Code Template for ACM-ICPC

UESTC Life and Lemons

October 13, 2016

Contents

1 DataStructure

1.1 Size Balanced Tree

```
const int N = 100005;
struct SBT {
    int lc, rc, sz, key;
    void init(int k) {
        lc = rc = 0;
        sz = 1;
        key = k;
    }
} T[N];
int tot;

inline void push_up(int x) {
    T[x].sz = T[T[x].lc].sz + T[T[x].rc].sz + 1;
}

void L_rotate(int &x) {
    int k = T[x].rc;
    T[x].rc = T[k].lc;
    T[k].lc = x;
    push_up(x);
    push_up(k);
    x = k;
}

void R_rotate(int &x) {
    int k = T[x].lc;
    T[x].lc = T[k].rc;
    T[k].rc = x;
    push_up(x);
    push_up(k);
    x = k;
}

void Maintain(int &x, bool fg) {
    if(fg) {
        if(T[T[T[x].rc].rc].sz > T[T[x].lc].sz) L_rotate(x);
        else if(T[T[T[x].rc].lc].sz > T[T[x].lc].sz) {
            R_rotate(T[x].rc);
            L_rotate(x);
        }
        else return;
    }
    else {
        if(T[T[T[x].lc].lc].sz > T[T[x].rc].sz) R_rotate(x);
        else if(T[T[T[x].lc].rc].sz > T[T[x].rc].sz) {
            L_rotate(T[x].lc);
            R_rotate(x);
        }
        else return;
    }
    Maintain(T[x].lc, 0);
    Maintain(T[x].rc, 1);
    Maintain(x, 0);
    Maintain(x, 1);
```

```
}

void Insert(int &x, int k) {
    if(!x) {
        x = ++tot;
        T[x].init(k);
    }
    else {
        Insert(k < T[x].key ? T[x].lc : T[x].rc, k);
        push_up(x);
        Maintain(x, k >= T[x].key);
    }
}

int d_key;
void Delete(int &x, int k) {
    if(T[x].key == k || (k < T[x].key && !T[x].lc) || (k > T[x].key && !T[x].rc)) {
        if(!T[x].lc || !T[x].rc) {
            d_key = T[x].key;
            x = T[x].lc + T[x].rc;
        }
        else {
            Delete(T[x].lc, k + 1);
            T[x].key = d_key;
        }
    }
    else Delete(k < T[x].key ? T[x].lc : T[x].rc, k);
    if(x) push_up(x);
}

int get_rank(int x, int k) {
    int res = 1;
    while(x) {
        if(T[x].key < k) {
            res += T[T[x].lc].sz + 1;
            x = T[x].rc;
        }
        else x = T[x].lc;
    }
    return res;
}

int get_kth(int x, int k) {
    while(T[T[x].lc].sz + 1 != k) {
        if(T[T[x].lc].sz + 1 < k) {
            k -= T[T[x].lc].sz + 1;
            x = T[x].rc;
        }
        else x = T[x].lc;
    }
    return T[x].key;
}

int get_pre(int x, int k) {
    int res;
    while(x) {
        if(T[x].key < k) {
            res = T[x].key;
            x = T[x].rc;
        }
        else x = T[x].lc;
    }
    return res;
}

int get_nxt(int x, int k) {
    int res;
    while(x) {
        if(T[x].key > k) {
            res = T[x].key;
            x = T[x].lc;
        }
        else x = T[x].rc;
    }
    return res;
}
```

1.2 Treap

```
const int N = 100005;
struct Treap {
    int key, val, sz;
    Treap *lc, *rc;
} pool[N], *nill, *root;
int tot;

void init() {
    srand(0);
    root = nill = pool;
    nill->sz = 0;
    tot = 0;
}

Treap* newnode(int v) {
    Treap *t = pool + (++tot);
    t->val = v;
    t->sz = 1;
    t->key = (rand() << 16) | rand();
    t->lc = t->rc = nill;
    return t;
}

inline void push_up(Treap *p) {
    p->sz = p->lc->sz + p->rc->sz + 1;
}

Treap* Merge(Treap *a, Treap *b) {
    if(a == nill) return b;
    if(b == nill) return a;
    if(a->key < b->key) {
        a->rc = Merge(a->rc, b);
        push_up(a);
        return a;
    }
    else {
        b->lc = Merge(a, b->lc);
        push_up(b);
        return b;
    }
}

typedef pair <Treap*, Treap*> pii;
pii Split(Treap *a, int k) {
    if(!k) return make_pair(nill, a);
    int cnt = a->lc->sz;
    if(cnt >= k) {
        pii u = Split(a->lc, k);
        a->lc = u.SE;
        push_up(a);
        return make_pair(u.FI, a);
    }
    else {
        pii u = Split(a->rc, k - cnt - 1);
        a->rc = u.FI;
        push_up(a);
        return make_pair(a, u.SE);
    }
}

int get_rank(int k) {
    Treap *p = root;
    int res = 1;
    while(p != nill) {
        if(p->val < k) {
            res += p->lc->sz + 1;
            p = p->rc;
        }
        else p = p->lc;
    }
    return res;
}

int get_kth(int k) {
    Treap *p = root;
    while(p->lc->sz + 1 != k) {
        if(p->lc->sz + 1 < k) {
            k -= p->lc->sz + 1;
            p = p->rc;
        }
        else p = p->lc;
    }
    return p->val;
}

int get_pre(int k) {
    int res;
    Treap *p = root;
    while(p != nill) {
        if(p->val < k) {
            res = p->val;
            p = p->rc;
        }
        else p = p->lc;
    }
    return res;
```

```
}

int get_nxt(int k) {
    int res;
    Treap *p = root;
    while(p != nill) {
        if(p->val > k) {
            res = p->val;
            p = p->lc;
        }
        else p = p->rc;
    }
    return res;
}

void Insert(int k) {
    Treap *t = newnode(k);
    pii u = Split(root, get_rank(k) - 1);
    root = Merge(u.FI, t);
    root = Merge(root, u.SE);
}

void Delete(int k) {
    int p = get_rank(k);
    pii a = Split(root, p - 1);
    pii b = Split(a.SE, 1);
    root = Merge(a.FI, b.SE);
}
```

1.3 Splay

```
#define keyTree (ch[ch[root][1]][0])
const int N = 100005;
int pre[N], ch[N][2], sz[N];
int val[N], mx[N], add[N];
int tot, root;

void init() {
    root = tot = 0;
    ch[0][0] = ch[0][1] = pre[0] = 0;
    sz[0] = val[0] = mx[0] = 0;
    add[0] = 0;
}

inline void push_up(int x) {
    sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + 1;
    mx[x] = max(val[x], max(mx[ch[x][0]], mx[ch[x][1]]));
}

inline void Add(int x, int v) {
    if(!x) return;
    val[x] += v;
    mx[x] += v;
    add[x] += v;
}

inline void push_down(int x) {
    if(!add[x]) return;
    Add(ch[x][0], add[x]);
    Add(ch[x][1], add[x]);
    add[x] = 0;
}

inline void newnode(int &x, int v, int fa) {
    x = ++tot;
    pre[x] = fa;
    ch[x][0] = ch[x][1] = 0;
    sz[x] = 1;
    mx[x] = val[x] = v;
    add[x] = 0;
}

inline void Rotate(int x, bool kind) {
    int y = pre[x];
    ch[y][!kind] = ch[x][kind];
    pre[ch[x][kind]] = y;
    pre[x] = pre[y];
    if(pre[x]) ch[pre[x]][ch[pre[x]][1] == y] = x;
    ch[x][kind] = y;
    pre[y] = x;
    push_up(y);
}

void P(int x) {
    if(!x) return;
    P(pre[x]);
    push_down(x);
}

void Splay(int x, int goal) {
    P(x);
    while(pre[x] != goal) {
        if(pre[pre[x]] == goal) Rotate(x, ch[pre[x]][0] == x);
        else {
            int y = pre[x];
            bool kind = (ch[pre[y]][0] == y);
            if(ch[pre[x]][kind] == x)
                Rotate(x, !kind), Rotate(x, kind);
            else
                Rotate(y, kind), Rotate(x, kind);
        }
    }
    push_up(x);
    if(!goal) root = x;
}

int get_kth(int k) {
    int x = root;
    push_down(x);
    while(sz[ch[x][0]] + 1 != k) {
        if(sz[ch[x][0]] + 1 < k) {
            k -= sz[ch[x][0]] + 1;
            x = ch[x][1];
        }
        else x = ch[x][0];
        push_down(x);
    }
    return x;
}

void RotateTo(int k, int goal) {
    int t = get_kth(k);
    Splay(t, goal);
}

void update(int l, int r, int v) {
    RotateTo(l, 0);
    RotateTo(r + 2, root);
    Add(keyTree, v);
    push_up(ch[root][1]);
    push_up(root);
}

void Delete(int x) {
    RotateTo(x, 0);
    RotateTo(x + 2, root);
    keyTree = 0;
    push_up(ch[root][1]);
    push_up(root);
}

void Insert(int x, int v) {
    RotateTo(x + 1, 0);
    RotateTo(x + 2, root);
    newnode(keyTree, v, ch[root][1]);
    push_up(ch[root][1]);
    push_up(root);
}
```

1.4 Scapegoat Tree

```
const int N = 100005;
const double alpha = 0.6;
struct Scapegoat_Tree {
    int lc, rc, sz, key;
    void init(int k) {
        lc = rc = 0;
        sz = 1;
        key = k;
    }
} T[N];
int tot, root, *Adjnode;

inline void push_up(int x) {
    T[x].sz = T[T[x].lc].sz + T[T[x].rc].sz + 1;
}

inline bool Balance(int x) {
    return max(T[T[x].lc].sz, T[T[x].rc].sz) <= T[x].sz * alpha;
}

int tr[N], cc;
void Travel(int x) {
    if(!x) return;
    Travel(T[x].lc);
    tr[cc++] = x;
    Travel(T[x].rc);
}

void build(int &x, int l, int r) {
    if(l > r) return;
    int mid = (l + r) >> 1;
    x = tr[mid];
    T[x].init(T[x].key);
    build(T[x].lc, l, mid - 1);
    build(T[x].rc, mid + 1, r);
    push_up(x);
}

void Adjust(int *x) {
    if(!x) return;
    cc = 0;
    Travel(*x);
    build(*x, 0, cc - 1);
}

void Insert(int &x, int k) {
    if(!x) {
        x = ++tot;
        T[x].init(k);
    }
    else {
        Insert(k < T[x].key ? T[x].lc : T[x].rc, k);
        push_up(x);
    }
    if(!Balance(x)) Adjnode = &x;
}

int d_key;
void Delete(int &x, int k) {
    if(T[x].key == k || (k < T[x].key && !T[x].lc) || (k > T[x].key && !T[x].rc)) {
        if(!T[x].lc || !T[x].rc) {
            d_key = T[x].key;
            x = T[x].lc + T[x].rc;
        }
        else {
            Delete(T[x].lc, k + 1);
            T[x].key = d_key;
        }
    }
    else Delete(k < T[x].key ? T[x].lc : T[x].rc, k);
    if(x) push_up(x);
    if(!Balance(x)) Adjnode = &x;
}
```

1.5 Leftist Tree

```
const int N = 100005;

struct LHeap {
    int dis, key;
    LHeap *lc, *rc;
} pool[N], *nill;
int tot;

inline void init() {
    tot = 0;
    nill = pool;
    nill->dis = -1;
}

inline LHeap* MakeTree(int v) {
    LHeap *t = pool + (++tot);
    t->lc = t->rc = nill;
    t->dis = 0;
    t->key = v;
    return t;
}

LHeap* Merge(LHeap *a, LHeap *b) {
    if(a == nill) return b;
    if(b == nill) return a;
    if(a->key > b->key) swap(a, b);
    a->rc = Merge(a->rc, b);
    if(a->rc->dis > a->lc->dis) swap(a->rc, a->lc);
    a->dis = a->rc->dis + 1;
    return a;
}

inline void Insert(LHeap* &a, int v) {
    LHeap *b = MakeTree(v);
    a = Merge(a, b);
}

inline int DeleteMin(LHeap* &a) {
    int t = a->key;
    a = Merge(a->lc, a->rc);
    return t;
}
```

1.6 Link Cut Tree

```
const int N = 100005;

struct Link_Cut_Tree {
    int pre[N], ch[N][2], bef[N];
    bool rev[N];
    inline void init() {

    }
    inline void Rev(int x) {
        if(!x) return;
        swap(ch[x][0], ch[x][1]);
        rev[x] ^= 1;
    }
    inline void push_down(int x) {
    }
    inline void P(int x) {
        if(pre[x]) P(pre[x]);
        push_down(x);
    }
    inline void push_up(int x) {

    }
    inline void Rotate(int x, bool kind) {
        int y = pre[x];
        ch[y][!kind] = ch[x][kind];
        pre[ch[x][kind]] = y;
        pre[x] = pre[y];
        if(pre[x]) ch[pre[x]][ch[pre[x]][1] == y] = x;
        ch[x][kind] = y;
        pre[y] = x;
        //push_up(y);
    }
    inline void Splay(int x) {
        P(x); int r = x;
        while(pre[r]) r = pre[r];
        if(r != x) bef[x] = bef[r], bef[r] = 0;
        while(pre[x]) {
            if(pre[pre[x]] == 0) Rotate(x, ch[pre[x]][0] == x);
            else {
                int y = pre[x], k = ch[pre[y]][0] == y;
                if(ch[pre[x]][k] == x)
                    Rotate(x, !k), Rotate(x, k);
                else
                    Rotate(y, k), Rotate(x, k);
            }
        }
        push_up(x);
    }
    inline void Access(int x) {
        int fa = 0;
        for(; x; x = bef[x]) {
            Splay(x);
            bef[ch[x][1]] = x;
            bef[fa] = 0;
            pre[ch[x][1]] = 0;
            ch[x][1] = fa;
            pre[fa] = x;
            fa = x;
            //push_up(x);
        }
    }
    inline int get_rt(int x) {
        Access(x);
        Splay(x);
        //push_down(x);
        while(ch[x][0]) {
            x = ch[x][0];
            //push_down(x);
        }
        Splay(x);
        return x;
    }
    inline void make_rt(int x) {
        Access(x);
        Splay(x);
        Rev(x);
    }
    inline void Cut(int u, int v) {
        make_rt(u);
        Access(v);
        Splay(v);
        pre[ch[v][0]] = 0;
        ch[v][0] = 0;
        //push_up(v);
    }
    inline void Link(int u, int v) {
        make_rt(v);
        bef[v] = u;
        Access(v);
        /*make_rt(v);
        push_down(v);
        Access(u);
        Splay(u);
        pre[u] = v;
        ch[v][0] = u;
        push_up(v);*/
    }
    inline int Query(int x, int y) {
        Access(y);
        for(y = 0; x; x = bef[x]) {
            Splay(x);
            if(!bef[x]) return max(mx[y], mx[ch[x][1]]);
            bef[ch[x][1]] = x;
            bef[y] = 0;
            pre[ch[x][1]] = 0;
            ch[x][1] = y;
            pre[y] = x;
            y = x;
            push_up(x);
        }
    }
} lct;
```

1.7 Partition Tree

```
int a[maxn], as[maxn];
int n, m;
int sum[20][maxn];
int tree[20][maxn];
void build(int c, int l, int r){
    int i, mid = (l + r) >> 1, lm = mid - l + 1, lp = l, rp = mid + 1;
    for (i = l; i <= mid; i++){
        if (as[i] < as[mid]){
            lm--;
        }
    }
    for (i = l; i <= r; i++){
        if (i == l){
            sum[c][i] = 0;
        }else{
            sum[c][i] = sum[c][i - 1];
        }
        if (tree[c][i] == as[mid]){
            if (lm){
                lm--;
                sum[c][i]++;
                tree[c + 1][lp++] = tree[c][i];
            }else
                tree[c + 1][rp++] = tree[c][i];
        } else if (tree[c][i] < as[mid]){
            sum[c][i]++;
            tree[c + 1][lp++] = tree[c][i];
        } else{
            tree[c + 1][rp++] = tree[c][i];
        }
    }
    if (l == r)return;
    build(c + 1, l, mid);
    build(c + 1, mid + 1, r);
}

int query(int c, int l, int r, int ql, int qr, int k){
    int s;
    int ss;
    int mid = (l + r) >> 1;
    if (l == r){
        return tree[c][l];
    }
    if (l == ql){
    s = 0;
    ss = sum[c][qr];
    }else{
        s = sum[c][ql - 1];
        ss = sum[c][qr] - s;
    }
    if (k <= ss){
        return query(c + 1, l, mid, l + s, l + s + ss - 1, k);
    }else{
        return query(c + 1, mid + 1, r, mid - l + 1 + ql - s, mid - l + 1 + qr - s - ss,k - ss);
    }
}

int main(){
    int i, j, k;
    while(~scanf("%d%d", &n, &m)){
        for (i = 1; i <= n; i++){
            scanf("%d", &a[i]);
            tree[0][i] = as[i] = a[i];
        }
        sort(as + 1, as + 1 + n);
        build(0, 1, n);
        while(m--){
            scanf("%d%d%d",&i,&j,&k);
            printf("%d\n", query(0, 1, n, i, j, k));
        }
    }
    return 0;
}
```

1.8 Range Minimum Query

```
void initRMQ(int n) {
    Lg[1] = 0;
    for(int i = 2; i <= n; ++i) Lg[i] = Lg[i >> 1] + 1;
    for(int j = 1; j < 20; ++j) {
        for(int i = 1; i <= n; ++i) {
            if(i + (1 << j) - 1 > n) break;
            minx[i][j] = min(minx[i][j - 1], minx[i + (1 << (j - 1))][j - 1]);
        }
    }
}

inline int query(int l, int r) {
    int t = Lg[r - l + 1];
    return min(minx[l][t], minx[r - (1 << t) + 1][t]);
}
```

1.9 BIT for Kth-Element

```
int findkth(int k) {
    int idx = 0;
    for(int i = 20; i >= 0; --i) {
        idx ^= 1 << i;
        if(idx <= N && bit[idx] < k) k -= bit[idx];
        else idx ^= 1 << i;
    }
    return idx + 1;
}
```

1.10 Neighbors for Tree Path

```
inline int query(int u, int v) {
    int ans = 0;
    int f1 = top[u], f2 = top[v];
    while(f1 ^ f2) {
        if(dep[f1] < dep[f2]) {
            swap(f1, f2);
            swap(u, v);
        }
        //Heavy son of the end of this chain
        if(son[u]) add(ans, sqr(bt1.query(L[son[u]], R[son[u]])));
        //All the light sons on this chain
        add(ans, bt2.query(L[f1], L[u]));
        //Subtract the value of the top of the chain, since we will count it when count the light
             sons on the above chain
        add(ans, -sqr(bt1.query(L[f1], R[f1])));
        u = fa[f1]; f1 = top[u];
    }
    if(dep[u] > dep[v]) swap(u, v);
    //All the light sons on the last chain
    add(ans, bt2.query(L[u], L[v]));
    //Heavy son of the bottom of the last chain
    if(son[v]) add(ans, sqr(bt1.query(L[son[v]], R[son[v]])));
    //Subtree above the LCA
    if(u != 1) add(ans, sqr(sum - bt1.query(L[u], R[u])));
    return ans;
}
```

1.11 Scan Line for Disjoint Circles

```
const int N = 150005;
int x[N], y[N], r[N], id[N];
int op[N];

struct Event {
    int x, tp, id;
    bool operator < (const Event &a) const {
        if(x != a.x) return x < a.x;
        return tp > a.tp;
    }
} C[N * 2];
int tot = 0;

void add(int x, int tp, int id) {
    C[tot].x = x;
    C[tot].tp = tp;
    C[tot++].id = id;
}

inline double sqr(double x) {
    return x * x;
}

inline int sgn(double x) {
    if(x < -eps) return -1;
    return x > eps;
}

double X;
struct HC {
    int id, up;
    double get_y() const {
        double v = sqr(r[id]) - sqr(x[id] - X);
        v = sqrt(max(v, 0.0));
        return up ? y[id] + v : y[id] - v;
    }
    bool operator < (const HC &a) const {
        int ck = sgn(get_y() - a.get_y());
        if(ck) return ck > 0;
        return up > a.up;
    }
};

inline bool OnCircle(int c, int p) {
    int dx = (x[c] - x[p]), dy = (y[c] - y[p]);
    return r[c] * r[c] == dx * dx + dy * dy;
}

set <HC> st;
int fa[N], belong[N];
```

```
vector <int> G[N];

int L[N], R[N], label = 0;
void dfs(int u) {
    L[u] = ++label;
    for(int i = 0; i < (int)G[u].size(); ++i) {
        int v = G[u][i];
        dfs(v);
    }
    R[u] = label;
}

struct BIT {
    int bit[N];
    void add(int x, int v) {
        for(; x <= label; x += x & -x) bit[x] += v;
    }
    int read(int x) {
        int res = 0;
        for(; x; x ^= x & -x) res += bit[x];
        return res;
    }
    void init() {
        memset(bit, 0, sizeof(bit));
    }
} t1, t2;

void get_fa(int &x, int up, int down) {
    if(up == down) x = up;
    else if(fa[down] == up) x = up;
    else if(fa[up] == down) x = down;
    else x = fa[up];
}

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 0; i < n; ++i) {
        op[i] = 1;
        scanf("%d", &id[i]);
        scanf("%d%d%d", &x[id[i]], &y[id[i]], &r[id[i]]);
        add(x[id[i]] - r[id[i]], -1, id[i]);
        add(x[id[i]] + r[id[i]], 1, id[i]);
    }
    for(int i = n; i < n + m; ++i) {
        op[i] = 2;
        scanf("%d", &id[i]);
        scanf("%d%d", &x[id[i]], &y[id[i]]);
        add(x[id[i]], 0, id[i]);
    }
    int q;
    scanf("%d", &q);
    q += n + m;
    for(int i = n + m; i < q; ++i) {
        scanf("%d%d", &op[i], &id[i]);
        if(op[i] == 1) {
            scanf("%d%d%d", &x[id[i]], &y[id[i]], &r[id[i]]);
            add(x[id[i]] - r[id[i]], -1, id[i]);
            add(x[id[i]] + r[id[i]], 1, id[i]);
        }
        else if(op[i] == 2) {
            scanf("%d%d", &x[id[i]], &y[id[i]]);
            add(x[id[i]], 0, id[i]);
        }
    }
    sort(C, C + tot);
    HC t;
    for(int i = 0; i < tot; ++i) {
        X = C[i].x;
        if(C[i].tp == -1) {
            t.id = C[i].id;
            t.up = 0;
            set <HC>::iterator it = st.upper_bound(t);
            int up = 0, down = 0;
            if(it != st.end()) down = it->id;
            if(it != st.begin()) up = (--it)->id;
            get_fa(fa[t.id], up, down);
            G[fa[t.id]].push_back(t.id);
            st.insert(t);
            t.up = 1;
            st.insert(t);
        }
        else if(C[i].tp == 0) {
            t.id = C[i].id;
            t.up = 1;
            set <HC>::iterator it = st.lower_bound(t);
            if(it == st.end()) continue;
            if(OnCircle(it->id, t.id)) {
                belong[t.id] = fa[it->id];
                continue;
            }
            if(it == st.begin()) continue;
            int down = it->id, up = (--it)->id;
            get_fa(belong[t.id], up, down);
        }
        else {
            t.id = C[i].id;
            t.up = 0;
            st.erase(t);
            t.up = 1;
            st.erase(t);
        }
    }
    dfs(0);
    t1.init(); t2.init();
    for(int i = 0; i < q; ++i) {
        if(op[i] == 1) {
            int u = id[i];
            t1.add(L[u], 1);
            t1.add(R[u] + 1, -1);
        }
        else if(op[i] == 2) {
            int u = belong[id[i]];
            t2.add(L[u], 1);
        }
        else if(op[i] == 3) {
            int u = id[i];
            if(r[u]) {
                t1.add(L[u], -1);
                t1.add(R[u] + 1, 1);
            }
            else {
                u = belong[u];
                t2.add(L[u], -1);
            }
        }
        else {
            int ans, u = id[i];
            if(r[u]) {
                ans = t2.read(R[u]) - t2.read(L[u] - 1);
            }
            else {
                ans = t1.read(L[belong[u]]);
            }
            printf("%d\n", ans);
        }
    }
    return 0;
}
```

1.12 Manhattan Distance MST

```
struct Point {
    int x, y, id;
} po[10005];
int data[10005], cc;

struct Edge {
    int u, v, l;
} ed[50005];
int ecnt = 0;

inline int Find( int x ) {
    return lower_bound( data, data + cc, x ) - data + 1;
}

inline bool cmp( Point a, Point b ) {
    return a.x > b.x || ( a.x == b.x && a.y > b.y );
}

inline int AB( int x ) {
    return x > 0 ? x : -x;
}

inline int Dis( Point a, Point b ) {
    return AB( a.x - b.x ) + AB( a.y - b.y );
}

inline void addedge( int u, int v, int l ) {
    ed[ecnt].u = u;
    ed[ecnt].v = v;
    ed[ecnt++].l = l;
}

int bitv[10005], bitid[10005];
```

```
inline void add( int x, int v, int id ) {
    x = cc - x + 1;
    for( ; x <= cc; x += x & -x ) if( bitv[x] > v ) {
            bitv[x] = v;
            bitid[x] = id;
    }
}

inline int read( int x ) {
    int v = INF, id = -1;
    x = cc - x + 1;
    for( ; x; x ^= x & -x ) if( bitv[x] < v ) {
            v = bitv[x];
            id = bitid[x];
    }
    return id;
}

inline bool ecmp( Edge a, Edge b ) {
    return a.l < b.l;
}

int F[10005];
int findroot( int x ) {
    return F[x] == x ? x : F[x] = findroot( F[x] );
}

int main() {
    int n, K;
    while( ~scanf( "%d%d", &n, &K ) ) {
        for( int i = 0; i < n; ++i ) {
            scanf( "%d%d", &po[i].x, &po[i].y );
            po[i].id = i;
        }
        for( int dir = 0; dir < 4; ++dir ) {
            if( dir == 1 || dir == 3 ) {
                for( int i = 0; i < n; ++i ) swap( po[i].x, po[i].y );
            } else if( dir == 2 ) {
                for( int i = 0; i < n; ++i ) po[i].x *= -1;
            }
            cc = 0;
            for( int i = 0; i < n; ++i ) data[cc++] = po[i].y - po[i].x;
            sort( data, data + cc );
            cc = unique( data, data + cc ) - data;
            sort( po, po + n, cmp );
            memset( bitv, 0x3f, sizeof( bitv ) );
            for( int i = 0; i < n; ++i ) {
                int v = Find( po[i].y - po[i].x );
                int id = read( v );
                if( id != -1 ) addedge( po[i].id, po[id].id, Dis( po[i], po[id] ) );
                add( v, po[i].x + po[i].y, i );
            }
        }
        sort( ed, ed + ecnt, ecmp );
        for( int i = 0; i < n; ++i ) F[i] = i;
        int cnt = 0;
        for( int i = 0; i < ecnt; ++i ) {
            int fu = findroot( ed[i].u ), fv = findroot( ed[i].v );
            if( fu == fv ) continue;
            ++cnt;
            if( cnt == n - K ) {
                printf( "%d\n", ed[i].l );
                break;
            }
            F[fu] = fv;
        }
    }
    return 0;
}
```

1.13 Dynamic MST

```
const int maxn = 50005;
struct Edge {
    int u, v, c, id;
} ed[20][maxn], e[maxn];
int cnt[20], val[maxn], pos[maxn];
struct Query {
    int id, c;
} qry[maxn];
LL ans[maxn];
int F[maxn];

inline void init(int m) {
    for(int i = 0; i < m; ++i) F[e[i].u] = e[i].u, F[e[i].v] = e[i].v;
}

int findroot(int x) {
    return F[x] == x ? x : F[x] = findroot(F[x]);
}

inline bool cmp(const Edge &a, const Edge &b) {
    return a.c < b.c;
}

int must[maxn];
void solve(int dep, int l, int r, LL sum) {
    if(l == r) val[qry[l].id] = qry[l].c;
    int m = cnt[dep];
    for(int i = 0; i < m; ++i) {
        e[i] = ed[dep][i];
        e[i].c = val[e[i].id];
    }
    if(l == r) {
        sort(e, e + m, cmp);
        init(m);
        for(int i = 0; i < m; ++i) {
            int fu = findroot(e[i].u), fv = findroot(e[i].v);
            if(fu == fv) continue;
            sum += e[i].c;
            F[fu] = fv;
        }
        ans[l] = sum;
        return;
    }
    for(int i = 0; i < m; ++i) pos[e[i].id] = i;
    for(int i = l; i <= r; ++i) e[pos[qry[i].id]].c = -1;
    sort(e, e + m, cmp);
    init(m);
    int mcnt = 0;
    for(int i = 0; i < m; ++i) {
        int fu = findroot(e[i].u), fv = findroot(e[i].v);
        if(fu == fv) continue;
        if(e[i].c != -1) sum += e[i].c, must[mcnt++] = i;
        F[fu] = fv;
    }
    init(m);
    for(int i = 0; i < mcnt; ++i) F[findroot(e[must[i]].u)] = findroot(e[must[i]].v);
    for(int i = 0; i < m; ++i) {
        if(e[i].c == -1) e[i].c = INF;
        e[i].u = findroot(e[i].u);
        e[i].v = findroot(e[i].v);
    }
    sort(e, e + m, cmp);
    cnt[dep + 1] = 0;
    for(int i = 0; i < m; ++i) {
        int fu = findroot(e[i].u), fv = findroot(e[i].v);
        if(fu == fv) {
            if(e[i].c == INF) ed[dep + 1][cnt[dep + 1]++] = e[i];
            continue;
        }
        ed[dep + 1][cnt[dep + 1]++] = e[i];
        F[fu] = fv;
    }
    int mid = (l + r) >> 1;
    solve(dep + 1, l, mid, sum);
    solve(dep + 1, mid + 1, r, sum);
}

int main() {
    int n, m, q;
    scanf("%d%d%d", &n, &m, &q);
    for(int i = 0; i < m; ++i) {
        scanf("%d%d%d", &ed[0][i].u, &ed[0][i].v, &ed[0][i].c);
        ed[0][i].id = i;
        val[i] = ed[0][i].c;
    }
    for(int i = 0; i < q; ++i) scanf("%d%d", &qry[i].id, &qry[i].c), --qry[i].id;
    cnt[0] = m;
    solve(0, 0, q - 1, 0);
    for(int i = 0; i < q; ++i) printf("%I64d\n", ans[i]);
    return 0;
}
```

1.14 Dynamic Convex

```
const double pi = acos(-1.0);
const double eps = 1e-8;
inline int sgn(double x) {
    if(x < -eps) return -1;
    return x > eps;
}
```

```
double Ox, Oy;
struct Vector {
    LL x, y;
    double arg;
    inline void read() {
        scanf("%I64d%I64d", &x, &y);
    }
    Vector(LL _x = 0, LL _y = 0) {
        x = _x;
        y = _y;
    }
    Vector operator +(Vector a) const {
        return Vector(x + a.x, y + a.y);
    }
    Vector operator +=(Vector a) {
        return *this = *this + a;
    }
    Vector operator -(Vector a) const {
        return Vector(x - a.x, y - a.y);
    }
    Vector operator -=(Vector a) {
        return *this = *this - a;
    }
    Vector operator *(double p) const {
        return Vector(x * p, y * p);
    }
    Vector operator *=(double p) {
        return *this = *this * p;
    }
    bool operator <(const Vector a) const {
        return arg < a.arg;
    }
    //bool operator ==(const Vector a) const { return sgn(x-a.x)==0 && sgn(y-a.y)==0; }
    double len() const {
        return sqrt(x * x + y * y);
    }
    double angle() const {
        return atan2(y - Oy, x - Ox);
    }
};

inline LL cross(Vector a, Vector b) {
    return a.x * b.y - a.y * b.x;
}

set <Vector> pst;
inline void init() {
    int t;
    Vector a, b, c;
    a.read();
    b.read();
    c.read();
    Ox = (a.x + b.x + c.x) / 3.0;
    Oy = (a.y + b.y + c.y) / 3.0;
    a.arg = a.angle();
    b.arg = b.angle();
    c.arg = c.angle();
    pst.insert(a);
    pst.insert(b);
    pst.insert(c);
}

typedef set <Vector>::iterator pit;
inline pit pre(pit it) {
    if(it == pst.begin()) it = pst.end();
    return --it;
}

inline pit next(pit it) {
    ++it;
    if(it == pst.end()) it = pst.begin();
    return it;
}

inline void updata(Vector p) {
    p.arg = p.angle();
    pit L = pre(pst.lower_bound(p));
    pit R = next(L);
    if(cross(*R - p, *L - p) <= 0) return;
    while(cross(*next(R) - p, *R - p) >= 0) R = next(R);
    while(cross(*L - p, *pre(L) - p) >= 0) L = pre(L);
    L = next(L);
    while(L != R) L = next(L), pst.erase(pre(L));
    pst.insert(p);
}

inline bool query(Vector p) {
    p.arg = p.angle();
    pit L = pre(pst.lower_bound(p));
    pit R = next(L);
    return cross(*R - p, *L - p) <= 0;
}
```

2 String

2.1 Manacher

```
const int N = 2e5 + 5;
char s[N], str[N];
int p[N];

int Manacher(char *s) {
    str[0] = ’$’;
    int cc = 1;
    for(int i = 0; s[i]; ++i) {
        str[cc++] = ’#’;
        str[cc++] = s[i];
    }
    str[cc++] = ’#’;
    str[cc] = 0;
    int mx = 0, id;
    for(int i = 1; str[i]; ++i) {
        if(mx > i) {
            p[i] = min(p[2 * id - i], mx - i);
        }
        else p[i] = 1;
        for(; str[i + p[i]] == str[i - p[i]]; ++p[i]);
        if(p[i] + i > mx) {
            mx = p[i] + i;
            id = i;
        }
    }
    return cc;
}
```

2.2 Minimum Representation

```
int minP(char s[])
{
    int l=strlen(s);
    int i = 0, j = 1, k = 0;
    while (1)
    {
        if (i + k >= l || j + k >= l) break;
        if (s[i + k] == s[j + k])
        {
            k++;
            continue;
        }
        else
        {
            if (s[j + k] > s[i + k]) j += k + 1;
            else i += k + 1;
            k = 0;
            if (i == j) j++;
        }
    }
    return min(i, j);
}
```

2.3 EX KMP

```
const int N = 1e5 + 5;
int next[N], extand[N];

void getnext(char *T) {
    int i, length = strlen(T);
    next[0] = length;
    for(i = 0; i < length - 1 && T[i] == T[i + 1]; i++);
    next[1] = i;
    int a = 1;
    for(int k = 2; k < length; k++) {
        int p = a + next[a] - 1, L = next[k - a];
        if((k - 1) + L >= p) {
            int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
            while(k + j < length && T[k + j] == T[j]) j++;
            next[k] = j, a = k;
        }
        else next[k] = L;
    }
}

void getextand(char *S, char *T) {
    memset(next, 0, sizeof(next));
    getnext(T);
    int Slen = strlen(S), Tlen = strlen(T), a = 0;
    int MinLen = Slen > Tlen ? Tlen : Slen;
    while(a < MinLen && S[a] == T[a]) a++;
    extand[0] = a, a = 0;
    for(int k = 1; k < Slen; k++) {
        int p = a + extand[a] - 1, L = next[k - a];
        if((k - 1) + L >= p) {
            int j = (p - k + 1) > 0 ? (p - k + 1) : 0;
            while(k + j < Slen && j < Tlen && S[k + j] == T[j]) j++;
            extand[k] = j; a = k;
        }
        else extand[k] = L;
    }
}
```

2.4 Palindromic Tree

```
const int MAXN = 100005 ;
const int N = 26 ;

struct Palindromic_Tree {
    int next[MAXN][N];
    int fail[MAXN];
    int cnt[MAXN];
    int num[MAXN];
    int len[MAXN];
    int S[MAXN];
    int last;
    int n;
    int p;

    int newnode(int l) {
        for(int i = 0; i < N; ++i) next[p][i] = 0;
        cnt[p] = 0;
        num[p] = 0;
        len[p] = l;
        return p++;
    }

    void init() {
        p = 0;
        newnode(0) ;
        newnode(-1);
        last = 0;
        n = 0;
        S[n] = -1;
        fail[0] = 1;
    }

    int get_fail(int x) {
        while(S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }

    void add(int c) {
        c -= 'a';
        S[++n] = c;
        int cur = get_fail(last);
        if(!next[cur][c]) {
            int now = newnode(len[cur] + 2);
            fail[now] = next[get_fail(fail[cur])][c];
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = next[cur][c];
        cnt[last]++;
    }

    void count() {
        for(int i = p - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
    }
} T;
```

2.5 Palindromic Tree AF

```
const int MAXN = 200005;
const int base = 100002;
const int N = 26;

struct Palindromic_Tree {
    int next[MAXN][N];
    int fail[MAXN];
    int num[MAXN];
    int len[MAXN];
    int S[MAXN];
    int suflast, prelast;
    int L, R;
    int p;

    int newnode(int l) {
        for(int i = 0; i < N; ++i) next[p][i] = 0;
        num[p] = 0;
        len[p] = l;
        return p++;
    }

    void init() {
        p = 0;
        newnode(0) ;
        newnode(-1);
        suflast = prelast = 0;
        L = base + 1; R = base;
        fail[0] = 1;
    }

    int get_back_fail(int x) {
        while(R - len[x] - 1 < L || S[R - len[x] - 1] != S[R]) x = fail[x];
        return x;
    }
    int get_front_fail(int x) {
        while(L + len[x] + 1 > R || S[L + len[x] + 1] != S[L]) x = fail[x];
        return x;
    }

    void add_back(int c) {
        c -= ’a’;
        S[++R] = c;
        int cur = get_back_fail(suflast);
        if(!next[cur][c]) {
            int now = newnode(len[cur] + 2);
            fail[now] = next[get_back_fail(fail[cur])][c];
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        suflast = next[cur][c];
        if(len[suflast] == R - L + 1) prelast = suflast;
    }

    void add_front(int c) {
        c -= ’a’;
        S[--L] = c;
        int cur = get_front_fail(prelast);
        if(!next[cur][c]) {
            int now = newnode(len[cur] + 2);
            fail[now] = next[get_front_fail(fail[cur])][c];
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        prelast = next[cur][c];
        if(len[prelast] == R - L + 1) suflast = prelast;
    }
} T;
```

2.6 Suffix Automaton

```
const int N = 1e5 + 5;
struct Sam {
    Sam *next[26], *par;
    int step;
} pool[N * 2], *root, *last;
int tot;

Sam* newnode(int step) {
    Sam *t = pool + (tot++);
    memset(t->next, 0, sizeof(t->next));
    t->par = NULL;
    t->step = step;
    return t;
}

void init() {
    tot = 0;
    last = root = newnode(0);
}
```

```
void Extend(int w) {
    Sam *p = last;
    Sam *newv = newnode(p->step + 1);
    for(; p && !p->next[w]; p = p->par) p->next[w] = newv;
    if(!p) newv->par = root;
    else {
        Sam *q = p->next[w];
        if(q->step == p->step + 1) newv->par = q;
        else {
            Sam *nq = newnode(p->step + 1);
            memcpy(nq->next, q->next, sizeof(q->next));
            nq->par = q->par;
            q->par = nq;
            newv->par = nq;
            for(; p && p->next[w] == q; p = p->par) p->next[w] = nq;
        }
    }
    last = newv;
}
```

2.7 Suffix Array

```
struct Suffix_Array {
    int wa[N], wb[N], wv[N], wd[N];
    inline int cmp(int *r, int a, int b, int l) {
        return r[a] == r[b] && r[a + l] == r[b + l];
    }
    void da(int *r, int *sa, int n, int m) {
        int i, j, p, *x = wa, *y = wb, *t;
        for(i = 0; i < m; ++i) wd[i] = 0;
        for(i = 0; i < n; ++i) wd[x[i] = r[i]]++;
        for(i = 1; i < m; ++i) wd[i] += wd[i - 1];
        for(i = n - 1; i >= 0; --i) sa[--wd[x[i]]] = i;
        for(j = 1, p = 1; p < n; j *= 2, m = p) {
            for(p = 0, i = n - j; i < n; ++i) y[p++] = i;
            for(i = 0; i < n; ++i) if(sa[i] >= j) y[p++] = sa[i] - j;
            for(i = 0; i < n; ++i) wv[i] = x[y[i]];
            for(i = 0; i < m; ++i) wd[i] = 0;
            for(i = 0; i < n; ++i) wd[wv[i]]++;
            for(i = 1; i < m; ++i) wd[i] += wd[i - 1];
            for(i = n - 1; i >= 0; --i) sa[--wd[wv[i]]] = y[i];
            for(t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; ++i)
                x[sa[i]] = cmp(y, sa[i - 1], sa[i],j) ? p - 1 : p++;
        }
    }
    int rank[N], height[N], data[N], sa[N];
    void calheight(int *r, int *sa, int n) {
        int i, j, k = 0;
        for(i = 1; i <= n; ++i) rank[sa[i]] = i;
        for(i = 0; i < n; height[rank[i++]] = k)
        for(k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; ++k);
    }
    int minx[N][20], Lg[N];
    void initRMQ(int n) {
        Lg[1] = 0;
        for(int i = 2; i <= n; ++i) Lg[i] = Lg[i >> 1] + 1;
        for(int j = 1; j < 20; ++j) {
            for(int i = 1; i <= n; ++i) {
                if(i + (1 << j) - 1 > n) break;
                minx[i][j] = min(minx[i][j - 1], minx[i + (1 << (j - 1))][j - 1]);
            }
        }
    }
    inline int lcp(int l, int r) {
        int t = Lg[r - l + 1];
        return min(minx[l][t], minx[r - (1 << t) + 1][t]);
    }
    int len;
    void work(char *s) {
        len = strlen(s);
        for(int i = 0; i < len; ++i) data[i] = s[i];
        data[len] = 0;
        da(data, sa, len + 1, 128);
        calheight(data, sa, len);
        for(int i = 1; i <= len; ++i) minx[i][0] = height[i];
        initRMQ(len);
    }
} sa;
```

2.8 Suffix Tree

```
const int SIGMA = 26;

const int N = 100005;

int alloc, curPos, actEdge, actLen, remaind;

struct node {
    int l, r;
    node *nxt[SIGMA], *slink;
    inline int edgeLen() {
        return min(r, curPos + 1) - l;
    }
} S[N + N], *root, *actNode, *needSL;

inline node* newnode(int l, int r = INF) {
    node *t = S + (alloc ++);
    t->l = l; t->r = r;
    t->slink = 0;
    memset(t->nxt, 0, sizeof(t->nxt));
    return t;
}

int text[N];

inline int actedge() {
    return text[actEdge];
}

inline void addSL(node *p) {
    if(needSL) needSL->slink = p;
    needSL = p;
}
```

```
bool walkDown(node *p) {
    if(actLen < p->edgeLen()) return false;
    actEdge += p->edgeLen();
    actLen -= p->edgeLen();
    actNode = p;
    return true;
}

void init() {
    needSL = 0; alloc = 0; curPos = -1;
    remaind = actEdge = actLen = 0;
    root = actNode = newnode(-1, -1);
}

void extend(int c) {
    text[++ curPos] = c;
    needSL = 0;
    ++ remaind;
    while(remaind > 0) {
        if(actLen == 0) actEdge = curPos;
        if(actNode->nxt[actedge()] == 0) {
            node* leaf = newnode(curPos);
            actNode->nxt[actedge()] = leaf;
            addSL(actNode);
        } else {
            node* nt = actNode->nxt[actedge()];
            if(walkDown(nt)) continue;
            if(text[nt->l + actLen] == c) {
                ++ actLen;
                addSL(actNode);
                break;
            }
            node* split = newnode(nt->l, nt->l + actLen);
            actNode->nxt[actedge()] = split;
            node* leaf = newnode(curPos);
            split->nxt[c] = leaf;
            nt->l += actLen;
            split->nxt[text[nt->l]] = nt;
            addSL(split);
        }
        -- remaind;
        if(actNode == root && actLen > 0) {
            -- actLen;
            actEdge = curPos - remaind + 1;
        } else {
            actNode = actNode->slink ? actNode->slink : root;
        }
    }
}
```

2.9 Suffix Tree DF

```
const int SIGMA = 26;

const int N = 100005;

int alloc, curPos, actEdge, actLen, remaind;
```

```
struct node {
    int l, r, son;
    node *nxt[SIGMA], *slink, *fa;
    inline int edgeLen() {
        return min(r, curPos + 1) - l;
    }
} S[N + N], *root, *actNode, *needSL;

inline node* newnode(int l, int r = INF) {
    node *t = S + (alloc ++);
    t->l = l; t->r = r;
    t->slink = t->fa = 0;
    t->son = 0;
    memset(t->nxt, 0, sizeof(t->nxt));
    return t;
}

int text[N];

inline int actedge() {
    return text[actEdge];
}

inline void addSL(node *p) {
    if(needSL) needSL->slink = p;
    needSL = p;
}

bool walkDown(node *p) {
    if(actLen < p->edgeLen()) return false;
    actEdge += p->edgeLen();
    actLen -= p->edgeLen();
    actNode = p;
    return true;
}

void doneins() {
    -- remaind;
    if(actNode == root && actLen > 0) {
        -- actLen;
        actEdge = curPos - remaind + 1;
    } else {
        actNode = actNode->slink ? actNode->slink : root;
    }
}

int head, tail;

node* leaves[N + N];

ll curAns;

void init() {
    curAns = head = tail = 0;
    needSL = 0; alloc = 0; curPos = -1;
    remaind = actEdge = actLen = 0;
    root = actNode = newnode(-1, -1);
}
```

```
void extend(int c) {
    text[++ curPos] = c;
    needSL = 0;
    ++ remaind;
    curAns += tail - head;
    while(remaind > 0) {
        if(actLen == 0) actEdge = curPos;
        if(actNode->nxt[actedge()] == 0) {
            node* leaf = newnode(curPos);
            actNode->nxt[actedge()] = leaf;
            leaf->fa = actNode;
            ++ actNode->son;
            addSL(actNode);
            leaves[tail ++] = leaf;
        } else {
            node* nt = actNode->nxt[actedge()];
            if(walkDown(nt)) continue;
            if(text[nt->l + actLen] == c) {
                ++ actLen;
                addSL(actNode);
                break;
            }
            node* split = newnode(nt->l, nt->l + actLen);
            actNode->nxt[actedge()] = split;
            split->fa = actNode;
            node* leaf = newnode(curPos);
            split->nxt[c] = leaf;
            leaf->fa = split;
            nt->l += actLen;
            split->nxt[text[nt->l]] = nt;
            nt->fa = split;
            addSL(split);
            split->son = 2;
            leaves[tail ++] = leaf;
        }
        doneins();
        ++ curAns;
    }
}

void erasefront() {
    while(actLen > 0 && actNode->nxt[actedge()] && walkDown(actNode->nxt[actedge()]));
    node* u = leaves[head ++], *f = u->fa;
    while(u != root && u->son == 0 && actNode != f) {
        curAns -= u->edgeLen();
        f->nxt[text[u->l]] = 0;
        -- f->son;
        u = f; f = u->fa;
    }
    if(u == root || u->son > 0) return;
    if(actLen == 0 || f->nxt[actedge()] != u) {
        curAns -= u->edgeLen();
        f->nxt[text[u->l]] = 0;
        if(-- f->son) return;
        if(remaind) doneins();
        if(f != root) {
            leaves[tail ++] = f;
            f->l = curPos - f->edgeLen() + 1;
            f->r = INF;
        }
    } else {
        curAns -= u->edgeLen() - actLen;
        u->l = curPos - actLen + 1;
        u->r = INF;
        doneins();
        leaves[tail ++] = u;
    }
}
```

2.10 Palindomic Factorization

```
/**
 * Alforithm from this paper -- A Subquadratic Algorithm for Minimum Palindromic Factorization
 *
 * pl[i][0] -- minimal even factorization
 * pl[i][1] -- minimal odd factorization
 * 1 <= i <= n
 *
 * time complexity: O(n log n)
 **/
const int MAXN = 300000 + 10;
int pl[MAXN][2], gpl[MAXN][2];

inline void set(int *a, int x, int y, int z) {
  a[0] = x, a[1] = y, a[2] = z;
}
inline void set(int *a, int *b) {
  a[0] = b[0], a[1] = b[1], a[2] = b[2];
}
inline void set(int pl[][2], int idx, int val) {
  if (val <= 0) return;
  pl[idx][val & 1] = val;
}
inline void upd(int pl[][2], int idx, int val) {
  if (val <= 0) return;
  int &r = pl[idx][val & 1];
  if (r == -1 || r > val) r = val;
}

void factorization(char s[]) {
  int n = strlen(s);
  for (int i = 0; i <= n; ++i) {
    gpl[i][0] = 1e9;
    gpl[i][1] = 1e9 + 1;
  }
  static int g[32][3], gp[32][3], gpp[32][3];
  int pg = 0;
  for (int j = 0; j < n; ++j) {
    // g->gp
    int pgp = 0;
    for (int u = 0; u < pg; ++u) {
      int i = g[u][0];
      if (i - 1 >= 0 && s[i - 1] == s[j]) {
        g[u][0]--;
        set(gp[pgp++], g[u]);
      }
    }
    // gp->gpp
    int pgpp = 0, r = -j - 2;
    for (int u = 0; u < pgp; ++u) {
      int i = gp[u][0], d = gp[u][1], k = gp[u][2];
      if (i - r != d) {
        set(gpp[pgpp++], i, i - r, 1);
        if (k > 1) set(gpp[pgpp++], i + d, d, k - 1);
      } else set(gpp[pgpp++], i, d, k);
      r = i + (k - 1) * d;
    }
    if (j - 1 >= 0 && s[j - 1] == s[j]) {
      set(gpp[pgpp++], j - 1, j - 1 - r, 1);
      r = j - 1;
    }
    set(gpp[pgpp++], j, j - r, 1);
    // gpp->g
    pg = 0;
    int *front = gpp[0];
    for (int u = 1; u < pgpp; ++u) {
      int *x = gpp[u];
      if (x[1] == front[1]) front[2] += x[2];
      else {
        set(g[pg++], front);
        front = x;
      }
    }
    set(g[pg++], front);
    // dp update
    if ((j + 1) % 2 == 0) {
      pl[j + 1][0] = j + 1;
      pl[j + 1][1] = 1e9 + 1;
    } else {
      pl[j + 1][0] = 1e9;
      pl[j + 1][1] = j + 1;
    }
    for (int u = 0; u < pg; ++u) {
      int i = g[u][0], d = g[u][1], k = g[u][2];
      r = i + (k - 1) * d;
      upd(pl, j + 1, pl[r][0] + 1);
      upd(pl, j + 1, pl[r][1] + 1);
      if (k > 1) {
        upd(pl, j + 1, gpl[i + 1 - d][0]);
        upd(pl, j + 1, gpl[i + 1 - d][1]);
      }
      if (i + 1 >= d) {
        if (k > 1) {
          upd(gpl, i + 1 - d, pl[r][0] + 1);
          upd(gpl, i + 1 - d, pl[r][1] + 1);
        } else {
          set(gpl, i + 1 - d, pl[r][0] + 1);
          set(gpl, i + 1 - d, pl[r][1] + 1);
        }
      }
    }
  }
}
```

2.11 Palindomic Factorization PT

```
const int MAXN = 300005;

const int N = 26;

const int inf = 0x3f3f3f3f;

struct Palindromic_Tree {
    int nxt[MAXN][N], fail[MAXN];
    int occ[MAXN], num[MAXN], len[MAXN];
    int S[MAXN], last, n, p;
    int sfail[MAXN], diff[MAXN], dp[2][MAXN], ans[2][MAXN];

    int newnode(int l) {
        memset(nxt[p], 0, N * sizeof(int));
        occ[p] = num[p] = 0;
        len[p] = l;
        return p ++;
    }

    void init() {
        p = 0;
        newnode(0) ;
        newnode(-1);
        last = 0;
        n = 0;
        S[n] = -1;
        fail[0] = 1;
        ans[0][0] = 0;
        ans[1][0] = inf;
    }

    int get_fail(int x) {
        while(S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }

    void add(int c) {
        c -= ’a’;
        S[++ n] = c;
        int cur = get_fail(last);
        if(!nxt[cur][c]) {
            int v = newnode(len[cur] + 2);
            fail[v] = nxt[get_fail(fail[cur])][c];
            nxt[cur][c] = v;
            num[v] = num[fail[v]] + 1;
            diff[v] = len[v] - len[fail[v]];
            sfail[v] = diff[v] ^ diff[fail[v]] ? fail[v] : sfail[fail[v]];
        }
        last = nxt[cur][c];
        occ[last] ++;
        update();
    }

    void update() {
        ans[0][n] = ans[1][n] = inf;
        for(int u = last; u; u = sfail[u]) {
            dp[0][u] = ans[1][n - len[sfail[u]] - diff[u]];
            dp[1][u] = ans[0][n - len[sfail[u]] - diff[u]];
            if(diff[u] == diff[fail[u]]) {
                dp[0][u] = min(dp[0][u], dp[0][fail[u]]);
                dp[1][u] = min(dp[1][u], dp[1][fail[u]]);
            }
            ans[0][n] = min(ans[0][n], dp[0][u] + 1);
            ans[1][n] = min(ans[1][n], dp[1][u] + 1);
        }
    }

        void count() {
        for(int i = p - 1; i >= 0; --i) occ[fail[i]] += occ[i];
    }

} T;
```

2.12 KMP

```
void getNext(char *p, int *next) {
    int j = 0, k = -1;
    next[0] = -1;
    int len = strlen(p);
    while(j < len) {
        if(k == -1 || p[j] == p[k]) {
            ++j; ++k;
            next[j] = k;
        }
        else k = next[k];
    }
}

int KMPMatch(char *s, char *p) {
    int i = 0, j = 0;
    int len = strlen(s), lenp = strlen(p);
    while(i < len) {
        if(j == -1 || s[i] == p[j]) {
            ++i;
            ++j;
        }
        else j = next[j];
        if(j == lenp) return i - lenp;
    }
    return -1;
}
```

2.13 AC Automaton

```
const int N = 1e5 + 5;
struct Trie {
    Trie *next[26];
    Trie *fail;
} pool[N], *root;
int tot;
```

```
Trie* newnode() {
    Trie *t = pool + (tot++);
    memset(t->next, 0, sizeof(t->next));
    t->fail = NULL;
    return t;
}

void init() {
    tot = 0;
    root = newnode();
}

void Insert(char *s) {
    Trie *p = root;
    for(int i = 0; s[i]; ++i) {
        int k = s[i] - 'a';
        if(!p->next[k]) p->next[k] = newnode();
        p = p->next[k];
    }
}

queue <Trie*> Q;
void Build_Ac() {
    Trie *p, *temp;
    Q.push(root);
    while(!Q.empty()) {
        temp = Q.front(); Q.pop();
        for(int i = 0; i < 26; ++i) if(temp->next[i]) {
            p = temp->fail;
            while(p) {
                if(p->next[i]) {
                    temp->next[i]->fail = p->next[i];
                    break;
                }
                p = p->fail;
            }
            if(!p) temp->next[i]->fail = root;
            Q.push(temp->next[i]);
        }
        else {
            temp->next[i] = temp->fail ? temp->fail->next[i] : root;
        }
    }
}
```

3 Graph

3.1 Directed MST

```
#define M 600
#define type int
const type inf = (1) << 30;
struct Node {
    int u, v;
    type cost;
} E[M * M + 5];
```

```
int pre[M], ID[M], vis[M];
type In[M];

type Directed_MST(int root, int NV, int NE) {
    type ret = 0;
    while(true) {
        for(int i = 0; i < NV; i++) In[i] = inf;
        for(int i = 0; i < NE; i++) {
            int u = E[i].u;
            int v = E[i].v;
            if(E[i].cost < In[v] && u != v) {
                pre[v] = u;
                In[v] = E[i].cost;
            }
        }
        for(int i = 0; i < NV; i++) {
            if(i == root) continue;
            if(In[i] == inf)  return -1;
        }
        int cntnode = 0;
        memset(ID, -1, sizeof(ID));
        memset(vis, -1, sizeof(vis));
        In[root] = 0;
        for(int i = 0; i < NV; i++) {
            ret += In[i];
            int v = i;
            while(vis[v] != i && ID[v] == -1 && v != root) {
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && ID[v] == -1) {
                for(int u = pre[v] ; u != v ; u = pre[u]) {
                    ID[u] = cntnode;
                }
                ID[v] = cntnode ++;
            }
        }
        if(cntnode == 0)      break;
        for(int i = 0; i < NV; i++) if(ID[i] == -1) {
            ID[i] = cntnode ++;
        }
        for(int i = 0; i < NE; i++) {
            int v = E[i].v;
            E[i].u = ID[E[i].u];
            E[i].v = ID[E[i].v];
            if(E[i].u != E[i].v) {
                E[i].cost -= In[v];
            }
        }
        NV = cntnode;
        root = ID[root];
    }
    return ret;
}
```

3.2 Directed MST SOL

```
const int N = 505;

const int DN = N << 1;

const int M = N * N + 5;

const int inf = 1 << 30;

struct EDGE { int u, v, w; };

struct D_MST {

    EDGE E[M];

    int pre[DN], ID[DN], vis[DN];

    int In[DN], inE[DN], ring;

    int nV[DN], nnV[DN];

    vector < pair <int, int> > R[DN];

    bool ans[M], newR[DN];

    map <int, int> dirE[DN];

    void del(int u, int e) {
        if(R[u].empty()) return;
        int pu = dirE[u][e];
        for(auto &x : R[u]) {
            if(x.first == pu) {
                ans[x.second] = false;
                del(x.first, e);
            } else {
                del(x.first, x.second);
            }
        }
    }

    int Directed_MST(int rt, int n, int m) {
        while(true) {
            for(int i = 0; i < n; ++ i) In[nV[i]] = inf;
            for(int i = 0; i < m; ++ i) {
                int u = E[i].u, v = E[i].v;
                if(E[i].w < In[v] && u != v) {
                    pre[v] = u;
                    In[v] = E[i].w;
                    inE[v] = i;
                }
            }
            for(int i = 0; i < n; ++ i) {
                if(nV[i] != rt && In[nV[i]] == inf)
                    return -1;
            }
            int cntnode = 0;
            memset(ID, -1, sizeof(ID));
            memset(vis, -1, sizeof(vis));
            memset(newR, 0, sizeof(newR));
            In[rt] = 0;
            for(int i = 0; i < n; ++ i) {
                int v = nV[i], s = v;
                while(vis[v] != s && ID[v] == -1 && v != rt) {
                    vis[v] = s;
                    v = pre[v];
                }
                if(v != rt && ID[v] == -1) {
                    for(int u = pre[v]; ; u = pre[u]) {
                        ID[u] = ring;
                        R[ring].push_back( {u, inE[u]} );
                        ans[inE[u]] = true;
                        newR[u] = true;
                        if(u == v) break;
                    }
                    nnV[cntnode ++] = ring ++;
                }
            }
            if(cntnode == 0) {
                for(int i = 0; i < n; ++ i) {
                    if(nV[i] == rt) continue;
                    ans[inE[nV[i]]] = true;
                    del(nV[i], inE[nV[i]]);
                }
                return 0;
            }
            for(int i = 0; i < n; ++ i) {
                int v = nV[i];
                if(ID[v] != -1) continue;
                ID[v] = v;
                nnV[cntnode ++] = v;
            }
            for(int i = 0; i < m; ++ i) {
                int v = E[i].v;
                E[i].u = ID[E[i].u];
                E[i].v = ID[E[i].v];
                if(!newR[v]) continue;
                if(E[i].u != E[i].v) {
                    E[i].w -= In[v];
                    dirE[E[i].v][i] = v;
                }
            }
            n = cntnode; rt = ID[rt];
            for(int i = 0; i < n; ++ i) nV[i] = nnV[i];
        }
    }

    vector <int> solve(int rt, int n, int m, EDGE *e) {
        for(int i = 0; i < m; ++ i) E[i] = e[i];
        memset(ans, 0, m * sizeof(bool));
        for(int i = 0; i < n; ++ i) nV[i] = i;
        ring = n; vector <int> ret;
        if(Directed_MST(rt, n, m) == -1) return ret;
        for(int i = 0; i < ring; ++ i)
            dirE[i].clear(), R[i].clear();
        for(int i = 0; i < m; ++ i) if(ans[i]) ret.push_back(i);
        return ret;
    }
```

```
} mst;
```

3.3 MCMF

```
const int N = 305, M = 100005;
int head[N];
struct Edge {
    int nxt, to, cow, cost;
    Edge() {}
    Edge(int _nxt, int _to, int _cow, int _cost) {
        nxt = _nxt; to = _to; cow = _cow; cost = _cost;
    }
} ed[M];
int ecnt, mx_flow, mi_cost;

void init() {
    mx_flow = mi_cost = ecnt = 0;
    memset(head, -1, sizeof(head));
}

void addedge(int u, int v, int cow, int cost) {
    ed[ecnt] = Edge(head[u], v, cow, cost);
    head[u] = ecnt++;
    ed[ecnt] = Edge(head[v], u, 0, -cost);
    head[v] = ecnt++;
}

queue <int> Q;
int dis[N], pre[N], inq[N];
bool Spfa(int S, int T) {
    memset(dis, 0x3f, sizeof(dis));
    dis[S] = 0;
    Q.push(S);
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        inq[u] = 0;
        for(int e = head[u]; ~e; e = ed[e].nxt) {
            if(!ed[e].cow) continue;
            int v = ed[e].to;
            if(dis[v] > dis[u] + ed[e].cost) {
                dis[v] = dis[u] + ed[e].cost;
                pre[v] = e;
                if(!inq[v]) {
                    inq[v] = 1;
                    Q.push(v);
                }
            }
        }
    }
    return dis[T] != INF;
}

void End(int S, int T) {
    int flow = INF;
    for(int u = T; u != S; u = ed[pre[u] ^ 1].to) {
        flow = min(flow, ed[pre[u]].cow);
    }
    for(int u = T; u != S; u = ed[pre[u] ^ 1].to) {
        ed[pre[u]].cow -= flow;
        ed[pre[u] ^ 1].cow += flow;
        mi_cost += flow * ed[pre[u]].cost;
    }
    mx_flow += flow;
}
```

3.4 ZKW MCMF

```
const int maxn=105,maxm=10005;
struct MaxFlow
{
    int size, n;
    int st, en, maxflow, mincost;
    bool vis[maxn];
    int net[maxn], pre[maxn], cur[maxn], dis[maxn];
    std::queue <int> Q;
    struct EDGE
    {
        int v, cap, cost, next;
        EDGE(){}
        EDGE(int a, int b, int c, int d)
        {
            v = a, cap = b, cost = c, next = d;
        }
    }E[maxm];
    void init(int _n)
    {
        n = _n, size = 0;
        memset(net, -1, sizeof(net));
    }
    void add(int u, int v, int cap, int cost)
    {
        E[size] = EDGE(v, cap, cost, net[u]);
        net[u] = size++;
        E[size] = EDGE(u, 0, -cost, net[v]);
        net[v] = size++;
    }
    bool modell()
    {
        int v, min = INF;
        for(int i = 0; i <= n; i++)
        {
            if(!vis[i])
                continue;
            for(int j = net[i]; v = E[j].v, j != -1; j = E[j].next)
                if(E[j].cap)
                    if(!vis[v] && dis[v]-dis[i]+E[j].cost < min)
                        min = dis[v] - dis[i] + E[j].cost;
        }
        if(min == INF)
            return false;
        for(int i = 0; i <= n; i++)
            if(vis[i])
                cur[i] = net[i], vis[i] = false, dis[i] += min;
        return true;
```

```
}
int augment(int i, int flow)
{
    if(i == en)
    {
        mincost += dis[st] * flow;
        maxflow += flow;
        return flow;
    }
    vis[i] = true;
    for(int j = cur[i], v; v = E[j].v, j != -1; j = E[j].next)
    {
        if(!E[j].cap)
            continue;
        if(vis[v] || dis[v]+E[j].cost != dis[i])
            continue;
        int delta = augment(v, std::min(flow, E[j].cap));
        if(delta)
        {
            E[j].cap -= delta;
            E[j^1].cap += delta;
            cur[i] = j;
            return delta;
        }
    }
    return 0;
}
void spfa()
{
    int u, v;
    for(int i = 0; i <= n; i++)
        vis[i] = false, dis[i] = INF;
    dis[st] = 0;
    Q.push(st);
    vis[st] = true;
    while(!Q.empty())
    {
        u = Q.front(), Q.pop();
        vis[u] = false;
        for(int i = net[u]; v = E[i].v, i != -1; i = E[i].next)
        {
            if(!E[i].cap || dis[v] <= dis[u] + E[i].cost)
                continue;
            dis[v] = dis[u] + E[i].cost;
            if(!vis[v])
            {
                vis[v] = true;
                Q.push(v);
            }
        }
    }
    for(int i = 0; i <= n; i++)
        dis[i] = dis[en] - dis[i];
}
int zkw(int s, int t)
{
    st = s, en = t;
    spfa();
    mincost=maxflow=0;
```

```
        for(int i = 0; i <= n; i++)
            vis[i] = false, cur[i] = net[i];
        do
        {
            while(augment(st, INF))
                memset(vis, false, sizeof(vis));
        }while(modell());
        return mincost;
    }
}cf;
```

3.5 SAP

```
const int MAXN = 20010;
const int MAXM = 880010;
const int INF = 0x3f3f3f3f;

struct Node {
    int from, to, next;
    int cap;
} edge[MAXM];
int tol;
int head[MAXN];
int dep[MAXN];
int gap[MAXN];

int n;

void init() {
    tol = 0;
    memset(head, -1, sizeof(head));
}

void addedge(int u, int v, int w) {
    edge[tol].from = u;
    edge[tol].to = v;
    edge[tol].cap = w;
    edge[tol].next = head[u];
    head[u] = tol++;
    edge[tol].from = v;
    edge[tol].to = u;
    edge[tol].cap = 0;
    edge[tol].next = head[v];
    head[v] = tol++;
}

void BFS(int start, int end) {
    memset(dep, -1, sizeof(dep));
    memset(gap, 0, sizeof(gap));
    gap[0] = 1;
    int que[MAXN];
    int front, rear;
    front = rear = 0;
    dep[end] = 0;
    que[rear++] = end;
    while(front != rear) {
        int u = que[front++];
        if(front == MAXN)front = 0;
        for(int i = head[u]; i != -1; i = edge[i].next) {
            int v = edge[i].to;
            if(dep[v] != -1)continue;
            que[rear++] = v;
            if(rear == MAXN)rear = 0;
            dep[v] = dep[u] + 1;
            ++gap[dep[v]];
        }
    }
}

int SAP(int start, int end) {
    int res = 0;
    BFS(start, end);
    int cur[MAXN];
    int S[MAXN];
    int top = 0;
    memcpy(cur, head, sizeof(head));
    int u = start;
    int i;
    while(dep[start] < n) {
        if(u == end) {
            int temp = INF;
            int inser;
            for(i = 0; i < top; i++)
                if(temp > edge[S[i]].cap) {
                    temp = edge[S[i]].cap;
                    inser = i;
                }
            for(i = 0; i < top; i++) {
                edge[S[i]].cap -= temp;
                edge[S[i] ^ 1].cap += temp;
            }
            res += temp;
            top = inser;
            u = edge[S[top]].from;
        }
        if(u != end && gap[dep[u] - 1] == 0)
            break;
        for(i = cur[u]; i != -1; i = edge[i].next)
            if(edge[i].cap != 0 && dep[u] == dep[edge[i].to] + 1)
                break;
        if(i != -1) {
            cur[u] = i;
            S[top++] = i;
            u = edge[i].to;
        } else {
            int min = n;
            for(i = head[u]; i != -1; i = edge[i].next) {
                if(edge[i].cap == 0)continue;
                if(min > dep[edge[i].to]) {
                    min = dep[edge[i].to];
                    cur[u] = i;
                }
            }
            --gap[dep[u]];
            dep[u] = min + 1;
            ++gap[dep[u]];
            if(u != start)u = edge[S[--top]].from;
        }
    }
    return res;
}
```

3.6 Global Minimum Cut

```
const int maxn = 510;

int G[maxn][maxn];

int n, m;

void contract(int x, int y) {
    for(int i = 0; i < n; ++i) if(i != x) G[x][i] += G[y][i], G[i][x] += G[i][y];
    for(int i = y + 1; i < n; ++i) for(int j = 0; j < n; ++j) {
            G[i - 1][j] = G[i][j];
            G[j][i - 1] = G[j][i];
    }
    n--;
}

int w[maxn], c[maxn];

int sx, tx;

int mincut() {
    int t, k;
    memset(c, 0, sizeof(c));
    c[0] = 1;
    for(int i = 0; i < n; ++i) w[i] = G[0][i];
    for(int i = 1; i + 1 < n; ++i) {
        t = k = -1;
        for(int j = 0; j < n; ++j) if(c[j] == 0 && w[j] > k) k = w[t = j];
        c[sx = t] = 1;
        for(int j = 0; j < n; ++j) w[j] += G[t][j];
    }
    for(int i = 0; i < n; ++i) if(c[i] == 0) return w[tx = i];
}

int main() {
    while(~scanf("%d%d", &n, &m)) {
        memset(G, 0, sizeof(G));
        while(m--) {
            int u, v, c;
            scanf("%d%d%d", &u, &v, &c);
            G[u][v] += c;
            G[v][u] += c;
        }
        int mint = INF;
        while(n > 1) {
            int t = mincut();
            mint = min(mint, t);
            contract(sx, tx);
        }
        printf("%d\n", mint);
    }
    return 0;
}
```

3.7 Blossom Tree

```
const int N = 250;

int belong[N];
int findb(int x) {
        return belong[x] == x ? x : belong[x] = findb(belong[x]);
}
void unit(int a, int b) {
        a = findb(a);
        b = findb(b);
        if (a != b) belong[a] = b;
}

int n, match[N];
vector<int> e[N];
int Q[N], rear;
int next[N], mark[N], vis[N];

int LCA(int x, int y) {
        static int t = 0; t++;
        while (true) {
                if (x != -1) {
                        x = findb(x);
                        if (vis[x] == t) return x;
                        vis[x] = t;
                        if (match[x] != -1) x = next[match[x]];
                        else x = -1;
                }
                swap(x, y);
        }
}

void group(int a, int p) {
        while (a != p) {
                int b = match[a], c = next[b];
                if (findb(c) != p) next[c] = b;
                if (mark[b] == 2) mark[Q[rear++] = b] = 1;
                if (mark[c] == 2) mark[Q[rear++] = c] = 1;
                unit(a, b); unit(b, c);
                a = c;
        }
}


void aug(int s) {
        for (int i = 0; i < n; i++)
                next[i] = -1, belong[i] = i, mark[i] = 0, vis[i] = -1;
        mark[s] = 1;
        Q[0] = s; rear = 1;
        for (int front = 0; match[s] == -1 && front < rear; front++) {
                int x = Q[front];
                for (int i = 0; i < (int)e[x].size(); i++) {
```

```
			int y = e[x][i];
			if (match[x] == y) continue;
			if (findb(x) == findb(y)) continue;
			if (mark[y] == 2) continue;
			if (mark[y] == 1) {
				int r = LCA(x, y);
				if (findb(x) != r) next[x] = y;
				if (findb(y) != r) next[y] = x;

				group(x, r);
				group(y, r);
			}
			else if (match[y] == -1) {
				next[y] = x;
				for (int u = y; u != -1; ) {
					int v = next[u];
					int mv = match[v];
					match[v] = u, match[u] = v;
					u = mv;
				}
				break;
			}
			else {
				next[y] = x;
				mark[Q[rear++] = match[y]] = 1;
				mark[y] = 2;
			}
		}
	}
}

bool g[N][N];
int main() {
	scanf("%d", &n);
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++) g[i][j] = false;

	int x, y; while (scanf("%d%d", &x, &y) != EOF) {
		x--, y--;
		if (x != y && !g[x][y])
			e[x].push_back(y), e[y].push_back(x);
		g[x][y] = g[y][x] = true;
	}

	for (int i = 0; i < n; i++) match[i] = -1;
	for (int i = 0; i < n; i++) if (match[i] == -1) aug(i);

	int tot = 0;
	for (int i = 0; i < n; i++) if (match[i] != -1) tot++;
	printf("%d\n", tot);
	for (int i = 0; i < n; i++) if (match[i] > i)
		printf("%d %d\n", i + 1, match[i] + 1);
	return 0;
}
```

3.8 KM

```
/******************************************************
Bipartite Graph Maximum Weighted Matching
(kuhn munkras algorithm O(m*m*n))
adjacent matrix: mat
notice: m <= n
init: for(i=0;i<MAXN;i++)
           for(j=0;j<MAXN;j++) mat[i][j]=-inf;
for existing edges: mat[i][j]=val;
********************************************************/

#define MAXN 310
#define inf 1000000000
#define _clr(x) memset(x,-1,sizeof(int)*MAXN)
int KM(int m, int n, int mat[][MAXN], int *match1, int *match2) {
    int s[MAXN], t[MAXN], l1[MAXN], l2[MAXN];
    int p, q, i, j, k, ret = 0;
    for(i = 0; i < m; i++) {
        l1[i] = -inf;
        for(j = 0; j < n; j++)
            l1[i] = mat[i][j] > l1[i] ? mat[i][j] : l1[i];
        if(l1[i] == -inf) return -1;
    }
    for(i = 0; i < n; i++)
        l2[i] = 0;
    _clr(match1);
    _clr(match2);
    for(i = 0; i < m; i++) {
        _clr(t);
        p = 0;
        q = 0;
        for(s[0] = i; p <= q && match1[i] < 0; p++) {
            for(k = s[p], j = 0; j < n && match1[i] < 0; j++) {
                if(l1[k] + l2[j] == mat[k][j] && t[j] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if(s[q] < 0) {
                        for(p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
        if(match1[i] < 0) {
            i--;
            p = inf;
            for(k = 0; k <= q; k++) {
                for(j = 0; j < n; j++) {
                    if(t[j] < 0 && l1[s[k]] + l2[j] - mat[s[k]][j] < p)
                        p = l1[s[k]] + l2[j] - mat[s[k]][j];
                }
            }
            for(j = 0; j < n; j++)
                l2[j] += t[j] < 0 ? 0 : p;
            for(k = 0; k <= q; k++)
                l1[s[k]] -= p;
        }
    }
    for(i = 0; i < m; i++)
        ret += mat[i][match1[i]];
    return ret;
}
```

3.9 General Graph MWM

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;

typedef long long s64;

const int INF = 2147483647;

const int MaxN = 400;
const int MaxM = 79800;

template <class T>
inline void tension(T &a, const T &b)
{
        if (b < a)
                a = b;
}
template <class T>
inline void relax(T &a, const T &b)
{
        if (b > a)
                a = b;
}
template <class T>
inline int size(const T &a)
{
        return (int)a.size();
}

inline int getint()
{
        char c;
        while (c = getchar(), ’0’ > c || c > ’9’);

        int res = c - ’0’;
        while (c = getchar(), ’0’ <= c && c <= ’9’)
                res = res * 10 + c - ’0’;
        return res;
}

const int MaxNX = MaxN + MaxN;

struct edge
{
        int v, u, w;
```

```
	edge(){}
	edge(const int &_v, const int &_u, const int &_w)
		: v(_v), u(_u), w(_w){}
};

int n, m;
edge mat[MaxNX + 1][MaxNX + 1];

int n_matches;
s64 tot_weight;
int mate[MaxNX + 1];
int lab[MaxNX + 1];

int q_n, q[MaxN];
int fa[MaxNX + 1], col[MaxNX + 1];
int slackv[MaxNX + 1];

int n_x;
int bel[MaxNX + 1], blofrom[MaxNX + 1][MaxN + 1];
vector<int> bloch[MaxNX + 1];

inline int e_delta(const edge &e) // does not work inside blossoms
{
	return lab[e.v] + lab[e.u] - mat[e.v][e.u].w * 2;
}
inline void update_slackv(int v, int x)
{
	if (!slackv[x] || e_delta(mat[v][x]) < e_delta(mat[slackv[x]][x]))
		slackv[x] = v;
}
inline void calc_slackv(int x)
{
	slackv[x] = 0;
	for (int v = 1; v <= n; v++)
		if (mat[v][x].w > 0 && bel[v] != x && col[bel[v]] == 0)
			update_slackv(v, x);
}

inline void q_push(int x)
{
	if (x <= n)
		q[q_n++] = x;
	else
	{
		for (int i = 0; i < size(bloch[x]); i++)
			q_push(bloch[x][i]);
	}
}
inline void set_mate(int xv, int xu)
{
	mate[xv] = mat[xv][xu].u;
	if (xv > n)
	{
		edge e = mat[xv][xu];
		int xr = blofrom[xv][e.v];
		int pr = find(bloch[xv].begin(), bloch[xv].end(), xr) - bloch[xv].begin();
		if (pr % 2 == 1)
		{
			reverse(bloch[xv].begin() + 1, bloch[xv].end());
```

```
                        pr = size(bloch[xv]) - pr;
                }

                for (int i = 0; i < pr; i++)
                        set_mate(bloch[xv][i], bloch[xv][i ^ 1]);
                set_mate(xr, xu);

                rotate(bloch[xv].begin(), bloch[xv].begin() + pr, bloch[xv].end());
        }
}
inline void set_bel(int x, int b)
{
        bel[x] = b;
        if (x > n)
        {
                for (int i = 0; i < size(bloch[x]); i++)
                        set_bel(bloch[x][i], b);
        }
}

inline void augment(int xv, int xu)
{
        while (true)
        {
                int xnu = bel[mate[xv]];
                set_mate(xv, xu);
                if (!xnu)
                        return;
                set_mate(xnu, bel[fa[xnu]]);
                xv = bel[fa[xnu]], xu = xnu;
        }
}
inline int get_lca(int xv, int xu)
{
        static bool book[MaxNX + 1];
        for (int x = 1; x <= n_x; x++)
                book[x] = false;
        while (xv || xu)
        {
                if (xv)
                {
                        if (book[xv])
                                return xv;
                        book[xv] = true;
                        xv = bel[mate[xv]];
                        if (xv)
                                xv = bel[fa[xv]];
                }
                swap(xv, xu);
        }
        return 0;
}

inline void add_blossom(int xv, int xa, int xu)
{
        int b = n + 1;
        while (b <= n_x && bel[b])
                b++;
        if (b > n_x)
```

```
                n_x++;

        lab[b] = 0;
        col[b] = 0;

        mate[b] = mate[xa];

        bloch[b].clear();
        bloch[b].push_back(xa);
        for (int x = xv; x != xa; x = bel[fa[bel[mate[x]]]])
                bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[x]]);
        reverse(bloch[b].begin() + 1, bloch[b].end());
        for (int x = xu; x != xa; x = bel[fa[bel[mate[x]]]])
                bloch[b].push_back(x), bloch[b].push_back(bel[mate[x]]), q_push(bel[mate[x]]);

        set_bel(b, b);

        for (int x = 1; x <= n_x; x++)
        {
                mat[b][x].w = mat[x][b].w = 0;
                blofrom[b][x] = 0;
        }
        for (int i = 0; i < size(bloch[b]); i++)
        {
                int xs = bloch[b][i];
                for (int x = 1; x <= n_x; x++)
                        if (mat[b][x].w == 0 || e_delta(mat[xs][x]) < e_delta(mat[b][x]))
                                mat[b][x] = mat[xs][x], mat[x][b] = mat[x][xs];
                for (int x = 1; x <= n_x; x++)
                        if (blofrom[xs][x])
                                blofrom[b][x] = xs;
        }
        calc_slackv(b);
}
inline void expand_blossom1(int b) // lab[b] == 1
{
        for (int i = 0; i < size(bloch[b]); i++)
                set_bel(bloch[b][i], bloch[b][i]);

        int xr = blofrom[b][mat[b][fa[b]].v];
        int pr = find(bloch[b].begin(), bloch[b].end(), xr) - bloch[b].begin();
        if (pr % 2 == 1)
        {
                reverse(bloch[b].begin() + 1, bloch[b].end());
                pr = size(bloch[b]) - pr;
        }

        for (int i = 0; i < pr; i += 2)
        {
                int xs = bloch[b][i], xns = bloch[b][i + 1];
                fa[xs] = mat[xns][xs].v;
                col[xs] = 1, col[xns] = 0;
                slackv[xs] = 0, calc_slackv(xns);
                q_push(xns);
        }
        col[xr] = 1;
        fa[xr] = fa[b];
        for (int i = pr + 1; i < size(bloch[b]); i++)
        {
                int xs = bloch[b][i];
                col[xs] = -1;
                calc_slackv(xs);
        }

        bel[b] = 0;
}
inline void expand_blossom_final(int b) // at the final stage
{
        for (int i = 0; i < size(bloch[b]); i++)
        {
                if (bloch[b][i] > n && lab[bloch[b][i]] == 0)
                        expand_blossom_final(bloch[b][i]);
                else
                        set_bel(bloch[b][i], bloch[b][i]);
        }
        bel[b] = 0;
}

inline bool on_found_edge(const edge &e)
{
        int xv = bel[e.v], xu = bel[e.u];
        if (col[xu] == -1)
        {
                int nv = bel[mate[xu]];
                fa[xu] = e.v;
                col[xu] = 1, col[nv] = 0;
                slackv[xu] = slackv[nv] = 0;
                q_push(nv);
        }
        else if (col[xu] == 0)
        {
                int xa = get_lca(xv, xu);
                if (!xa)
                {
                        augment(xv, xu), augment(xu, xv);
                        for (int b = n + 1; b <= n_x; b++)
                                if (bel[b] == b && lab[b] == 0)
                                        expand_blossom_final(b);
                        return true;
                }
                else
                        add_blossom(xv, xa, xu);
        }
        return false;
}

bool match()
{
        for (int x = 1; x <= n_x; x++)
                col[x] = -1, slackv[x] = 0;

        q_n = 0;
        for (int x = 1; x <= n_x; x++)
                if (bel[x] == x && !mate[x])
                        fa[x] = 0, col[x] = 0, slackv[x] = 0, q_push(x);
        if (q_n == 0)
                return false;
```

```
while (true)
{
        for (int i = 0; i < q_n; i++)
        {
                int v = q[i];
                for (int u = 1; u <= n; u++)
                        if (mat[v][u].w > 0 && bel[v] != bel[u])
                        {
                                int d = e_delta(mat[v][u]);
                                if (d == 0)
                                {
                                        if (on_found_edge(mat[v][u]))
                                                return true;
                                }
                                else if (col[bel[u]] == -1 || col[bel[u]] == 0)
                                        update_slackv(v, bel[u]);
                        }
        }

        int d = INF;
        for (int v = 1; v <= n; v++)
                if (col[bel[v]] == 0)
                        tension(d, lab[v]);
        for (int b = n + 1; b <= n_x; b++)
                if (bel[b] == b && col[b] == 1)
                        tension(d, lab[b] / 2);
        for (int x = 1; x <= n_x; x++)
                if (bel[x] == x && slackv[x])
                {
                        if (col[x] == -1)
                                tension(d, e_delta(mat[slackv[x]][x]));
                        else if (col[x] == 0)
                                tension(d, e_delta(mat[slackv[x]][x]) / 2);
                }

        for (int v = 1; v <= n; v++)
        {
                if (col[bel[v]] == 0)
                        lab[v] -= d;
                else if (col[bel[v]] == 1)
                        lab[v] += d;
        }
        for (int b = n + 1; b <= n_x; b++)
                if (bel[b] == b)
                {
                        if (col[bel[b]] == 0)
                                lab[b] += d * 2;
                        else if (col[bel[b]] == 1)
                                lab[b] -= d * 2;
                }

        q_n = 0;
        for (int v = 1; v <= n; v++)
                if (lab[v] == 0) // all unmatched vertices' labels are zero! cheers!
                        return false;
        for (int x = 1; x <= n_x; x++)
                if (bel[x] == x && slackv[x] && bel[slackv[x]] != x &&
                    e_delta(mat[slackv[x]][x]) == 0)
                {
```

```
				if (on_found_edge(mat[slackv[x]][x]))
					return true;
			}
		for (int b = n + 1; b <= n_x; b++)
			if (bel[b] == b && col[b] == 1 && lab[b] == 0)
				expand_blossom1(b);
	}
	return false;
}

void calc_max_weight_match()
{
	for (int v = 1; v <= n; v++)
		mate[v] = 0;

	n_x = n;
	n_matches = 0;
	tot_weight = 0;

	bel[0] = 0;
	for (int v = 1; v <= n; v++)
		bel[v] = v, bloch[v].clear();
	for (int v = 1; v <= n; v++)
		for (int u = 1; u <= n; u++)
			blofrom[v][u] = v == u ? v : 0;

	int w_max = 0;
	for (int v = 1; v <= n; v++)
		for (int u = 1; u <= n; u++)
			relax(w_max, mat[v][u].w);
	for (int v = 1; v <= n; v++)
		lab[v] = w_max;

	while (match())
		n_matches++;

	for (int v = 1; v <= n; v++)
		if (mate[v] && mate[v] < v)
			tot_weight += mat[v][mate[v]].w;
}

int main()
{
	n = getint(), m = getint();

	for (int v = 1; v <= n; v++)
		for (int u = 1; u <= n; u++)
			mat[v][u] = edge(v, u, 0);

	for (int i = 0; i < m; i++)
	{
		int v = getint(), u = getint(), w = getint();
		mat[v][u].w = mat[u][v].w = w;
	}

	calc_max_weight_match();

	printf("%lld\n", tot_weight);
	for (int v = 1; v <= n; v++)
```

```
            printf("%d ", mate[v]);
        printf("\n");

        return 0;
}
```

4 Geometry

4.1 Closet Point Pair

```
const int Max=100005;
struct Point
{
    double x, y;
};
Point p[Max], *px[Max], *py[Max];

inline int Cmp_x(Point *a, Point *b)
{
    return a->x < b->x;
}

inline int Cmp_y(Point *a, Point *b)
{
    return a->y < b->y;
}

inline double Dis(Point *a, Point *b)
{
    return sqrt((a->x - b->x) * (a->x - b->x) + (a->y - b->y) * (a->y - b->y));
}

double Close(int l, int r)
{
    if (l + 1 == r)
        return Dis(px[l], px[r]);
    else if (l + 2 == r)
        return min(min(Dis(px[l], px[l + 1]), Dis(px[l + 1], px[r])), Dis(
            px[l], px[r]));
    int mid = (l + r) / 2;
    double ans = min(Close(l, mid), Close(mid + 1, r));
    int i, j, cnt;
    for (i = l, cnt = 0; i <= r; i++)
        if (px[i]->x >= px[mid]->x - ans && px[i]->x <= px[mid]->x + ans)
            py[cnt++] = px[i];
    sort(py, py + cnt, Cmp_y);
    for (i = 0; i < cnt; i++)
        for (j = i + 1; j < cnt; j++)
        {
            if (py[j]->y - py[i]->y >= ans)
                break;
            ans = min(ans, Dis(py[i], py[j]));
        }
    return ans;
}
```

4.2 Geometry Lweb

```
const double pi = acos(-1.0);

inline int sgn(double x) {
    if(x < -eps) return -1;
    return x > eps;
}

struct Vector {
    double x, y;
    inline void read() {
        scanf("%lf%lf", &x, &y);
    }
    Vector(double _x = 0, double _y = 0) {
        x = _x;
        y = _y;
    }
    Vector operator +(Vector a) const {
        return Vector(x + a.x, y + a.y);
    }
    Vector operator +=(Vector a) {
        return *this = *this + a;
    }
    Vector operator -(Vector a) const {
        return Vector(x - a.x, y - a.y);
    }
    Vector operator -=(Vector a) {
        return *this = *this - a;
    }
    Vector operator *(double p) const {
        return Vector(x * p, y * p);
    }
    Vector operator *=(double p) {
        return *this = *this * p;
    }
    Vector normal() {
        return Vector(x / len(), y / len());
    }
    bool operator <(const Vector a) const {
        if(x == a.x) return y < a.y;
        return x < a.x;
    }
    bool operator ==(const Vector a) const {
        return sgn(x - a.x) == 0 && sgn(y - a.y) == 0;
    }
    double len() const {
        return sqrt(x * x + y * y);
    }
    double angle() const {
        return atan2(y, x);
    }
};

inline double cross(Vector a, Vector b) {
    return a.x * b.y - a.y * b.x;
}
inline double scalar(Vector a, Vector b) {
    return a.x * b.x + a.y * b.y;
}
inline double dist(Vector A, Vector B) {
    return sqrt(1.0 * (A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
}
inline bool parallel(Vector a, Vector b, Vector c, Vector d) {
    return sgn(cross(b - a, d - c)) == 0;
}
Vector Rotate(Vector A, double rad) {
    return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
}

Vector GetLineProjection(Vector P, Vector A, Vector B) {
    //Projection of point P on line AB
    Vector v = B - A;
    Vector v1 = P - A;
    double t = scalar(v, v1) / scalar(v, v);
    return A + v * t;
}

Vector GetLineSymmetry(Vector P, Vector A, Vector B) {
    //Symmetry point of P according to line AB
    double dis = DistoLine(A, B, P) * 2;
    if(sgn(dis) == 0) return P;
    Vector v = Rotate(B - A, pi / 2).normal();
    if(sgn(scalar(P - A, v)) > 0) v = v * -1;
    return P + (v * dis);
}

inline bool between(Vector a, Vector b, Vector c) {
    //a is between bc
    return sgn(a.x - min(b.x, c.x)) >= 0 && sgn(a.x - max(b.x, c.x)) <= 0 && sgn(a.y - min(b.y,
        c.y)) >= 0 &&
          sgn(a.y - max(b.y, c.y)) <= 0;
}

//Distance of C to segment AB
inline double DistoSegment(Vector A, Vector B, Vector C) {
    if(dist(A, B) < eps) return dist(B, C);
    if(scalar(B - A, C - A) < -eps) return dist(A, C);
    if(scalar(A - B, C - B) < -eps) return dist(B, C);
    return fabs(cross(B - A, C - A) / dist(A, B));
}

//Distance of C to line AB
inline double DistoLine(Vector A, Vector B, Vector C) {
    return fabs(cross(B - A, C - A) / dist(A, B));
}

inline int segmentsegment(Vector a, Vector b, Vector c, Vector d) {
    //-1 No intersection point
    //0 Parallel && No intersection point
    //1 Parallel && Inf intersection point
    //2 Parallel && Intersect at end point
    //3 Intersect at end point
    //4 Strictly intersect
    double d1 = cross(a - c, a - d), d2 = cross(b - c, b - d), d3 = cross(c - a, c - b), d4 =
        cross(d - a, d - b);
    if(sgn(d1 * d2) < 0 && sgn(d3 * d4) < 0) return 4;
    if(sgn(d1) == 0 && sgn(d2) == 0 && sgn(d3) == 0 && sgn(d4) == 0) {
        if(a == c && !between(d, a, b)) return 2;
        if(a == d && !between(c, a, b)) return 2;
        if(b == c && !between(d, a, b)) return 2;
        if(b == d && !between(c, a, b)) return 2;
        if(between(c, a, b) || between(d, a, b)) return 1;
        if(between(a, c, d) || between(b, c, d)) return 1;
        return 0;
    }
    if(sgn(d1) == 0 && between(a, c, d)) return 3;
    if(sgn(d2) == 0 && between(b, c, d)) return 3;
    if(sgn(d3) == 0 && between(c, a, b)) return 3;
    if(sgn(d4) == 0 && between(d, a, b)) return 3;
    return -1;
}

inline bool in(Vector t, Vector *po, int n) {
    int cnt = 0;
    for(int i = 0; i < n; ++i) {
        int a = i, b = (i + 1) % n;
        if(sgn(po[a].y - po[b].y) < 0) swap(a, b);
        int x = sgn(cross(po[a] - t, po[b] - t));
        if(!x) {
            if(between(t, po[a], po[b])) return 1;
        } else if(x < 0 && sgn(po[a].y - t.y) >= 0 && sgn(t.y - po[b].y) > 0) ++cnt;
    }
    return cnt & 1;
}

inline int get_convex(Vector *p, Vector *convex, int n) {
    sort(p, p + n);
    int cnt = 0;
    for(int i = 0; i < n; ++i) {
        while(cnt >= 2 && sgn(cross(convex[cnt - 1] - p[i], convex[cnt - 2] - p[i])) <= 0) --cnt;
        convex[cnt++] = p[i];
    }
    int half = cnt;
    for(int i = n - 2; i >= 0; --i) {
        while(cnt - half >= 1 && sgn(cross(convex[cnt - 1] - p[i], convex[cnt - 2] - p[i])) <= 0)
            --cnt;
        convex[cnt++] = p[i];
    }
    return cnt - 1;
}

inline double PolygonArea(Vector *p, int n) {
    double area = 0;
    for(int i = 1; i < n - 1; i++) area += cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}


double CircleCircleArea(double r1, double r2, double s) {
    //Circle area intersection
    //r1 r2: radius
    //s: Circle center distance
    if(r1 + r2 < s)
        return 0;
    else if(r2 - r1 >= s)
```

```
        return pi * r1 * r1;
    else if(r1 - r2 >= s)
        return pi * r2 * r2;
    double q1 = acos((r1 * r1 + s * s - r2 * r2) / (2 * r1 * s));
    double q2 = acos((r2 * r2 + s * s - r1 * r1) / (2 * r2 * s));
    return (r1 * r1 * q1 + r2 * r2 * q2 - r1 * s * sin(q1));
}

typedef Vector Point;
struct Circle {
    Point c;
    double r;
    Circle(Point c = Point(0, 0), double r = 0): c(c), r(r) {}
    Point point(double a) {
        return Point(c.x + r * cos(a), c.y + r * sin(a));
    }
};

struct Line {
    Point p;
    Vector v;
    Line(Point p = Point(0, 0), Vector v = Vector(0, 1)): p(p), v(v) {}
    Point point(double t) {
        return Point(p + v * t);
    }
};

int getLineCircleIntersection(Line L, Circle C, double &t1, double &t2, vector<Point> &sol) {
    double a = L.v.x;
    double b = L.p.x - C.c.x;
    double c = L.v.y;
    double d = L.p.y - C.c.y;
    double e = a * a + c * c;
    double f = 2 * (a * b + c * d);
    double g = b * b + d * d - C.r * C.r;
    double delta = f * f - 4 * e * g;
    if(sgn(delta) < 0) return 0;
    if(sgn(delta) == 0) {
        t1 = t2 = -f / (2 * e);
        sol.push_back(L.point(t1));
        return 1;
    } else {
        t1 = (-f - sqrt(delta)) / (2 * e);
        t2 = (-f + sqrt(delta)) / (2 * e);
        sol.push_back(L.point(t1));
        sol.push_back(L.point(t2));
        return 2;
    }
}

double angle(Vector v) {
    return atan2(v.y, v.x);
}
double Length(Vector v) {
    return sqrt(scalar(v, v));
}
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point> &sol) {
    double d = Length(C1.c - C2.c);
    if(sgn(d) == 0) {
        if(sgn(C1.r - C2.r) == 0) return -1; // coincidence
        else return 0;  //  contains
    }
    if(sgn(C1.r + C2.r - d) < 0) return 0; // away from
    if(sgn(fabs(C1.r - C2.r) - d) > 0) return 0; // contains
    double a = angle(C2.c - C1.c);
    double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
    Point p1 = C1.point(a - da);
    Point p2 = C1.point(a + da);
    sol.push_back(p1);
    if(p1 == p2) return 1; // tangent
    else {
        sol.push_back(p2);
        return 2;
    }
}

int getTangents(Point p, Circle C, vector <Point> &sol) {
    //Tangent for p to C
    Vector u = C.c - p;
    double dist = Length(u);
    if(sgn(dist - C.r) < 0) return 0;
    else if(sgn(dist - C.r) == 0) {
        sol.push_back(Rotate(u, pi / 2));
        return 1;
    } else {
        double ang = asin(C.r / dist);
        sol.push_back(Rotate(u, -ang));
        sol.push_back(Rotate(u, +ang));
        return 2;
    }
}

bool flag = 0;
int getTangents(Circle A, Circle B, Point *a, Point *b) {
    //Common tangent for A B
    int cnt = 0;
    if(A.r < B.r) {
        flag = 1;
        swap(A, B);
        swap(a, b);
    }
    double d = Length(A.c - B.c);
    double rdiff = A.r - B.r;
    double rsum = A.r + B.r;
    if(sgn(d - rdiff) < 0) return 0; // contains
    double base = angle(B.c - A.c);
    if(sgn(d) == 0 && sgn(rdiff) == 0) return -1 ; // coincidence
    if(sgn(d - rdiff) == 0) {    // tangent inside
        a[cnt] = A.point(base);
        b[cnt] = B.point(base);
        cnt++;
        return 1;
    }

    double ang = acos(rdiff / d);
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(base + ang);
    cnt++;
    a[cnt] = A.point(base - ang);
    b[cnt] = B.point(base - ang);
    cnt++;
    if(sgn(d - rsum) == 0) { // tagent outside
        a[cnt] = A.point(base);
        b[cnt] = B.point(base + pi);
        cnt++;
    } else if(sgn(d - rsum) > 0) { // away from
        double ang_in = acos(rsum / d);
        a[cnt] = A.point(base + ang_in);
        b[cnt] = B.point(base + ang_in + pi);
        cnt++;
        a[cnt] = A.point(base - ang_in);
        b[cnt] = B.point(base - ang_in + pi);
        cnt++;
    }
    return cnt;
}
```

4.3 Geometry Hezhu

```
Point GetLineIntersection(Point p, Vector v, Point q, Vector w) {
  Vector u = p - q;
  double t = Cross(w, u) / Cross(v, w);
  return p + v * t;
}
Point GetLineProjection(Point P, Point A, Point B) {
  Vector v = B - A;
  return A + v * (Dot(v, P - A) / Dot(v, v));
}
double DistanceToLine(Point P, Point A, Point B) {
  Vector v1 = B - A, v2 = P - A;
  return fabs(Cross(v1, v2)) / Length(v1);
}
bool OnSegment(Point p, Point a1, Point a2) {
  return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 - p, a2 - p)) < 0;
}
void getLineGeneralEquation(const Point &p1, const Point &p2, double &a,
             double &b, double &c) {
  a = p2.y - p1.y;
  b = p1.x - p2.x;
  c = -a * p1.x - b * p1.y;
}
double DistanceToSegment(Point p, Point a, Point b) {
  if (a == b) return Length(p - a);
  Vector v1 = b - a, v2 = p - a, v3 = p - b;
  if (dcmp(Dot(v1, v2)) < 0)
    return Length(v2);
  else if (dcmp(Dot(v1, v3)) > 0)
    return Length(v3);
  else
    return fabs(Cross(v1, v2)) / Length(v1);
}
double dis_pair_seg(Point p1, Point p2, Point p3, Point p4) {
  return min(
    min(DistanceToSegment(p1, p3, p4), DistanceToSegment(p2, p3, p4)),
    min(DistanceToSegment(p3, p1, p2), DistanceToSegment(p4, p1, p2)));
```

```
}
bool SegmentIntersection(Point a1, Point a2, Point b1, Point b2) {
  double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1),
       c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
  return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
struct Line {
  Point P;
  Vector v;
  double ang;
  Line() {}
  Line(Point P, Vector v) : P(P), v(v) { ang = atan2(v.y, v.x); }
  Point point(double a) { return p + (v * a); }
  bool operator<(const Line &L) const { return ang < L.ang; }
};
bool OnLeft(const Line &L, const Point &p) { return Cross(L.v, p - L.P) > 0; }
vector<Point> HalfplaneIntersection(vector<Line> L) {
  int n = L.size();
  sort(L.begin(), L.end());
  int first, last;
  vector<Point> p(n);
  vector<Line> q(n);
  vector<Point> ans;
  q[first = last = 0] = L[0];
  for (int i = 1; i < n; i++) {
    while (first < last && !OnLeft(L[i], p[last - 1])) last--;
    while (first < last && !OnLeft(L[i], p[first])) first++;
    q[++last] = L[i];
    if (fabs(Cross(q[last].v, q[last - 1].v)) < eps) {
      last--;
      if (OnLeft(q[last], L[i].P)) q[last] = L[i];
    }
    if (first < last)
      p[last - 1] = GetLineIntersection(q[last - 1], q[last]);
  }
  while (first < last && !OnLeft(q[first], p[last - 1])) last--;
  if (last - first <= 1) return ans;
  p[last] = GetLineIntersection(q[last], q[first]);
  for (int i = first; i <= last; i++) ans.push_back(p[i]);
  return ans;
}
Point PolyGravity(Point *p, int n) {
  Point tmp, g = Point(0, 0);
  double sumArea = 0, area;
  for (int i = 2; i < n; ++i) {
    area = Cross(p[i - 1] - p[0], p[i] - p[0]);
    sumArea += area;
    tmp.x = p[0].x + p[i - 1].x + p[i].x;
    tmp.y = p[0].y + p[i - 1].y + p[i].y;
    g.x += tmp.x * area;
    g.y += tmp.y * area;
  }
  g.x /= (sumArea * 3.0);
  g.y /= (sumArea * 3.0);
  return g;
}
vector<Point> ConvexHull(vector<Point> &p) {
  sort(p.begin(), p.end());
  p.erase(unique(p.begin(), p.end()), p.end());
  int n = p.size();
  int m = 0;
  vector<Point> ch(n + 1);
  for (int i = 0; i < n; i++) {
    while (m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0)
      m--;
    ch[m++] = p[i];
  }
  int k = m;
  for (int i = n - 2; i >= 0; i--) {
    while (m > k && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0)
      m--;
    ch[m++] = p[i];
  }
  if (n > 1) m--;
  ch.resize(m);
  return ch;
}
int isPointInPolygon(Point p, Polygon poly) {
  int wn = 0;
  int n = poly.size();
  for (int i = 0; i < n; i++) {
    if (OnSegment(p, poly[i], poly[(i + 1) % n])) return -1;
    int k = dcmp(Cross(poly[(i + 1) % n] - poly[i], p - poly[i]));
    int d1 = dcmp(poly[i].y - p.y);
    int d2 = dcmp(poly[(i + 1) % n].y - p.y);
    if (k > 0 && d1 <= 0 && d2 > 0) wn++;
    if (k < 0 && d2 <= 0 && d1 > 0) wn--;
  }
  if (wn != 0) return 1;
  return 0;
}
int diameter2(vector<Point> &points) {
  vector<Point> p = ConvexHull(points);
  int n = p.size();
  if (n == 1) return 0;
  if (n == 2) return Dist2(p[0], p[1]);
  p.push_back(p[0]);
  int ans = 0;
  for (int u = 0, v = 1; u < n; u++) {
    for (;;) {
      int diff = Cross(p[u + 1] - p[u], p[v + 1] - p[v]);
      if (diff <= 0) {
        ans = max(ans, Dist2(p[u], p[v]));
        if (diff == 0) ans = max(ans, Dist2(p[u], p[v + 1]));
        break;
      }
      v = (v + 1) % n;
    }
  }
  return ans;
}
double RC_Distance(Point *ch1, Point *ch2, int n, int m) {
  int q = 0, p = 0;
  REP(i, n) if (ch1[i].y - ch1[p].y < -eps) p = i;
  REP(i, m) if (ch2[i].y - ch2[q].y > eps) q = i;
  ch1[n] = ch1[0];
  ch2[m] = ch2[0];
  double tmp, ans = 1e100;
  REP(i, n) {
    while ((tmp = Cross(ch1[p + 1] - ch1[p], ch2[q + 1] - ch1[p]) -
            Cross(ch1[p + 1] - ch1[p], ch2[q] - ch1[p])) > eps)
      q = (q + 1) % m;
    if (tmp < -eps)
      ans = min(ans, DistanceToSegment(ch2[q], ch1[p], ch1[p + 1]));
    else
      ans =
        min(ans, dis_pair_seg(ch1[p], ch1[p + 1], ch2[q], ch2[q + 1]));
    p = (p + 1) % n;
  }
  return ans;
}
double RC_Triangle(Point *res, int n) {
  if (n < 3) return 0;
  double ans = 0, tmp;
  res[n] = res[0];
  int j, k;
  REP(i, n) {
    j = (i + 1) % n;
    k = (j + 1) % n;
    while ((j != k) && (k != i)) {
      while (Cross(res[j] - res[i], res[k + 1] - res[i]) >
           Cross(res[j] - res[i], res[k] - res[i]))
        k = (k + 1) % n;
      tmp = Cross(res[j] - res[i], res[k] - res[i]);
      if (tmp > ans) ans = tmp;
      j = (j + 1) % n;
    }
  }
  return ans;
}
double fermat_point(Point *pt, int n, Point &ptres) {
  Point u, v;
  double step = 0.0, curlen, explen, minlen;
  int i, j, k, idx;
  bool flag;
  u.x = u.y = v.x = v.y = 0.0;
  REP(i, n) {
    step += fabs(pt[i].x) + fabs(pt[i].y);
    u.x += pt[i].x;
    u.y += pt[i].y;
  }
  u.x /= n;
  u.y /= n;
  flag = 0;
  while (step > eps) {
    for (k = 0; k < 10; step /= 2, ++k)
      for (i = -1; i <= 1; ++i)
        for (j = -1; j <= 1; ++j) {
          v.x = u.x + step * i;
          v.y = u.y + step * j;
          curlen = explen = 0.0;
          REP(idx, n) {
            curlen += dist(u, pt[idx]);
            explen += dist(v, pt[idx]);
          }
          if (curlen > explen) {
            u = v;
          minlen = explen;
          flag = 1;
        }
      }
  }
  ptres = u;
  return flag ? minlen : curlen;
}
bool cmpy(const int &a, const int &b) { return point[a].y < point[b].y; }
double Closest_Pair(int left, int right) {
  double d = INF;
  if (left == right) return d;
  if (left + 1 == right) return dis(left, right);
  int mid = (left + right) >> 1;
  double d1 = Closest_Pair(left, mid);
  double d2 = Closest_Pair(mid + 1, right);
  d = min(d1, d2);
  int i, j, k = 0;
  for (i = left; i <= right; i++) {
    if (fabs(point[mid].x - point[i].x) <= d) tmpt[k++] = i;
  }
  sort(tmpt, tmpt + k, cmpy);
  for (i = 0; i < k; i++) {
    for (j = i + 1; j < k && point[tmpt[j]].y - point[tmpt[i]].y < d; j++) {
      double d3 = dis(tmpt[i], tmpt[j]);
      if (d > d3) d = d3;
    }
  }
  return d;
}
int getLineCircleIntersection(Line L, Circle C, double &t1, double &t2, vector<Point> &sol) {
  double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
  double e = a * a + c * c, f = 2 * (a * b + c * d),
       g = b * b + d * d - C.r * C.r;
  double delta = f * f - 4 * e * g;
  if (dcmp(delta) < 0) return 0;
  if (dcmp(delta) == 0) {
    t1 = t2 = -f / (2 * e);
    sol.push_back(L.point(t1));
    return 1;
  }
  t1 = (-f - sqrt(delta)) / (2 * e);
  sol.push_back(L.point(t1));
  t2 = (-f + sqrt(delta)) / (2 * e);
  sol.push_back(L.point(t2));
  return 2;
}
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point> &sol) {
  double d = Length(C1.c - C2.c);
  if (dcmp(d) == 0) {
    if (dcmp(C1.r - C2.r) == 0) return -1;
    return 0;
  }
  if (dcmp(C1.r + C2.r - d) < 0) return 0;
  if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;
  double a = angle(C2.c - C1.c);
  double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
  Point p1 = C1.point(a - da), p2 = C1.point(a + da);
  sol.push_back(p1);
  if (p1 == p2) return 1;
  sol.push_back(p2);
  return 2;
}
int getTangents(Point p, Circle C, Vector *v) {
  Vector u = C.c - p;
  double dist = Length(u);
  if (dist < C.r)
    return 0;
  else if (dcmp(dist - C.r) == 0) {
    v[0] = Rotate(u, PI / 2);
    return 1;
  } else {
    double ang = asin(C.r / dist);
    v[0] = Rotate(u, -ang);
    v[1] = Rotate(u, +ang);
    return 2;
  }
}
int getTangents(Circle A, Circle B, Point *a, Point *b) {
  int cnt = 0;
  if (A.r < B.r) swap(A, B), swap(a, b);
  int d2 =
    (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y);
  int rdiff = A.r - B.r;
  int rsum = A.r + B.r;
  if (d2 < rdiff * rdiff) return 0;
  double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
  if (d2 == 0 && A.r == B.r) return -1;
  if (d2 == rdiff * rdiff) {
    a[cnt] = A.point(base);
    b[cnt] = B.point(base);
    cnt++;
    return 1;
  }
  double ang = acos((A.r - B.r) / sqrt(d2));
  a[cnt] = A.point(base + ang);
  b[cnt] = B.point(base + ang);
  cnt++;
  a[cnt] = A.point(base - ang);
  b[cnt] = B.point(base - ang);
  cnt++;
  if (d2 == rsum * rsum) {
    a[cnt] = A.point(base);
    b[cnt] = B.point(PI + base);
    cnt++;
  } else if (d2 > rsum * rsum) {
    double ang = acos((A.r + B.r) / sqrt(d2));
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(PI + base + ang);
    cnt++;
    a[cnt] = A.point(base - ang);
    b[cnt] = B.point(PI + base - ang);
    cnt++;
  }
  return cnt;
}
Circle CircumscribedCircle(Point p1, Point p2, Point p3) {
  double Bx = p2.x - p1.x, By = p2.y - p1.y;
  double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
  double D = 2 * (Bx * Cy - By * Cx);
  double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
  double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
  Point p = Point(cx, cy);
  return Circle(p, Length(p1 - p));
}
Circle InscribedCircle(Point p1, Point p2, Point p3) {
  double a = Length(p2 - p3);
  double b = Length(p3 - p1);
  double c = Length(p1 - p2);
  Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
  return Circle(p, DistanceToLine(p, p1, p2));
}
vector<Point> CircleThroughPointTangentToLineGivenRadius(Point p, Line L, double r) {
  vector<Point> ans;
  double t1, t2;
  getLineCircleIntersection(L.move(-r), Circle(p, r), t1, t2, ans);
  getLineCircleIntersection(L.move(r), Circle(p, r), t1, t2, ans);
  return ans;
}
vector<Point> CircleTangentToLinesGivenRadius(Line a, Line b, double r) {
  vector<Point> ans;
  Line L1 = a.move(-r), L2 = a.move(r);
  Line L3 = b.move(-r), L4 = b.move(r);
  ans.push_back(GetLineIntersection(L1, L3));
  ans.push_back(GetLineIntersection(L1, L4));
  ans.push_back(GetLineIntersection(L2, L3));
  ans.push_back(GetLineIntersection(L2, L4));
  return ans;
}
vector<Point> CircleTangentToTwoDisjointCirclesWithRadius(Circle c1, Circle c2, double r) {
  vector<Point> ans;
  Vector v = c2.c - c1.c;
  double dist = Length(v);
  int d = dcmp(dist - c1.r - c2.r - r * 2);
  if (d > 0) return ans;
  getCircleCircleIntersection(Circle(c1.c, c1.r + r), Circle(c2.c, c2.r + r), ans);
  return ans;
}
int getSegCircleIntersection(Line L, Circle C, Point *sol) {
  Vector nor = normal(L.v);
  Line pl = Line(C.c, nor);
  Point ip = GetIntersection(pl, L);
  double dis = Length(ip - C.c);
  if (dcmp(dis - C.r) > 0) return 0;
  Point dxy = vecunit(L.v) * sqrt(sqr(C.r) - sqr(dis));
  int ret = 0;
  sol[ret] = ip + dxy;
  if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
  sol[ret] = ip - dxy;
  if (OnSegment(sol[ret], L.p, L.point(1))) ret++;
  return ret;
}
double SegCircleArea(Circle C, Point a, Point b) {
  double a1 = angle(a - C.c);
  double a2 = angle(b - C.c);
  double da = fabs(a1 - a2);
  if (da > PI) da = PI * 2.0 - da;
  return dcmp(Cross(b - C.c, a - C.c)) * da * sqr(C.r) / 2.0;
}
double PolyCiclrArea(Circle C, Point *p, int n) {
  double ret = 0.0;
  Point sol[2];
  p[n] = p[0];
  REP(i, n) {
    double t1, t2;
    int cnt = getSegCircleIntersection(Line(p[i], p[i + 1] - p[i]), C, sol);
    if (cnt == 0) {
      if (!OnOrInCircle(p[i], C) || !OnOrInCircle(p[i + 1], C))
        ret += SegCircleArea(C, p[i], p[i + 1]);
      else
        ret += Cross(p[i + 1] - C.c, p[i] - C.c) / 2.0;
    }
    if (cnt == 1) {
      if (OnOrInCircle(p[i], C) && !OnOrInCircle(p[i + 1], C))
        ret += Cross(sol[0] - C.c, p[i] - C.c) / 2.0,
          ret += SegCircleArea(C, sol[0], p[i + 1]);
      else
        ret += SegCircleArea(C, p[i], sol[0]),
          ret += Cross(p[i + 1] - C.c, sol[0] - C.c) / 2.0;
    }
    if (cnt == 2) {
      if ((p[i] < p[i + 1]) ^ (sol[0] < sol[1])) swap(sol[0], sol[1]);
      ret += SegCircleArea(C, p[i], sol[0]);
      ret += Cross(sol[1] - C.c, sol[0] - C.c) / 2.0;
      ret += SegCircleArea(C, sol[1], p[i + 1]);
    }
  }
  return fabs(ret);
}
double area[N];
int n;
struct cp {
  double x, y, r, angle;
  int d;
  cp() {}
  cp(double xx, double yy, double ang = 0, int t = 0) {
    x = xx;
    y = yy;
    angle = ang;
    d = t;
  }
  void get() {
    scanf("%lf%lf%lf", &x, &y, &r);
    d = 1;
  }
} cir[N], tp[N * 2];
double dis(cp a, cp b) { return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)); }
double cross(cp p0, cp p1, cp p2) {
  return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
}
bool circmp(const cp &u, const cp &v) { return dcmp(u.r - v.r) < 0; }
bool cmp(const cp &u, const cp &v) {
  if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
  return u.d > v.d;
}
double calc(cp cir, cp cp1, cp cp2) {
  double ans = (cp2.angle - cp1.angle) * sqr(cir.r) - cross(cir, cp1, cp2) +
         cross(cp(0, 0), cp1, cp2);
  return ans / 2;
}
void CirUnion(cp cir[], int n) {
  cp cp1, cp2;
  sort(cir, cir + n, circmp);
  for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j)
      if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0)
        cir[i].d++;
  for (int i = 0; i < n; ++i) {
    int tn = 0, cnt = 0;
    for (int j = 0; j < n; ++j) {
      if (i == j) continue;
      if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r, cp2, cp1) < 2)
        continue;
      cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
      cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
      cp1.d = 1;
      tp[tn++] = cp1;
      cp2.d = -1;
      tp[tn++] = cp2;
      if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
    }
    tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
    tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
    sort(tp, tp + tn, cmp);
    int p, s = cir[i].d + tp[0].d;
    for (int j = 1; j < tn; ++j) {
      p = s;
      s += tp[j].d;
      area[p] += calc(cir[i], tp[j - 1], tp[j]);
    }
  }
}
void solve() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) cir[i].get();
  memset(area, 0, sizeof(area));
  CirUnion(cir, n);
  for (int i = 1; i <= n; ++i) { area[i] -= area[i + 1]; }
  double tot = 0;
  for (int i = 1; i <= n; i++) tot += area[i];
  printf("%f\n", tot);
}
inline double cross(point o, point a, point b) { return (a - o) * (b - o); }
PDI s[maxN * maxp * 2];
Polygon P[maxN];
double S, ts;
int N;
inline double seg(point o, point a, point b) {
  if (cmp(b.x - a.x) == 0) return (o.y - a.y) / (b.y - a.y);
  return (o.x - a.x) / (b.x - a.x);
}
double PolygonUnion() {
  int M, c1, c2;
  double s1, s2, ret = 0;
  for (int i = 0; i < N; i++)
```

```
    for (int ii = 0; ii < P[i].n; ii++) {
      M = 0;
      s[M++] = mp(0.00, 0);
      s[M++] = mp(1.00, 0);
      for (int j = 0; j < N; j++)
        if (i != j)
          for (int jj = 0; jj < P[j].n; jj++) {
            c1 = cmp(cross(P[i][ii], P[i][ii + 1], P[j][jj]));
            c2 = cmp(cross(P[i][ii], P[i][ii + 1], P[j][jj + 1]));
            if (c1 == 0 && c2 == 0) {
              if (((P[i][ii + 1] - P[i][ii]) ^
                 (P[j][jj + 1] - P[j][jj])) > 0 &&
                i > j) {
                s[M++] = mp(
                  seg(P[j][jj], P[i][ii], P[i][ii + 1]), 1);
                s[M++] = mp(
                  seg(P[j][jj + 1], P[i][ii], P[i][ii + 1]),
                  -1);
              }
            } else {
              s1 = cross(P[j][jj], P[j][jj + 1], P[i][ii]);
              s2 = cross(P[j][jj], P[j][jj + 1], P[i][ii + 1]);
              if (c1 >= 0 && c2 < 0)
                s[M++] = mp(s1 / (s1 - s2), 1);
              else if (c1 < 0 && c2 >= 0)
                s[M++] = mp(s1 / (s1 - s2), -1);
            }
          }
      sort(s, s + M);
      double pre = min(max(s[0].x, 0.0), 1.0), now;
      double sum = 0;
      int cov = s[0].y;
      for (int j = 1; j < M; j++) {
        now = min(max(s[j].x, 0.0), 1.0);
        if (!cov) sum += now - pre;
        cov += s[j].y;
        pre = now;
      }
      ret += P[i][ii] * P[i][ii + 1] * sum;
    }
  return ret / 2;
}
int main() {
  for (int i = 0; i < N; i++) {
    P[i].n = 4;
    P[i].input();
    ts = P[i].Area();
    if (cmp(ts < 0)) {
      reverse(P[i].p, P[i].p + P[i].n);
      P[i][P[i].n] = P[i][0];
      ts = -ts;
    }
    S += ts;
  }
  printf("%.9lf\n", S / PolygonUnion());
}
// count(c / a + 1, c % a, a, b) + c / a + 1
long long count(long long n, long long a, long long b, long long m) {
  if (b == 0) { return n * (a / m); }
  if (a >= m) { return n * (a / m) + count(n, a % m, b, m); }
  if (b >= m) { return (n - 1) * n / 2 * (b / m) + count(n, a, b % m, m); }
  return count((a + b * n) / m, (a + b * n) % m, m, b);
}
bool TriSegIntersection(Point3 P0, Point3 P1, Point3 P2, Point3 A, Point3 B, Point3 &P) {
  Vector3 n = Cross(P1 - P0, P2 - P0);
  if (dcmp(Dot(n, B - A)) == 0) return false;
  double t = Dot(n, P0 - A) / Dot(n, B - A);
  if (dcmp(t) < 0 || dcmp(t - 1) > 0) return false;
  P = A + (B - A) * t;
  return PointInTri(P, P0, P1, P2);
}
struct Face {
  int v[3];
  Vector3 normal(Point3 *P) const {
    return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);
  }
  int cansee(Point3 *p, int i) const {
    return Dot(P[i] - P[v[0]], normal(P)) > 0 ? 1 : 0;
  }
};
vector<Face> CH3D(Point3 *P, int n) {
  vector<Face> cur;
  cur.push_back((Face){{0, 1, 2}});
  cur.push_back((Face){{2, 1, 0}});
  for (int i = 3; i < n; ++i) {
    vector<Face> next;
    for (int j = 0; j < cur.size(); ++j) {
      Face &f = cur[j];
      int res = f.cansee(P, i);
      if (!res) next.push_back(f);
      for (int k = 0; k < 3; ++k) vis[f.v[k]][f.v[(k + 1) % 3]] = res;
    }
    for (int j = 0; j < cur.size(); ++j)
      for (int k = 0; k < 3; ++k) {
        int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];
        if (vis[a][b] != vis[b][a] && vis[a][b])
          next.push_back((Face){{a, b, i}});
      }
    cur = next;
  }
  return cur;
}
```

4.4 3D Convex

```
const int MAXN = 100;
const double EPS = 1e-8;

struct Point
{
      double x,y,z;
      Point(){}
      Point(double xx,double yy,double zz):x(xx),y(yy),z(zz){}

      Point operator -(const Point p1)
      {
           return Point(x-p1.x,y-p1.y,z-p1.z);
      }

      Point operator *(Point p)
      {
           return Point(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);
      }

      double operator ^(Point p)
      {
            return (x*p.x+y*p.y+z*p.z);
      }
};

struct CH3D
{
      struct face
      {
            int a,b,c;
            bool ok;
      };

      int n;
      Point P[MAXN];

      int num;
      face F[8*MAXN];

      int g[MAXN][MAXN];

      double vlen(Point a)
      {
            return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
      }

      Point cross(const Point &a, const Point &b, const Point &c)
      {
           return Point((b.y-a.y)*(c.z-a.z)-(b.z-a.z)*(c.y-a.y),-((b.x-a.x)*(c.z-a.z)
               -(b.z-a.z)*(c.x-a.x)),(b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x));
      }
      double area(Point a,Point b,Point c)                            //*2
      {
            return vlen((b-a)*(c-a));
      }

      double volume(Point a,Point b,Point c,Point d)                 //*6
      {
            return (b-a)*(c-a)^(d-a);
      }

      double dblcmp(Point &p,face &f)
      {
            Point m=P[f.b]-P[f.a];
            Point n=P[f.c]-P[f.a];
            Point t=p-P[f.a];
            return (m*n)^t;
      }

      void deal(int p,int a,int b)
```

```
{
    int f=g[a][b];
    face add;
    if(F[f].ok)
    {
        if(dblcmp(P[p],F[f])>EPS)
            dfs(p,f);
        else
        {
            add.a=b;
            add.b=a;
            add.c=p;
            add.ok=1;
            g[p][b]=g[a][p]=g[b][a]=num;
            F[num++]=add;
        }
    }
}

void dfs(int p,int now)
{
    F[now].ok=0;
    deal(p,F[now].b,F[now].a);
    deal(p,F[now].c,F[now].b);
    deal(p,F[now].a,F[now].c);
}

bool same(int s,int t)
{
    Point &a=P[F[s].a];
    Point &b=P[F[s].b];
    Point &c=P[F[s].c];
    return fabs(volume(a,b,c,P[F[t].a]))<EPS && fabs(volume(a,b,c,P[F[t].b]))<EPS
        && fabs(volume(a,b,c,P[F[t].c]))<EPS;
}

void solve()
{
    int i,j,tmp;
    face add;
    bool flag=true;
    num=0;
    if(n<4)
       return;
    for(i=1;i<n;i++)
    {
        if(vlen(P[0]-P[i])>EPS)
        {
              swap(P[1],P[i]);
              flag=false;
              break;
        }
    }
    if(flag)
        return;
    flag=true;
    for(i=2;i<n;i++)
    {
         if(vlen((P[0]-P[1])*(P[1]-P[i]))>EPS)
         {
              swap(P[2],P[i]);
              flag=false;
              break;
         }
    }
    if(flag)
        return;
    flag=true;
    for(i=3;i<n;i++)
    {
          if(fabs((P[0]-P[1])*(P[1]-P[2])^(P[0]-P[i]))>EPS)
          {
               swap(P[3],P[i]);
               flag=false;
               break;
          }
    }
    if(flag)
        return;
    for(i=0;i<4;i++)
    {
           add.a=(i+1)%4;
           add.b=(i+2)%4;
           add.c=(i+3)%4;
           add.ok=true;
           if(dblcmp(P[i],add)>0)
               swap(add.b,add.c);
           g[add.a][add.b]=g[add.b][add.c]=g[add.c][add.a]=num;
           F[num++]=add;
    }
    for(i=4;i<n;i++)
    {
        for(j=0;j<num;j++)
        {
             if(F[j].ok && dblcmp(P[i],F[j])>EPS)
             {
                  dfs(i,j);
                  break;
             }
        }
    }
    tmp=num;
    for(i=num=0;i<tmp;i++)
      if(F[i].ok)
      {
             F[num++]=F[i];
      }
}

double area()
{
      double res=0.0;
      if(n==3)
      {
           Point p=cross(P[0],P[1],P[2]);
           res=vlen(p)/2.0;
           return res;
      }
      for(int i=0;i<num;i++)
         res+=area(P[F[i].a],P[F[i].b],P[F[i].c]);
      return res/2.0;
}

double volume()
{
      double res=0.0;
      Point tmp(0,0,0);
      for(int i=0;i<num;i++)
         res+=volume(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);
      return fabs(res/6.0);
}

int triangle()
{
      return num;
}

int polygon()
{
    int i,j,res,flag;
    for(i=res=0;i<num;i++)
    {
        flag=1;
        for(j=0;j<i;j++)
         if(same(i,j))
         {
             flag=0;
             break;
         }
        res+=flag;
    }
    return res;
}
Point getcent()
{
    Point ans(0,0,0),temp=P[F[0].a];
    double v = 0.0,t2;
    for(int i=0;i<num;i++){
        if(F[i].ok == true){
           Point p1=P[F[i].a],p2=P[F[i].b],p3=P[F[i].c];
           t2 = volume(temp,p1,p2,p3)/6.0;
           if(t2>0){
              ans.x += (p1.x+p2.x+p3.x+temp.x)*t2;
              ans.y += (p1.y+p2.y+p3.y+temp.y)*t2;
              ans.z += (p1.z+p2.z+p3.z+temp.z)*t2;
              v += t2;
           }
        }
    }
    ans.x /= (4*v); ans.y /= (4*v); ans.z /= (4*v);
    return ans;
 }
 double function(Point fuck){
    double min=99999999;
    for(int i=0;i<num;i++){
        if(F[i].ok==true){
           Point p1=P[F[i].a] , p2=P[F[i].b] , p3=P[F[i].c];
```

```
                    double a = ( (p2.y-p1.y)*(p3.z-p1.z)-(p2.z-p1.z)*(p3.y-p1.y) );
                    double b = ( (p2.z-p1.z)*(p3.x-p1.x)-(p2.x-p1.x)*(p3.z-p1.z) );
                    double c = ( (p2.x-p1.x)*(p3.y-p1.y)-(p2.y-p1.y)*(p3.x-p1.x) );
                    double d = ( 0-(a*p1.x+b*p1.y+c*p1.z) );
                    double temp = fabs(a*fuck.x+b*fuck.y+c*fuck.z+d)/sqrt(a*a+b*b+c*c);
                    if(temp<min)min = temp;
                }
            }
            return min;
        }

};
```

5 Math

5.1 EX GCD

```
void gcd(int a, int b, int &d, int &x, int &y) {
    if(!b) {
        d = a;
        x = 1;
        y = 0;
    } else {
        gcd(b, a % b, d, y, x);
        y -= x * (a / b);
    }
}

//ax + by = gcd(a, b)
//when gcd(a, b) = 1, ax = 1 (mod b)
```

5.2 Romberg

```
const int MAX = 18;

double f(double x) {

}

double Romberg (double a, double b) {
#define MAX_N 18

    int i, j, temp2, min;
    double h, R[2][MAX_N], temp4;

    for (i = 0; i < MAX_N; i++) {
        R[0][i] = 0.0;
        R[1][i] = 0.0;
    }
    h = b - a;
    min = (int)(log(h * 10.0) / log(2.0)); //h should be at most 0.1
    R[0][0] = (f(a) + f(b)) * h * 0.50;
    i = 1;
    temp2 = 1;
    while (i < MAX_N) {
        i++;
        R[1][0] = 0.0;
        for (j = 1; j <= temp2; j++)
            R[1][0] += f(a + h * ((double)j - 0.50));
        R[1][0] = (R[0][0] + h * R[1][0]) * 0.50;
        temp4 = 4.0;
        for (j = 1; j < i; j++) {
            R[1][j] = R[1][j - 1] + (R[1][j - 1] - R[0][j - 1]) / (temp4 - 1.0);
            temp4 *= 4.0;
        }
        if ((fabs(R[1][i - 1] - R[0][i - 2]) < eps) && (i > min))
            return R[1][i - 1];
        h *= 0.50;
        temp2 *= 2;
        for (j = 0; j < i; j++)
            R[0][j] = R[1][j];
    }
    return R[1][MAX_N - 1];
}
```

5.3 Miller Rabin-Pollard rho

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#include<iostream>
#include<algorithm>
using namespace std;

//****************************************************************
// Miller_Rabin Prime Test
// for number < 2^63
//****************************************************************
const int S = 20; //test times


//cal (a*b)%c.
//  a,b,c <2^63
long long mult_mod(long long a, long long b, long long c) {
    a %= c;
    b %= c;
    long long ret = 0;
    while(b) {
        if(b & 1) {
            ret += a;
            ret %= c;
        }
        a <<= 1;
        if(a >= c)a %= c;
        b >>= 1;
    }
    return ret;
}
```

```
//cal x^n %c
long long pow_mod(long long x, long long n, long long mod) { //x^n%c
    if(n == 1)return x % mod;
    x %= mod;
    long long tmp = x;
    long long ret = 1;
    while(n) {
        if(n & 1) ret = mult_mod(ret, tmp, mod);
        tmp = mult_mod(tmp, tmp, mod);
        n >>= 1;
    }
    return ret;
}

//true for composite numbers
bool check(long long a, long long n, long long x, long long t) {
    long long ret = pow_mod(a, x, n);
    long long last = ret;
    for(int i = 1; i <= t; i++) {
        ret = mult_mod(ret, ret, n);
        if(ret == 1 && last != 1 && last != n - 1) return true;
        last = ret;
    }
    if(ret != 1) return true;
    return false;
}

bool Miller_Rabin(long long n) {
    if(n < 2)return false;
    if(n == 2)return true;
    if((n & 1) == 0) return false; // even number
    long long x = n - 1;
    long long t = 0;
    while((x & 1) == 0) {
        x >>= 1;
        t++;
    }
    for(int i = 0; i < S; i++) {
        long long a = rand() % (n - 1) + 1;
        if(check(a, n, x, t))
            return false; //composite numbers
    }
    return true;
}


//**************************************************
//pollard_rho prime factor decomposition
//**************************************************
long long factor[100]; //decomposition result

int tol; //prime factor count

long long gcd(long long a, long long b) {
    if(a == 0)return 1;
    if(a < 0) return gcd(-a, b);
    while(b) {
        long long t = a % b;
        a = b;
        b = t;
    }
    return a;
}

long long Pollard_rho(long long x, long long c) {
    long long i = 1, k = 2;
    long long x0 = rand() % x;
    long long y = x0;
    while(1) {
        i++;
        x0 = (mult_mod(x0, x0, x) + c) % x;
        long long d = gcd(y - x0, x);
        if(d != 1 && d != x) return d;
        if(y == x0) return x;
        if(i == k) {
            y = x0;
            k += k;
        }
    }
}

//decomposition n
void findfac(long long n) {
    if(n == 1) return;
    if(Miller_Rabin(n)) {
        factor[tol++] = n;
        return;
    }
    long long p = n;
    while(p >= n)p = Pollard_rho(p, rand() % (n - 1) + 1);
    findfac(p);
    findfac(n / p);
}

int main() {
    srand(time(NULL));
    long long n;
    while(scanf("%I64d", &n) != EOF) {
        tol = 0;
        findfac(n);
        for(int i = 0; i < tol; i++)printf("%I64d ", factor[i]);
        printf("\n");
        if(Miller_Rabin(n))printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

5.4 FWT XOR

```
inline int ck(int x) {
    if(x < 0) x += mod;
    if(x >= mod) x -= mod;
    return x;
}
```

```
int tmp[1 << 16];

void tf(int a[], int ta[], int n) {
    if(n == 1) {
        ta[0] = a[0];
        return;
    }
    int x = n >> 1;
    tf(a, ta, x);
    tf(a + x, ta + x, x);
    for(int i = 0; i < x; ++ i) {
        tmp[i] = ck(ta[i] - ta[x + i]);
        tmp[x + i] = ck(ta[i] + ta[x + i]);
    }
    for(int i = 0; i < n; ++ i) ta[i] = tmp[i];
}

LL inv2;

void utf(int a[], int ta[], int n) {
    if(n == 1) {
        ta[0] = a[0];
        return;
    }
    int x = n >> 1;
    for(int i = 0; i < x; ++ i) {
        tmp[i] = (a[i] + a[i + x]) * inv2 % mod;
        tmp[i + x] = (a[i + x] - a[i] + mod) * inv2 % mod;
    }
    for(int i = 0; i < n; ++ i) a[i] = tmp[i];
    utf(a, ta, x);
    utf(a + x, ta + x, x);
}
```

5.5 FNT Base 2

```
//1004535809 3
//211812353 3
//10000000025100289 22
//11000000009994241 17
//100000000135659521 3
//mod:prime g:prime root
const int mod = 998244353, g = 3;
int _g[30], _ig[30];
LL invl;

int pow_mod(LL a, int b) {
    LL c = 1;
    while(b) {
        if(b & 1) c = c * a % mod;
        b >>= 1;
        a = a * a % mod;
    }
    return c;
}
```

```
void init() {
    for(int i = 0; i < 30; ++i) {
        _g[i] = pow_mod(g, (mod - 1) / (1 << i));
        _ig[i] = pow_mod(_g[i], mod - 2);
    }
}

void FNT(int F[], int len, int f) {
    int i, j, k, cnt = 1;
    LL x, y, w = 1, wn;
    for(i = 1, j = len >> 1; i < len - 1; ++i) {
        if(i < j) swap(F[i], F[j]);
        k = len >> 1;
        while(j >= k) j ^= k, k >>= 1;
        j |= k;
    }
    for(i = 1; i < len; i <<= 1) {
        wn = f ? _g[cnt++] : _ig[cnt++];
        for(j = 0; j < len; j += i << 1, w = 1) {
            for(k = j; k < j + i; ++k, w = w * wn % mod) {
                x = F[k]; y = w * F[k + i] % mod;
                F[k] = x + y;
                F[k + i] = x - y;
                if(F[k] >= mod) F[k] -= mod;
                if(F[k + i] < 0) F[k + i] += mod;
            }
        }
    }
    if(!f) {
        for(i = 0; i < len; ++i) {
            F[i] = F[i] * invl % mod;
        }
    }
}

int a[150000], b[150000];
void conv(int ca[], int l1, int cb[], int l2, int c[], int &l) {
    l = 1; while(l < l1 + l2) l <<= 1;
    init(); invl = pow_mod(l, mod - 2);
    for(int i = 0; i < l; ++i) {
        a[i] = i < l1 ? ca[i] : 0;
        b[i] = i < l2 ? cb[i] : 0;
    }
    FNT(a, l, 1); FNT(b, l, 1);
    for(int i = 0; i < l; ++i) a[i] = (LL)a[i] * b[i] % mod;
    FNT(a, l, 0);
    for(int i = 0; i < l; ++i) c[i] = a[i];
}
```

5.6 FNT Base 3

```
const int P = 258280327, G = 5, B = 3;

const int N = 531441;

int pow_mod(ll a, int b) {
    int c = 1;
    while(b) {
        if(b & 1) c = c * a % P;
        b >>= 1;
        a = a * a % P;
    }
    return c;
}

int w[2][N], rev[N];

void init() {
    ll t = pow_mod(G, (P - 1) / N);
    w[0][0] = w[1][0] = 1;
    for(int i = 1; i < N; ++ i) {
        w[0][i] = w[0][i - 1] * t % P;
    }
    for(int i = 1; i < N; ++ i) {
        w[1][i] = w[0][N - i];
    }
    for(int i = 0; i < N; ++ i) {
        for(int j = 1; j < N; j *= B) {
            (rev[i] *= B) += i / j % B;
        }
    }
}

void ntt(int *a, int n, int o) {
    int tt = N / n, d = N / B;
    for(int i = 0; i < n; ++ i) {
        int j = rev[i] / tt;
        if(i < j) swap(a[i], a[j]);
    }
    for(int i = 1; i < n; i *= B) {
        for(int j = 0, t = N / (i * B); j < n; j += i * B) {
            for(int k = 0, l = 0; k < i; ++ k, l += t) {
                int x = a[j + k], y = a[j + k + i], z = a[j + k + i + i];
                a[j + k] = (x + (ll)y * w[o][l] + (ll)z * w[o][l + l]) % P;
                a[j + k + i] = (x + (ll)y * w[o][l + d] + (ll)z * w[o][(l + l + d + d) % N]) % P;
                a[j + k + i + i] = (x + (ll)y * w[o][l + d + d] + (ll)z * w[o][(l + d + d) * 2 - N])
                  % P;
            }
        }
    }
    if(o == 1) {
        ll inv = pow_mod(n, P - 2);
        for(int i = 0; i < n; ++ i) {
            a[i] = a[i] * inv % P;
        }
    }
}

int getlen(int n) {
    int r = 1;
    while(r < n) r *= B;
    return r;
}
```

5.7 FNT All

```
const int P = 258280327, G = 5;

struct Number_Theory_Transform {
    #define size 531441
    int N, *W, w[2][size], tmp[size];
    int pow_mod(ll a, int b) {
        int c = 1;
        while(b) {
            if(b & 1) c = c * a % P;
            b >>= 1;
            a = a * a % P;
        }
        return c;
    }
    void prepare(int n) {
        if(N == n) return;
        N = n;
        int x = pow_mod(G, (P - 1) / n);
        int y = pow_mod(x, P - 2);
        w[0][0] = w[1][0] = 1;
        for(int i = 1; i < n; ++ i) {
            w[0][i] = (ll)w[0][i - 1] * x % P;
            w[1][i] = (ll)w[1][i - 1] * y % P;
        }
    }
    void work(int *A, int n) {
        if(n == 1) return;
        int i, j, k = 0, l, x, m, u = W[N / n], w = 1;
        for(x = 2; n % x; ++ x); m = n / x;
        for(i = 0; i < x; ++ i) {
            for(j = i; j < n; j += x) {
                tmp[k ++] = A[j];
            }
        }
        for(i = 0; i < n; ++ i) A[i] = tmp[i];
        for(i = l = 0; i < x; ++ i, l += m) work(A + l, m);
        for(i = j = 0; i < n; ++ i) {
            for(l = j + x * m - m, k = x, tmp[i] = 0; k; -- k, l -= m) {
                tmp[i] = ((ll)w * tmp[i] + A[l]) % P;
            }
            w = (ll) w * u % P;
            if(++ j == m) j = 0;
        }
        for(int i = 0; i < n; ++ i) A[i] = tmp[i];
    }
    void DFT(int *A, int n) {
        prepare(n);
        W = w[0];
        work(A, n);
    }
    void IDFT(int *A, int n) {
        prepare(n);
        W = w[1];
        work(A, n);
        for(int i = 0, x = pow_mod(n, P - 2); i < n; ++ i) {
            A[i] = (ll)A[i] * x % P;
        }
    }
    #undef size
} NTT;
```

5.8 FFT Normal

```
const double PI = acos(-1.0);

struct Virt {
    double r, i;

    Virt(double r = 0.0, double i = 0.0) {
        this->r = r;
        this->i = i;
    }

    Virt operator + (const Virt &x) {
        return Virt(r + x.r, i + x.i);
    }

    Virt operator - (const Virt &x) {
        return Virt(r - x.r, i - x.i);
    }

    Virt operator * (const Virt &x) {
        return Virt(r * x.r - i * x.i, i * x.r + r * x.i);
    }
};

void Rader(Virt F[], int len) {
    int i, j, k;
    for(i = 1, j = len / 2; i < len - 1; i++) {
        if(i < j) swap(F[i], F[j]);
        k = len / 2;
        while(j >= k) {
            j -= k;
            k >>= 1;
        }
        if(j < k) j += k;
    }
}

void FFT(Virt F[], int len, int on) {
    Rader(F, len);
    for(int h = 2; h <= len; h <<= 1) {
        Virt wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for(int j = 0; j < len; j += h) {
            Virt w(1, 0);
            for(int k = j; k < j + h / 2; k++) {
                Virt u = F[k];
                Virt t = w * F[k + h / 2];
                F[k] = u + t;
                F[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0; i < len; i++)
            F[i].r /= len;
}
```

5.9 FFT

```
#include <algorithm>
#include <cmath>

using namespace std;

const int mod = 1e9 + 7;

const int max0 = 1 << 17;

struct comp
{
        double x, y;

        comp() : x(0), y(0) {}
        comp(const double &_x, const double &_y) : x(_x), y(_y) {}
};

inline comp operator+(const comp &a, const comp &b)
{
        return comp(a.x + b.x, a.y + b.y);
}

inline comp operator-(const comp &a, const comp &b)
{
        return comp(a.x - b.x, a.y - b.y);
}

inline comp operator*(const comp &a, const comp &b)
{
        return comp(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
}

inline comp conj(const comp &a) { return comp(a.x, -a.y); }

const double PI = acos(-1);

int N, L;

comp w[max0 + 5];
int bitrev[max0 + 5];

void fft(comp *a, const int &n)
{
        for (int i = 0; i < n; ++i)
                if (i < bitrev[i])
                        swap(a[i], a[bitrev[i]]);
        for (int i = 2, lyc = n >> 1; i <= n; i <<= 1, lyc >>= 1)
                for (int j = 0; j < n; j += i)
                {
                        comp *l = a + j, *r = a + j + (i >> 1), *p = w;
                        for (int k = 0; k < i>> 1; ++k)
                        {
                                comp tmp = *r * *p;
                                *r = *l - tmp, *l = *l + tmp;
                                ++l, ++r, p += lyc;
                        }
                }
}

inline void fft_prepare()
{
        for (int i = 0; i < N; ++i)
                bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (L - 1));
        for (int i = 0; i < N; ++i)
                w[i] = comp(cos(2 * PI * i / N), sin(2 * PI * i / N));
}

inline vector<int> conv(const vector<int> &x, const vector<int> &y)
{
        static comp a[max0 + 5], b[max0 + 5];
        static comp dfta[max0 + 5], dftb[max0 + 5], dftc[max0 + 5], dftd[max0 + 5];
        L = 0;
        while ((1 << L) < x.size() + y.size() - 1)
                ++L;
        N = 1 << L;
        fft_prepare();
        for (int i = 0; i < N; ++i)
                a[i] = b[i] = comp(0, 0);
        for (int i = 0; i < x.size(); ++i)
                a[i] = comp(x[i] & 32767, x[i] >> 15);
        for (int i = 0; i < y.size(); ++i)
                b[i] = comp(y[i] & 32767, y[i] >> 15);
        fft(a, N), fft(b, N);
        for (int i = 0; i < N; ++i)
        {
                int j = (N - i) & (N - 1);
                static comp da, db, dc, dd;
                da = (a[i] + conj(a[j])) * comp(0.5, 0);
                db = (a[i] - conj(a[j])) * comp(0, -0.5);
                dc = (b[i] + conj(b[j])) * comp(0.5, 0);
                dd = (b[i] - conj(b[j])) * comp(0, -0.5);
                dfta[j] = da * dc;
                dftb[j] = da * dd;
                dftc[j] = db * dc;
                dftd[j] = db * dd;
        }
        for (int i = 0; i < N; ++i)
                a[i] = dfta[i] + dftb[i] * comp(0, 1);
        for (int i = 0; i < N; ++i)
                b[i] = dftc[i] + dftd[i] * comp(0, 1);
        fft(a, N), fft(b, N);
        vector<int> z(x.size() + y.size() - 1);
        for (int i = 0; i < x.size() + y.size() - 1; ++i)
        {
                int da = (long long)(a[i].x / N + 0.5) % mod;
                int db = (long long)(a[i].y / N + 0.5) % mod;
                int dc = (long long)(b[i].x / N + 0.5) % mod;
                int dd = (long long)(b[i].y / N + 0.5) % mod;
```

```
              z[i] = (da + ((long long)(db + dc) << 15) + ((long long)dd << 30)) % mod;
       }
       return z;
}
```

6 Others

6.1 Big Number

```
#include<iostream>
#include<string>
#include<iomanip>
#include<algorithm>
using namespace std;

#define MAXN 9999
#define DLEN 4

class BigNum{
private:
   int a[300];//DLEN digs for a position
   int len;
public:
   BigNum(){len = 1;memset(a,0,sizeof(a));}
   BigNum(const int b);
   BigNum(const BigNum & T);

   bool    Bigger(const BigNum &) const;
   BigNum & operator=(const BigNum &);
   BigNum & Add(const BigNum &);
   BigNum & Sub(const BigNum &);
   BigNum operator+(const BigNum &) const;
   BigNum operator-(const BigNum &) const;
   BigNum operator*(const BigNum &) const;
   BigNum operator/(const int &) const;
   void Print();
};
BigNum::BigNum(const int b)
{
   int c,d = b;

   len = 0;
   memset(a,0,sizeof(a));
   while(d > MAXN){
      c = d - d / (MAXN + 1) * (MAXN + 1);
      d = d / (MAXN + 1);
      a[len++] = c;
   }
   a[len++] = d;
}
BigNum::BigNum(const BigNum & T) : len(T.len)
{
   int i;
   memset(a,0,sizeof(a));
   for(i = 0 ; i < len ; i++)
      a[i] = T.a[i];
}
bool BigNum::Bigger(const BigNum & T) const
{
   int ln;
   if(len > T.len) return true;
   else if(len == T.len){
      ln = len - 1;
      while(a[ln] == T.a[ln] && ln >= 0) ln--;
      if(ln >= 0 && a[ln] > T.a[ln]) return true;
      else return false;
   }
   else return false;
}
BigNum & BigNum::operator=(const BigNum & n)
{
   len = n.len;
   memset(a,0,sizeof(a));
   for(int i = 0 ; i < len ; i++)
      a[i] = n.a[i];
   return *this;
}
BigNum & BigNum::Add(const BigNum & T)
{
   int i,big;

   big = T.len > len ? T.len : len;
   for(i = 0 ; i < big ; i++)
   {
      a[i] = a[i] + T.a[i];
      if(a[i] > MAXN)
      {
         a[i + 1]++;
         a[i] = a[i] - MAXN - 1;
      }
   }
   if(a[big] != 0) len = big + 1;
   else len = big;

   return *this;
}
BigNum & BigNum::Sub(const BigNum & T)
{
   int i,j,big;

   big = T.len > len ? T.len : len;
   for(i = 0 ; i < big ; i++){
      if(a[i] < T.a[i]){
         j = i + 1;
         while(a[j] == 0) j++;
         a[j--]--;
         while(j > i) a[j--] += MAXN;
         a[i] = a[i] + MAXN + 1 - T.a[i];
      }
      else a[i] -= T.a[i];
   }
   len = big;
   while(a[len - 1] == 0 && len > 1) len--;
   return *this;
}
```

```
BigNum BigNum::operator+(const BigNum & n) const
{
   BigNum a = *this;

   a.Add(n);
   return a;
}
BigNum BigNum::operator-(const BigNum & T) const
{
   BigNum b = *this;

   b.Sub(T);
   return b;
}
BigNum BigNum::operator*(const BigNum & T) const
{
   BigNum ret;
   int i,j,up;
   int temp,temp1;

   for(i = 0 ; i < len ; i++){
      up = 0;
      for(j = 0 ; j < T.len ; j++){
         temp = a[i] * T.a[j] + ret.a[i + j] + up;
         if(temp > MAXN){
            temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
            up = temp / (MAXN + 1);
            ret.a[i + j] = temp1;
         }
         else {
            up = 0;
            ret.a[i + j] = temp;
         }
      }
      if(up != 0)
         ret.a[i + j] = up;
   }
   ret.len = i + j;
   while(ret.a[ret.len - 1] == 0 && ret.len > 1) ret.len--;
   return ret;
}
BigNum BigNum::operator/(const int & b) const
{
   BigNum ret;
   int i,down = 0;

   for(i = len - 1 ; i >= 0 ; i--){
      ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
      down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
   }
   ret.len = len;
   while(ret.a[ret.len - 1] == 0) ret.len--;
   return ret;
}
void BigNum::Print()
{
   int i;

   cout << a[len - 1];
   for(i = len - 2 ; i >= 0 ; i--){
      cout.width(DLEN);
      cout.fill('0');
      cout << a[i];
   }
   cout << endl;
}
```

6.2 Long Long Mul

```
long long Mul(long long x, long long y) {
    return (x * y - (long long)(x / (long double)P * y + 1e-3) * P + P) % P;
}
```

6.3 Fast IO

```
inline void R(int &x) {
  char c; bool sign = false;
  for (c = getchar(); c<'0' || c>'9'; c = getchar()) if (c=='-') sign = true;
  for (x = 0; c>='0' && c<='9'; c = getchar()) x = x*10+c-'0';
  sign && (x=-x);
}
```