Standard Code Library

ONGLU

North Eastern University

August 2021

Contents

初始化 **4**

Clion 环境配置 **4**

数据结构 **4**

可撤销并查集 . . . 4
树分治 . . . 4
线段树上二分 . . . 6
ST 表 . . . 6
二维哈希 . . . 6
线性基 . . . 7
轻重链剖分 . . . 7
线段树合并 . . . 8
二维树状数组 . . . 8
平衡树 . . . 9
K-D Tree . . . 10
可持久化数据结构 . . . 12
可持久化 Trie . . . 12
主席树（静态第 k 小） . . . 13
cdq 分治三维偏序 . . . 14
整体二分求区间静态 k 小 . . . 15
Link Cut Tree . . . 16

数学 **18**

数论 . . . 18
欧拉函数 . . . 18
排列组合 . . . 19
逆元 . . . 19
拓展欧几里得 . . . 20
拓展中国剩余定理 . . . 20
Miller_rabbin 素数测试 . . . 20
多项式 . . . 21
结论 . . . 21
拉格朗日插值法 . . . 21
FFT 快速傅里叶变换 . . . 21
NTT 快速数论变换 . . . 22
FWT 快速沃尔什变换 . . . 23
子集卷积 . . . 24
群论 . . . 25
结论 . . . 25
BurnSide 引理 . . . 25
Polya 定理 . . . 25
线性代数 . . . 25
矩阵运算全家桶 . . . 25
高斯消元 . . . 26

图论 **27**

树论 . . . 27
树的直径 . . . 27
求 LCA . . . 28
树上启发式合并 . . . 29
图论 . . . 29
第 k 短路 . . . 29
二分图匹配 . . . 30
结论 . . . 30
匈牙利算法 . . . 30

KM 算法二分图最大权匹配 . . . 31
网络流 . . . 32
Dinic 算法 . . . 32
EK 算法费用流 . . . 32
无源汇上下界可行流 . . . 33
连通性算法 . . . 33
Tarjan 强连通分量 . . . 33
kosaraju 算法 . . . 33
点双连通 . . . 34
边双连通 . . . 35
2-SAT . . . 36

计算几何 **37**
公式 . . . 37
结构体定义 . . . 37
基本操作 . . . 38
线 . . . 38
点与线 . . . 39
线与线 . . . 39
多边形 . . . 39
凸包 . . . 40
旋转卡壳 . . . 40
半平面交 . . . 40
圆 . . . 42
直线和圆 . . . 42
最小圆覆盖 . . . 43
三维几何 . . . 44
tips . . . 45

字符串 **45**
Lemma . . . 45
最小表示法 . . . 45
双哈希 . . . 45
字串哈希 . . . 46
后缀数组 . . . 47
SA-IS . . . 48
SAM . . . 49
后缀树 . . . 50
Trie . . . 50
KMP 算法 . . . 51
manacher 算法 . . . 51
回文自动机 . . . 51
AC 自动机 . . . 52

区间本质不同子串个数 **53**

杂项 **54**
取模还原成分数 . . . 54
快读 . . . 54
int128 . . . 54
Java,BigInteger . . . 55
奇技淫巧 . . . 55
对拍 . . . 55
快速乘 . . . 56
子集枚举 . . . 56
mt19937_64 随机数生成器 . . . 56
tips: . . . 56

经典题 **57**
区间回文子串计数（非本质不同） 57

初始化

Clion 环境配置

program : g++ argument :

```
-std=c++14 "$FileDir$\$FileName$" -o "$ProjectFileDir$\a.exe" -D MYLOCAL -Wl,--stack,524288000
```

数据结构

可撤销并查集

采用按秩合并的并查集，没有路径压缩后可以撤销。

```
struct UNDO_DSU {
    int pre[N], dis[N], siz[N];
    stack<pair<int, int> >mg;
    void init(int x) {
        for(int i = 0; i <= x; i++) {
            pre[i] = i;
            dis[i] = 0;
            siz[i] = 1;
        }
    }
    pair<int, int> fid(int x) {
        if(x == pre[x]) return {x, 0};
        auto item = fid(pre[x]);
        item.second += dis[x];
        return item;
    }
    int merge(int x, int y) {
        auto a = fid(x);
        auto b = fid(y);
        int fx = a.first;
        int fy = b.first;
        if(fx == fy) {
            mg.push({-1, -1});
            return 0;
        }
        if(siz[fx] < siz[fy]) swap(fx, fy), swap(a, b);
        siz[fx] += siz[fy];
        dis[fy] = a.second + b.second + 1;
        pre[fy] = fx;
        mg.push({fx, fy});
        return 1;
    }
    int undo() {
        if(mg.empty()) return 0;
        auto t = mg.top(); mg.pop();
        if(t.first == -1) return -1;
        siz[t.first] -= siz[t.second];
        pre[t.second] = t.second;
        dis[t.second] = 0;
        return 1;
    }
};
```

树分治

点分治，思想是枚举所有路径，但是路径要支持可合并。
递归切割整棵树，无根树选定重心作为根。
枚举每棵子树向上到达根的路径，与其他子树的答案合并，组成经过当前根节点的所有路径。
计算完经过当前点的路径后，递归计算每棵子树，同样也是选取重心作为分割点。

```
const int M = 1e7 + 1009;
const int N = 2e5 + 1009;
```

```cpp
struct Edge {
    int to, w;
};
int sum;
int n, m, q[N], ans[N], rt;
int maxSize[N], siz[N], vis[N];
vector<Edge> ver[N];
void findRoot(int x, int pre) {
    maxSize[x] = 0;
    siz[x] = 1;
    for(auto e : ver[x]) {
        int y = e.to, w = e.w;
        if(y == pre || vis[y]) continue;
        findRoot(y, x);
        maxSize[x] = max(maxSize[x], siz[y]);
        siz[x] += siz[y];
    }
    // 这里一定要是 sum 而不能是 n，因为要切割子树
    maxSize[x] = max(maxSize[x], sum - siz[x]);
    if(rt == -1 || maxSize[x] < maxSize[rt]) rt = x;
}
int dis[N], distot, bul[M];
void getDist(int x, int pre, int dd) {
    dis[++distot] = dd;
    for(auto e : ver[x]) {
        int y = e.to, w = e.w;
        if(y != pre && !vis[y]) {
            getDist(y, x, dd + w);
        }
    }
}
queue<int> clr;
void dfz(int x) {
    vis[x] = 1;
    bul[0] = 1;
    clr.push(0);
    for(auto e : ver[x]) {
        // 一定要每棵子树单独计算链，否则会出现两条同一子树内的链合并的情况，造成多算
        int y = e.to, w = e.w;
        if(vis[y]) continue;
        distot = 0;
        getDist(y, x, w);
        for(int i = 1; i <= distot; i++) {
            for(int j = 1; j <= m; j++) {
                if(q[j] >= dis[i] && bul[q[j] - dis[i]]) {
                    ans[j] = 1;
                }
            }
        }
        // 合并完子树和其他树的链后，再将链加入桶
        for(int i = 1; i <= distot; i++) if(dis[i] < M) bul[dis[i]] = 1, clr.push(dis[i]);
    }
    while(!clr.empty()) bul[clr.front()] = 0, clr.pop();
    for(auto e : ver[x]) {
        int y = e.to, w = e.w;
        if(vis[y]) continue;
        // 子树大小一定要设置成 siz[y]，否则复杂度出错
        sum = siz[y];
        rt = -1;
        findRoot(y, x);
        findRoot(rt, rt);
        dfz(rt);
    }
}
void work() {
    cin >> n >> m;
    for(int i = 1; i < n; i++) {
        int x, y, w;
        cin >> x >> y >> w;
        ver[x].push_back({y, w});
        ver[y].push_back({x, w});
```

```
    }
    for(int i = 1; i <= m; i++) cin >> q[i];
    rt = -1;
    sum = n;
    findRoot(1, 1);
    findRoot(rt, rt);
    dfz(rt);
    for(int i = 1; i <= m; i++) {
        if(ans[i]) cout << "AYE" << endl;
        else cout << "NAY" << endl;
    }
}
```

线段树上二分

要先 check 是否存在第 k 大线段树上权值 <10，判断 L,R 右边权值都等于 0 或者 9 的最长长度。

```
int countval(int l, int r, int rt, int L, int R, int val) {
    if(L == l && R == r && sum[rt] == val * (r - l + 1)) {
        return (r - l + 1);
    }
    if(l == r) return sum[rt] == val;
    pushdown(l, r, rt);
    if(L > Mid) return countval(Mid + 1, r, rson, L, R, val);
    if(R <= Mid) return countval(l, Mid, lson, L, R, val);
    int ans = countval(Mid + 1, r, rson, Mid + 1, R, val);
    if(ans < (R - Mid)) return ans;
    ans += countval(l, Mid, lson, L, Mid, val);
    return ans;
}
```

ST 表

```
int query(int l, int r) {
    int p = log2(r - l + 1);
    return max(st[p][l], st[p][r - (1 << p) + 1]);
}
void init() {
    for(int i = 2; i <= n; i++) LOG2[i] = LOG2[i / 2] + 1;
    for(int i = 1; i <= n; i++) st[0][i] = h[i];
    for(int i = 1; i < 25; i++) {
        for(int j = 1; j + (1 << i - 1) <= n; j++) {
            st[i][j] = max(st[i - 1][j], st[i - 1][j + (1 << i - 1)]);
        }
    }
}
```

二维哈希

```
ull hs[109][109], pw1[10009], pw2[100009];
ull gethash(int lx, int ly, int rx, int ry) {
    ull hs1 = hs[lx][ly] - pw2[ry - ly + 1] * hs[lx][ry + 1];
    ull hs2 = hs[rx + 1][ly] - pw2[ry - ly + 1] * hs[rx + 1][ry + 1];
    return hs1 - pw1[rx - lx + 1] * hs2;
}
pw1[0] = pw2[0] = 1;
for(int i = 1; i <= 1000; i++) pw1[i] = pw1[i - 1] * 19260817;
for(int i = 1; i <= 1000; i++) pw2[i] = pw2[i - 1] * 135;
for(int i = n; i >= 1; i--) {
    for(int j = 1; j <= m; j++) {
        if(i == n) hs[i][j] = sum[i][j] + 2;
        else hs[i][j] = hs[i + 1][j] * 19260817 + sum[i][j] + 2;
    }
}
for(int i = 1; i <= n; i++) {
    for(int j = m - 1; j; j--) {
        hs[i][j] = hs[i][j + 1] * 135 + hs[i][j];
    }
}
```

线性基

最大异或和

```
struct Basis {
    long long base[64];
    void insert(long long x) {
        for(int i = 62; i >= 0; i--) {
            if(x == 0) return;
            if(x >> i & 1) {
                if(base[i] != 0) x ^= base[i];
                else {
                    base[i] = x;
                    break;
                }
            }
        }
    }
    int contains(long long x) {
        for(int i = 62; i >= 0; i--) if(x >> i & 1) {
            if(!base[i]) return 0;
            x ^= base[i];
        }
        return 1;
    }
    int maxContains() {
        long long ans = 0;
        for(int i = 62; i >= 0; i--) {
            if(ans >> i & 1) continue;
            ans ^= base[i];
        }
        return ans;
    }
};
```

轻重链剖分

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

    }
    dth[x] < dth[y] && (x ^= y ^= x ^= y);
    modify(1, n, id[y], id[x], 1, z);
}
int link_query(int x, int y) {
    int ans = 0;
    while(ltp[x] != ltp[y]) {
```

```
        dth[ltp[x]] < dth[ltp[y]] && (x ^= y ^= x ^= y);
        ans = (1ll * ans + query(1, n, id[ltp[x]], id[x], 1)) % mod;
        x = fa[ltp[x]];
    }
    dth[x] < dth[y] && (x ^= y ^= x ^= y);
    ans = (1ll * ans + query(1, n, id[y], id[x], 1)) % mod;
    return ans;
}
```

线段树合并

搞个动态开点线段树出来

```
#define mval(x) tree[x].mval
#define mpos(x) tree[x].mpos
#define lson(x) tree[x].lson
#define rson(x) tree[x].rson
struct node {
    int mpos, mval, lson, rson;
} tree[N * 50];
void update(int rt) {
    if(mval(lson(rt)) >= mval(rson(rt))) {
        mval(rt) = mval(lson(rt));
        mpos(rt) = mpos(lson(rt));
    } else {
        mval(rt) = mval(rson(rt));
        mpos(rt) = mpos(rson(rt));

    }
}
void modify(int l, int r, int x, int v, int &rt) {
    if(!rt) rt = ++idtot;
    if(l == r) {
        mval(rt) += v;
        mpos(rt) = l;
        return ;
    }
    if(x <= Mid) modify(l, Mid, x, v, lson(rt));
    else modify(Mid + 1, r, x, v, rson(rt));
    update(rt);
}
int merge(int l, int r, int rt1, int rt2) {
    if(!rt1 || !rt2) return rt1 + rt2;
    if(l == r) {
        mval(rt1) += mval(rt2);
        mpos(rt1) = l;
        return rt1;
    }
    lson(rt1) = merge(l, Mid, lson(rt1), lson(rt2));
    rson(rt1) = merge(Mid + 1, r, rson(rt1), rson(rt2));
    update(rt1);
    return rt1;
}
```

二维树状数组

- 矩阵修改，矩阵查询

  查询前缀和公式：

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

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

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

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

平衡树

- luogu P3369 【模板】普通平衡树

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

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

K-D Tree

用方差最大的那一维坐标作为当前的划分点集，然后选取该维度的中位数点划分成左右两个点集。

```
#include <bits/stdc++.h>
#define pt(x) cout << x << endl;
#define Mid ((l + r) / 2)
#define low(x, k) tree[x].low[k]
#define high(x, k) tree[x].high[k]
#define lson(x) tree[x].lson
#define rson(x) tree[x].rson
using namespace std;
int read() {
    char c; int num, f = 1;
    while(c = getchar(),!isdigit(c)) if(c == '-') f = -1; num = c - '0';
    while(c = getchar(), isdigit(c)) num = num * 10 + c - '0';
```

```
    return f * num;
}
const int N = 5e5 + 1009;

namespace KD_Tree{

    const int dimension = 2;
    struct node {
        int lson, rson;
        int low[dimension], high[dimension];
    } tree[N];
    struct Point {
        int id;
        int v[dimension];
    } p[N];
    void update(int rt) {
        for(int i = 0; i < dimension; i++) {
            low(rt, i) = high(rt, i) = p[rt].v[i];
            if(lson(rt)) {
                low(rt, i) = min(low(rt, i), low(lson(rt), i));
                high(rt, i) = max(high(rt, i), high(lson(rt), i));
            }
            if(rson(rt)) {
                low(rt, i) = min(low(rt, i), low(rson(rt), i));
                high(rt, i) = max(high(rt, i), high(rson(rt), i));
            }

        }
    }
    int build(int l, int r) {
        if(l > r) return 0;
        double av[dimension] = {0};
        double va[dimension] = {0};
        for(int i = 0; i < dimension; i++)
            low(Mid, i) = high(Mid, i) = p[Mid].v[i];
        for(int i = l; i <= r; i++)
            for(int j = 0; j < dimension; j++)
                av[j] += p[i].v[j];
        for(int i = 0; i < dimension; i++)
            av[i] /= (double) (r - l + 1);
        for(int i = l; i <= r; i++)
            for(int j = 0; j < dimension; j++)
                va[j] += (p[i].v[j] - av[j]) * (p[i].v[j] - av[j]);
        int maxdi = 0;
        for(int i = 1; i < dimension; i++)
            if(va[i] > va[maxdi])
                maxdi = i;
        nth_element(p + l, p + Mid, p + 1 + r, [maxdi](const Point &a, const Point &b)  -> int{return a.v[maxdi] <
↪  b.v[maxdi];});
        lson(Mid) = build(l, Mid - 1);
        rson(Mid) = build(Mid + 1, r);
        update(Mid);
        return Mid;
    }
    int isIn(const Point &a, const Point &ld, const Point &ru) {
        for(int i = 0; i < dimension; i++)
            if(a.v[i] < ld.v[i] || a.v[i] > ru.v[i])
                return false;
        return true;
    }
    void debug(int rt, int l, int r) {
        if(l > r) return ;
        printf("%d\n", p[rt].id);
        debug(lson(rt), l, Mid - 1);
        debug(rson(rt), Mid + 1, r);

    }
    //只能处理二维
    void getNodeset(int rt, int l, int r, vector<int> &v, const Point &ld, const Point &ru) {
        if(l > r) return ;
        for(int i = 0; i < dimension; i++) {
```

```
            if(low(rt, i) > ru.v[i] || high(rt, i) < ld.v[i]) {
                return ;
            }
        }
        if(isIn(p[Mid], ld, ru))
            v.push_back(p[Mid].id);
        getNodeset(lson(rt), l, Mid - 1, v, ld, ru);
        getNodeset(rson(rt), Mid + 1, r, v, ld, ru);
    }
}
using namespace KD_Tree;
int n, q, root;
signed main()
{
    n = read();
    for(int i = 1; i <= n; i++) {
        p[i].v[0] = read();
        p[i].v[1] = read();
        p[i].id = i - 1;
    }
    root = build(1, n);
    q = read();
    for(int i = 1; i <= q; i++) {
        int x = read(), xx = read();
        int y = read(), yy = read();
        Point ld, ru;
        ld.v[0] = x; ld.v[1] = y;
        ru.v[0] = xx; ru.v[1] = yy;
        vector<int> v;
        v.clear();
        getNodeset(root, 1, n, v, ld, ru);
        sort(v.begin(), v.end());
        for(auto x : v)
            printf("%d\n", x);
        printf("\n");
    }
    return 0;
}
```

可持久化数据结构

可持久化 Trie

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

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

主席树（静态第 k 小）

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

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <assert.h>
#define Mid ((l + r) / 2)
#define lson (rt << 1)
#define rson (rt << 1 | 1)
using namespace std;
int read() {
    char c; int num, f = 1;
    while(c = getchar(),!isdigit(c)) if(c == '-') f = -1; num = c - '0';
    while(c = getchar(), isdigit(c)) num = num * 10 + c - '0';
    return f * num;
}
const int N = 1e7 + 1009;
const int M = 2e5 + 1009;
struct node {
    int ls, rs, v;
} tree[N];
int tb;
int n, m, tot, a[M], b[M], rt[M];
int _new(int ls, int rs, int v) {
    tree[++tot].ls = ls;
    tree[tot].rs = rs;
    tree[tot].v = v;
    return tot;
}
void update(int rt) {
    tree[rt].v = tree[tree[rt].ls].v + tree[tree[rt].rs].v;
}
int build(int l, int r) {
    if(l == r) return _new(0, 0, 0);
    int x = _new(build(l, Mid), build(Mid + 1, r), 0);
    update(x);
    return x;
}
int add(int l, int r, int p, int rt, int v) {
    int x = ++tot;
    tree[x] = tree[rt];
    if(l == r) {
        tree[x].v += v;
        return x;
    }
    if(p <= Mid) tree[x].ls = add(l, Mid, p, tree[x].ls, v);
    else tree[x].rs = add(Mid + 1, r, p, tree[x].rs, v);
    update(x);
    return x;
}
int query(int l, int r, int rt1, int rt2, int k) {
    if(l == r) return l;
    if(k <= tree[tree[rt1].ls].v - tree[tree[rt2].ls].v) return query(l, Mid, tree[rt1].ls, tree[rt2].ls, k);
```

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

cdq 分治三维偏序

先按照第一维，第二维，第三维的顺序排序，再去重。目的是为了保证右边元素不会对左边元素产生贡献。
然后对第二维归并, 归并时计算左边第三维小于等于右边第三维的数量。
注意清空树状数组的时候要与值域相关。

```
//
// Created by onglu on 2022/8/3.
//

#include <bits/stdc++.h>

#define all(a) a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()

#define endl '\n'
#define lson (rt << 1)
#define rson (rt << 1 | 1)
#define Mid ((l + r) / 2)
//#define int long long
using namespace std;
const int N = 2e6 + 1009;
//const int N = 2e5 + 1009;
//const int N = 5009;
//const int N = 309;
int n, m;
struct Point {
    int id, x, y, z, cnt, ans;
} a[N], b[N];
int tree[N];
void add(int x, int y) {
    for(; x <= m; x += x & -x)
        tree[x] += y;
}
void clear(int x) {
    for( ; x <= m; x += x & -x)
        tree[x] = 0;
}
int query(int x) {
    int ans = 0;
    for( ; x; x -= x & -x)
        ans += tree[x];
    return ans;
```

```
}
void solve(int l, int r) {
    if(l == r) return ;
    solve(l, Mid); solve(Mid + 1, r);
    int i = l, tot = i;
    for(int j = Mid + 1; j <= r; j++) {
        while(i <= Mid && a[i].y <= a[j].y) {
            add(a[i].z, a[i].cnt);
            b[tot++] = a[i++];
        }
        a[j].ans += query(a[j].z);
        b[tot++] = a[j];
    }

    while(i <= Mid) b[tot++] = a[i++];
    for(int j = l; j <= r; j++) {
        clear(b[j].z);
        a[j] = b[j];
    }
}
void work() {
    cin >> n >> m;
    for(int i = 1; i <= n; i++) {
        cin >> a[i].x >> a[i].y >> a[i].z;
        a[i].id = i;
    }
    sort(a + 1, a + 1 + n, [](const Point &a, const Point &b) {
        if(a.x != b.x) return a.x < b.x;
        if(a.y != b.y) return a.y < b.y;
        return a.z < b.z;
    });
    int tot = 0;
    for(int i = 1; i <= n; i++) {
        if(i == 0 || a[i].x != a[i - 1].x || a[i].y != a[i - 1].y || a[i].z != a[i - 1].z) {
            a[++tot] = a[i];
            a[tot].cnt = 1;
        } else {
            a[tot].cnt++;
        }
    }
    solve(1, n);
    vector<int> ans_cnt(n + 1);
    for(int i = 1; i <= n; i++) {
        ans_cnt[a[i].ans + a[i].cnt - 1] += a[i].cnt;
    }
    for(int i = 0; i < n; i++) cout << ans_cnt[i] << endl;
}
```

整体二分求区间静态 k 小

二分时用 vector 存在答案在 [l,r] 之间的询问。统计答案 [l, Mid] 对所有询问的影响，影响足够的放到 [l, Mid] 中继续处理。影响不够的**消除影响后**放到 [Mid + 1, r] 中继续处理。

```
//
// Created by onglu on 2022/8/3.
//

#define Mid ((l + r) / 2)
using namespace std;
const int N = 2e6 + 1009;
int n, m, a[N], ans[N];
struct query {
    int id, l, r, k;
};
vector<int> b;
vector<int> pos[N];
void solve(int l, int r, vector<query> v) {
    vector<int> nums;
    for(int i = l; i <= Mid; i++) {
        for(auto x : pos[i]) {
```

```
            nums.push_back(x);
        }
    }
    std::sort(nums.begin(), nums.end());
    vector<query> vl, vr;
    for(auto x : v) {
        int lpos = std::lower_bound(nums.begin(), nums.end(),x.l) - nums.begin();
        int rpos = std::upper_bound(nums.begin(), nums.end(),x.r) - nums.begin();
        if(rpos - lpos >= x.k) {
            vl.push_back(x);
        } else {
            vr.push_back(x);
            vr.back().k -= (rpos - lpos);
        }
    }
    if(l == r) {
        for(auto x : vl) ans[x.id] = l;
        for(auto x : vr) ans[x.id] = -1;
        return ;
    }
    if(vl.size()) solve(l, Mid, vl);
    if(vr.size()) solve(Mid + 1, r, vr);
}
void work() {
    cin >> n >> m;
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
        b.push_back(a[i]);
    }
    std::sort(b.begin(), b.end());
    b.resize(std::unique(b.begin(), b.end()) - b.begin());
    for(int i = 1; i <= n; i++) {
        a[i] = std::lower_bound(b.begin(), b.end(), a[i]) - b.begin() + 1;
        pos[a[i]].push_back(i);
    }
    for(int i = 1; i <= b.size(); i++) {
        std::sort(pos[i].begin(), pos[i].end());
    }
    vector<query> v(m);
    for(int i = 0; i < m; i++) {
        v[i].id = i;
        cin >> v[i].l >> v[i].r >> v[i].k;
    }
    solve(1, b.size(), v);
    for(int i = 0; i < m; i++) {
        if(ans[i] == -1) cout << "No Answer" << endl;
        else cout << b[ans[i] - 1] << endl;
    }
}
```

Link Cut Tree

access(x)：将 x 与 x 所在树的路径构造到同一个 splay 里面。makeroot(x)：将 x 置为树的根。

```
struct LCT {
    int fa[N], ch[N][2], val[N], sum[N], tmp[N];
    bool rev[N];
    bool isroot(int x) {return !fa[x] || ch[fa[x]][0] != x && ch[fa[x]][1] != x;}
    void pushrev(int x) {if(!x)return; swap(ch[x][0], ch[x][1]); rev[x] ^= 1;}
    void pushdown(int x) {if(rev[x]) {pushrev(ch[x][0]); pushrev(ch[x][1]); rev[x] = 0;}}
    void update(int x) {
        sum[x] = val[x];
        if(ch[x][0]) sum[x] += sum[ch[x][0]];
        if(ch[x][1]) sum[x] += sum[ch[x][1]];
    }
    void rotate(int x) {
        int y = fa[x], w = ch[y][1] == x;
        ch[y][w] = ch[x][w ^ 1];
        if(ch[x][w ^ 1]) fa[ch[x][w ^ 1]] = y;
        if(fa[y]) {
            int z = fa[y];
```

```
            if(ch[z][0] == y) ch[z][0] = x; else if(ch[z][1] == y) ch[z][1] = x;
        }
        fa[x] = fa[y]; fa[y] = x; ch[x][w ^ 1] = y; update(y);
    }
    void splay(int x) {
        int s = 1, i = x, y; tmp[1] = i;
        while(!isroot(i)) tmp[++s] = i = fa[i];
        while(s) pushdown(tmp[s--]);
        while(!isroot(x)) {
            y = fa[x];
            if(!isroot(y)) {
                if((ch[fa[y]][0] == y) ^ (ch[y][0] == x)) rotate(x);
                else rotate(y);
            }
            rotate(x);
        }
        update(x);
    }
    int access(int x) { int y; for(y = 0; x; y = x, x = fa[x]) {splay(x); ch[x][1] = y; update(x);}return y;}
    int root(int x) { access(x); splay(x); while(ch[x][0]) pushdown(x), x = ch[x][0]; return x; }
    void makeroot(int x) { access(x); splay(x); pushrev(x); }
    void link(int x, int y) { makeroot(x); if(root(y) != x) fa[x] = y;}
    void cut(int x, int y) {
        makeroot(x);
        if(root(y) == x && fa[x] == y && ch[y][0] == x) {
            fa[x] = ch[y][0] = 0;
            update(x);
        }
    }
    int ask(int x, int y) { makeroot(x); access(y); splay(y); return sum[y]; }
    int lca(int root, int x, int y) {return makeroot(root), access(x), access(y);}
    void fix(int x, int y) { splay(x); val[x] = y; update(x);}
};
```

带子树查询的 LCT（[BJOI2014] 大融合）

计算树上经过一条边的路径数量，模板用于计算连通块 size

```
struct LCT {
    int fa[N], ch[N][2], siz[N], siz2[N], tmp[N];
    bool rev[N];
    bool isroot(int x) {return !fa[x] || ch[fa[x]][0] != x && ch[fa[x]][1] != x;}
    void pushrev(int x) {if(!x)return; swap(ch[x][0], ch[x][1]); rev[x] ^= 1;}
    void pushdown(int x) {if(rev[x]) {pushrev(ch[x][0]); pushrev(ch[x][1]); rev[x] = 0;}}
    void update(int x) {
        siz[x] = 1 + siz2[x];
        if(ch[x][0]) siz[x] += siz[ch[x][0]];
        if(ch[x][1]) siz[x] += siz[ch[x][1]];
    }
    void rotate(int x) {
        int y = fa[x], w = ch[y][1] == x;
        ch[y][w] = ch[x][w ^ 1];
        if(ch[x][w ^ 1]) fa[ch[x][w ^ 1]] = y;
        if(fa[y]) {
            int z = fa[y];
            if(ch[z][0] == y) ch[z][0] = x; else if(ch[z][1] == y) ch[z][1] = x;
        }
        fa[x] = fa[y]; fa[y] = x; ch[x][w ^ 1] = y; update(y);
    }
    void splay(int x) {
        int s = 1, i = x, y; tmp[1] = i;
        while(!isroot(i)) tmp[++s] = i = fa[i];
        while(s) pushdown(tmp[s--]);
        while(!isroot(x)) {
            y = fa[x];
            if(!isroot(y)) {
                if((ch[fa[y]][0] == y) ^ (ch[y][0] == x)) rotate(x);
                else rotate(y);
            }
            rotate(x);
        }
        update(x);
```

```
    }
    void access(int x) {
        for(int y = 0; x; y = x, x = fa[x]) {
            splay(x);
            if(ch[x][1]) siz2[x] += siz[ch[x][1]];
            if(y) siz2[x] -= siz[y];
            ch[x][1] = y;
            update(x);
        }
    }
    int root(int x) { access(x); splay(x); while(ch[x][0]) pushdown(x), x = ch[x][0]; return x; }
    void makeroot(int x) { access(x); splay(x); pushrev(x); }
    void link(int x, int y) {
        makeroot(x);
        if(root(y) != x) {
            fa[x] = y;
            siz2[y] += siz[x];
        }
    }
    void cut(int x, int y) {
        makeroot(x);
        if(root(y) == x && fa[x] == y && ch[y][0] == x) {
            fa[x] = ch[y][0] = 0;
            update(x);
            update(y);
        }
    }
} lct;
int n, m;
void work() {
    cin >> n >> m;
    for(int i = 1; i <= m; i++) {
        char c;
        int x, y;
        cin >> c >> x >> y;
        if(c == 'A') {
            lct.link(x, y);
        } else {
            lct.makeroot(x);
            lct.access(y);
            lct.splay(y);
            cout << 1ll * (lct.siz[y] - lct.siz[x]) * lct.siz[x] << endl;
        }
    }
}
```

数学

数论

欧拉函数

性质

1 和任何数互质。

+ $\phi(1) = 1$ + $\phi(p) = p - 1$(p 为质数) + $\phi(x \times p) = \phi(x) \times p(p \mid x), \phi(x \times p) = \phi(x) \times p(p \nmid x)$

线性欧拉函数筛

```
int phi[N], f[N], pri[N], tot;
void getphi() {
    int k;
    phi[1] = 1;
    for(int i = 2; i < N; i++) {
        if(!f[i]) phi[pri[++tot] = i] = i - 1;
        for(int j = 1; j <= tot && (k = i * pri[j]) < N; j++) {
            f[k] = 1;
            if(i % pri[j]) phi[k] = phi[i] * (pri[j] - 1);
            else {
                phi[k] = phi[i] * pri[j];
```

```
                break;
            }
        }
    }
}
```

$O(\sqrt{n})$ 求欧拉函数

```
int getphi(int x) {
    int phi = 1;
    for(int i = 2; i * i <= x; i++) if(x % i == 0) {
        phi *= (i - 1); x /= i;
        while(x % i == 0) {
            phi = phi * i;
            x /= i;
        }
    }
    if(x > 1) phi *= x - 1;
    return phi;
}
```

排列组合

斯特林近似求组合 (≥ 15 时收敛)

精度容易不够，推荐使用 python Demical 类

$$\ln n! \simeq n\ln n - n + \frac{1}{6}\ln(8n^3 + 4n^2 + n + \frac{1}{30}) + \frac{1}{2}\ln\pi$$

```
double lnfac(int n) {
    return n * log(n) - n + 1.0 / 6 * log(8 * n * n * n + 4 * n * n + n + 1.0 / 30) + 0.5 * log(acos(-1.0));
}
double C(int n, int m) {
    return exp(lnfac(n) - lnfac(n - m) - lnfac(m));
}
```

Lucas 定理

$$\binom{n}{m} = \binom{n \mod p}{m \mod p} \times \binom{n/p}{m/p}$$

```
int C(int n, int m) {
    if(m > n) return 0;
    if(n < mod) return 1ll * fac[n] * inv[n - m] % mod * inv[m] % mod;
    else return 1ll * C(n / mod, m / mod) * C(n % mod, m % mod) % mod;
}
```

Min-Max 容斥

$$max(S) = \sum_{T\subseteq S}(-1)^{|T|-1}min(T)$$

逆元

线性推

```
inv[1] = inv[0] = 1;
for(int i = 2; i < N; i++)  inv[i] = (1ll * mod - mod / i) * inv[mod % i] % mod;
```

费马小定理 (模数为质数)

```
int inv(int x) {
    return Pow(x % mod, mod - 2);
}
```

exgcd(ap 互质)

```cpp
int inv(int x) {
    int x, y;
    exgcd(x, y, a, p);
    return (x % p + p) % p;
}
```

拓展欧几里得

求解的是类似 $ax + by = gcd(a, b)$ 的一组解。

```cpp
void exgcd(int &x, int &y, int a, int b) {
    if(b == 0) return (void)(x = 1, y = 0);
    exgcd(y, x, b, a % b);
    y = y - a / b * x;
}
```

拓展中国剩余定理

拓展中国剩余定理用于解决同余方程组。

$$x \equiv a_i (mod\, b_i)$$

构造 $M_k = lcm_{i=1}^{k-1} b_i$
假设前面的解为 p 显然新解 $p + M_k \times y$ 仍然是前面方程的解。
exgcd 求出 $M_k \times x + b_i \times y = gcd(M_k, b_i)$ 的解。
于是 $p' = p + x \times M_k \times (a_i - p)/gcd(M_k, b_i)$。
实际处理的时候可以直接让 $b_i = b_i/gcd(b_i, M_k)$ 防止溢出。

```cpp
#define long long ll
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}
ll lcm(ll a, ll b) {
    return a / gcd(a, b) * b;
}
ll exgcd(ll &x, ll &y, ll a, ll b) {
    if(b == 0) return x = 1, y = 0, a;
    ll t = exgcd(y, x, b, a % b);
    y -= a / b * x;
    return t;
}
inline ll mul(ll x, ll y, ll mod){
    return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
}

ll excrt(ll n, ll *a, ll *b) {
    ll ans = a[1], M = b[1];
    for(ll i = 2; i <= n; i++) {
        ll c = ((a[i] - ans) % b[i] + b[i]) % b[i], x, y;
        ll t = exgcd(x, y, M, b[i]), pb = b[i] / t;
        if(c % t != 0) return -1;
        x = mul(x, c / t, pb);
        ans = ans + x * M;
        M = M *pb;
        ans = (ans % M + M) % M;
    }
    return ans;
}
```

Miller_rabbin 素数测试

```cpp
namespace Isprime{
    ll mul(ll x, ll y, ll mod){
        return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
    }
    ll Pow(ll a, ll p, ll mod) {
        ll ans = 1;
        for( ; p; p >>= 1, a = mul(a, a, mod))
```

```
            if(p & 1)
                ans = mul(ans, a, mod);
        return ans % mod;
    }
    int check(ll P){
        const ll test[11] = {0, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
        if(P == 1) return false;
        if(P > 6 && P % 6 != 1 && P % 6 != 5) return false;
        ll k = 0, t = P - 1;
        while(!(t & 1)) k++, t >>= 1;
        for(int i = 1; i <= 10 && test[i] <= P; i++) {
            if(P == test[i]) return true;
            ll nxt, a = Pow(test[i], t, P);
            for(int j = 1; j <= k; j++) {
                nxt = mul(a, a, P);
                if(nxt == 1 && a != 1 && a != P - 1) return false;
                a = nxt;
            }
            if(a != 1)return false;
        }
        return true;
    }
}
```

多项式

结论

1. 自然数幂之和 $s(n)=\sum_{i=0}^{n} i^k$ 是关于 n 的 $k+1$ 次多项式

拉格朗日插值法

令拉格朗日函数

$$l_i(x)=\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

注意到这个函数有一些性质：
1. 次数为 n
2. 在 $x=x_i$ 位置值为 1,$x=x_j(j\neq i)$ 位置值为 0
于是可以凑出唯一的多项式表达式为：

$$f(x)=\sum_{i=0}^{n} y_i\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}$$

如果要取模的话得求逆元，逆元先求好分母再一起求即可。

```
int interpolation(int *x, int *y, int n) {
    int f = 0;
    for(int i = 1; i <= n; i++) {
        int s1 = 1, s2 = 1;
        for(int j = 1; j <= n; j++) {
            if(i != j) {
                s1 = 1ll * s1 * (k - x[j] + mod) % mod;
                s2 = 1ll * s2 * (x[i] - x[j] + mod) % mod;
            }
        }
        f = (f + 1ll * y[i] * s1 % mod * inv(s2) % mod) % mod;
    }
    return f;
}
```

FFT 快速傅里叶变换

FFT 的想法是把第 k 号位置变成 $f(\omega_n^k)$，注意到 $\omega_n^k=-\omega_n^{k+n/2}$，于是可以进行变换。
几条公式：

$$\omega_n^n=1$$

$$\omega_n^k = \omega_{2n}^{2k}$$

$$\omega_{2n}^{k+n} = -\omega_{2n}^k$$

蝴蝶变换：相邻的位置为二进制的 reverse
DFT 变换公式 ($DFT(f)$ 为矩阵)：
令

$$G(x) = a_0 + a_2 x + a_4 x^2 + ...$$
$$H(x) = a_1 + a_3 x + a_5 x^3 + ...$$

则有

$$f(x) = G(x^2) + x \times H(x^2)$$
$$DFT(f(\omega_n^k)) = DFT(G(\omega_{n/2}^k) + \omega_n^k \times DFT(H(\omega_{n/2}^k)))$$
$$DFT(f(\omega_n^{k+n/2})) = DFT(G(\omega_{n/2}^k) - \omega_n^k \times DFT(H(\omega_{n/2}^k)))$$

$DFT(G(\omega_{n/2}^k), DFT(H(\omega_{n/2}^k)))$ 可递归计算

NTT 快速数论变换

NTT 使用原根代替复数进行运算。
原根 g 的重要性质：$g^t \equiv k \mod n, t \in [0, n-2]$，$k$ 遍取 1~$n-1$
原根存在的充要条件是：模数 $n = 2, 4, p^\alpha, 2p^\alpha$($p$ 为奇质数)。
对于一个质数 $p = qn + 1 (n = 2^m)$, 原根满足性质 $g^{qn} \equiv 1 \mod p$。
它满足和复数近似的性质，我们把 q 看成复数中的 2π，就可以套用 FFT 实现 NTT 了。
$g_n^n \equiv 1, g_n^n \equiv -1$

通常取

$$p = 1004535809 = 7 \times 479 \times 2^{21} + 1, g = 3$$

$$p = 998244353 = 7 \times 17 \times 2^{23} + 1, g = 3$$

```cpp
const int P = 998244353, G = 3, Gi = 332748118;
struct Complex {double x, y;};
Complex operator+(const Complex &a, const Complex &b) {return (Complex) {a.x + b.x, a.y + b.y};}
Complex operator-(const Complex &a, const Complex &b) {return (Complex) {a.x - b.x, a.y - b.y};}
Complex operator*(const Complex &a, const Complex &b) {return (Complex) {a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x};}
namespace Polynomial {
    const double Pi = acos(-1.0);
    int rev[N];
    template <typename T>
    void change(T *y, int n) {
        for(int i = 0; i < n; i++)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? (n >> 1) : 0);
        for(int i = 0; i < n; i++)
            if(i < rev[i])
                swap(y[i], y[rev[i]]);
    }
    void FFT(Complex *A, int n, int type) {
        //type = 1 DFT
        //type = -1 IDFT
        //确保 n 是 2 的幂次
        change(A, n);
        for(int m = 1; m < n; m <<= 1) {
            Complex Wn = (Complex) {cos(Pi / m), type * sin(Pi / m)};
            for(int i = 0; i < n; i += 2 * m) {
                Complex w = (Complex) {1.0, 0};
                for(int j = 0; j < m; j++, w = w * Wn) {
```

```
                    Complex x = A[i + j], y = w * A[i + j + m];
                    A[i + j] = x + y;
                    A[i + j + m] = x - y;
                }
            }
        }
        if(type == -1) {
            for(int i = 0; i < n; i++)
                A[i].x = A[i].x / n;
        }
    }
    void NTT(int *A, int n, int type) {
        //type = 1 DFT
        //type = -1 IDFT
        change(A, n);
        for(int m = 1; m < n; m <<= 1) {
            int Wn = Pow(type == 1 ? G : Gi, (P - 1) / (m << 1));
            for(int i = 0; i < n; i += 2 * m) {
                int w = 1;
                for(int j = 0; j < m; j++, w = 1ll * w * Wn % P) {
                    int x = A[i + j], y = 1ll * w * A[i + j + m] % P;
                    A[i + j] = (x + y) % P;
                    A[i + j + m] = (x - y + P) % P;
                }
            }
        }
        if(type == -1) {
            int inv = Pow(n, P - 2);
            for(int i = 0; i < n; i++)
                A[i] = 1ll * A[i] * inv % P;
        }
    }

}
//以下代码加在主函数内
limit = 1;
while(limit <= n + m) limit <<= 1;
Polynomial :: FFT(A, limit, 1);
Polynomial :: FFT(B, limit, 1);
for(int i = 0; i < limit; i++) A[i] = A[i] * B[i];
Polynomial :: FFT(A, limit, -1);
```

FWT 快速沃尔什变换

FWT 用于计算下列多项式

$$C[k] = \sum_{i \oplus j = k} A[i] \times B[j]$$

先通过 FWT 将 A, B 变为 $FWT(A), FWT(B)$，这样有 $FWT(C) = FWT(A) \times FWT(B)$。
当然位运算符不同的时候对应的变换形式也需要改变。

$a \in S, b \in S$ 可以表示为 $a|b \in S$
FWT 为线性变换 $\sum FWT(F) = FWT(\sum F)$

与卷积

当 $\oplus = and$ 的时候
$FWT(A) = (FWT(A_0) + FWT(A_1), FWT(A_1))$
$FWT(A) = A$(长度为 1)
$IFWT(A) = (IFWT(A_0) - IFWT(A_1), IFWT(A_1))$

或卷积

当 $\oplus = or$ 的时候
$FWT(A) = (FWT(A_0), FWT(A_0) + FWT(A_1))$
$FWT(A) = A$(长度为 1)
$IFWT(A) = (IFWT(A_0), IFWT(A_1) - IFWT(A_0))$

异或卷积

当 $\oplus = xor$ 的时候

$FWT(A) = (FWT(A_0) + FWT(A_1), FWT(A_0) - FWT(A_1))$

$FWT(A) = A$(长度为 1)

$IFWT(A) = (\frac{IFWT(A_0)+IFWT(A_1)}{2}, \frac{IFWT(A_0)-IFWT(A_1)}{2})$

```
namespace Polynomial {
    void FWT_or(int *A, int n, int type) {
        for(int m = 1; m < n; m <<= 1) {
            for(int i = 0; i < n; i += 2 * m) {
                for(int j = 0; j < m; j++) {
                    A[i + j + m] = (1ll * A[i + j + m] + A[i + j] * type + mod) % mod;
                }
            }
        }
    }
    void FWT_and(int *A, int n, int type) {
        for(int m = 1; m < n; m <<= 1) {
            for(int i = 0; i < n; i += 2 * m) {
                for(int j = 0; j < m; j++) {
                    A[i + j] = (1ll * A[i + j + m] * type + A[i + j] + mod) % mod;
                }
            }
        }
    }
    void FWT_xor(int *A, int n, int type) {
        int inv_2 = Pow(2, mod - 2);
        for(int m = 1; m < n; m <<= 1) {
            for(int i = 0; i < n; i += 2 * m) {
                for(int j = 0; j < m; j++) {
                    int x = A[i + j], y = A[i + j + m];
                    A[i + j] = (1ll * x + y) * (type == 1 ? 1 : inv_2) % mod;
                    A[i + j + m] = (1ll * x - y + mod) * (type == 1 ? 1 : inv_2) % mod;
                }
            }
        }
    }
}
```

子集卷积

子集卷积求的是下面一个式子:

$$c_k = \sum_{i|j=k, i\&j=0} a_i \times b_j$$

就是把集合 k 划分成两个集合。

后面那个与的条件通过 $|k| = |i| + |j|$ 干掉，加一维集合元素个数，就变成了

$$c[i+j][mask_k] = \sum_{i|j=k} a[i][mask_i] \times b[j][mask_j]$$

这个可以用 FWT 算。

```
namespace ssc{
    int f[21][1 << 21], g[21][1 << 21], ans[21][1 << 21];
    void subset_convolution(int *A, int *B, int *C, int n, int lim) {
    //  memset(f, 0, sizeof(f));
    //  memset(g, 0, sizeof(g));
        for(int i = 0; i < lim; i++) f[__builtin_popcount(i)][i] = A[i];
        for(int i = 0; i < lim; i++) g[__builtin_popcount(i)][i] = B[i];
        for(int i = 0; i <= n; i++) FWT_or(f[i], lim, 1), FWT_or(g[i], lim, 1);
        for(int i = 0; i <= n; i++)
            for(int j = 0; j <= i; j++)
                for(int k = 0; k < lim; k++)
                    ans[i][k] = (ans[i][k] + 1ll * f[j][k] * g[i - j][k] % mod) % mod;
        for(int i = 0; i <= n; i++) FWT_or(ans[i], lim, -1);
        for(int i = 0; i < lim; i++) C[i] = ans[__builtin_popcount(i)][i];
```

```
    }
}
```

群论

结论

1. **子群检验法**: 群 G 是群 H 的子群的充分必要条件：对于所有元素 h,g，只需检查 $g^{-1} \cdot h \in H$。

BurnSide 引理

定义 AB 同构为在群 G 中存在一个运算 f 使得 $f(A) = B$，则本质不同的元素个数为

$$\frac{\sum_{f \in G} c(f)}{|G|}$$

$c(f)$ 为 $\sum[f(A) == A]$，也就是 f 的不动点数量。

Polya 定理

在 BurnSide 的基础上，染色数为 m，则本质不同的染色方案数为

$$\frac{\sum_{f \in G} m^{cnt_f}}{|G|}$$

cnt_f 为置换 f 的循环节个数
> 在 Burnside 的计算不动点过程中，如果两个状态置换后相同，那么同一个子循环置换中颜色一定相同，不同子循环置换中颜色选取独立。

项链计数问题:
一个 n 元环，m 染色，旋转同构，方案数为:

$$\frac{\sum_{i=1}^{n} m^{gcd(n,i)}}{n}$$

线性代数

矩阵运算全家桶

```cpp
struct mat {
    int g[5][5], n, m;
};
void operator+=(mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "+= size error" << endl, exit(0);
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            a.g[i][j] = (a.g[i][j] + b.g[i][j]);
            if(a.g[i][j] >= mod) a.g[i][j] -= mod;
        }
}
void operator-=(mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "-= size error" << endl, exit(0);
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            a.g[i][j] -= b.g[i][j];
            if(a.g[i][j] < 0) a.g[i][j] += mod;
        }
}
mat operator+(const mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "+ size error" << endl, exit(0);
    mat c;
    c.n = a.n; c.m = a.m;
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            c.g[i][j] = (a.g[i][j] + b.g[i][j]);
```

```
            if(c.g[i][j] >= mod) c.g[i][j] -= mod;
        }
    return c;
}
mat operator-(const mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "- size error" << endl, exit(0);
    mat c;
    c.n = a.n; c.m = a.m;
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            c.g[i][j] = (a.g[i][j] - b.g[i][j]);
            if(c.g[i][j] < 0) c.g[i][j] += mod;
        }
    return c;
}
mat operator*(const mat &a, const mat &b) {
    if(a.m != b.n) cerr << "* size error" << endl, exit(0);
    mat c;
    c.n = a.n; c.m = b.m;
    for(int i = 1; i <= a.n; i++) {
        for(int j = 1; j <= b.m; j++) {
            c.g[i][j] = 0;
            for(int k = 1; k <= a.m; k++) {
                c.g[i][j] = c.g[i][j] + 1ll * a.g[i][k] * b.g[k][j] % mod;
                if(c.g[i][j] >= mod) c.g[i][j] -= mod;
            }
        }
    }
    return c;
}
mat Pow(mat a, int p) {
    if(a.n != a.m) cerr << "* size error" << endl, exit(0);
    mat ans;
    ans.n = ans.m = a.n;
    memset(ans.g, 0, sizeof(ans.g));
    for(int i = 1; i <= ans.n; i++) ans.g[i][i] = 1;
    for(; p; p >>= 1, a = a * a)
        if(p & 1)
            ans = ans * a;
    return ans;
}
```

高斯消元

```
namespace Gauss {
    int n, m;
    double g[N][N];
    int iszero(double x) {return fabs(x) < eps;}
    void exchange(int i, int j) {
        for(int k = 1; k <= m; k++)
            swap(g[i][k], g[j][k]);
    }
    void minus(int i, int j, double t) {
        for(int k = 1; k <= m; k++)
            g[j][k] -= g[i][k] * t;
    }
    void div(int i, double d) {
        for(int k = 1; k <= m; k++)
            g[i][k] /= d;
    }
    void solve() {
        for(int i = 1; i <= n; i++) {
            if(iszero(g[i][i])) {
                for(int j = i + 1; j <= n; j++) {
                    if(!iszero(g[j][i])) {
                        exchange(i, j);
                        break;
                    }
                }
                if(iszero(g[i][i])) continue;
            }
```

```
            div(i, g[i][i]);
            for(int j = 1; j <= n; j++) if(i != j && !iszero(g[j][i])){
                minus(i, j, g[j][i]);
            }
        }
    }
}
```

图论

树论

树的直径

模板：POJ - 1985

- 两遍 DFS

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

- 树形 DP

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

```
        E[y].push_back((Edge) {x, w});
    }
    dfs(1, 0);
    printf("%d\n", d);
    return 0;
}
```

求 LCA

- 树链剖分

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

- 倍增

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

```
            }
        return fa[x][0];
    }
}
```

树上启发式合并

长春站的痛.jpg

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

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

}
```

图论

第 k 短路

模板：HDU-6351

估值函数：$h(x) = f(x) + g(x)$，其中 $f(x)$ 为从起点到现在的距离，$g(x)$ 为起点到当前点的最短路。

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

```
        if(dis[x] < d) continue;
        for(int i = 0; i < re[x].size(); i++) {
            int y = re[x][i].y, w = re[x][i].w;
            if(dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                q.push((node) {y, dis[y], 0});
            }
        }
    }
    for(int i = 1; i <= n; i++) cnt[i] = k;
    cnt[s]++;
    q.push((node) {s, 0, dis[s]});
    while(q.size()) {
        int x = q.top().x, f = q.top().f, g = q.top().g;
        q.pop();
        if(cnt[x] == 0) continue;
        cnt[x]--;
        if(x == t && cnt[x] == 0) {
            printf("%lld\n", f);
            return 0;
        }
        for(int i = 0; i < E[x].size(); i++) {
            int y = E[x][i].y, w = E[x][i].w;
            q.push((node) {y, f + w, dis[y]});
        }
    }
    printf("-1\n");
    return 0;
}
```

二分图匹配

结论

最大匹配数：最大匹配的匹配边的数目

最小点/边覆盖数：选取最少的点/边，使任意一条边至少有一个点被选择 / 点至少连有一条边。

最大独立数：选取最多的点，使任意所选两点均不相连

最小路径覆盖数：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以为 0（即单个点）。

1. 最大匹配数 = 最小点覆盖数（这是 Konig 定理）
2. 最大匹配数 = 最大独立数
3. 最小路径覆盖数 = 顶点数 - 最大匹配数
4. 原图的最大团 = 补图的最大独立集原图的最大独立集 = 补图的最大团
5. 最小边覆盖 = 顶点数 - 最大匹配数

在一般图中：

最小不相交路径覆盖：每个点拆点为 $2x-1, 2x$，那么一条边 (x, y)，则连边 $(2x-1, 2y)$，答案是 $n - maxmatch$

最小可相交路径覆盖：跑一遍传递闭包，按传递闭包上的边建边之后转化为最小不相交路径覆盖。

二分图最大匹配的必须边：

在完备匹配中：

匹配边从左到右方向，非匹配边从右到左方向，则一条边为必须边当且仅当边在最大匹配中，并且边所连的两个点**不在**同一个强连通分量中。

在非完备匹配中：

匈牙利算法

```
int dfs(int x) {
    for(int i = head[x]; i; i = nxt[i]) {
        int y = ver[i];
        if(vis[y]) continue;
        vis[y] = 1;
```

```
        if(!match[y] || dfs(match[y])) {
            match[y] = x;
            return true;
        }
    }
    return false;
}
for(int i = 1; i <= n; i++) {
    memset(vis, 0, sizeof(vis));
    if(dfs(i)) ans++;
}
```

KM 算法二分图最大权匹配

KM 算法只支持二分图最大权完美匹配，若图不一定存在完美匹配，注意补 0 边和补点。

KM 算法引入了顶标的概念，用 $la[x]$ 和 $lb[x]$ 分别保存两侧点的顶标，顶标必须满足大于所有边。
每次对每个点进行循环匹配，匹配中统计一个 $delta$ 表示最小的权值使得一条边可以加入。
然后修改顶标再继续匹配。

```
int la[N], lb[N], va[N], vb[N], delta, match[N], g[N][N], n;
int dfs(int x) {
    va[x] = 1;
    for(int y = 1; y <= n; y++) {
        if(!vb[y]) {
            if(la[x] + lb[y] - g[x][y] == 0) {
                vb[y] = 1;
                if(!match[y] || dfs(match[y])) {
                    match[y] = x;
                    return true;
                }
            } else delta = min(delta, la[x] + lb[y] - g[x][y]);
        }
    }
    return false;
}
void work() {
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            g[i][j] = read();
    memset(match, 0, sizeof(match));
    for(int i = 1; i <= n; i++) {
        la[i] = g[i][1];
        lb[i] = 0;
        for(int j = 2; j <= n; j++)
            la[i] = max(la[i], g[i][j]);
    }
    for(int i = 1; i <= n; i++) {
        while(true) {
            memset(va, 0, sizeof(va));
            memset(vb, 0, sizeof(vb));
            delta = 0x3f3f3f3f;
            if(dfs(i)) break;
            for(int j = 1; j <= n; j++) {
                if(va[j]) la[j] -= delta;
                if(vb[j]) lb[j] += delta;
            }
        }
    }
    long long ans = 0;
    for(int i = 1; i <= n; i++)
        ans += g[match[i]][i];
    printf("%lld\n", ans);
}
```

网络流

Dinic 算法

```
const int inf = 0x3f3f3f3f;
queue<int> q;
int d[N];
int bfs() {
    memset(d, 0, sizeof(int) * (t + 10)); d[s] = 1;
    while(q.size()) q.pop(); q.push(s);
    while(q.size()) {
        int x = q.front(); q.pop();
        for(int i = head[x]; i; i = nxt[i]) {
            if(d[ver[i]]) continue;
            if(edge[i] <= 0) continue;
            d[ver[i]] = d[x] + 1;
            q.push(ver[i]);
        }
    }
    return d[t];
}
int dinic(int x, int flow) {
    if(x == t) return flow;
    int k, res = flow;
    for(int i = head[x]; i && res; i = nxt[i]) {
        if(d[ver[i]] != d[x] + 1 || edge[i] <= 0) continue;
        k = dinic(ver[i], min(res, edge[i]));
        if(k == 0) d[ver[i]] = 0;
        edge[i] -= k;
        edge[i ^ 1] += k;
        res -= k;
    }
    return flow - res;
}
```

EK 算法费用流

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

```
}
```

无源汇上下界可行流

$x->y$，则 s 向 y，s 向 x 连 l,x 向 y 连 $r-l$，有可行流的条件是 s 出边全满流，解通过残量网络构造出。

```
for(int i = 1; i <= m; i++) {
    int x = read(), y = read();
    int l = read(), r = read();
    low[i] = l;
    add(x, y, r - l); add(y, x, 0);
    id[i] = tot;
    add(s, y, l); add(y, s, 0);
    add(x, t, l); add(t, x, 0);
}
while(bfs())
    dinic(s, inf);
int f = 1;
for(int i = head[s]; i; i = nxt[i]) {
    f &= (edge[i] == 0);
}
printf("%s\n", f ? "YES" : "NO");
if(!f) return 0;
for(int i = 1; i <= m; i++) {
    printf("%d\n", edge[id[i]] + low[i]);
}
```

连通性算法

Tarjan 强连通分量

$dfn[x]$：dfs 序。

$low[x]$：追溯值，指 x 的子树内部，通过一条非树边能到达的最小的 dfn 值。

如果 $dfn[x]==low[x]$，当前栈中，x 以后的元素为一个强连通。

```
void tarjan(int x) {
    low[x] = dfn[x] = ++dfncnt;
    s[++t] = x; vis[x] = 1;
    for(int i = head[x]; i; i = nxt[i]) {
        if(!dfn[ver[i]]) {
            tarjan(ver[i]);
            low[x] = min(low[x], low[ver[i]]);
        } else if(vis[ver[i]]) {
            low[x] = min(low[x], dfn[ver[i]]);
        }
    }
    if(dfn[x] == low[x]) {
        int z = -1;
        ++sc;
        while(z != x) {
            scc[s[t]] = sc;
            siz[sc]++;
            vis[s[t]] = 0;
            z = s[t];
            t--;
        }
    }
}
//从任意点开始跑，但是注意如果图不连通，需要每个点跑一次
for(int i = 1; i <= n; i++)
    if(!dfn[i])
        tarjan(i);
```

kosaraju 算法

先正着跑一边 dfs，在出栈的时候把点加入栈。
再倒着跑，此时栈顶是 DAG 的末端，每次跑到的点在同一个 scc 里面。

```cpp
pair<int, vector<int> > kosaraju(vector<vector<int> > ver) {
    int n = ver.size() - 1;
    vector<int> scc(n + 1);
    int cnt = 0;
    vector<vector<int> > rver(n + 1);
    for(int i = 1; i <= n; i++) {
        for(auto y : ver[i]) {
            rver[y].push_back(i);
        }
    }
    vector<int> vis(n + 1);
    stack<int> q;
    auto dfs1 = [&](auto &&me, int x) -> void {
        vis[x] = 1;
        for(auto y : ver[x]) if(!vis[y]) {
            me(me, y);
        }
        q.push(x);
    };
    auto dfs2 = [&](auto &&me, int x) -> void {
        vis[x] = 0;
        scc[x] = cnt;
        for(auto y : rver[x]) if(vis[y]) {
            me(me, y);
        }
    };
    for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(dfs1, i);
    while(q.size()) {
        int x = q.top();
        q.pop();
        if(vis[x]) {
            cnt++;
            dfs2(dfs2, x);
        }
    }
    return {cnt, scc};
}
vector<vector<int> > reduction(vector<vector<int> > ver, vector<int> scc, int scnt) {
    int n = ver.size() - 1;
    vector<vector<int> > sver(scnt + 1);
    map<pair<int, int>, int> M;
    for(auto x = 1; x <= n; x++) {
        for(auto y : ver[x]) {
            if(scc[x] != scc[y] && !M.count({scc[x], scc[y]})) {
                M[{scc[x], scc[y]}] = 1;
                sver[scc[x]].push_back(scc[y]);
            }
        }
    }
    return sver;
}
```

点双连通

Tarjan 割点判定

```cpp
int cut[N];
namespace v_dcc {
    int root, low[N], dfn[N], dfntot;
    void tarjan(int x) {
        low[x] = dfn[x] = ++dfntot;
        int flag = 0;
        for(int i = head[x]; i; i = nxt[i]) {
            int y = ver[i];
            if(!dfn[y]) {
                tarjan(y);
                low[x] = min(low[x], low[y]);
                if(low[y] >= dfn[x]) {
                    flag++;
                    if(x != root || flag > 1) cut[x] = 1;

```

```
                }
            } else low[x] = min(low[x], dfn[y]);
        }
    }
    void getcut() {
        for(int i = 1; i <= n; i++)
            if(!dfn[i])
                tarjan(root = i);
    }
}
```

求点双连通分量

点双连通分量比较复杂，一个点可能存在于多个点双连通分量当中，一个点删除与搜索树中的儿子节点断开时，不能在栈中弹掉父亲点，但是父亲点属于儿子的 v-dcc。

```
int cut[N];
vector<int> dcc[N];
namespace v_dcc {
    int s[N], t, root;
    int es[N], et;
    void tarjan(int x) {
        dfn[x] = low[x] = ++dfntot;
        s[++t] = x;
        if(x == root && head[x] == 0) {
            dcc[++dc].clear();
            dcc[dc].push_back(x);
            return ;
        }
        int flag = 0;
        for(int i = head[x]; i; i = nxt[i]) {
            int y = ver[i];
            if(!dfn[y]) {
                tarjan(y);
                low[x] = min(low[x], low[y]);
                if(low[y] >= dfn[x]) {
                    flag++;
                    if(x != root || flag > 1) cut[x] = true;
                    dcc[++dc].clear();
                    int z = -1;
                    while(z != y) {
                        z = s[t--];
                        dcc[dc].push_back(z);
                    }
                    dcc[dc].push_back(x);
                }
            } else low[x] = min(low[x], dfn[y]);
        }
    }
    void get_cut() {
        for(int i = 1; i <= n; i++)
            if(!dfn[i])
                tarjan(root = i);
    }
}
```

边双连通

搜索树上的点 x，若它的一个儿子 y，满足严格大于号 $low[y] > dfn[x]$，那么这条边就是桥。

注意由于会有重边，不能仅仅考虑他的父亲编号，而应该记录入边编号。

```
namespace e_dcc {
    int low[N], dfn[N], dfntot;
    vector<int> E[N];
    void tarjan(int x, int in_edge) {
        low[x] = dfn[x] = ++dfntot;
        for(int i = head[x]; i; i = nxt[i]) {
            int y = ver[i];
            if(!dfn[y]) {
                tarjan(y, i);
```

```
                low[x] = min(low[x], low[y]);
                if(low[y] > dfn[x])
                    bridge[i] = bridge[i ^ 1] = true;
            } else if(i != (in_edge ^ 1))
            //注意运算优先级
                low[x] = min(low[x], dfn[y]);
        }
    }
    void getbridge() {
        for(int i = 1; i <= n; i++)
            if(!dfn[i])
                tarjan(i, 0);
    }
    void dfs(int x) {
        dcc[x] = dc;
        for(int i = head[x]; i; i = nxt[i]) {
            if(!dcc[ver[i]] && !bridge[i]) {
                dfs(ver[i]);
            }
        }
    }
    void getdcc() {
        for(int i = 1; i <= n; i++) {
            if(!dcc[i]) {
                ++dc;
                dfs(i);
            }
        }
    }
    void getgraphic() {
        for(int x = 1; x <= n; x++) {
            for(int i = head[x]; i; i = nxt[i]) {
                if(dcc[ver[i]] != dcc[x]) {
                    E[dcc[x]].push_back(dcc[ver[i]]);
                    E[dcc[ver[i]]].push_back(dcc[x]);
                }
            }
        }
    }
}
```

2-SAT

2-SAT 用于解决每个变量的 01 取值问题，用于判断是否存在一种不冲突取值方法。

建边方法：假如选了 A 之后，B 的取值**确定**，那么就 A 的这个取值向 B 的这个取值建边，否则不要建边。

判定方法：如果，$\exists A$, 使得 A 和 $\neg A$ 在同一个强连通分量里面，说明不存在一种合法取值，否则存在。

输出方案：自底向上确定每个变量的取值, 由于 tarjan 求解强连通分量是自底向上，所以编号比较小的强连通是位于 DAG 底部的。

基于 tarjan 的方案输出就变得十分简单了，只要判断一个点和对立节点哪个 scc 的编号小就行了。

例如：$A-> B-> C$，那么 C 的编号最小。

```
for(int i = 1; i <= m; i++) {
    int x = read() + 1, y = read() + 1;
    int w = read();
    char c[10];
    scanf("%s", c + 1);
    if(c[1] == 'A') {
        if(w) {
            add(2 * x - 0, 2 * x - 1);
            add(2 * y - 0, 2 * y - 1);
        } else {
            add(2 * x - 1, 2 * y - 0);
            add(2 * y - 1, 2 * x - 0);
        }
    }
    if(c[1] == 'O') {
        if(w) {
```

```
            add(2 * x - 0, 2 * y - 1);
            add(2 * y - 0, 2 * x - 1);
        } else {
            add(2 * x - 1, 2 * x - 0);
            add(2 * y - 1, 2 * y - 0);
        }
    }
    if(c[1] == 'X') {
        if(w) {
            add(2 * x - 0, 2 * y - 1);
            add(2 * x - 1, 2 * y - 0);
            add(2 * y - 0, 2 * x - 1);
            add(2 * y - 1, 2 * x - 0);
        } else {
            add(2 * x - 0, 2 * y - 0);
            add(2 * x - 1, 2 * y - 1);
            add(2 * y - 0, 2 * x - 0);
            add(2 * y - 1, 2 * x - 1);
        }
    }
}
for(int i = 1; i <= 2 * n; i++)
    if(!dfn[i])
        tarjan(i);
for(int i = 1; i <= n; i++) {
    if(scc[2 * i - 0] == scc[2 * i - 1]) {
        printf("NO\n");
        return 0;
    }
}
printf("YES\n");
//2 * x - a -> 2 * y - b 的边表示，假如 x 取值为 a，那么 y 的取值必须为 b

//输出方案
for(int i = 2; i <= 2 * n; i += 2) {
    if(scc[i - 0] == scc[i - 1]) {
        printf("NO\n");
        return 0;
    } else ans[(i + 1) / 2] = scc[i - 1] < scc[i - 0];
}
```

计算几何

公式

三角形内心和重心公式（点为 A,B,C, 对边为 a,b,c）：
+ 内心：$\frac{aA+bB+cC}{a+b+c}$
+ 重心：$\frac{A+B+C}{3}$
+ 外心，垂心：用两直线交点计算

结构体定义

```
const double Pi = acos(-1.0);
const double eps = 1e-11;
// 三态函数
int sgn(double x) {
    if(fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}
struct line;
struct Point;
struct Point {
    double x, y;
    Point() : x(0), y(0) {}
    Point(double x, double y) : x(x), y(y) {}
};
struct line{
    Point s, t;
```

```
    line() {}
    line(const Point &s, const Point &t) : s(s), t(t) {}
};

struct circle{
    Point c;
    double r;
    circle() : c(Point(0,0)), r(0) {}
    circle(const Point &c, double r) : c(c), r(r) {}
    Point point(double a) {
        return Point(c.x + cos(a)*r, c.y + sin(a)*r);
    }
};
typedef Point Vector;
Point operator+(const Point &a, const Point &b) { return Point(a.x + b.x, a.y + b.y); }
Point operator-(const Point &a, const Point &b) { return Point(a.x - b.x, a.y - b.y); }
Point operator*(const Point &a, const double &c) { return Point(c * a.x, c * a.y); }
Point operator/(const Point &a, const double &c) { return Point(a.x / c, a.y / c); }
inline bool operator < (const Point &a, const Point &b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const Point& a, const Point& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
// 点积
double dot(const Vector& a, const Vector& b) { return a.x * b.x + a.y * b.y; }
// 叉积
double det(const Vector& a, const Vector& b) { return a.x * b.y - a.y * b.x; }
double cross(const Point& s, const Point& t, const Point& o = Point()) { return det(s - o, t - o); }
```

基本操作

```
// 点到原点距离
double abs(const Point &a){ return sqrt(a.x * a.x + a.y * a.y); }
// 点旋转 theta 角度
Point rot(const Point &a, double theta){ return Point(a.x * cos(theta) - a.y * sin(theta), a.x * sin(theta) + a.y *
 ↪  cos(theta)); }
// 逆时针旋转 90 度
Point rotCCW90(const Point &a) { return Point(-a.y, a.x); }
// 顺时针旋转 90 度
Point rotCW90(const Point &a) { return Point(a.y, -a.x); }
// 点的幅角`
double arg(const Point &a){
    double t = atan2(a.y, a.x);
    return t < 0 ? t + 2 * Pi:t;
}
//极角排序
// 1 浮点数坐标排序
// 顺序（象限）: 3 -> 4 -> 1 -> 2
int cmp(const node &a, const node &b) {
    if(atan2(a.y, a.x) != atan2(b.y, b.x)) {
        return atan2(a.y, a.x) < atan2(b.y, b.x);
    }
    return a.x < b.x;
}
// 2 整数坐标排序
// 顺序（象限）: 1 -> 2 -> 3 -> 4
int up(const node &a) {
    return a.y > 0 || (a.y == 0 && a.x >= 0);
}
int cmp(const node &a, const node &b) {
    if(up(a) != up(b)) return up(a) > up(b);
    return det(a, b) > 0;
}
```

线

```
// 是否平行
bool parallel(const line &a, const line &b) {
    return !sgn(det(a.t - a.s, b.t - b.s));
}
// 直线是否相等
```

```cpp
bool l_eq(const line& a, const line& b) {
    return parallel(a, b) && parallel(line(a.s, b.t), line(b.s, a.t));
}
```

点与线

```cpp
// 点是否在线段上，<= 包含端点
bool p_on_seg(const Point &p, const Line &seg) {
    return !sgn(det(p - seg.s, p - seg.t)) && sgn(dot(p - seg.s, p - seg.t)) <= 0;
}
// 点到直线距离
double dist_to_line(const Point &p, const Line &l) {
    return fabs(det(l.s - p, l.t - p)) / (l.s - l.t).len();
}
// 点到线段距离
double dist_to_seg(const Point &p, const line &l) {
    if (l.s == l.t) return abs(p - l.s);
    Vector vs = p - l.s, vt = p - l.t;
    if (sgn(dot(Point(l), vs)) < 0) return abs(vs);
    else if (sgn(dot(Point(l), vt)) > 0) return abs(vt);
    else return dist_to_line(p, l);
}
```

线与线

```cpp
// 直线交点，需保证存在
Point l_intersection(const Line& a, const Line& b) {
    double s1 = det(a.t - a.s, b.s - a.s), s2 = det(a.t - a.s, b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}
// 线段和直线是否有交   1 = 规范，2 = 不规范 >= 1 表示相交
int s_l_cross(const Line &seg, const Line &line) {
    int d1 = sgn(det(line.s - seg.s, line.t - seg.s));
    int d2 = sgn(det(line.s - seg.t, line.t - seg.t));
    if ((d1 ^ d2) == -2) return 1; // proper
    if (d1 == 0 || d2 == 0) return 2;
    return 0;
}
// 线段的交   1 = 规范，2 = 不规范   >= 1 表示相交
// 如果是不规范相交，p_on_seg 函数要改成 <=
int s_cross(const Line &a, const Line &b, Point &p) {
    int d1 = sgn(det(a.t - a.s, b.s - a.s)), d2 = sgn(det(a.t - a.s, b.t - a.s));
    int d3 = sgn(det(b.t - b.s, a.s - b.s)), d4 = sgn(det(b.t - b.s, a.t - b.s));
    if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) { p = l_intersection(a, b); return 1; }
    if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
    if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
    if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
    if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
    return 0;
}
```

多边形

```cpp
#define nxt(i) ((i + 1) % s.size())
typedef vector<Point> Polygon;
// 多边形面积
double poly_area(const Polygon &s){
    double area = 0;
    for(int i = 1; i < s.size() - 1; i++)
        area += cross(s[i], s[i + 1], s[0]);
    return area / 2;
}
// 点是否在多边形中 0 = 在外部 1 = 在内部 -1 = 在边界上
int p_in_poly(Point p, const vector<Point> &s){
    int cnt = 0;
    for(int i = 0; i < s.size(); i++) {
        Point a = s[i], b = s[nxt(i)];
        if (p_on_seg(p, line(a, b))) return -1;
        //p 在多边形边上
```

```
        if (sgn(a.y - b.y) <= 0) swap(a, b);
        if (sgn(p.y - a.y) > 0) continue;
        if (sgn(p.y - b.y) <= 0) continue;
        //一条边包含它较高的点，不包含较低的点
        cnt += sgn(det(b - p, a - p)) > 0;
        //如果 p 在这条线段左边
    }
    return bool(cnt & 1);
}
```

凸包

andrew 算法，

```
// 构建凸包 点不可以重复 < 0 边上可以有点， <= 0 则不能
// 会改变输入点的顺序
vector<Point> Convex_hull(vector<Point> &s) {
    sort(s.begin(), s.end());
    vector<Point> ret(s.size() * 2);
    int sz = 0;
    for(int i = 0; i < s.size(); i++) {
        while(sz > 1 && sgn(det(ret[sz - 1] - ret[sz - 2], s[i] - ret[sz - 2])) <= 0) sz--;
        ret[sz++] = s[i];
    }
    int k = sz;
    for(int i = s.size() - 2; i >= 0; i--) {
        while(sz > k && sgn(det(ret[sz - 1] - ret[sz - 2], s[i] - ret[sz - 2])) <= 0) sz--;
        ret[sz++] = s[i];
    }
    ret.resize(sz - (s.size() > 1));
    return ret;
}
```

旋转卡壳

用平行线夹多边形，根据两个向量的叉积判断支点变化

```
int cmp(const Point &a, const Point &b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
double rotatingCalipers(const vector<Point> &s) {
    if(s.size() == 2) return abs(s[1] - s[0]);
    int i = 0, j = 0;
    for(int k = 0; k < s.size(); k++) {
        if( cmp(s[i], s[k])) i = k;
        if(!cmp(s[j], s[k])) j = k;
    }
    double ans = 0;
    int si = i, sj = j;
    do{
        ans = max(ans, abs(s[i] - s[j]));
        if(sgn(det(s[(i + 1) % s.size()] - s[i], s[(j + 1) % s.size()] - s[j])) < 0)
            i = (i + 1) % s.size();
        else j = (j + 1) % s.size();
    } while(i != si || j != sj);
    return ans;
}
```

半平面交

(多边形面积交)

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e6 + 1009;
const double eps = 1e-9;
int sgn(double x) {
    if(-eps < x && x < eps) return 0;
    else return x > 0 ? 1 : -1;
}
```

```cpp
struct Point {
    double x, y;
    int quad() const {return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) >= 0);}
    Point(double a, double b) : x(a), y(b) {}
};
Point operator-(const Point &a, const Point &b) {return {a.x - b.x, a.y - b.y};}
Point operator+(const Point &a, const Point &b) {return {a.x + b.x, a.y + b.y};}
Point operator*(const Point &a, double b) {return {a.x * b, a.y * b};}
Point operator*(double b, const Point &a) {return {a.x * b, a.y * b};}
Point operator/(double b, const Point &a) {return {a.x / b, a.y / b};}
Point operator/(const Point &a, double b) {return {a.x / b, a.y / b};}
double dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}
double det(const Point &a, const Point &b) {
    return a.x * b.y - a.y * b.x;
}
struct Line {
    Point s, t;
    bool include(const Point &p) const { return sgn(det(t - s, p - s)) > 0;}
    Line(Point a, Point b) : s(a), t(b) {}
};
Point l_intersection(const Line& a, const Line& b) {
    double s1 = det(a.t - a.s, b.s - a.s), s2 = det(a.t - a.s, b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1);
}
bool operator<(const Point &a, const Point &b) {
    if(a.quad() != b.quad()) {
        return a.quad() < b.quad();
    }
    return sgn(det(a, b)) > 0;
}
bool parallel(const Line &a, const Line &b) {
    return !sgn(det(a.t - a.s, b.t - b.s));
}
bool sameDir(const Line &a, const Line &b) {
    return parallel(a, b) && (sgn(dot(a.t - a.s, b.t - b.s)) == 1);
}
bool operator<(const Line &a, const Line &b) {
    if(sameDir(a, b)) {
        return b.include(a.s);
    } else {
        return (a.t - a.s) < (b.t - b.s);
    }
}
bool check(const Line &u, const Line &v, const Line &w) {
    return w.include(l_intersection(u, v));
}
vector<Point> half_intersection(vector<Line> &l) {
    sort(l.begin(), l.end());
    deque<Line> q;
    for(int i = 0; i < l.size(); i++) {
        if(i && sameDir(l[i], l[i - 1])) {
            continue;
        }
        while(q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i])) q.pop_back();
        while(q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
        q.push_back(l[i]);
    }
    while(q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
    while(q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front();
    vector<Point> ret;
    for (int i = 0; i < q.size(); i++) ret.push_back(l_intersection(q[i], q[(i + 1) % q.size()]));
    return ret;
}
int n, m;
void work() {
    scanf("%d", &n);
    vector<Line> v;
    for(int i = 1; i <= n; i++) {
        scanf("%d", &m);
```

```
        vector<Point> p;
        for(int j = 1; j <= m; j++) {
            double x, y;
            scanf("%lf%lf", &x, &y);
            p.push_back({x, y});
        }
        for(int j = 0; j < m; j++) {
            v.push_back({p[j], p[(j + 1) % m]});
        }
    }
    vector<Point> p = half_intersection(v);
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        area += det(p[i], p[(i + 1) % p.size()]);
    }
    printf("%.3f\n", fabs(area) / 2);

}
```

圆

直线和圆

```
// 直线与圆交点
vector<Point> l_c_intersection(const line &l, const circle &o) {
    vector<Point> ret;
    Point b = l.t - l.s, a = l.s - o.c;
    double x = dot(b, b), y = dot(a, b), z = dot(a, a) - o.r * o.r;
    double D = y * y - x * z;
    if (sgn(D) < 0) return ret;
    ret.push_back(o.c + a + b * (-y + sqrt(D + eps)) / x);
    if (sgn(D) > 0) ret.push_back(o.c + a + b * (-y - sqrt(D)) / x);
    return ret;
}
// 点到圆的切点
vector<Point> p_c_tangent(const Point &p, const circle &o) {
    vector<Point> ret;
    double d = abs(p - o.c), x = dot(p - o.c, p - o.c) - o.r * o.r;
    if(sgn(x) < 0) ;
    else if(sgn(x) == 0) ret.push_back(p);
    else {
        Vector base = p + (o.c - p) * x / dot(p - o.c, p - o.c);
        Vector e = rotCW90(o.c - p) / d;
        ret.push_back(base + e * sqrt(x) * o.r / d);
        ret.push_back(base - e * sqrt(x) * o.r / d);
    }
    return ret;
}
// 圆与圆的交点
vector<Point> c_c_intersection(const circle &a, const circle &b) {
    vector<Point> ret;
    double d = abs(b.c - a.c);
    if(sgn(d) == 0 || sgn(d - a.r - b.r) > 0 || sgn(d + min(a.r, b.r) - max(a.r, b.r)) < 0)
        return ret;
    double x = (a.r * a.r + dot(b.c - a.c, b.c - a.c) - b.r * b.r) / (2 * d);
    double y = sqrt(a.r * a.r - x * x);
    Point v = (b.c - a.c) / d;
    ret.push_back(a.c + v * x + rotCW90(v) * y);
    if(sgn(y) > 0) ret.push_back(a.c + v * x - rotCW90(v) * y);
    return ret;
}
```

在 res 中存放的线上的两点分别是在 c1,c2 上的切点。

```
int tangent(const circle &C1, const circle &C2, vector<line> &res){
    double d = abs(C1.c - C2.c);
    if(d < eps) return 0;

    int c=0;
    // 内公切线
    if(C1.r + C2.r < d - eps){
```

```
        double t = acos((C1.r + C2.r) / d);
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c), t), C2.c + rot(C2.r / d * (C1.c - C2.c), t)));
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c),-t), C2.c + rot(C2.r / d * (C1.c - C2.c),-t)));
        c += 2;
    } else if(C1.r + C2.r < d + eps){
        Point p = C1.c + C1.r / d * (C2.c - C1.c);
        res.push_back(line(p, p + rot(C2.c - C1.c, Pi / 2)));
        c++;
    }

    // 外公切线
    if(abs(C1.r - C2.r) < d - eps){
        double t1 = acos((C1.r - C2.r) / d), t2 = Pi - t1;
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c), t1), C2.c + rot(C2.r / d * (C1.c - C2.c),-t2)));
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c),-t1), C2.c + rot(C2.r / d * (C1.c - C2.c), t2)));
        c+=2;
    } else if(abs(C1.r - C2.r) < d + eps){
        Point p = C1.c + C1.r / d * (C2.c - C1.c);
        res.push_back(line(p, p + rot(C2.c - C1.c, Pi / 2)));
        c++;
    }

    return c;
}
```

最小圆覆盖

```
Line norm(const Line &a) {
    Point mid = (a.t + a.s) / 2;
    Point dir = a.t - a.s;
    dir = {dir.y, -dir.x};
    return Line(mid, mid + dir);
}
double abs(const Point &a, const Point &b) {
    return sqrt(dot(a - b, a - b));
}
struct Circle {
    Point o;
    double r;
    Circle(const Point &o, const double &r) : o(o), r(r) {}
    Circle(const Point &a, const Point &b, const Point &c) : o(0, 0) {
        Line ab = norm(Line(b, a));
        Line bc = norm(Line(b, c));
        o = l_intersection(ab, bc);
        r = abs(o, a);
    }
};

int n, m, a[N];
vector<Point> v;
void work() {
    cin >> n;
    srand(12391278);
    for(int i = 1; i <= n; i++) {
        double x, y;
        cin >> x >> y;
        v.push_back({x, y});
    }
    random_shuffle(v.begin(), v.end());
    Circle ans = {v[0], 0};
    for(int i = 1; i < n; i++) {
        if(abs(ans.o, v[i]) > ans.r + eps) {
            ans = {v[i], 0};
            for(int j = 0; j < i; j++) {
                if(abs(ans.o, v[j]) > ans.r + eps) {
                    ans = {(v[i] + v[j]) / 2, abs(v[i], v[j]) / 2};
                    for(int k = 0; k < j; k++) {
                        if(abs(ans.o, v[k]) > ans.r + eps) {
                            ans = Circle(v[i], v[j], v[k]);
                        }
                    }
```

```
                }
            }
        }
    }
    cout << fixed << setprecision(10) << ans.o.x << " " << ans.o.y << " " << ans.r << endl;
}
```

三维几何

三维凸包（最小面积）

```
const double eps = 1e-10;
double rand_eps() {
    return ((double)rand() / RAND_MAX - 0.5) * eps;
}
struct Point {
    double x, y, z;
    Point(double x, double y, double z) : x(x), y(y), z(z) {}
    double len() {
        return sqrt(x * x + y * y + z * z);
    }
    void shake() {
        x += rand_eps();
        y += rand_eps();
        z += rand_eps();
    }
};
Point operator+(const Point &a, const Point &b) { return {a.x + b.x, a.y + b.y, a.z + b.z}; }
Point operator-(const Point &a, const Point &b) { return {a.x - b.x, a.y - b.y, a.z - b.z}; }
Point operator*(const Point &a, const double &b) { return {a.x * b, a.y * b, a.z * b}; }
Point operator/(const Point &a, const double &b) { return {a.x / b, a.y / b, a.z / b}; }
double dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
Point det(const Point &a, const Point &b) {
    return {a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x};
}
vector<Point> p;
struct Plane {
    int v[3];
    Plane(const int &a, const int &b, const int &c) : v{a, b, c} {}
    Point norm() {
        return det(p[v[0]] - p[v[1]], p[v[2]] - p[v[1]]);
    }
    int above(const Point &q) {
        return dot((q - p[v[0]]), norm()) >= 0;
    }
    double area() {
        return norm().len() / 2;
    }
};
int n;
vector<Plane> convex_hull_3d() {
    for(int i = 0; i < p.size(); i++) p[i].shake();
    vector<Plane> plane, tmp;
    vector<vector<int> > vis(n);
    for(int i = 0; i < n; i++)
        vis[i].resize(n);
    plane.emplace_back(0, 1, 2);
    plane.emplace_back(0, 2, 1);
    for(int i = 3; i < n; i++) {
        tmp = plane;
        plane.clear();
        for(auto t : tmp) {
            bool f = t.above(p[i]);
            if(!f) plane.push_back(t);
            for(int k = 0; k < 3; k++) {
                vis[t.v[k]][t.v[(k + 1) % 3]] = f;
            }
        }
        for(auto t : tmp) {
```

```
            for(int k = 0; k < 3; k++) {
                int a = t.v[k], b = t.v[(k + 1) % 3];
                if(vis[a][b] != vis[b][a] && vis[a][b]) {
                    plane.push_back(Plane{a, b, i});
                }
            }
        }
    }
    return plane;

}
void work() {
    srand(1230213);
    cin >> n;
    for(int i = 0; i < n; i++) {
        double x, y, z;
        cin >> x >> y >> z;
        p.push_back({x, y, z});
    }
    vector<Plane> convex = convex_hull_3d();
    double area = 0;
    for(auto x : convex) {
        area += x.area();
    }
    cout << fixed << setprecision(6) << area << endl;
}
```

tips

- atan2(y, x) 函数：点 (x, y) 到原点的方位角，值域在 $(-\pi, \pi)$ 在一二象限为正，三四象限为负。

字符串

Lemma

1. 若 r 是 S 的 border，则 $|S| - r$ 是字符串的周期。

2. 若 p, q 是 S 的周期，且 $p + q - gcd(p, q) \leq |S|$，那么 $gcd(p, q)$ 是 S 的周期。
3. 若 S 存在一个匹配 u, **满足**$2|u| \geq |S|$，S 中所有匹配位置构成一个等差数列。并且若存在超过 3 个匹配，则这个等差数列的公差 d **等于**u 的最小周期 $per(u)$，并且 $per(u) \leq |u|/2$

最小表示法

```
int k = 0, i = 0, j = 1;
while (k < n && i < n && j < n) {
  if (sec[(i + k) % n] == sec[(j + k) % n]) {
    k++;
  } else {
    sec[(i + k) % n] > sec[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
    if (i == j) i++;
    k = 0;
  }
}
i = min(i, j);
```

双哈希

```
struct Hash {
    int a, b;
    Hash() : a(0), b(0) {}
    Hash(int x) : a(x), b(x) {}
    Hash operator+(Hash x) {
        Hash tmp;
        tmp.a = (a + x.a >= p1) ? a + x.a - p1 : a + x.a;
        tmp.b = (b + x.b >= p2) ? b + x.b - p2 : b + x.b;
        return tmp;
```

```
    }
    Hash operator-(Hash x) {
        Hash tmp;
        tmp.a = (a - x.a < 0) ? a - x.a + p1 : a - x.a;
        tmp.b = (b - x.b < 0) ? b - x.b + p2 : b - x.b;
        return tmp;
    }
    Hash operator*(Hash x) {
        Hash tmp;
        tmp.a = (long long) a * x.a % p1;
        tmp.b = (long long) b * x.b % p2;
        return tmp;
    }
    Hash operator*(int x) {
        Hash tmp;
        tmp.a = (long long) a * x % p1;
        tmp.b = (long long) b * x % p2;
        return tmp;
    }
    bool operator==(Hash x) {
        return a == x.a && b == x.b;
    }
};
```

字串哈希

```
namespace String {
    const int x = 133;
    const int p1 = 1e9 + 7, p2 = 1e9 + 9;
    ull xp1[N], xp2[N], xp[N];
    void init_xp() {
        xp1[0] = xp2[0] = xp[0] = 1;
        for(int i = 1; i < N; i++) {
            xp1[i] = xp1[i - 1] * x % p1;
            xp2[i] = xp2[i - 1] * x % p2;
            xp[i] = xp[i - 1] * x;
        }
    }
    struct HashString {
        char s[N];
        int length, subsize;
        bool sorted;
        ull h[N], hl[N];
        ull init(const char *t) {
            if(xp[0] != 1) init_xp();
            length = strlen(t);
            strcpy(s, t);
            ull res1 = 0, res2 = 0;
            h[length] = 0;
            for(int j = length - 1; j >= 0; j--) {
            #ifdef ENABLE_DOUBLE_HASH
                res1 = (res1 * x + s[j]) % p1;
                res2 = (res2 * x + s[j]) % p2;
                h[j] = (res1 << 32) | res2;
            #else
                res1 = res1 * x + s[j];
                h[j] = res1;
            #endif
                }
            return h[0];
        }
        //获取子串哈希，左闭右开
        ull get_substring_hash(int left, int right)  {
            int len = right - left;
        #ifdef ENABLE_DOUBLE_HASH
            unsigned int mask32 = ~(0u);
            ull left1 = h[left] >> 32, right1 = h[right] >> 32;
            ull left2 = h[left] & mask32, right2 = h[right] & mask32;
            return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
                   (((left2 - right2 * xp2[len] % p2 + p2) % p2));
        #else
```

```
            return h[left] - h[right] * xp[len];
        #endif
        }
        void get_all_subs_hash(int sublen) {
            subsize = length - sublen + 1;
            for (int i = 0; i < subsize; ++i)
                hl[i] = get_substring_hash(i, i + sublen);
            sorted = 0;
        }

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

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

后缀数组

sa[i] 定义为字典序排名第 i 的后缀的起始位置。
rk[i] 定义为以 i 起始的后缀的排名。
height[i] 定义为 sa[i] 和 sa[i - 1] 的最长公共前缀的长度。
定义 h[i] 为排名第 i 的字符串和排名第 i - 1 的字符串的最长公共前缀长度。
有 height[i]>=height[i - 1] - 1。
$lcp(sa[i], sa[j]) = min(lcp(sa[i], sa[k]), lcp(sa[k], sa[j]))\forall sa[i] \le sa[k] \le sa[j]$
$lcp(sa[i], sa[j]) = min\{height[i+1, \cdots, j]\}$

```
int sa[N], id[N], rk[N], oldrk[N], cnt[N], height[N];
void get_sa(char *c) {
    n = strlen(c + 1);
    m = 300;
    for(int i = 1; i <= n; i++) cnt[rk[i] = c[i]]++;
    for(int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
    for(int i = n; i >= 1; i--) sa[cnt[rk[i]]--] = i;
    for(int w = 1; w < n; w <<= 1) {
        int tt = 0;
        for(int i = 1; i <= w; i++) id[++tt] = i + n - w;
        for(int i = 1; i <= n; i++) if(sa[i] > w) {
                id[++tt] = sa[i] - w;
            }
        memset(cnt, 0, sizeof(int) * (m + 1));
        for(int i = 1; i <= n; i++) cnt[rk[i]]++;
        for(int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
        for(int i = n; i >= 1; i--) sa[cnt[rk[id[i]]]--] = id[i];
        memcpy(oldrk, rk, sizeof(int) * (n + 1));
        int p = 1;
        rk[sa[1]] = 1;
        for(int i = 2; i <= n; i++) {
            rk[sa[i]] = (oldrk[sa[i]] == oldrk[sa[i - 1]] && oldrk[sa[i] + w] == oldrk[sa[i - 1] + w]) ? p : ++p;
        }
        if(p == n) break;
        m = p;
    }
}
void get_height(char *c) {
    for(int i = 1, k = 0; i <= n; i++) {
        if(rk[i] == 1) continue;
        if(k) k--;
        int j = sa[rk[i] - 1];
        while(i + k <= n && j + k <= n && c[i + k] == c[j + k]) k++;
        height[rk[i]] = k;
    }
}
```

SA-IS

会开两倍空间，如果传入的是整数数组，字符集请从 1 开始，不能出现 0。

```cpp
template<size_t size>
struct suffix_array {
    bool type[size << 1];
    int ht[size], rk[size], sa[size], bk[size], bk1[size];

    inline bool isLMS(int i, bool *type) {
        return (i > 0 && type[i] && !type[i - 1]);
    }

    template<class T>
    inline void induced_sort(T s, int *sa, int len, int bs, int sigma,
                             bool *type, int *bk, int *cb, int *p) {
        memset(bk, 0, sizeof(int) * sigma), memset(sa, -1, sizeof(int) * len);
        for (int i = 0; i < len; ++i) bk[s[i]]++;
        cb[0] = bk[0];
        for (int i = 1; i < sigma; ++i) cb[i] = cb[i - 1] + bk[i];
        for (int i = bs - 1; i >= 0; --i) sa[--cb[s[p[i]]]] = p[i];
        for (int i = 1; i < sigma; ++i) cb[i] = cb[i - 1] + bk[i - 1];
        for (int i = 0; i < len; ++i)
            if (sa[i] > 0 && !type[sa[i] - 1])
                sa[cb[s[sa[i] - 1]]++] = sa[i] - 1;
        cb[0] = bk[0];
        for (int i = 1; i < sigma; ++i) cb[i] = cb[i - 1] + bk[i];
        for (int i = len - 1; i >= 0; --i)
            if (sa[i] > 0 && type[sa[i] - 1])
                sa[--cb[s[sa[i] - 1]]] = sa[i] - 1;
    }

    template<class T>
    inline void sais(T s, int *sa, int len, bool *type, int *bk,
                     int *bk1, int sigma) {
        int p = -1, j, cnt = 0, bs = 0, *cb = bk + sigma, x;
        type[len - 1] = true;
        for (int i = len - 2; i >= 0; --i)
            type[i] = (s[i] < s[i + 1] || (s[i] == s[i + 1] && type[i + 1]));
        for (int i = 1; i < len; ++i)
            if (type[i] && !type[i - 1]) bk1[bs++] = i;
        induced_sort(s, sa, len, bs, sigma, type, bk, cb, bk1);
        for (int i = bs = 0; i < len; ++i)
            if (isLMS(sa[i], type)) sa[bs++] = sa[i];
        for (int i = bs; i < len; ++i) sa[i] = -1;
        for (int i = 0; i < bs; ++i) {
            x = sa[i];
            for (int j = 0; j < len; ++j) {
                if (p == -1 || s[x + j] != s[p + j] || type[x + j]
                                                    != type[p + j]) { cnt++, p = x; break; }
                else if (j > 0 && (isLMS(p + j, type)
                                   || isLMS(x + j, type))) break;
            }
            x >>= 1, sa[bs + x] = cnt - 1;
        }
        for (int i = j = len - 1; i >= bs; --i)
            if (sa[i] >= 0) sa[j--] = sa[i];
        int *s1 = sa + len - bs, *bk2 = bk1 + bs;
        if (cnt < bs) sais(s1, sa, bs, type + len, bk, bk2, cnt);
        else for (int i = 0; i < bs; ++i) sa[s1[i]] = i;
        for (int i = 0; i < bs; ++i) bk2[i] = bk1[sa[i]];
        induced_sort(s, sa, len, bs, sigma, type, bk, cb, bk2);
    }

    template<class T>
    inline void get_height(T *s, int len, int *sa) {
        for (int i = 0, k = 0; i < len; ++i) {
            if (rk[i] == 0) k = 0;
            else {
                if (k > 0) k--;
                int j = sa[rk[i] - 1];
                while (i + k < len && j + k < len && s[i + k] == s[j + k]) k++;
```

```
            }
            ht[rk[i]] = k;
        }
    }

    template<class T>
    inline void init(T s, int len, int sigma = 124) {
        // len 是字符串长度, sigma 是字符集大小
        // 计算完 sa[1...len], 存放 0...len-1, 表示排名第 i 的后缀的开始位置
        // rk[0...len-1] 表示从 i 开始的后缀的排名
        // h[2...len] 表示排名为 i 的后缀, 与排名为 i-1 的后缀的 lcp 是多少
        len += 1;
        sais(s, sa, len, type, bk, bk1, sigma);
        for (int i = 1; i < len; ++i) rk[sa[i]] = i;
        get_height(s, len, sa);
    }
};
```

SAM

字符集要放缩到 0-sigma

```
template<size_t size, size_t sigma>
struct SAM{
    int ch[size << 1][sigma], fail[size << 1], siz[size << 1], len[size << 1], endpos[size << 1];
    int t[size << 1], A[size << 1];
    //int jump[size << 1][19], endsta[size << 1];
    long long cnt[size << 1];
    int num = 1, last = 1;
    void insert(int c, int pos) {
        int now = ++ num, node = last;
        len[now] = len[last] + 1;
        siz[now] = 1; endpos[now] = pos;
        //endsta[pos] = now;
        while(node && !ch[node][c]) ch[node][c] = now, node = fail[node];
        if(!node) fail[now] = 1;
        else {
            int cur = ch[node][c];
            if(len[cur] == len[node] + 1) fail[now] = cur;
            else {
                int clone = ++ num;
                for(int i = 0; i < sigma; i ++) ch[clone][i] = ch[cur][i];
                len[clone] = len[node] + 1;
                fail[clone] = fail[cur];
                while(node && ch[node][c] == cur)
                    ch[node][c] = clone, node = fail[node];
                fail[now] = fail[cur] = clone;
            }
        }
        last = now;
    }
    void prework() {
        for(int i = 1; i <= num; i ++) t[len[i]] ++;
        for(int i = 1; i <= num; i ++) t[i] += t[i - 1];
        for(int i = 1; i <= num; i ++) A[t[len[i]] --] = i;
        for(int i = num; i >= 1; i --)
            siz[fail[A[i]]] += siz[A[i]],
                    endpos[fail[A[i]]] = max(endpos[fail[A[i]]], endpos[A[i]]);
        for(int i = 1; i <= num; i ++)
            cnt[A[i]] = cnt[fail[A[i]]] + 1ll * (len[A[i]] - len[fail[A[i]]]) * siz[A[i]];
        // for(int i = 1; i <= num; i++) jump[i][0] = fail[i];
        // jump[1][0] = 1;
        // for(int j = 1; j < 19; j++) {
        //     for(int i = 1; i <= num; i++) {
        //         jump[i][j] = jump[jump[i][j - 1]][j - 1];
        //     }
        // }
    }
    void clear() {
        memset(len, 0, sizeof(int) * (num + 1));
        memset(fail, 0, sizeof(int) * (num + 1));
```

```
        memset(cnt, 0, sizeof(int) * (num + 1));
        memset(endpos, 0, sizeof(int) * (num + 1));
        memset(siz, 0, sizeof(int) * (num + 1));
        memset(t, 0, sizeof(int) * (num + 1));
        memset(A, 0, sizeof(int) * (num + 1));
        memset(ch, 0, sizeof(int) * (num + 1) * sigma);
        last = num = 1;
    }
    void build(const string &s) {
        clear();
        for(int i = 0; i < s.size(); i++) insert(s[i] - 'a', i);
        prework();
    }
    // int fidNode(int l, int r) {
    //  // 定位子串 [l, r] 对应的节点
    //     int now = endsta[r];
    //     for(int i = 18; i >= 0; i--) {
    //         if(len[jump[now][i]] >= (r - l + 1)) {
    //             now = jump[now][i];
    //         }
    //     }
    //     return now;
    // }
};
SAM<N, 26> sam;
```

后缀树

后缀树是将字符串的所有后缀插入到 Trie 树里面形成的数据结构，它等于字符串反串 SAM 的 parent 树。

```
// 放到 SAM 的 prework 中
for(int i = 2; i <= num; i++) {
    ver[fail[i]].push_back({s[endpos[i] - len[fail[i]]] - 'a', i});
}
// 字典序
for(int i = 1; i <= num; i++) std::sort(ver[i].begin(), ver[i].end());
```

Trie

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

KMP 算法

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

manacher 算法

p 数组代表的是插入’#’ 的串的每个点开始的最长回文半径
比如 #a#b#a# 的 p[3] = 4，表示从 b 开始往外 3 个字母是最大的回文串
s 和 p 都是从 0 开始的。

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

回文自动机

回文自动机中，每个点表示一个本质不同的回文串。
有两个根，偶根 0，长度为 0，奇根 1，长度为-1.
偶根的 fail 指向奇根，奇根没有 fail，因为不可能失配 (匹配后长度为 1)。
fail 表示一个点的最长回文后缀

cnt 表示当前节点所能代表的回文串的数量
text 表示文本
son[x][y] 表示 x 点的回文串，前后各加一个 y，形成新的回文串。

```
namespace PAM {
    int all, son[N][S], fail[N], len[N], text[N], last, tot;
    int slink[N], dif[N], cnt[N];
    // 回文后缀的 border 可以划分成 log 个等差数列
    // dif 是跟回文后缀的长度差值，slink 是第一次差值不同的位置
    int newnode(int l) {
        memset(son[tot], 0, sizeof(int) * (S));
        len[tot] = l;
        return tot++;
    }
    void init() {
        // 一定注意主函数是否运行了这一句
        last = tot = all = 0;
        newnode(0); newnode(-1);
        text[0] = -1; fail[0] = 1;
        slink[0] = 1;
    }
    int getfail(int x) {
        while(text[all - len[x] - 1] != text[all]) x = fail[x];
        return x;
    }
    void add(int w) {
        text[++all] = w;
        int x = getfail(last);
        if(!son[x][w]) {
            int y = newnode(len[x] + 2);
            fail[y] = son[getfail(fail[x])][w];
            int fa = fail[y];
            dif[y] = len[y] - len[fa];
            slink[y] = (dif[y] == dif[fa]) ? slink[fa] : fa;
            son[x][w] = y;
        }
        cnt[last = son[x][w]]++;
    }
    void count() {for(int i = tot - 1; ~i; i--) cnt[fail[i]] += cnt[i];}
}
```

回文子串划分方案数

```
f[0] = 1;
for(int i = 1; i <= n; i++) {
    int c = s[i - 1] - 'a';
    PAM::add(c);
    for(int k = PAM::last; k; k = PAM::slink[k]) {
        int dlt = PAM::len[PAM::slink[k]] + PAM::dif[k];
        g[k] = f[i - dlt];
        if(PAM::fail[k] != PAM::slink[k]) {
            g[k] = (g[k] + g[PAM::fail[k]]) % mod;
        }
        f[i] = (f[i] + g[k]) % mod;
    }
}
```

AC 自动机

```
struct ac_automaton {
    int t[N][26], danger[N], tot, fail[N];
    int dp[N][N];
    void init() {
        tot = -1;
        _new();
    }
    int _new() {
        tot++;
        memset(t[tot], 0, sizeof(t[tot]));
        danger[tot] = 0;
        fail[tot] = 0;
```

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

区间本质不同子串个数

```
// 写一个线段树，写一个 SAM（记录每个字符加入时对应的节点位置），补全 LCT 的 splay，rotate
struct LCT {
    void pushtag(int x, int y) {
        if(x == 0) return;
        val[x] = y;
        tag[x] = y;
    }
    void pushdown(int x) {
        if(tag[x]) {
            if(ch[x][0]) pushtag(ch[x][0], tag[x]);
            if(ch[x][1]) pushtag(ch[x][1], tag[x]);
            tag[x] = 0;
        }
    }
    void access(int x, int v) {
        int y;
        for(y = 0; x; y = x, x = fa[x]) {
            splay(x);
            ch[x][1] = y;
            if(val[x]) {
```

```
                sgt.modify(val[x] - sam.len[x] + 1, val[x] - sam.len[fa[x]], -1);
            }
        }
        pushtag(y, v); sgt.modify(1, v, 1);
    }
} lct;
string s;
vector<pair<int, int> > v[N];
int n, ans[N];
void work() {
    cin >> s;
    cin >> n;
    sgt.setLimit(s.size() + 3);
    for(int i = 1; i <= n; i++) {
        int l, r;
        cin >> l >> r;
        v[r].push_back({i, l});
    }
    for(int i = 0; i < s.size(); i++) sam.insert(s[i] - 'a', i + 1);
    for(int i = 2; i <= sam.num; i++) lct.fa[i] = sam.fail[i];
    for(int i = 1; i <= s.size(); i++) {
        lct.access(sam.p[i], i);
        for(auto item : v[i]) {
            ans[item.first] = sgt.query(item.second, i);
        }
    }
    for(int i = 1; i <= n; i++) cout << ans[i] << endl;
}
```

杂项

取模还原成分数

摘自 EI 的博客

```
pair<int, int> approx(int p, int q, int A) {
  int x = q, y = p, a = 1, b = 0;
  while (x > A) {
    swap(x, y); swap(a, b);
    a -= x / y * b;
    x %= y;
  }
  return make_pair(x, a);
}
```

快读

```
#define gc()(is==it?it=(is=in)+fread(in,1,Q,stdin),(is==it?EOF:*is++):*is++)
const int Q=(1<<24)+1;
char in[Q],*is=in,*it=in,c;
void read(int &n){
    for(n=0;(c=gc())<'0'||c>'9';);
    for(;c<='9'&&c>='0';c=gc())n=n*10+c-48;
}
```

int128

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

```cpp
inline void output(u128 res) {
    if(res >= 16)
        output(res / 16);
    putchar(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
    //std::cout.put(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
}
```

Java,BigInteger

```
public BigInteger add(BigInteger val)    返回当前大整数对象与参数指定的大整数对象的和
public BigInteger subtract(BigInteger val)  返回当前大整数对象与参数指定的大整数对象的差
public BigInteger multiply(BigInteger val)    返回当前大整数对象与参数指定的大整数对象的积
public BigInteger devide(BigInteger val)     返回当前大整数对象与参数指定的大整数对象的商
public BigInteger remainder(BigInteger val)     返回当前大整数对象与参数指定的大整数对象的余
public int compareTo(BigInteger val)     返回当前大整数对象与参数指定的大整数对象的比较结果，返回值是 1、-1、0，分别表示当前大整数对象大于、小于或等于参数指定的大整数。
public BigInteger abs()     返回当前大整数对象的绝对值
public BigInteger pow(int exponent)    返回当前大整数对象的 exponent 次幂。
public String toString()     返回当前当前大整数对象十进制的字符串表示。
public String toString(int p)    返回当前大整数对象 p 进制的字符串表示。
public BigInteger negate()    返回当前大整数的相反数。
```

奇技淫巧

**_builtin_ 内建函数 **

- ~~__builtin_popcount(unsigned int n) 该函数是判断 n 的二进制中有多少个 1~~
- __builtin_parity(unsigned int n) 该函数是判断 n 的二进制中 1 的个数的奇偶性
- __builtin_ffs(unsigned int n) 该函数判断 n 的二进制末尾最后一个 1 的位置，从一开始
- __builtin_ctz(unsigned int n) 该函数判断 n 的二进制末尾后面 0 的个数，当 n 为 0 时，和 n 的类型有关
- __builtin_clz (unsigned int x) 返回前导的 0 的个数

真·popcount

```cpp
int _popcount(int x) {
    return __builtin_popcount(x & (0ull - 1)) + __builtin_popcount(x >> 32);
}
```

随机数种子

```cpp
srand(std :: chrono :: system_clock :: now().time_since_epoch().count());
```

T(5) 求任意 int log2

```cpp
inline int LOG2_1(unsigned x){
    static const int tb[32]={0,9,1,10,13,21,2,29,11,14,16,18,22,25,3,30,8,12,20,28,15,17,24,7,19,27,23,6,26,5,4,31};
    x|=x>>1; x|=x>>2; x|=x>>4; x|=x>>8; x|=x>>16;
    return tb[x*0x07C4ACDDu>>27];
}
```

O(1) 求 2 的整幂次 log2

```cpp
inline int LOG2(unsigned x){  //x=2^k
    static const int tb[32]={31,0,27,1,28,18,23,2,29,21,19,12,24,9,14,3,30,26,17,22,20,11,8,13,25,16,10,7,15,6,5,4};
    return tb[x*263572066>>27];
}
```

对拍

注意关闭流同步，取消 define endl

```cpp
system("g++ -std=c++14 J.cpp -o J.exe");
system("g++ -std=c++14 bfJ.cpp -o bf.exe");
system("g++ -std=c++14 rdJ.cpp -o rd.exe");
while(1) {
```

```
    system("rd.exe>data.in");
    system("J.exe<data.in>data.myans");
    system("bf.exe<data.in>data.bfans");
    if(system("fc data.myans data.bfans")) {
        break;
    } else {
        static int cnt = 0; cnt++;
        cout << cnt << endl;
    }
}
```

快速乘

```
ll mul(ll x, ll y, ll mod){
    return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
}
ll mul(ll a, ll b, ll MOD) {
    __int128 x = a, y = b, m = MOD;
    return (ll)(x * y % m);
}
```

子集枚举

枚举 s 的子集

```
for(int i = s; i; i = (i - 1) & s))
```

枚举所有大小为 r 的集合

```
for(int s = (1 << r) - 1; s < (1 << n); ) {
    int x = s & -s;
    int y = s + x;
    s = ((y ^ s) >> __builtin_ctz(x) + 2) | y;
}
```

mt19937_64 随机数生成器

```
std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());
template <typename T>
T rd(T l, T r) {
    std::uniform_int_distribution<T> u(l, r);
    return u(rng);
}
template <>
double rd<double>(double l, double r) {
    std::uniform_real_distribution<double> u(l, r);
    return u(rng);
}
```

tips:

- 如果使用 sort 比较两个函数，不能出现 $a < b$ 和 $a > b$ 同时为真的情况，否则会运行错误。
- 多组数据清空线段树的时候，不要忘记清空全部数组（比如说 lazytag 数组）。
- 注意树的深度和节点到根的距离是两个不同的东西，深度是点数，距离是边长，如果求 LCA 时用距离算会出错。
- 连通性专题：注意判断 $dfn[x]$ 和 $low[y]$ 的关系时是否不小心两个都达成 low 了
- 推不等式确定范围的时候，仅需要考虑所有不等式限定的范围，然后判断左端点是否大于右端点，不要加额外的臆想条件。
- 矩阵快速幂如果常数十分大的时候，可以考虑 unordered_map 保存结果，可以明显加速。
- **__builtin_popcount** 只支持 unsigned int 型，不支持 long long!!!!!!!!
- multiset 删除某个元素的所有对象：s.erase(x)，删除某个元素的一个对象：s.erase(s.find(x))

经典题

区间回文子串计数（非本质不同）

对于一个询问，考虑一个回文中心向两边拓展，在一个区间左半边的会被左端点挡住，右半边会被右端点挡住。所以我们左右半边分别计算答案。

```
input:
3 3
aba
1 3
1 2
2 2
output:
4
2
1
```

```cpp
const int N = 4e6 + 1009;
namespace manacher {
    char tmp[N];
    int p[N], len;
    void getP(char *s, int n) {
        len = 0;
        for(int i = 0; i < n; i++) {
            tmp[len++] = '#';
            tmp[len++] = s[i];
        }
        tmp[len++] = '#';
        memset(p, 0, sizeof(int) * (len + 10));
        int c = 0, r = 0;
        for(int i = 0; i < len; i++) {
            if(i <= r) p[i] = min(p[2 * c - i], r - i);
            else p[i] = 1;
            while(i - p[i] >= 0 && i + p[i] < len && tmp[i - p[i]] == tmp[i + p[i]]) p[i]++;
            if(i + p[i] - 1 > r) {
                r = i + p[i] - 1;
                c = i;
            }
        }
        for(int i = 0; i < len; i++)p[i]--;
    }
};
struct node {
    int l, r, id;
};
struct SGT {
    long long sum[N];
    int lzt[N];
    int n;
    void set(int x) {
        n = x;
        build(1, n, 1);
    }
    void update(int rt) {
        sum[rt] = sum[lson] + sum[rson];
    }
    void pushdown(int l, int r, int rt) {
        if(lzt[rt] == 0) return ;
        sum[lson] += (Mid - l + 1) * lzt[rt]; lzt[lson] += lzt[rt];
        sum[rson] += (r - Mid) * lzt[rt]; lzt[rson] += lzt[rt];
        lzt[rt] = 0;
    }
    void build(int l, int r, int rt) {
        lzt[rt] = 0;
        if(l == r) {
            sum[rt] = 0;
            return ;
```

```
        }
        build(l, Mid, lson);
        build(Mid + 1, r, rson);
        update(rt);
    }
    void modify(int l, int r, int L, int R, int rt, int x) {
        if(L <= l && r <= R) {
            sum[rt] += (r - l + 1) * x;
            lzt[rt] += x;
            return ;
        }
        pushdown(l, r, rt);
        if(L <= Mid) modify(l, Mid, L, R, lson, x);
        if(Mid <  R) modify(Mid + 1, r, L, R, rson, x);
        update(rt);
    }
    long long query(int l, int r, int L, int R, int rt) {
        if(L <= l && r <= R) return sum[rt];
        pushdown(l, r, rt);
        long long ans = 0;
        if(L <= Mid) ans += query(l, Mid, L, R, lson);
        if(Mid <  R) ans += query(Mid + 1, r, L, R, rson);
        return ans;
    }
    long long query(int l, int r) {
        if(l > r) return 0;
        return query(1, n,l, r, 1);
    }
    void modify(int l, int r, int x) {
        if(l > r) return ;
        modify(1, n, l, r, 1, x);
    }
} T;
vector<node> v;
long long ans[N];
int n, q;
char s[N];
void work() {
    cin >> n >> q;
    cin >> s;
    manacher::getP(s, n);
    for(int i = 1; i <= q; i++) {
        int l, r;
        cin >> l >> r;
        l = l * 2 - 1;
        r = r * 2 + 1;
        v.push_back({l, r, i});
    }
    int j = 1;
    std::sort(v.begin(), v.end(), [](const node &a, const node &b) {
        return (a.l + a.r) / 2 < (b.l + b.r) / 2;
    });
    T.set(2 * n + 1);
    for(int i = 0; i < q; i++) {
        int mid = (v[i].l + v[i].r) / 2;
        while(j <= mid) {
            j++;
        }
        ans[v[i].id] += T.query(v[i].l, mid);
    }

    std::sort(v.begin(), v.end(), [](const node &a, const node &b) {
        return (a.l + a.r) / 2 > (b.l + b.r) / 2;
    });
    j = 2 * n + 1;
    T.set(2 * n + 1);
    for(int i = 0; i < q; i++) {
        int mid = (v[i].l + v[i].r) / 2 + 1;
        while(j >= mid) {
            T.modify(j, j + manacher::p[j - 1], 1);
            j--;
```

```
122             }
123             ans[v[i].id] += T.query(mid, v[i].r);
124         }
125         std::sort(v.begin(), v.end(), [](const node &a, const node &b) {
126             return a.id < b.id;
127         });
128         for(int i = 1; i <= q; i++) {
129             ans[i] -= (v[i - 1].r - v[i - 1].l + 2) / 2;
130             cout << ans[i] / 2 << endl;
131         }
132     }
```