0;

        }

        tot = 1;

        last = 1;

    }

    // 延长一个字符，通常为[1-26]

    void extend(int w)

    {

        int p = ++tot, x = last, r, q;

        // TODO

        endpos[p] = 1;

        // END

        for (len[last = p] = len[x] + 1; x && !ch[x][w]; x = link[x])

            ch[x][w] = p;

        if (!x)

            link[p] = 1;

        else if (len[x] + 1 == len[q = ch[x][w]])

            link[p] = q;

        else

        {

            link[r = ++tot] = link[q];

            memcpy(ch[r], ch[q], sizeof ch[r]);

            len[r] = len[x] + 1;

            link[p] = link[q] = r;

            for (; x && ch[x][w] == q; x = link[x])

                ch[x][w] = r;

        }

    }

    //\*注意vector占用的空间

    vector<int> p[MAXN << 1]; // 建立parent树，以便从上到下dfs

    void dfs(int u)

    {

        int v;

        for (int i = 0; i < p[u].size(); i++)

        {

            v = p[u][i];

            dfs(v);

            endpos[u] += endpos[v];

        }

    }

    // 注意！在使用该方法前，endpos[]代表每个点作为“终结点”的次数

    // 使用该方法后，endpos[]指在串中出现总次数，即原数组的子树求和

    void get\_endpos()

    {

        for (int i = 1; i <= tot; i++)

            p[i].clear();

        for (int i = 2; i <= tot; i++)

        {

            p[link[i]].push\_back(i); // 建立parent树，以便从上到下dfs

        }

        dfs(1);

        for (int i = 1; i <= tot; i++)

            p[i].clear();

    }

    //\*在您不确定是否有抄写错误时再使用该方法

    //\*必须在输入任何数据前自检，此前的数据会被清空

    static const int STC = 998244353;

    void self\_test()

    {

        clear();

        for (int i = 1; i <= 1000; i++)

            extend(i \* i % 26 + 1);

        int tmp = 107 \* last + 301 \* tot;

        for (int i = 1; i <= tot; i++)

        {

            tmp = (tmp \* 33 + link[i] \* 101 + len[i] \* 97) % STC;

            for (int k = 1; k < MAXS; k++)

                tmp = (tmp + k \* ch[i][k]) % STC;

        }

        assert("stage 1" && tmp == 393281314); // stage1 : 检查建树是否正确

        tmp = 0;

        get\_endpos();

        for (int i = 1; i <= tot; i++)

            tmp = (tmp \* 33 + endpos[i]) % STC;

        assert("stage 2" && tmp == 178417668); // stage2 : 检查endpos计算是否正确，如果您修改了endpos[]的含义则会报错

        cout << "Self Test Passed.Remember to delete this function's use." << endl;

        clear();

    }

    // 调试时可调用

    void debug\_print()

    {

        for (int i = 1; i <= tot; i++)

        {

            cout << "node:" << i << " father:" << link[i] << " endpos:" << endpos[i] << " len:" << len[i] << endl;

        }

    }

    string str;

    void solve()

    {

        // TODO

        string s;

        cin >> s;

        for (auto x : s)

        {

            extend(x - 'a' + 1);

        }

        get\_endpos();

        ll ans = 0;

        for (int i = 1; i <= tot; i++)

        {

            if (endpos[i] > 1)

                ans = max(ans, (ll)endpos[i] \* len[i]);

        }

        cout << ans << endl;

        // END

    }

} sam;

string tmp;

void prepare()

{

    sam.solve();

}

signed main()

{

    T = 1;

    while (T--)

    {

        prepare();

    }

    return 0;

}

/\*

给定一个只包含小写字母的字符串 S。

请你求出 S 的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值。

\*/

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define int long long

int T;

struct SAM

{

    // edited by piaoyun from some other's code

    // 必须#define int long long

    static const int MAXN = 5e5 + 10, MAXS = 28;

    int tot = 1, last = 1, link[MAXN << 1], ch[MAXN << 1][MAXS], len[MAXN << 1], endpos[MAXN << 1], sum[MAXN << 1], vis[MAXN << 1];

    // 总点数tot，点的index属于[1-tot]，空串/根为1

    // last为上一次插入的点

    // link为点的parent树父节点 / 最长 出现位置与自己不同 的后缀

    // ch[n][s] 指节点n末尾加字符s所转移到的点

    // len指该节点的串的 最长长度，注意到 最短长度 等于 len[link[n]] + 1 即父节点最长 + 1

    // endpos[n] 参考get\_endpos()的注释

    void clear()

    {

        for (int i = 0; i <= tot; i++)

        {

            link[i] = len[i] = endpos[i] = 0;

            for (int k = 0; k < MAXS; k++)

                ch[i][k] = 0;

        }

        tot = 1;

        last = 1;

    }

    // 延长一个字符，通常为[1-26]

    void extend(int w)

    {

        int p = ++tot, x = last, r, q;

        // TODO

        endpos[p] = 1;

        // END

        for (len[last = p] = len[x] + 1; x && !ch[x][w]; x = link[x])

            ch[x][w] = p;

        if (!x)

            link[p] = 1;

        else if (len[x] + 1 == len[q = ch[x][w]])

            link[p] = q;

        else

        {

            link[r = ++tot] = link[q];

            memcpy(ch[r], ch[q], sizeof ch[r]);

            len[r] = len[x] + 1;

            link[p] = link[q] = r;

            for (; x && ch[x][w] == q; x = link[x])

                ch[x][w] = r;

        }

    }

    //\*注意vector占用的空间

    vector<int> p[MAXN << 1]; // 建立parent树，以便从上到下dfs

    void dfs(int u)

    {

        int v;

        for (int i = 0; i < p[u].size(); i++)

        {

            v = p[u][i];

            dfs(v);

            endpos[u] += endpos[v];

        }

    }

    // 注意！在使用该方法前，endpos[]代表每个点作为“终结点”的次数

    // 使用该方法后，endpos[]指在串中出现总次数，即原数组的子树求和

    void get\_endpos()

    {

        for (int i = 1; i <= tot; i++)

            p[i].clear();

        for (int i = 2; i <= tot; i++)

        {

            p[link[i]].push\_back(i); // 建立parent树，以便从上到下dfs

        }

        dfs(1);

        for (int i = 1; i <= tot; i++)

            p[i].clear();

    }

    //\*在您不确定是否有抄写错误时再使用该方法

    //\*必须在输入任何数据前自检，此前的数据会被清空

    static const int STC = 998244353;

    void self\_test()

    {

        clear();

        for (int i = 1; i <= 1000; i++)

            extend(i \* i % 26 + 1);

        int tmp = 107 \* last + 301 \* tot;

        for (int i = 1; i <= tot; i++)

        {

            tmp = (tmp \* 33 + link[i] \* 101 + len[i] \* 97) % STC;

            for (int k = 1; k < MAXS; k++)

                tmp = (tmp + k \* ch[i][k]) % STC;

        }

        assert("stage 1" && tmp == 393281314); // stage1 : 检查建树是否正确

        tmp = 0;

        get\_endpos();

        for (int i = 1; i <= tot; i++)

            tmp = (tmp \* 33 + endpos[i]) % STC;

        assert("stage 2" && tmp == 178417668); // stage2 : 检查endpos计算是否正确，如果您修改了endpos[]的含义则会报错

        cout << "Self Test Passed.Remember to delete this function's use." << endl;

        clear();

    }

    // 调试时可调用

    void debug\_print()

    {

        for (int i = 1; i <= tot; i++)

        {

            cout << "node:" << i << " father:" << link[i] << " endpos:" << endpos[i] << " len:" << len[i] << endl;

        }

    }

    int dfs2(int x)

    {

        if (vis[x])

            return sum[x];

        vis[x] = 1;

        for (int i = 1; i <= 26; i++)

        {

            if (!ch[x][i])

                continue;

            sum[x] += dfs2(ch[x][i]);

        }

        return sum[x];

    }

    void find(int x, int k)

    {

        if (k <= endpos[x])

            return;

        k -= endpos[x];

        for (int i = 1; i <= 26; i++)

        {

            if (!ch[x][i])

                continue;

            if (k <= sum[ch[x][i]])

            {

                putchar(i + 'a' - 1);

                find(ch[x][i], k);

                break;

            }

            k -= sum[ch[x][i]];

        }

    }

    string str;

    void solve()

    {

        // TODO

        string s;

        cin >> s;

        int t, k;

        cin >> t >> k;

        for (auto x : s)

        {

            extend(x - 'a' + 1);

        }

        get\_endpos();

        if (t == 0)

        {

            for (int i = 1; i <= tot; i++)

                sum[i] = endpos[i] = 1;

        }

        else

        {

            for (int i = 1; i <= tot; i++)

                sum[i] = endpos[i];

        }

        sum[1] = endpos[1] = 0;

        dfs2(1);

        if (sum[1] < k)

        {

            cout << -1 << endl;

            return;

        }

        find(1, k);

        // END

    }

} sam;

string tmp;

void prepare()

{

    sam.solve();

}

signed main()

{

    T = 1;

    while (T--)

    {

        prepare();

    }

    return 0;

}

/\*

求第k小子串，t=0时不同位置相同子串算一个，t=1时不同位置相同子串算多个

\*/

### AC自动机

struct AhoCorasick

{

    static constexpr int ALPHABET = 26;

    struct Node

    {

        int len;

        int link;

        array<int, ALPHABET> next;

        Node() : link{}, next{} {}

    };

    vector<Node> t;

    AhoCorasick()

    {

        init();

    }

    void init()

    {

        t.assign(2, Node());

        t[0].next.fill(1);

        t[0].len = -1;

    }

    int newNode()

    {

        t.emplace\_back();

        return t.size() - 1;

    }

    int add(const vector<int> &a)

    {

        int p = 1;

        for (auto x : a)

        {

            if (t[p].next[x] == 0)

            {

                t[p].next[x] = newNode();

                t[t[p].next[x]].len = t[p].len + 1;

            }

            p = t[p].next[x];

        }

        return p;

    }

    int add(const string &a, char offset = 'a')

    {

        vector<int> b(a.size());

        for (int i = 0; i < a.size(); i++)

        {

            b[i] = a[i] - offset;

        }

        return add(b);

    }

    void work()

    {

        queue<int> q;

        q.push(1);

        while (!q.empty())

        {

            int x = q.front();

            q.pop();

            for (int i = 0; i < ALPHABET; i++)

            {

                if (t[x].next[i] == 0)

                {

                    t[x].next[i] = t[t[x].link].next[i];

                }

                else

                {

                    t[t[x].next[i]].link = t[t[x].link].next[i];

                    q.push(t[x].next[i]);

                }

            }

        }

    }

    int next(int p, int x)

    {

        return t[p].next[x];

    }

    int next(int p, char c, char offset = 'a')

    {

        return next(p, c - 'a');

    }

    int link(int p)

    {

        return t[p].link;

    }

    int len(int p)

    {

        return t[p].len;

    }

    int size()

    {

        return t.size();

    }

};

### 回文自动机

struct PAM

{

    static constexpr int ALPHABET\_SIZE = 28;

    struct Node

    {

        int len;

        int link;

        int cnt;

        array<int, ALPHABET\_SIZE> next;

        Node() : len{}, link{}, cnt{}, next{} {}

    };

    vector<Node> t;

    int suff;

    string s;

    PAM()

    {

        init();

    }

    void init()

    {

        t.assign(2, Node());

        t[0].len = -1;

        suff = 1;

        s.clear();

    }

    int newNode()

    {

        t.emplace\_back();

        return t.size() - 1;

    }

    bool add(char c, char offset = 'a')

    {

        int pos = s.size();

        s += c;

        int let = c - offset;

        int cur = suff, curlen = 0;

        while (true)

        {

            curlen = t[cur].len;

            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])

                break;

            cur = t[cur].link;

        }

        if (t[cur].next[let])

        {

            suff = t[cur].next[let];

            return false;

        }

        int num = newNode();

        suff = num;

        t[num].len = t[cur].len + 2;

        t[cur].next[let] = num;

        if (t[num].len == 1)

        {

            t[num].link = 1;

            t[num].cnt = 1;

            return true;

        }

        while (true)

        {

            cur = t[cur].link;

            curlen = t[cur].len;

            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])

            {

                t[num].link = t[cur].next[let];

                break;

            }

        }

        t[num].cnt = 1 + t[t[num].link].cnt;

        return true;

    }

};

## 其他

### 多项式

namespace NFTS

{

    int M = 998244353, g = 3;

    vector<int> rev, roots{0, 1};

    int powMod(int x, int n)

    {

        int r(1);

        while (n)

        {

            if (n & 1)

            {

                r = r \* x % M;

            }

            n >>= 1;

            x = x \* x % M;

        }

        return r;

    }

    void dft(vector<int> &a)

    {

        int n = a.size();

        if ((int)rev.size() != n)

        {

            int k = \_\_builtin\_ctz(n) - 1;

            rev.resize(n);

            for (int i = 0; i < n; ++i)

            {

                rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;

            }

        }

        if ((int)roots.size() < n)

        {

            int k = \_\_builtin\_ctz(roots.size());

            roots.resize(n);

            while ((1 << k) < n)

            {

                int e = powMod(g, (M - 1) >> (k + 1));

                for (int i = 1 << (k - 1); i < (1 << k); ++i)

                {

                    roots[2 \* i] = roots[i];

                    roots[2 \* i + 1] = 1LL \* roots[i] \* e % M;

                }

                ++k;

            }

        }

        for (int i = 0; i < n; ++i)

            if (rev[i] < i)

            {

                swap(a[i], a[rev[i]]);

            }

        for (int k = 1; k < n; k \*= 2)

        {

            for (int i = 0; i < n; i += 2 \* k)

            {

                for (int j = 0; j < k; ++j)

                {

                    int u = a[i + j];

                    int v = 1LL \* a[i + j + k] \* roots[k + j] % M;

                    int x = u + v, y = u - v;

                    if (x >= M)

                        x -= M;

                    if (y < 0)

                        y += M;

                    a[i + j] = x;

                    a[i + j + k] = y;

                }

            }

        }

    }

    void idft(vector<int> &a)

    {

        int n = a.size();

        reverse(a.begin() + 1, a.end());

        dft(a);

        int inv = powMod(n, M - 2);

        for (int i = 0; i < n; ++i)

        {

            a[i] = a[i] \* inv % M;

        }

    }

} // namespace NFTS

// 如果需要换模 M，那么就把 M 当做全局变量提出来，NFT 中 g, rev, root 需要重新初始化。

// 如果只需要换模几个常数 M，可使用 template<LL M>（但是不是特别推荐）

class PolyS

{

    void rev()

    {

        reverse(a.begin(), a.end());

    }

public:

    static inline const int M = NFTS::M;

    static inline const int inv2 = (M + 1) / 2;

    vector<int> a;

    PolyS() {}

    PolyS(int x)

    {

        if (x)

            a = {x};

    }

    PolyS(const vector<int> \_a) : a(\_a) {}

    int size() const { return a.size(); }

    int &operator[](int id) { return a[id]; }

    int at(int id) const

    {

        if (id < 0 || id >= (int)a.size())

            return 0;

        return a[id];

    }

    PolyS operator-() const

    {

        auto A = \*this;

        for (auto &x : A.a)

            x = (x == 0 ? 0 : M - x);

        return A;

    }

    PolyS mulXn(int n) const

    {

        auto b = a;

        b.insert(b.begin(), n, 0);

        return PolyS(b);

    }

    PolyS modXn(int n) const

    {

        if (n > size())

            return \*this;

        return PolyS({a.begin(), a.begin() + n});

    }

    PolyS divXn(int n) const

    {

        if (size() <= n)

            return PolyS();

        return PolyS({a.begin() + n, a.end()});

    }

    PolyS &operator+=(const PolyS &rhs)

    {

        if (size() < rhs.size())

            a.resize(rhs.size());

        for (int i = 0; i < rhs.size(); ++i)

        {

            if ((a[i] += rhs.a[i]) >= M)

                a[i] -= M;

        }

        return \*this;

    }

    PolyS &operator-=(const PolyS &rhs)

    {

        if (size() < rhs.size())

            a.resize(rhs.size());

        for (int i = 0; i < rhs.size(); ++i)

        {

            if ((a[i] -= rhs.a[i]) < 0)

                a[i] += M;

        }

        return \*this;

    }

    PolyS &operator\*=(PolyS rhs)

    {

        int n = size(), m = rhs.size(), tot = max(1LL, n + m - 1);

        int sz = 1 << \_\_lg(tot \* 2 - 1);

        a.resize(sz);

        rhs.a.resize(sz);

        NFTS::dft(a);

        NFTS::dft(rhs.a);

        for (int i = 0; i < sz; ++i)

        {

            a[i] = a[i] \* rhs.a[i] % M;

        }

        NFTS::idft(a);

        return \*this;

    }

    PolyS &operator/=(PolyS rhs)

    {

        int n = size(), m = rhs.size();

        if (n < m)

            return (\*this) = PolyS();

        rev();

        rhs.rev();

        (\*this) \*= rhs.inv(n - m + 1);

        a.resize(n - m + 1);

        rev();

        return \*this;

    }

    PolyS &operator%=(PolyS rhs)

    {

        return (\*this) -= (\*this) / rhs \* rhs;

    }

    PolyS operator+(const PolyS &rhs) const

    {

        return PolyS(\*this) += rhs;

    }

    PolyS operator-(const PolyS &rhs) const

    {

        return PolyS(\*this) -= rhs;

    }

    PolyS operator\*(PolyS rhs) const

    {

        return PolyS(\*this) \*= rhs;

    }

    PolyS operator/(PolyS rhs) const

    {

        return PolyS(\*this) /= rhs;

    }

    PolyS operator%(PolyS rhs) const

    {

        return PolyS(\*this) %= rhs;

    }

    PolyS powModPoly(int n, PolyS p)

    {

        PolyS r(1), x(\*this);

        while (n)

        {

            if (n & 1)

                (r \*= x) %= p;

            n >>= 1;

            (x \*= x) %= p;

        }

        return r;

    }

    int inner(const PolyS &rhs)

    {

        int r = 0, n = min(size(), rhs.size());

        for (int i = 0; i < n; ++i)

        {

            r = (r + 1LL \* a[i] \* rhs.a[i]) % M;

        }

        return r;

    }

    PolyS derivation() const

    {

        if (a.empty())

            return PolyS();

        int n = size();

        vector<int> r(n - 1);

        for (int i = 1; i < n; ++i)

            r[i - 1] = 1LL \* a[i] \* i % M;

        return PolyS(r);

    }

    PolyS integral() const

    {

        if (a.empty())

            return PolyS();

        int n = size();

        vector<int> r(n + 1), inv(n + 1, 1);

        for (int i = 2; i <= n; ++i)

            inv[i] = 1LL \* (M - M / i) \* inv[M % i] % M;

        for (int i = 0; i < n; ++i)

            r[i + 1] = 1LL \* a[i] \* inv[i + 1] % M;

        return PolyS(r);

    }

    PolyS inv(int n) const

    {

        assert(a[0] != 0);

        PolyS x(NFTS::powMod(a[0], M - 2));

        int k = 1;

        while (k < n)

        {

            k \*= 2;

            x \*= (PolyS(2) - modXn(k) \* x).modXn(k);

        }

        return x.modXn(n);

    }

    // 需要保证首项为 1

    PolyS log(int n) const

    {

        return (derivation() \* inv(n)).integral().modXn(n);

    }

    // 需要保证首项为 0

    PolyS exp(int n) const

    {

        PolyS x(1);

        int k = 1;

        while (k < n)

        {

            k \*= 2;

            x = (x \* (PolyS(1) - x.log(k) + modXn(k))).modXn(k);

        }

        return x.modXn(n);

    }

    // 需要保证首项为 1，开任意次方可以先 ln 再 exp 实现。

    PolyS sqrt(int n) const

    {

        PolyS x(1);

        int k = 1;

        while (k < n)

        {

            k \*= 2;

            x += modXn(k) \* x.inv(k);

            x = x.modXn(k) \* inv2;

        }

        return x.modXn(n);

    }

    // 减法卷积，也称转置卷积 {\rm MULT}(F(x),G(x))=\sum\_{i\ge0}(\sum\_{j\ge 0}f\_{i+j}g\_j)x^i

    PolyS mulT(PolyS rhs) const

    {

        if (rhs.size() == 0)

            return PolyS();

        int n = rhs.size();

        reverse(rhs.a.begin(), rhs.a.end());

        return ((\*this) \* rhs).divXn(n - 1);

    }

    int eval(int x)

    {

        int r = 0, t = 1;

        for (int i = 0, n = size(); i < n; ++i)

        {

            r = (r + 1LL \* a[i] \* t) % M;

            t = 1LL \* t \* x % M;

        }

        return r;

    }

    // 多点求值新科技：https://jkloverdcoi.github.io/2020/08/04/转置原理及其应用/

    vector<int> evals(vector<int> x) const

    {

        if (size() == 0)

            return vector<int>(x.size());

        int n = x.size();

        vector<int> ans(n);

        vector<PolyS> g(4 \* n);

        function<void(int, int, int)> build = [&](int l, int r, int p)

        {

            if (r - l == 1)

            {

                // g[p] = vector<int>{1, x[i] ? M - x[l] : 0};

                g[p] = PolyS({1, x[l] ? M - x[l] : 0});

            }

            else

            {

                int m = (l + r) / 2;

                build(l, m, 2 \* p);

                build(m, r, 2 \* p + 1);

                g[p] = g[2 \* p] \* g[2 \* p + 1];

            }

        };

        build(0, n, 1);

        function<void(int, int, int, const PolyS &)> solve = [&](int l, int r, int p, const PolyS &f)

        {

            if (r - l == 1)

            {

                ans[l] = f.at(0);

            }

            else

            {

                int m = (l + r) / 2;

                solve(l, m, 2 \* p, f.mulT(g[2 \* p + 1]).modXn(m - l));

                solve(m, r, 2 \* p + 1, f.mulT(g[2 \* p]).modXn(r - m));

            }

        };

        solve(0, n, 1, mulT(g[1].inv(size())).modXn(n));

        return ans;

    }

};

### FFT

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl '\n'

const int N = 1e7 + 10;

const double Pi = acos(-1.0);

struct Complex

{

    double x, y;

    Complex(double xx = 0, double yy = 0) { x = xx, y = yy; }

} a[N], b[N];

Complex operator+(Complex a, Complex b) { return Complex(a.x + b.x, a.y + b.y); }

Complex operator-(Complex a, Complex b) { return Complex(a.x - b.x, a.y - b.y); }

Complex operator\*(Complex a, Complex b) { return Complex(a.x \* b.x - a.y \* b.y, a.x \* b.y + a.y \* b.x); }

int n, m;

int l, r[N];

int limit = 1;

void fft(Complex A[], int type)

{

    for (int i = 0; i < limit; i++)

        if (i < r[i])

            swap(A[i], A[r[i]]);

    for (int mid = 1; mid < limit; mid <<= 1)

    {

        Complex Wn(cos(Pi / mid), type \* sin(Pi / mid));

        for (int R = mid << 1, j = 0; j < limit; j += R)

        {

            Complex w(1, 0);

            for (int k = 0; k < mid; k++, w = w \* Wn)

            {

                Complex x = A[j + k], y = w \* A[j + mid + k];

                A[j + k] = x + y;

                A[j + mid + k] = x - y;

            }

        }

    }

}

signed main()

{

    ios::sync\_with\_stdio(0);

    cin.tie(0), cout.tie(0);

    cin >> n >> m;

    for (int i = 0; i <= n; i++)

        cin >> a[i].x;

    for (int i = 0; i <= m; i++)

        cin >> b[i].x;

    while (limit <= n + m)

        limit <<= 1, l++;

    for (int i = 0; i < limit; i++)

    {

        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));

    }

    fft(a, 1);

    fft(b, 1);

    for (int i = 0; i <= limit; i++)

        a[i] = a[i] \* b[i];

    fft(a, -1);

    for (int i = 0; i <= n + m; i++)

    {

        cout << (int)(a[i].x / limit + 0.5) << " ";

    }

    return 0;

}

### 计算几何

#include <bits/stdc++.h>

using namespace std;

using db = double;

mt19937 eng(time(0));

const db eps = 1e-6;

const db pi = acos(-1);

int sgn(db k)

{

    if (k > eps)

        return 1;

    else if (k < -eps)

        return -1;

    return 0;

}

// -1: < | 0: == | 1: >

int cmp(db k1, db k2) { return sgn(k1 - k2); }

// k3 in [k1, k2]

int inmid(db k1, db k2, db k3) { return sgn(k1 - k3) \* sgn(k2 - k3) <= 0; }

// 点 (x, y)

struct point

{

    db x, y;

    point operator+(const point &k1) const { return (point){k1.x + x, k1.y + y}; }

    point operator-(const point &k1) const { return (point){x - k1.x, y - k1.y}; }

    point operator\*(db k1) const { return (point){x \* k1, y \* k1}; }

    point operator/(db k1) const { return (point){x / k1, y / k1}; }

    int operator==(const point &k1) const { return cmp(x, k1.x) == 0 && cmp(y, k1.y) == 0; }

    // 逆时针旋转 k1 弧度

    point rotate(db k1) { return (point){x \* cos(k1) - y \* sin(k1), x \* sin(k1) + y \* cos(k1)}; }

    // 逆时针旋转 90°

    point rotleft() { return (point){-y, x}; }

    // 优先比较 x 坐标

    bool operator<(const point k1) const

    {

        int a = cmp(x, k1.x);

        if (a == -1)

            return 1;

        else if (a == 1)

            return 0;

        else

            return cmp(y, k1.y) == -1;

    }

    // 模长

    db abs() { return sqrt(x \* x + y \* y); }

    // 模长的平方

    db abs2() { return x \* x + y \* y; }

    // 与点 k1 的距离

    db dis(point k1) { return ((\*this) - k1).abs(); }

    // 化为单位向量, require: abs() > 0

    point unit()

    {

        db w = abs();

        return (point){x / w, y / w};

    }

    // 读入

    void scan()

    {

        double k1, k2;

        scanf("%lf%lf", &k1, &k2);

        x = k1;

        y = k2;

    }

    // 输出

    void print() { printf("%.11lf %.11lf\n", x, y); }

    // 方向角 atan2(y, x)

    db getw() { return atan2(y, x); }

    // 将向量对称到 (-pi, pi] 半平面中

    point getdel()

    {

        if (sgn(x) == -1 || (sgn(x) == 0 && sgn(y) == -1))

            return (\*this) \* (-1);

        else

            return (\*this);

    }

    // (-pi, 0] -> 0, (0, pi] -> 1

    int getP() const { return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) == -1); }

};

/\* 点与线段的位置关系及交点 \*/

// k3 在 矩形 [k1, k2] 中

int inmid(point k1, point k2, point k3) { return inmid(k1.x, k2.x, k3.x) && inmid(k1.y, k2.y, k3.y); }

db cross(point k1, point k2) { return k1.x \* k2.y - k1.y \* k2.x; }

db dot(point k1, point k2) { return k1.x \* k2.x + k1.y \* k2.y; }

// 从 k1 转到 k2 的方向角

db rad(point k1, point k2) { return atan2(cross(k1, k2), dot(k1, k2)); }

// k1 k2 k3 逆时针 1 顺时针 -1 否则 0

int clockwise(point k1, point k2, point k3)

{

    return sgn(cross(k2 - k1, k3 - k1));

}

// 按 (-pi, pi] 顺序进行极角排序

int cmpangle(point k1, point k2)

{

    return k1.getP() < k2.getP() || (k1.getP() == k2.getP() && sgn(cross(k1, k2)) > 0);

}

// 点 q 在线段 k1, k2 上

int onS(point k1, point k2, point q) { return inmid(k1, k2, q) && sgn(cross(k1 - q, k2 - k1)) == 0; }

// q 到直线 k1,k2 的投影

point proj(point k1, point k2, point q)

{

    point k = k2 - k1;

    return k1 + k \* (dot(q - k1, k) / k.abs2());

}

// q 关于直线 k1,k2 的镜像

point reflect(point k1, point k2, point q) { return proj(k1, k2, q) \* 2 - q; }

// 判断 直线 (k1, k2) 和 直线 (k3, k4) 是否相交

int checkLL(point k1, point k2, point k3, point k4)

{

    return cmp(cross(k3 - k1, k4 - k1), cross(k3 - k2, k4 - k2)) != 0;

}

// 求直线 (k1, k2) 和 直线 (k3, k4) 的交点

point getLL(point k1, point k2, point k3, point k4)

{

    db w1 = cross(k1 - k3, k4 - k3), w2 = cross(k4 - k3, k2 - k3);

    return (k1 \* w2 + k2 \* w1) / (w1 + w2);

}

int intersect(db l1, db r1, db l2, db r2)

{

    if (l1 > r1)

        swap(l1, r1);

    if (l2 > r2)

        swap(l2, r2);

    return cmp(r1, l2) != -1 && cmp(r2, l1) != -1;

}

// 线段与线段相交判断（非严格相交）

int checkSS(point k1, point k2, point k3, point k4)

{

    return intersect(k1.x, k2.x, k3.x, k4.x) && intersect(k1.y, k2.y, k3.y, k4.y) &&

           sgn(cross(k3 - k1, k4 - k1)) \* sgn(cross(k3 - k2, k4 - k2)) <= 0 &&

           sgn(cross(k1 - k3, k2 - k3)) \* sgn(cross(k1 - k4, k2 - k4)) <= 0;

}

// 点 q 到 直线 (k1, k2) 的距离

db disLP(point k1, point k2, point q)

{

    return fabs(cross(k1 - q, k2 - q)) / k1.dis(k2);

}

// 点 q 到 线段 (k1, k2) 的距离

db disSP(point k1, point k2, point q)

{

    point k3 = proj(k1, k2, q);

    if (inmid(k1, k2, k3))

        return q.dis(k3);

    else

        return min(q.dis(k1), q.dis(k2));

}

// 线段 (k1, k2) 到 线段 (k3, k4) 的距离

db disSS(point k1, point k2, point k3, point k4)

{

    if (checkSS(k1, k2, k3, k4))

        return 0;

    else

        return min(min(disSP(k1, k2, k3), disSP(k1, k2, k4)), min(disSP(k3, k4, k1), disSP(k3, k4, k2)));

}

/\* 直线与半平面交 \*/

// 直线 p[0] -> p[1]

struct line

{

    point p[2];

    line() {}

    line(point k1, point k2)

    {

        p[0] = k1;

        p[1] = k2;

    }

    point &operator[](int k) { return p[k]; }

    // k 严格位于直线左侧 / 半平面 p[0] -> p[1]

    int include(point k) { return sgn(cross(p[1] - p[0], k - p[0])) > 0; }

    // 方向向量

    point dir() { return p[1] - p[0]; }

    // 向左平移 d, 默认为 eps

    line push(db d = eps)

    {

        point delta = (p[1] - p[0]).rotleft().unit() \* d;

        return {p[0] + delta, p[1] + delta};

    }

};

// 直线与直线交点

point getLL(line k1, line k2) { return getLL(k1[0], k1[1], k2[0], k2[1]); }

// 两直线平行

int parallel(line k1, line k2) { return sgn(cross(k1.dir(), k2.dir())) == 0; }

// 平行且同向

int sameDir(line k1, line k2) { return parallel(k1, k2) && sgn(dot(k1.dir(), k2.dir())) == 1; }

// 同向则左侧优先，否则按极角排序，用于半平面交

int operator<(line k1, line k2)

{

    if (sameDir(k1, k2))

        return k2.include(k1[0]);

    return cmpangle(k1.dir(), k2.dir());

}

// k3 (半平面) 包含 k1,k2 的交点, 用于半平面交

int checkpos(line k1, line k2, line k3) { return k3.include(getLL(k1, k2)); }

// 求半平面交, 半平面是逆时针方向, 输出按照逆时针

vector<line> getHL(vector<line> L)

{

    sort(L.begin(), L.end());

    deque<line> q;

    for (int i = 0; i < (int)L.size(); i++)

    {

        if (i && sameDir(L[i], L[i - 1]))

            continue;

        while (q.size() > 1 && !checkpos(q[q.size() - 2], q[q.size() - 1], L[i]))

            q.pop\_back();

        while (q.size() > 1 && !checkpos(q[1], q[0], L[i]))

            q.pop\_front();

        q.push\_back(L[i]);

    }

    while (q.size() > 2 && !checkpos(q[q.size() - 2], q[q.size() - 1], q[0]))

        q.pop\_back();

    while (q.size() > 2 && !checkpos(q[1], q[0], q[q.size() - 1]))

        q.pop\_front();

    vector<line> ans;

    for (int i = 0; i < q.size(); i++)

        ans.push\_back(q[i]);

    return ans;

}

db closepoint(vector<point> &A, int l, int r)

{ // 最近点对 , 先要按照 x 坐标排序

    if (r - l <= 5)

    {

        db ans = 1e20;

        for (int i = l; i <= r; i++)

            for (int j = i + 1; j <= r; j++)

                ans = min(ans, A[i].dis(A[j]));

        return ans;

    }

    int mid = l + r >> 1;

    db ans = min(closepoint(A, l, mid), closepoint(A, mid + 1, r));

    vector<point> B;

    for (int i = l; i <= r; i++)

        if (abs(A[i].x - A[mid].x) <= ans)

            B.push\_back(A[i]);

    sort(B.begin(), B.end(), [](point k1, point k2)

         { return k1.y < k2.y; });

    for (int i = 0; i < B.size(); i++)

        for (int j = i + 1; j < B.size() && B[j].y - B[i].y < ans; j++)

            ans = min(ans, B[i].dis(B[j]));

    return ans;

}

/\* 圆基础操作 \*/

// 圆 (o, r)

struct circle

{

    point o;

    db r;

    void scan()

    {

        o.scan();

        scanf("%lf", &r);

    }

    int inside(point k) { return cmp(r, o.dis(k)); }

};

// 两圆位置关系（两圆公切线数量）

int checkposCC(circle k1, circle k2)

{

    if (cmp(k1.r, k2.r) == -1)

        swap(k1, k2);

    db dis = k1.o.dis(k2.o);

    int w1 = cmp(dis, k1.r + k2.r), w2 = cmp(dis, k1.r - k2.r);

    if (w1 > 0)

        return 4;

    else if (w1 == 0)

        return 3;

    else if (w2 > 0)

        return 2;

    else if (w2 == 0)

        return 1;

    else

        return 0;

}

// 直线与圆交点，沿 k2->k3 方向给出, 相切给出两个

vector<point> getCL(circle k1, point k2, point k3)

{

    point k = proj(k2, k3, k1.o);

    db d = k1.r \* k1.r - (k - k1.o).abs2();

    if (sgn(d) == -1)

        return {};

    point del = (k3 - k2).unit() \* sqrt(max((db)0.0, d));

    return {k - del, k + del};

}

// 两圆交点，沿圆 k1 逆时针给出, 相切给出两个

vector<point> getCC(circle k1, circle k2)

{

    int pd = checkposCC(k1, k2);

    if (pd == 0 || pd == 4)

        return {};

    db a = (k2.o - k1.o).abs2(), cosA = (k1.r \* k1.r + a - k2.r \* k2.r) / (2 \* k1.r \* sqrt(max(a, (db)0.0)));

    db b = k1.r \* cosA, c = sqrt(max((db)0.0, k1.r \* k1.r - b \* b));

    point k = (k2.o - k1.o).unit(), m = k1.o + k \* b, del = k.rotleft() \* c;

    return {m - del, m + del};

}

// 点到圆的切点,沿圆 k1 逆时针给出, 注意未判位置关系!!

vector<point> TangentCP(circle k1, point k2)

{

    db a = (k2 - k1.o).abs(), b = k1.r \* k1.r / a, c = sqrt(max((db)0.0, k1.r \* k1.r - b \* b));

    point k = (k2 - k1.o).unit(), m = k1.o + k \* b, del = k.rotleft() \* c;

    return {m - del, m + del};

}

// 外公切线

vector<line> TangentoutCC(circle k1, circle k2)

{

    int pd = checkposCC(k1, k2);

    if (pd == 0)

        return {};

    if (pd == 1)

    {

        point k = getCC(k1, k2)[0];

        return {(line){k, k}};

    }

    if (cmp(k1.r, k2.r) == 0)

    {

        point del = (k2.o - k1.o).unit().rotleft().getdel();

        return {(line){k1.o - del \* k1.r, k2.o - del \* k2.r}, (line){k1.o + del \* k1.r, k2.o + del \* k2.r}};

    }

    else

    {

        point p = (k2.o \* k1.r - k1.o \* k2.r) / (k1.r - k2.r);

        vector<point> A = TangentCP(k1, p), B = TangentCP(k2, p);

        vector<line> ans;

        for (int i = 0; i < A.size(); i++)

            ans.push\_back((line){A[i], B[i]});

        return ans;

    }

}

// 内公切线

vector<line> TangentinCC(circle k1, circle k2)

{

    int pd = checkposCC(k1, k2);

    if (pd <= 2)

        return {};

    if (pd == 3)

    {

        point k = getCC(k1, k2)[0];

        return {(line){k, k}};

    }

    point p = (k2.o \* k1.r + k1.o \* k2.r) / (k1.r + k2.r);

    vector<point> A = TangentCP(k1, p), B = TangentCP(k2, p);

    vector<line> ans;

    for (int i = 0; i < A.size(); i++)

        ans.push\_back((line){A[i], B[i]});

    return ans;

}

// 所有公切线

vector<line> TangentCC(circle k1, circle k2)

{

    int flag = 0;

    if (k1.r < k2.r)

        swap(k1, k2), flag = 1;

    vector<line> A = TangentoutCC(k1, k2), B = TangentinCC(k1, k2);

    for (line k : B)

        A.push\_back(k);

    if (flag)

        for (line &k : A)

            swap(k[0], k[1]);

    return A;

}

// 圆 k1 与三角形 k2 k3 k1.o 的有向面积交

db getarea(circle k1, point k2, point k3)

{

    point k = k1.o;

    k1.o = k1.o - k;

    k2 = k2 - k;

    k3 = k3 - k;

    int pd1 = k1.inside(k2), pd2 = k1.inside(k3);

    vector<point> A = getCL(k1, k2, k3);

    if (pd1 >= 0)

    {

        if (pd2 >= 0)

            return cross(k2, k3) / 2;

        return k1.r \* k1.r \* rad(A[1], k3) / 2 + cross(k2, A[1]) / 2;

    }

    else if (pd2 >= 0)

    {

        return k1.r \* k1.r \* rad(k2, A[0]) / 2 + cross(A[0], k3) / 2;

    }

    else

    {

        int pd = cmp(k1.r, disSP(k2, k3, k1.o));

        if (pd <= 0)

            return k1.r \* k1.r \* rad(k2, k3) / 2;

        return cross(A[0], A[1]) / 2 + k1.r \* k1.r \* (rad(k2, A[0]) + rad(A[1], k3)) / 2;

    }

}

// 多边形与圆面积交

db getarea(vector<point> A, circle c)

{

    int n = A.size();

    if (n <= 2)

        return 0.0;

    A.push\_back(A[0]);

    db res = 0.0;

    for (int i = 0; i < n; i++)

    {

        point k1 = A[i], k2 = A[i + 1];

        res += getarea(c, k1, k2);

    }

    return fabs(res);

}

// 三角形外接圆

circle getcircle(point k1, point k2, point k3)

{

    db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 \* a1 + b1 \* b1) / 2;

    db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 \* a2 + b2 \* b2) / 2;

    db d = a1 \* b2 - a2 \* b1;

    point o = (point){k1.x + (c1 \* b2 - c2 \* b1) / d, k1.y + (a1 \* c2 - a2 \* c1) / d};

    return (circle){o, k1.dis(o)};

}

// 最小圆覆盖

circle getScircle(vector<point> A)

{

    shuffle(A.begin(), A.end(), eng);

    circle ans = (circle){A[0], 0};

    for (int i = 1; i < A.size(); i++)

        if (ans.inside(A[i]) == -1)

        {

            ans = (circle){A[i], 0};

            for (int j = 0; j < i; j++)

                if (ans.inside(A[j]) == -1)

                {

                    ans.o = (A[i] + A[j]) / 2;

                    ans.r = ans.o.dis(A[i]);

                    for (int k = 0; k < j; k++)

                        if (ans.inside(A[k]) == -1)

                            ans = getcircle(A[i], A[j], A[k]);

                }

        }

    return ans;

}

/\* 多边形 \*/

// 多边形有向面积

db area(vector<point> A)

{

    db ans = 0;

    for (int i = 0; i < A.size(); i++)

        ans += cross(A[i], A[(i + 1) % A.size()]);

    return ans / 2;

}

// 判断是否为逆时针凸包

int checkconvex(vector<point> A)

{

    int n = A.size();

    A.push\_back(A[0]);

    A.push\_back(A[1]);

    for (int i = 0; i < n; i++)

        if (sgn(cross(A[i + 1] - A[i], A[i + 2] - A[i])) == -1)

            return 0;

    return 1;

}

// 点与简单多边形位置关系：2 内部 1 边界 0 外部

int contain(vector<point> A, point q)

{

    int pd = 0;

    A.push\_back(A[0]);

    for (int i = 1; i < A.size(); i++)

    {

        point u = A[i - 1], v = A[i];

        if (onS(u, v, q))

            return 1;

        if (cmp(u.y, v.y) > 0)

            swap(u, v);

        if (cmp(u.y, q.y) >= 0 || cmp(v.y, q.y) < 0)

            continue;

        if (sgn(cross(u - v, q - v)) < 0)

            pd ^= 1;

    }

    return pd << 1;

}

// flag=0 不严格 flag=1 严格

vector<point> ConvexHull(vector<point> A, int flag = 1)

{

    int n = A.size();

    if (n == 1)

        return A;

    if (n == 2)

    {

        if (A[0] == A[1])

            return {A[0]};

        else

            return A;

    }

    vector<point> ans(n \* 2);

    sort(A.begin(), A.end());

    int now = -1;

    for (int i = 0; i < A.size(); i++)

    {

        while (now > 0 && sgn(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) < flag)

            now--;

        ans[++now] = A[i];

    }

    int pre = now;

    for (int i = n - 2; i >= 0; i--)

    {

        while (now > pre && sgn(cross(ans[now] - ans[now - 1], A[i] - ans[now - 1])) < flag)

            now--;

        ans[++now] = A[i];

    }

    ans.resize(now);

    return ans;

}

// 凸包直径

db convexDiameter(vector<point> A)

{

    int now = 0, n = A.size();

    db ans = 0;

    for (int i = 0; i < A.size(); i++)

    {

        now = max(now, i);

        while (1)

        {

            db k1 = A[i].dis(A[now % n]), k2 = A[i].dis(A[(now + 1) % n]);

            ans = max(ans, max(k1, k2));

            if (k2 > k1)

                now++;

            else

                break;

        }

    }

    return ans;

}

// 直线切凸包，保留 k1,k2,p 逆时针的所有点

vector<point> convexcut(vector<point> A, point k1, point k2)

{

    int n = A.size();

    A.push\_back(A[0]);

    vector<point> ans;

    for (int i = 0; i < n; i++)

    {

        int w1 = clockwise(k1, k2, A[i]), w2 = clockwise(k1, k2, A[i + 1]);

        if (w1 >= 0)

            ans.push\_back(A[i]);

        if (w1 \* w2 < 0)

            ans.push\_back(getLL(k1, k2, A[i], A[i + 1]));

    }

    return ans;

}

// 多边形 A 和 直线 (线段) k1->k2 严格相交, 注释部分为线段

int checkPoS(vector<point> A, point k1, point k2)

{

    struct ins

    {

        point m, u, v;

        int operator<(const ins &k) const { return m < k.m; }

    };

    vector<ins> B;

    // if (contain(A,k1)==2||contain(A,k2)==2) return 1;

    vector<point> poly = A;

    A.push\_back(A[0]);

    for (int i = 1; i < A.size(); i++)

        if (checkLL(A[i - 1], A[i], k1, k2))

        {

            point m = getLL(A[i - 1], A[i], k1, k2);

            if (inmid(A[i - 1], A[i], m) /\*&&inmid(k1,k2,m)\*/)

                B.push\_back((ins){m, A[i - 1], A[i]});

        }

    if (B.size() == 0)

        return 0;

    sort(B.begin(), B.end());

    int now = 1;

    while (now < B.size() && B[now].m == B[0].m)

        now++;

    if (now == B.size())

        return 0;

    int flag = contain(poly, (B[0].m + B[now].m) / 2);

    if (flag == 2)

        return 1;

    point d = B[now].m - B[0].m;

    for (int i = now; i < B.size(); i++)

    {

        if (!(B[i].m == B[i - 1].m) && flag == 2)

            return 1;

        int tag = sgn(cross(B[i].v - B[i].u, B[i].m + d - B[i].u));

        if (B[i].m == B[i].u || B[i].m == B[i].v)

            flag += tag;

        else

            flag += tag \* 2;

    }

    // return 0;

    return flag == 2;

}

int checkinp(point r, point l, point m)

{

    if (cmpangle(l, r))

    {

        return cmpangle(l, m) && cmpangle(m, r);

    }

    return cmpangle(l, m) || cmpangle(m, r);

}

// 快速检查线段是否和多边形严格相交

int checkPosFast(vector<point> A, point k1, point k2)

{

    if (contain(A, k1) == 2 || contain(A, k2) == 2)

        return 1;

    if (k1 == k2)

        return 0;

    A.push\_back(A[0]);

    A.push\_back(A[1]);

    for (int i = 1; i + 1 < A.size(); i++)

        if (checkLL(A[i - 1], A[i], k1, k2))

        {

            point now = getLL(A[i - 1], A[i], k1, k2);

            if (inmid(A[i - 1], A[i], now) == 0 || inmid(k1, k2, now) == 0)

                continue;

            if (now == A[i])

            {

                if (A[i] == k2)

                    continue;

                point pre = A[i - 1], ne = A[i + 1];

                if (checkinp(pre - now, ne - now, k2 - now))

                    return 1;

            }

            else if (now == k1)

            {

                if (k1 == A[i - 1] || k1 == A[i])

                    continue;

                if (checkinp(A[i - 1] - k1, A[i] - k1, k2 - k1))

                    return 1;

            }

            else if (now == k2 || now == A[i - 1])

                continue;

            else

                return 1;

        }

    return 0;

}

/\* 普通凸包中的二分 \*/

// 求经过点 x 切凸包 A 的两个切点，返回下标。方向：A 上 [fi, se] 为点 x 能看到的区域。

// 需要保证 x 严格在凸包 A 外侧，A 的点数 >= 3

// 需要保证 A 是严格凸包，即无三点共线

pair<int, int> getTangentCoP(const vector<point> &A, point x)

{

    int sz = A.size();

    assert(sz >= 3);

    int res[2];

    int flag = 1;

    if (clockwise(A[sz - 1], A[0], x) == -1)

        flag = -1;

    int l = 0, r = sz - 1, ans = 0;

    while (l < r)

    {

        int mid = ((l + r) >> 1);

        if (clockwise(A[mid], A[mid + 1], x) == flag && clockwise(A[0], A[mid + 1], x) == flag)

            ans = mid + 1, l = mid + 1;

        else

            r = mid;

    }

    res[0] = ans;

    l = ans, r = sz - 1, ans = sz - 1;

    while (l < r)

    {

        int mid = ((l + r) >> 1);

        if (clockwise(A[mid], A[mid + 1], x) == flag)

            ans = mid, r = mid;

        else

            l = mid + 1;

    }

    res[1] = ans;

    if (flag == -1)

        swap(res[0], res[1]);

    return {res[0], res[1]};

}

// 判断点是否在凸多边形 A 内部，flag = 1 严格，0 不严格

bool containCoP(const vector<point> &A, point x, int flag = 1)

{

    int sz = A.size();

    assert(sz >= 3);

    if (!flag && (onS(A[0], A[1], x) || onS(A[sz - 1], A[0], x)))

        return 1;

    if (!(clockwise(A[0], A[1], x) == 1 && clockwise(A[sz - 1], A[0], x) == 1))

        return 0;

    int l = 1, r = sz - 1, ans = 1;

    while (l < r)

    {

        int mid = l + r >> 1;

        if (clockwise(A[0], A[mid], x) == 1)

            ans = mid, l = mid + 1;

        else

            r = mid;

    }

    return clockwise(A[ans], A[ans + 1], x) >= flag;

}

/\* 上下凸包中的二分 \*/

// 拆分凸包成上下凸壳 凸包尽量都随机旋转一个角度来避免出现相同横坐标

// 尽量特判只有一个点的情况 凸包逆时针

void getUDP(vector<point> A, vector<point> &U, vector<point> &D)

{

    db l = 1e100, r = -1e100;

    for (int i = 0; i < A.size(); i++)

        l = min(l, A[i].x), r = max(r, A[i].x);

    int wherel, wherer;

    for (int i = 0; i < A.size(); i++)

        if (cmp(A[i].x, l) == 0)

            wherel = i;

    for (int i = A.size(); i; i--)

        if (cmp(A[i - 1].x, r) == 0)

            wherer = i - 1;

    U.clear();

    D.clear();

    int now = wherel;

    while (1)

    {

        D.push\_back(A[now]);

        if (now == wherer)

            break;

        now++;

        if (now >= A.size())

            now = 0;

    }

    now = wherel;

    while (1)

    {

        U.push\_back(A[now]);

        if (now == wherer)

            break;

        now--;

        if (now < 0)

            now = A.size() - 1;

    }

}

// 需要保证凸包点数大于等于 3, 2 内部 ,1 边界 ,0 外部

int containCoP(const vector<point> &U, const vector<point> &D, point k)

{

    db lx = U[0].x, rx = U[U.size() - 1].x;

    if (k == U[0] || k == U[U.size() - 1])

        return 1;

    if (cmp(k.x, lx) == -1 || cmp(k.x, rx) == 1)

        return 0;

    int where1 = lower\_bound(U.begin(), U.end(), (point){k.x, -1e100}) - U.begin();

    int where2 = lower\_bound(D.begin(), D.end(), (point){k.x, -1e100}) - D.begin();

    int w1 = clockwise(U[where1 - 1], U[where1], k), w2 = clockwise(D[where2 - 1], D[where2], k);

    if (w1 == 1 || w2 == -1)

        return 0;

    else if (w1 == 0 || w2 == 0)

        return 1;

    return 2;

}

// d 是方向 , 输出上方切点和下方切点

pair<point, point> getTangentCow(const vector<point> &U, const vector<point> &D, point d)

{

    if (sgn(d.x) < 0 || (sgn(d.x) == 0 && sgn(d.y) < 0))

        d = d \* (-1);

    point whereU, whereD;

    if (sgn(d.x) == 0)

        return {U[0], U[U.size() - 1]};

    int l = 0, r = U.size() - 1, ans = 0;

    while (l < r)

    {

        int mid = l + r >> 1;

        if (sgn(cross(U[mid + 1] - U[mid], d)) <= 0)

            l = mid + 1, ans = mid + 1;

        else

            r = mid;

    }

    whereU = U[ans];

    l = 0, r = D.size() - 1, ans = 0;

    while (l < r)

    {

        int mid = l + r >> 1;

        if (sgn(cross(D[mid + 1] - D[mid], d)) >= 0)

            l = mid + 1, ans = mid + 1;

        else

            r = mid;

    }

    whereD = D[ans];

    return {whereU, whereD};

}

// 先检查 contain, 逆时针给出

pair<point, point> getTangentCoP(const vector<point> &U, const vector<point> &D, point k)

{

    db lx = U[0].x, rx = U[U.size() - 1].x;

    if (k.x < lx)

    {

        int l = 0, r = U.size() - 1, ans = U.size() - 1;

        while (l < r)

        {

            int mid = l + r >> 1;

            if (clockwise(k, U[mid], U[mid + 1]) == 1)

                l = mid + 1;

            else

                ans = mid, r = mid;

        }

        point w1 = U[ans];

        l = 0, r = D.size() - 1, ans = D.size() - 1;

        while (l < r)

        {

            int mid = l + r >> 1;

            if (clockwise(k, D[mid], D[mid + 1]) == -1)

                l = mid + 1;

            else

                ans = mid, r = mid;

        }

        point w2 = D[ans];

        return {w1, w2};

    }

    else if (k.x > rx)

    {

        int l = 1, r = U.size(), ans = 0;

        while (l < r)

        {

            int mid = l + r >> 1;

            if (clockwise(k, U[mid], U[mid - 1]) == -1)

                r = mid;

            else

                ans = mid, l = mid + 1;

        }

        point w1 = U[ans];

        l = 1, r = D.size(), ans = 0;

        while (l < r)

        {

            int mid = l + r >> 1;

            if (clockwise(k, D[mid], D[mid - 1]) == 1)

                r = mid;

            else

                ans = mid, l = mid + 1;

        }

        point w2 = D[ans];

        return {w2, w1};

    }

    else

    {

        int where1 = lower\_bound(U.begin(), U.end(), (point){k.x, -1e100}) - U.begin();

        int where2 = lower\_bound(D.begin(), D.end(), (point){k.x, -1e100}) - D.begin();

        if ((k.x == lx && k.y > U[0].y) || (where1 && clockwise(U[where1 - 1], U[where1], k) == 1))

        {

            int l = 1, r = where1 + 1, ans = 0;

            while (l < r)

            {

                int mid = l + r >> 1;

                if (clockwise(k, U[mid], U[mid - 1]) == 1)

                    ans = mid, l = mid + 1;

                else

                    r = mid;

            }

            point w1 = U[ans];

            l = where1, r = U.size() - 1, ans = U.size() - 1;

            while (l < r)

            {

                int mid = l + r >> 1;

                if (clockwise(k, U[mid], U[mid + 1]) == 1)

                    l = mid + 1;

                else

                    ans = mid, r = mid;

            }

            point w2 = U[ans];

            return {w2, w1};

        }

        else

        {

            int l = 1, r = where2 + 1, ans = 0;

            while (l < r)

            {

                int mid = l + r >> 1;

                if (clockwise(k, D[mid], D[mid - 1]) == -1)

                    ans = mid, l = mid + 1;

                else

                    r = mid;

            }

            point w1 = D[ans];

            l = where2, r = D.size() - 1, ans = D.size() - 1;

            while (l < r)

            {

                int mid = l + r >> 1;

                if (clockwise(k, D[mid], D[mid + 1]) == -1)

                    l = mid + 1;

                else

                    ans = mid, r = mid;

            }

            point w2 = D[ans];

            return {w1, w2};

        }

    }

}

// 三维计算几何

struct P3

{

    db x, y, z;

    P3 operator+(P3 k1) { return (P3){x + k1.x, y + k1.y, z + k1.z}; }

    P3 operator-(P3 k1) { return (P3){x - k1.x, y - k1.y, z - k1.z}; }

    P3 operator\*(db k1) { return (P3){x \* k1, y \* k1, z \* k1}; }

    P3 operator/(db k1) { return (P3){x / k1, y / k1, z / k1}; }

    db abs2() { return x \* x + y \* y + z \* z; }

    db abs() { return sqrt(x \* x + y \* y + z \* z); }

    P3 unit() { return (\*this) / abs(); }

    int operator<(const P3 k1) const

    {

        if (cmp(x, k1.x) != 0)

            return x < k1.x;

        if (cmp(y, k1.y) != 0)

            return y < k1.y;

        return cmp(z, k1.z) == -1;

    }

    int operator==(const P3 k1)

    {

        return cmp(x, k1.x) == 0 && cmp(y, k1.y) == 0 && cmp(z, k1.z) == 0;

    }

    void scan()

    {

        double k1, k2, k3;

        scanf("%lf%lf%lf", &k1, &k2, &k3);

        x = k1;

        y = k2;

        z = k3;

    }

};

P3 cross(P3 k1, P3 k2) { return (P3){k1.y \* k2.z - k1.z \* k2.y, k1.z \* k2.x - k1.x \* k2.z, k1.x \* k2.y - k1.y \* k2.x}; }

db dot(P3 k1, P3 k2) { return k1.x \* k2.x + k1.y \* k2.y + k1.z \* k2.z; }

// p=(3,4,5),l=(13,19,21),theta=85 ans=(2.83,4.62,1.77)

P3 turn3D(db k1, P3 l, P3 p)

{

    l = l.unit();

    P3 ans;

    db c = cos(k1), s = sin(k1);

    ans.x = p.x \* (l.x \* l.x \* (1 - c) + c) + p.y \* (l.x \* l.y \* (1 - c) - l.z \* s) + p.z \* (l.x \* l.z \* (1 - c) + l.y \* s);

    ans.y = p.x \* (l.x \* l.y \* (1 - c) + l.z \* s) + p.y \* (l.y \* l.y \* (1 - c) + c) + p.z \* (l.y \* l.z \* (1 - c) - l.x \* s);

    ans.z = p.x \* (l.x \* l.z \* (1 - c) - l.y \* s) + p.y \* (l.y \* l.z \* (1 - c) + l.x \* s) + p.z \* (l.x \* l.x \* (1 - c) + c);

    return ans;

}

typedef vector<P3> VP;

typedef vector<VP> VVP;

db Acos(db x) { return acos(max(-(db)1, min(x, (db)1))); }

// 球面距离 , 圆心原点 , 半径 1

db Odist(P3 a, P3 b)

{

    db r = Acos(dot(a, b));

    return r;

}

db r;

P3 rnd;

vector<db> solve(db a, db b, db c)

{

    db r = sqrt(a \* a + b \* b), th = atan2(b, a);

    if (cmp(c, -r) == -1)

        return {0};

    else if (cmp(r, c) <= 0)

        return {1};

    else

    {

        db tr = pi - Acos(c / r);

        return {th + pi - tr, th + pi + tr};

    }

}

vector<db> jiao(P3 a, P3 b)

{

    // dot(rd+x\*cos(t)+y\*sin(t),b) >= cos(r)

    if (cmp(Odist(a, b), 2 \* r) > 0)

        return {0};

    P3 rd = a \* cos(r), z = a.unit(), y = cross(z, rnd).unit(), x = cross(y, z).unit();

    vector<db> ret = solve(-(dot(x, b) \* sin(r)), -(dot(y, b) \* sin(r)), -(cos(r) - dot(rd, b)));

    return ret;

}

db norm(db x, db l = 0, db r = 2 \* pi)

{ // change x into [l,r)

    while (cmp(x, l) == -1)

        x += (r - l);

    while (cmp(x, r) >= 0)

        x -= (r - l);

    return x;

}

db disLP(P3 k1, P3 k2, P3 q)

{

    return (cross(k2 - k1, q - k1)).abs() / (k2 - k1).abs();

}

db disLL(P3 k1, P3 k2, P3 k3, P3 k4)

{

    P3 dir = cross(k2 - k1, k4 - k3);

    if (sgn(dir.abs()) == 0)

        return disLP(k1, k2, k3);

    return fabs(dot(dir.unit(), k1 - k2));

}

VP getFL(P3 p, P3 dir, P3 k1, P3 k2)

{

    db a = dot(k2 - p, dir), b = dot(k1 - p, dir), d = a - b;

    if (sgn(fabs(d)) == 0)

        return {};

    return {(k1 \* a - k2 \* b) / d};

}

VP getFF(P3 p1, P3 dir1, P3 p2, P3 dir2)

{ // 返回一条线

    P3 e = cross(dir1, dir2), v = cross(dir1, e);

    db d = dot(dir2, v);

    if (sgn(abs(d)) == 0)

        return {};

    P3 q = p1 + v \* dot(dir2, p2 - p1) / d;

    return {q, q + e};

}

// 3D Covex Hull Template

db getV(P3 k1, P3 k2, P3 k3, P3 k4)

{ // get the Volume

    return dot(cross(k2 - k1, k3 - k1), k4 - k1);

}

db rand\_db() { return 1.0 \* rand() / RAND\_MAX; }

VP convexHull2D(VP A, P3 dir)

{

    P3 x = {(db)rand(), (db)rand(), (db)rand()};

    x = x.unit();

    x = cross(x, dir).unit();

    P3 y = cross(x, dir).unit();

    P3 vec = dir.unit() \* dot(A[0], dir);

    vector<point> B;

    for (int i = 0; i < A.size(); i++)

        B.push\_back((point){dot(A[i], x), dot(A[i], y)});

    B = ConvexHull(B);

    A.clear();

    for (int i = 0; i < B.size(); i++)

        A.push\_back(x \* B[i].x + y \* B[i].y + vec);

    return A;

}

namespace CH3

{

    VVP ret;

    set<pair<int, int>> e;

    int n;

    VP p, q;

    void wrap(int a, int b)

    {

        if (e.find({a, b}) == e.end())

        {

            int c = -1;

            for (int i = 0; i < n; i++)

                if (i != a && i != b)

                {

                    if (c == -1 || sgn(getV(q[c], q[a], q[b], q[i])) > 0)

                        c = i;

                }

            if (c != -1)

            {

                ret.push\_back({p[a], p[b], p[c]});

                e.insert({a, b});

                e.insert({b, c});

                e.insert({c, a});

                wrap(c, b);

                wrap(a, c);

            }

        }

    }

    VVP ConvexHull3D(VP \_p)

    {

        p = q = \_p;

        n = p.size();

        ret.clear();

        e.clear();

        for (auto &i : q)

            i = i + (P3){rand\_db() \* 1e-4, rand\_db() \* 1e-4, rand\_db() \* 1e-4};

        for (int i = 1; i < n; i++)

            if (q[i].x < q[0].x)

                swap(p[0], p[i]), swap(q[0], q[i]);

        for (int i = 2; i < n; i++)

            if ((q[i].x - q[0].x) \* (q[1].y - q[0].y) > (q[i].y - q[0].y) \* (q[1].x - q[0].x))

                swap(q[1], q[i]), swap(p[1], p[i]);

        wrap(0, 1);

        return ret;

    }

}

VVP reduceCH(VVP A)

{

    VVP ret;

    map<P3, VP> M;

    for (VP nowF : A)

    {

        P3 dir = cross(nowF[1] - nowF[0], nowF[2] - nowF[0]).unit();

        for (P3 k1 : nowF)

            M[dir].push\_back(k1);

    }

    for (pair<P3, VP> nowF : M)

        ret.push\_back(convexHull2D(nowF.second, nowF.first));

    return ret;

}

//  把一个面变成 ( 点 , 法向量 ) 的形式

pair<P3, P3> getF(VP F)

{

    return {F[0], cross(F[1] - F[0], F[2] - F[0]).unit()};

}

// 3D Cut 保留 dot(dir,x-p)>=0 的部分

VVP ConvexCut3D(VVP A, P3 p, P3 dir)

{

    VVP ret;

    VP sec;

    for (VP nowF : A)

    {

        int n = nowF.size();

        VP ans;

        int dif = 0;

        for (int i = 0; i < n; i++)

        {

            int d1 = sgn(dot(dir, nowF[i] - p));

            int d2 = sgn(dot(dir, nowF[(i + 1) % n] - p));

            if (d1 >= 0)

                ans.push\_back(nowF[i]);

            if (d1 \* d2 < 0)

            {

                P3 q = getFL(p, dir, nowF[i], nowF[(i + 1) % n])[0];

                ans.push\_back(q);

                sec.push\_back(q);

            }

            if (d1 == 0)

                sec.push\_back(nowF[i]);

            else

                dif = 1;

            dif |= (sgn(dot(dir, cross(nowF[(i + 1) % n] - nowF[i], nowF[(i + 1) % n] - nowF[i]))) == -1);

        }

        if (ans.size() > 0 && dif)

            ret.push\_back(ans);

    }

    if (sec.size() > 0)

        ret.push\_back(convexHull2D(sec, dir));

    return ret;

}

db vol(VVP A)

{

    if (A.size() == 0)

        return 0;

    P3 p = A[0][0];

    db ans = 0;

    for (VP nowF : A)

        for (int i = 2; i < nowF.size(); i++)

            ans += abs(getV(p, nowF[0], nowF[i - 1], nowF[i]));

    return ans / 6;

}

VVP init(db INF)

{

    VVP pss(6, VP(4));

    pss[0][0] = pss[1][0] = pss[2][0] = {-INF, -INF, -INF};

    pss[0][3] = pss[1][1] = pss[5][2] = {-INF, -INF, INF};

    pss[0][1] = pss[2][3] = pss[4][2] = {-INF, INF, -INF};

    pss[0][2] = pss[5][3] = pss[4][1] = {-INF, INF, INF};

    pss[1][3] = pss[2][1] = pss[3][2] = {INF, -INF, -INF};

    pss[1][2] = pss[5][1] = pss[3][3] = {INF, -INF, INF};

    pss[2][2] = pss[4][3] = pss[3][1] = {INF, INF, -INF};

    pss[5][0] = pss[4][0] = pss[3][0] = {INF, INF, INF};

    return pss;

}