Standrad Code Library

TheWaySoFar @ Beihang University

May 13, 2016

目录

1 Data Structure

1.1 Treap

```
template <class T> struct Treap {
    int nodecnt , prior[M];
    int cnt[M] , size[M] , c[M][2];
    T key[M] , GCD[M];
    void clear() {
        nodecnt = 1;
        prior[0] = -1 << 30;
        c[0][0] = c[0][1] = 0;
        key[0] = GCD[0] = cnt[0] = size[0] = 0;
    }
    Treap () {
        clear();
    }
    inline void pushup(int p) {
        size[p] = size[c[p][0]] + size[c[p][1]] + cnt[p];
        GCD[p] = __gcd(__gcd(GCD[c[p][0]] , GCD[c[p][1]]) , key[p]);
    }
    inline void rotate (int& x , int t) {
        int y = c[x][t];
        c[x][t] = c[y][!t] , c[y][!t] = x;
        pushup(x) , pushup(y) , x = y;
    }
    inline void newnode(int& p , T w) {
        p = nodecnt ++;
        key[p] = GCD[p] = w , cnt[p] = size[p] = 1;
        prior[p] = rand() << 15 | rand(), c[p][0] = c[p][1] = 0;
    }
    void insert(int& p , T w) {
        if (!p) {
            newnode(p , w);
            return;
        }
        if (key[p] == w)
            ++ cnt[p];
        else {
            int t = key[p] < w;
            insert(c[p][t] , w);
            if (prior[c[p][t]] > prior[p])
                rotate(p , t);
        }
        pushup(p);
    }
    void erase(int& p , T w) {
        if (!p) return;
        if (key[p] == w) {
            if (cnt[p] == 1) {
                if (!c[p][0] && !c[p][1])
                    p = 0;
                else {
                    rotate(p , prior[c[p][0]] < prior[c[p][1]]);
                    erase(p , w);
                }
            } else
                -- cnt[p];
        } else
            erase(c[p][key[p] < w] , w);
        pushup(p);
    }
    T getKth(int p , int K) {
        if (K <= size[c[p][0]])
            return getKth(c[p][0] , K);
        K -= size[c[p][0]] + cnt[p];
        if (K <= 0) return key[p];
        return getKth(c[p][1] , K);
    }
    T lower_bound(int p , T w) {
        if (!p) return 1 << 30;
        if (key[p] >= w)
            return min(lower_bound(c[p][0] , w) , key[p]);
        else
            return lower_bound(c[p][1] , w);
    }
    T range(int p , int l , int r) {
        if (!p || l > r) return 0;
        if (l <= key[p] && key[p] <= r) {
            int ans = key[p];
            if (l == -1 << 30) {
                ans = __gcd(ans , GCD[c[p][0]]);
                ans = __gcd(ans , range(c[p][1] , l , r));
            } else if (r == 1 << 30) {
                ans = __gcd(ans , GCD[c[p][1]]);
                ans = __gcd(ans , range(c[p][0] , l , r));
            } else {
                ans = __gcd(ans , range(c[p][0] , l , 1 << 30));
                ans = __gcd(ans , range(c[p][1] , -1 << 30 , r));
            }
            return ans;
        }
        if (r < key[p])
            return range(c[p][0] , l , r);
        else
            return range(c[p][1] , l , r);
    }
    void merge(int& p , int& q) {
        if (!p) return;
        merge(c[p][0] , q);
        merge(c[p][1] , q);
        insert(q , key[p]);
        erase(p , key[p]);
    }
};
```

1.2 序列维护 Treap

```
struct Treap {
    int nodecnt;
    int L[N] , R[N] , cnt[N];
    int key[N];
    int Min[N] , add[N] , rev[N];
    bool hey(int A , int B) {
        return rand() % (cnt[A] + cnt[B]) < cnt[A];
    }
    int newnode(int val) {
        ++ nodecnt , L[nodecnt] = R[nodecnt] = 0;
        cnt[nodecnt] = 1 , Min[nodecnt] = key[nodecnt] = val;
        rev[nodecnt] = add[nodecnt] = 0;
        return nodecnt;
    }
    void pushup(int x) {
        cnt[x] = 1 , Min[x] = key[x];
        if (L[x]) cnt[x] += cnt[L[x]] , Min[x] = min(Min[x] , Min[L[x]]);
        if (R[x]) cnt[x] += cnt[R[x]] , Min[x] = min(Min[x] , Min[R[x]]);
    }
    void pushdown(int x) {
        if (rev[x]) {
            if (L[x]) rev[L[x]] ^= 1 , swap(L[L[x]] , R[L[x]]);
            if (R[x]) rev[R[x]] ^= 1 , swap(L[R[x]] , R[R[x]]);
            rev[x] = 0;
        }
        if (add[x]) {
            if (L[x]) add[L[x]] += add[x] , Min[L[x]] += add[x] , key[L[x]] += add[x];
            if (R[x]) add[R[x]] += add[x] , Min[R[x]] += add[x] , key[R[x]] += add[x];;
            add[x] = 0;
        }
    }
    void merge(int& p , int x , int y) {
        if (!x || !y)
            p = x | y;
        else if ( hey(x , y) ) // key[x] < key[y]
            pushdown(x) , merge(R[x] , R[x] , y) , pushup(p = x);
        else
            pushdown(y) , merge(L[y] , x , L[y]) , pushup(p = y);
    }
    void split(int p , int& x , int& y , int size) {
        if (!size) {
            x = 0 , y = p;
            return;
        }   pushdown(p);
        if (cnt[L[p]] >= size)
            y = p , split(L[p] , x , L[y] , size) , pushup(y);
        else
            x = p , split(R[p] , R[x] , y , size - cnt[L[p]] - 1) , pushup(x);
    }
};
```

1.3 Splay

```
const int N = 500005;
struct Node {
    Node *ch[2] , *p;
    int size;
    int val , sum , lm , rm , sm;
    int same;
    bool rev;
    Node () {
        size = val = sum = 0;
        lm = rm = sm = -1e9;
        same = 1 << 30 , rev = 0;
    }
    bool d() {
        return this == p->ch[1];
    }
    void setc(Node *c , int d) {
        ch[d] = c;
        c -> p = this;
    }
    void setsame(int x) {
        same = val = x;
        sum = val * size;
        lm = rm = sm = max(val , sum);
    }
    void reverse() {
        rev ^= 1;
        swap(ch[0] , ch[1]);
        swap(lm , rm);
    }
    void pushdown();
    void pushup() {
        size = ch[0] -> size + ch[1] -> size + 1;
        sum = ch[0] -> sum + val + ch[1] -> sum;
        lm = max(ch[0] -> lm , max(ch[0] -> sum + val , ch[0] -> sum + val + ch[1] -> lm));
        rm = max(ch[1] -> rm , max(ch[1] -> sum + val , ch[1] -> sum + val + ch[0] -> rm));
        sm = max(val , max(ch[0] -> sm , ch[1] -> sm));
        sm = max(sm , max(ch[0]->rm , ch[1]->lm) + val);
        sm = max(sm , ch[0]->rm + val + ch[1]->lm);
    }
}Tnull , *null = &Tnull;
Node mem[N] , *C = mem;
Node* rub[N];
int rubsize;
void Node::pushdown() {
    if (rev) {
        if (ch[0] != null)
            ch[0]->reverse();
        if (ch[1] != null)
            ch[1]->reverse();
        rev ^= 1;
    }
    if (same != 1 << 30) {
        if (ch[0] != null)
            ch[0]->setsame(same);
        if (ch[1] != null)
            ch[1]->setsame(same);
        same = 1 << 30;
    }
}
Node* newnode(int v) {
    Node *p = rubsize ? rub[-- rubsize] : C ++;
    p->ch[0] = p->ch[1] = p->p = null;
    p->size = 1;
    p->val = p->sum = p->lm = p->rm = p->sm = v;
    p->same = 1 << 30 , p->rev = 0;
    return p;
}
void rotate(Node *t) {
    Node *p = t->p;
    int d = t->d();
    p->p->setc(t , p->d());
    p->setc(t->ch[!d] , d);
    t->setc(p , !d);
    p->pushup();
}
void update(Node *t) {
    static Node* Stack[N];
    int top = 0;
    while (t != null) {
        Stack[top ++] = t;
        t = t->p;
    }
    for (int i = top - 1 ; i >= 0 ; -- i)
        Stack[i]->pushdown();
}
void splay(Node *t , Node *f = null) {
    update(t);
    while (t->p != f) {
        if (t->p->p == f)
            rotate(t);
        else {
            if (t->d() == t->p->d())
                rotate(t->p) , rotate(t);
            else
                rotate(t) , rotate(t);
        }
    }
    t->pushup();
}
Node* select(Node *t , int k) {
    while (1) {
        t->pushdown();
        int c = 1 + t->ch[0]->size;
        if (k == c)
            return t;
        if (k > c)
            k -= c , t = t->ch[1];
        else
            t = t->ch[0];
    }
}
void split(Node *p , Node *&x , Node *&y , int K) {
    if (K == 0) {
        x = null , y = p;
    } if (K == p->size) {
        x = p , y = null;
    } else {
        y = select(p , K + 1);
        splay(y);
        x = y->ch[0];
        y->ch[0] = x->p = null;
        y->pushup();
    }
}
void merge(Node *&p , Node *x , Node *y) {
    if (x == null)
        p = y;
    else if (y == null)
        p = x;
    else {
        x->pushdown();
        p = select(x , x->size);
        splay(p);
        p->setc(y , 1);
        p->pushup();
    }
}
int n , m , a[N];
Node* build(int l , int r) {
    if (l > r)
        return null;
    int mid = l + r >> 1;
    Node *t = newnode(a[mid]);
    t->setc(build(l , mid - 1) , 0);
    t->setc(build(mid + 1 , r) , 1);
    t->pushup();
    return t;
}
void del(Node *p) {
    if (p == null)
        return;
    rub[rubsize ++] = p;
    del(p->ch[0]);
    del(p->ch[1]);
}
int main() {
    scanf("%d%d" , &n , &m);
    for (int i = 1 ; i <= n ; ++ i)
        scanf("%d" , &a[i]);
    Node *root = build(1 , n);
    return 0;
}
```

1.4 树状数组

```
int getKth(int k) {
    int x = 0 , i;
    for (i = 16 ; i >= 0 ; -- i)
        if (x + (1 << i) <= D && c[x + (1 << i)] < k) {
            x += 1 << i;
            k -= c[x];
        }
    return x + 1;
}
struct CHU_2_BIT {
    int n;
    LL B[N] , C[N];
    void init(int size) {
        n = size;
        memset(B , 0 , n + 1 << 3);
        memset(C , 0 , n + 1 << 3);
    }
    CHU_2_BIT() {}
    CHU_2_BIT(int size) {
        init(size);
    }
    inline LL _sum(LL* c , int x) {
        LL res = 0;
        for ( ; x > 0 ; x -= x & -x)
            res += c[x];
        return res;
    }
    void add(int l , int r , LL w) {
        for (int i = l ; i <= n ; i += i & -i)
            B[i] += w , C[i] += w * l;
        ++ r;
        for (int i = r ; i <= n ; i += i & -i)
            B[i] -= w , C[i] -= w * r;
    }
    LL sum(int l , int r) {
        LL res = 0; -- l;
        res += (r + 1) * _sum(B , r) - _sum(C , r);
        res -= (l + 1) * _sum(B , l) - _sum(C , l);
        return res;
    }
}T;
```

1.5 Link-Cut-Tree

```
const int N = 100005;
const int INF = -1 << 30;
struct Node {
    Node *ch[2] , *p , *fa;
    int size;
    int val , add;
    int same;
    pair<int , int> mx[2];
    bool rev;
    Node () {
        mx[0].first = mx[1].first = INF;
        size = val = add = 0;
        same = 1 << 30 , rev = 0;
    }
    bool d() {
        return this == p->ch[1];
    }
    void setc(Node *c , int d) {
        ch[d] = c;
        c->p = this;
    }
    void addwei(int x) {
        if (same != 1 << 30)
            same += x;
        add += x , val += x;
        mx[0].first += x , mx[1].first += x;
    }
    void setsame(int x) {
        same = val = x;
        add = 0;
        mx[0] = make_pair(x , size);
        mx[1] = make_pair(INF , 0);
    }
    void reverse() {
        rev ^= 1;
        swap(ch[0] , ch[1]);
    }
    void pushdown();
    void pushup() {
        size = ch[0]->size + ch[1]->size + 1;
        int j = 0 , k = 0 , l = 0;
        for (int i = 0 ; i < 2 ; ++ i) {
            int x = max(ch[0]->mx[j].first , ch[1]->mx[l].first) , y = 0;
            if (k < 1) x = max(x , val);
            if (x == ch[0]->mx[j].first) y += ch[0]->mx[j ++].second;
            if (k < 1 && x == val) ++ y , ++ k;
            if (x == ch[1]->mx[l].first) y += ch[1]->mx[l ++].second;
            mx[i] = make_pair(x , y);
        }
    }
}Tnull , *null = &Tnull;
Node mem[N] , *C = mem;
void Node::pushdown() {
    if (rev) {
        if (ch[0] != null)
            ch[0]->reverse();
        if (ch[1] != null)
            ch[1]->reverse();
        rev ^= 1;
    }
    if (add) {
        if (ch[0] != null)
            ch[0]->addwei(add);
        if (ch[1] != null)
            ch[1]->addwei(add);
        add = 0;
    }
    if (same != 1 << 30) {
        if (ch[0] != null)
            ch[0]->setsame(same);
        if (ch[1] != null)
            ch[1]->setsame(same);
        same = 1 << 30;
    }
}

Node* newnode(int v) {
    Node *p = C ++;
    p->ch[0] = p->ch[1] = p->p = p->fa = null;
    p->size = 1;
    p->val = v;
    p->same = 1 << 30 , p->rev = p->add = 0;
    p->mx[0] = make_pair(v , 1);
    p->mx[1] = make_pair(INF , 0);
    return p;
}
void rotate(Node *t) {
    Node *p = t->p;
    int d = t->d();
    p->p->setc(t , p->d());
    p->setc(t->ch[!d] , d);
    t->setc(p , !d);
    p->pushup();
    t->fa = p->fa;
}
void update(Node *t) {
    static Node* Stack[N];
    int top = 0;
    while (t != null) {
        Stack[top ++] = t;
        t = t->p;
    }
    for (int i = top - 1 ; i >= 0 ; -- i)
        Stack[i]->pushdown();
}
```

```
void splay(Node *t , Node *f = null) {
    update(t);
    while (t->p != f) {
        if (t->p->p == f)
            rotate(t);
        else {
            if (t->d() == t->p->d())
                rotate(t->p) , rotate(t);
            else
                rotate(t) , rotate(t);
        }
    }
    t->pushup();
}
Node* expose(Node *x) {
    Node *y = null;
    while (x != null) {
        splay(x);
        Node *z = x->ch[1];
        z->p = null;
        z->fa = x;
        x->setc(y , 1);
        y->fa = null;
        x->pushup();
        y = x , x = x->fa;
    }
    return y;
}
void setroot(Node *x) {
    expose(x);
    splay(x);
    x->reverse();
}
void link(Node *x , Node *y) {
    setroot(x);
    x->fa = y;
    expose(x);
}
void cut(Node *x , Node *y) {
    setroot(x);
    expose(y);
    splay(x);
    x->setc(null , 1);
    x->pushup();
    y->fa = y->p = null;
}
int n , m , pre[N] , mcnt , ca;
struct edge {
    int x , next;
}e[N << 1];
bool vis[N];
Node *V[N];
void work() {
    printf("Case #%d:\n" , ++ ca);
    scanf("%d%d" , &n , &m);
    C = mem;
    for (int i = 1 ; i <= n ; ++ i) {
        int x;
        scanf("%d" , &x);
        V[i] = newnode(x);
    }

    queue<int> Q;
    memset(vis , 0 , sizeof(vis));
    Q.push(1) , vis[1] = 1;
    while (!Q.empty()) {
        int x = Q.front() ; Q.pop();
        for (int i = pre[x] ; ~i ; i = e[i].next) {
            int y = e[i].x;
            if (!vis[y]) {
                V[y]->fa = V[x];
                vis[y] = 1;
                Q.push(y);
            }
        }
    }
}
```

1.6 树边分治

```
/*QTREE4,里面自带rebuild过程重建树为适合边分治的结构。还自带手写的二叉堆。*/
const int N = 200005;
int n , m , Q , pre[N] , mcnt;
struct edge {
    int x , w , next;
}e[N << 2];
int tmp[N];
bool color[N];
void build(vector<int>& child , int x , int l , int r) {
    if (l == r) {
        int y = e[child[l]].x;
        e[mcnt] = (edge) {y , e[child[l]].w , tmp[x]} , tmp[x] = mcnt ++;
        e[mcnt] = (edge) {x , e[child[l]].w , tmp[y]} , tmp[y] = mcnt ++;
    } else {
        int mid = l + r >> 1;
        int rt = ++ n ; color[rt] = 1;
        e[mcnt] = (edge) {x , 0 , tmp[n]} , tmp[n] = mcnt ++;
        e[mcnt] = (edge) {n , 0 , tmp[x]} , tmp[x] = mcnt ++;
        if (l <= mid) build(child , rt , l , mid);
        if (mid < r) build(child , rt , mid + 1 , r);
    }
}
void rebuild(int x , int fa) {
    vector<int> child;
    for (int i = pre[x] ; ~i ; i = e[i].next) {
        int y = e[i].x;
        if (y != fa) {
            rebuild(y , x);
            child.push_back(i);
        }
    }
    if (!child.empty())
        build(child , x , 0 , child.size() - 1);
}
int s[N] , size;
bool f[N];
pair<int , int> Find(int x , int fa , int cnt) {
    s[x] = 1;
    pair<int , int> res = make_pair(1 << 30 , -1);
    for (int i = pre[x] ; ~i ; i = e[i].next) {
        int y = e[i].x;
        if (!f[i >> 1] && y != fa) {
            res = min(res , Find(y , x , cnt));
            s[x] += s[y];
            res = min(res , make_pair(max(s[y] , cnt - s[y]) , i));
        }
    }
    return res;
}
pair<int , int> res[N];
int flag[N];
vector< pair<int , int> > b[N];
priority_queue< pair<int , int> > PQ[N << 1];
void Getdis(int x , int fa , int d , int id) {
    b[x].push_back(make_pair(id , d));
    if (!color[x])
        PQ[id].push(make_pair(d , x));
    ++ size;
    for (int i = pre[x] ; ~i ; i = e[i].next) {
        if (!f[i >> 1]) {
            int y = e[i].x;
            if (y != fa)
                Getdis(y , x , d + e[i].w , id);
        }
    }
}
void divide(int x , int cnt) {
    if (cnt <= 1) return;
    int k = Find(x , 0 , cnt).second;
    f[k >> 1] = 1;

    size = 0 , PQ[k].push(make_pair(-1 << 29 , -1));
    Getdis(e[k].x , 0 , 0 , k);
    s[e[k].x] = size;

    size = 0 , PQ[k ^ 1].push(make_pair(-1 << 29 , -1));
    Getdis(e[k ^ 1].x , 0 , 0 , k ^ 1);
    s[e[k ^ 1].x] = size;

    res[k >> 1] = make_pair(PQ[k].top().first + PQ[k ^ 1].
     top().first + e[k].w , k >> 1);

    divide(e[k].x , s[e[k].x]);
    divide(e[k ^ 1].x , s[e[k ^ 1].x]);
}
void down(int x) {
    int i = x , j = i << 1 | 1;
    pair<int , int> t = res[i];
    if (j + 1 < m && res[j + 1] > res[j])
        ++ j;
    while (j < m && t < res[j]) {
        flag[res[j].second] = i , res[i] = res[j];
        i = j , j = i << 1 | 1;
        if (j + 1 < m && res[j + 1] > res[j])
            ++ j;
    }
    res[i] = t , flag[t.second] = i;
}
void up(int x) {
    int i = x , j = (i + 1 >> 1) – 1;
    pair<int , int> t = res[i];
    while (j >= 0 && res[j] < t) {
        flag[res[j].second] = i , res[i] = res[j];
        i = j , j = (i + 1 >> 1) – 1;
    }
    res[i] = t , flag[t.second] = i;
}

void work() {
    int i , j , x , y , z;
    char str[10];
    scanf("%d",&n);
    memset(pre , –1 , sizeof(pre)) , mcnt = n * 6;
    for (i = 1 ; i < n ; ++ i) {
        scanf("%d%d%d",&x,&y,&z);
        e[mcnt] = (edge) {y , z , pre[x]} , pre[x] = mcnt
        ++;
        e[mcnt] = (edge) {x , z , pre[y]} , pre[y] = mcnt
        ++;
    }
    int cnt = n;
    mcnt = 0;
    memset(tmp , –1 , sizeof(tmp));
    rebuild(1 , 0);
    memcpy(pre , tmp , sizeof(tmp));
    divide(1 , n);
    m = mcnt >> 1;

    make_heap(res , res + m);
    for (i = 0 ; i < m ; ++ i)
        flag[res[i].second] = i;

    scanf("%d",&Q);
    while (Q ––) {
        scanf("%s" , str);
        if (*str == 'A') {
            if (!cnt)
                puts("They have disappeared.");
            else if (cnt == 1)
                puts("0");
            else
                printf("%d\n" , max(0 , res[0].first));
        } else {
            scanf("%d",&x);
            if (color[x])
                ++ cnt;
            else
                –– cnt;
            color[x] ^= 1;
            for (i = 0 ; i < b[x].size() ; ++ i) {
                j = b[x][i].first , z = b[x][i].second;
                if (!color[x])
                    PQ[j].push(make_pair(z , x));
                while (~PQ[j].top().second && color[PQ[j].
                top().second])
                    PQ[j].pop();
                res[flag[j >> 1]].first = PQ[j].top().
                first + PQ[j ^ 1].top().first + e[j].w;
                down(flag[j >> 1]) , up(flag[j >> 1]);
            }
        }
    }
}

int main() {
    work();
    return 0;
}
```

1.7 K-d Tree

```
struct Point3D {
    int x , y , z;
    bool operator < (const Point3D& R) const {
        if (x != R.x)
            return x < R.x;
        if (y != R.y)
            return y < R.y;
        return z < R.z;
    }
}P[N];
typedef pair<int , int> Point;
typedef pair<int , int> Value;
Point a[N];
bool cmpX(const Point& A , const Point& B) {
    return A < B;
}
bool cmpY(const Point& A , const Point& B) {
    return make_pair(A.second , A.first) < make_pair(B.
    second , B.first);
}
inline void add(Value& A , Value B) {
    if (B.first > A.first)
        A = B;
    else if (B.first == A.first)
        A.second += B.second;
}
int cnt , root;
struct Node {
    Point u , low , high;
    Value val , mx;
    int c[2];
    Node () {}
    Node(int K , Point p) {
        u = low = high = p;
        c[0] = c[1] = 0;
        val = mx = make_pair(–1 << 30 , 0);
    }
    void merge(const Node& R) {
        low.first = min(low.first , R.low.first);
        low.second = min(low.second , R.low.second);
        high.first = max(high.first , R.high.first);
        high.second = max(high.second , R.high.second);
        add(mx , R.mx);
    }
}t[N];
inline void update(int p) {
    t[p].mx = t[p].val;
    t[p].high = t[p].low = t[p].u;
    if (t[p].c[0])
        t[p].merge(t[t[p].c[0]]);
    if (t[p].c[1])
        t[p].merge(t[t[p].c[1]]);
}
void build(int& p , int k , int l , int r) {
    p = cnt ++;
    int mid = l + r >> 1;
    nth_element(a + l , a + mid , a + r , k ? cmpY : cmpX)
    ;
    t[p] = Node(k , a[mid]);
    if (l < mid)
        build(t[p].c[0] , k ^ 1 , l , mid);
    if (mid + 1 < r)
        build(t[p].c[1] , k ^ 1 , mid + 1 , r);
    update(p);
}
LL cnt1 , cnt2;
void query(int p , int k , const Point& P , Value& res) {
    if (!p) return;
    ++ cnt1;
    if (t[p].high.first <= P.first && t[p].high.second <=
    P.second)
        return add(res , t[p].mx);
    if (t[p].u.first <= P.first && t[p].u.second <= P.
    second)
        add(res , t[p].val);
    if (k) {
        query(t[p].c[0] , k ^ 1 , P , res);
        if (t[p].u.second <= P.second)
            query(t[p].c[1] , k ^ 1 , P , res);
    } else {
        query(t[p].c[0] , k ^ 1 , P , res);
        if (t[p].u.first <= P.first)
            query(t[p].c[1] , k ^ 1 , P , res);
    }
}
void modify(int p , int k , const Point& P , const Value& val) {
    if (!p) return;
    ++ cnt2;
    if (t[p].u == P) {
        t[p].val = val;
    } else {
        if (k) {
            if (P.second <= t[p].u.second)
                modify(t[p].c[0] , k ^ 1 , P , val);
            if (P.second >= t[p].u.second)
                modify(t[p].c[1] , k ^ 1 , P , val);
        } else {
            if (P.first <= t[p].u.first)
                modify(t[p].c[0] , k ^ 1 , P , val);
            if (P.first >= t[p].u.first)
                modify(t[p].c[1] , k ^ 1 , P , val);
        }
    }
    update(p);
}

void work() {
    int i , n;
    scanf("%d",&n);
    for (i = 0 ; i < n ; ++ i) {
        scanf("%d%d%d",&P[i].x , &P[i].y , &P[i].z);
        a[i] = make_pair(P[i].y , P[i].z);
    }
    sort(P , P + n) , sort(a , a + n);
    int m = unique(a , a + n) - a;
    cnt = 1 , root = 0;
    build(root , 0 , 0 , m);

    Value ans(-1 << 30 , 0);
    for (i = 0 ; i < n ; ++ i) {
        Point p = make_pair(P[i].y , P[i].z);
        Value w = make_pair(0 , 1);
        query(root , 0 , p , w);
        ++ w.first , add(ans , w);
        modify(root , 0 , p , w);
    }
    printf("%d %d\n" , ans.first , ans.second & ((1 << 30) - 1));
}
void getNearest(int p , int k) {
  if (!p) return;
  if (t[p].vis) {
        LL dis = dist(P , t[p].u);
        if (dis < res || (dis == res && t[p].o < t[ret].o))
            res = dis , ret = p;
  }
    if (k) {
        if (cmpY(P , t[p].u)) {
            getNearest(t[p].c[0] , k ^ 1);
            if (sqr(P.second - t[p].u.second) <= res)
                getNearest(t[p].c[1] , k ^ 1);
        } else {
            getNearest(t[p].c[1] , k ^ 1);
            if (sqr(P.second - t[p].u.second) <= res)
                getNearest(t[p].c[0] , k ^ 1);
        }
    } else {
        if (cmpX(P , t[p].u)) {
            getNearest(t[p].c[0] , k ^ 1);
            if (sqr(P.first - t[p].u.first) <= res)
                getNearest(t[p].c[1] , k ^ 1);
        } else {
            getNearest(t[p].c[1] , k ^ 1);
            if (sqr(P.first - t[p].u.first) <= res)
                getNearest(t[p].c[0] , k ^ 1);
        }
    }
}
```

1.8 2D-Segment-Tree

```
int n , m , Q;
int mx[N << 1][N << 1] , mn[N << 1][N << 1];
pair<int , int> A , B;
int val , LL , RR , Max , Min;
inline int id(int l , int r) {return l + r | l != r;}
#define MID int mid = (l + r) >> 1
#define Left l , mid
#define Right mid + 1 , r
void QUERY(int p , int l , int r) {
    int q = id(l , r);
    if (B.first <= l && r <= B.second) {
        Max = max(Max , mx[p][q]);
        Min = min(Min , mn[p][q]);
        return;
    } MID;
    if (B.first <= mid)
        QUERY(p , Left);
    if (B.second > mid)
        QUERY(p , Right);
}
void query(int l , int r) {
    int p = id(l , r);
    if (A.first <= l && r <= A.second) {
        QUERY(p , 1 , m);
        return;
    } MID;
    if (A.first <= mid)
        query(Left);
    if (A.second > mid)
        query(Right);
}
void UPDATE(int p , int l , int r) {
    int q = id(l , r);
    if (l == r) {
        if (p & 1) {
            mx[p][q] = max(mx[LL][q] , mx[RR][q]);
            mn[p][q] = min(mn[LL][q] , mn[RR][q]);
        } else {
            mx[p][q] = mn[p][q] = val;
        }
        return;
    } MID;
    if (A.second <= mid)
        UPDATE(p , Left);
    else
        UPDATE(p , Right);
    mx[p][q] = max(mx[p][id(Left)] , mx[p][id(Right)]);
    mn[p][q] = min(mn[p][id(Left)] , mn[p][id(Right)]);
}
void update(int l , int r) {
    int p = id(l , r);
    if (l == r) {
        UPDATE(p , 1 , m);
        return;
    } MID;
    if (A.first <= mid)
        update(Left);
    else
        update(Right);
    LL = id(Left) , RR = id(Right);
    UPDATE(p , 1 , m);
}
void work()
{
    int i , j;
    char str[5];
    scanf("%d%d",&n,&m);
    memset(mx , 0x80 , sizeof(mx));
    memset(mn , 0x7F , sizeof(mn));
    for (i = 1 ; i <= n ; ++ i)
        for (j = 1 ; j <= m ; ++ j) {
            scanf("%d",&val);
            A = make_pair(i , j);
            update(1 , n);
        }
    scanf("%d",&Q);
    while (Q --) {
        scanf("%s" , str);
        if (*str == 'c') {
            scanf("%d%d%d",&A.first , &A.second , &val);
            update(1 , n);
        } else {
            scanf("%d%d%d%d",&A.first , &B.first , &A.second , &B.second);
            Max = -1 << 30 , Min = 1 << 30;
            query(1 , n);
            printf("%d %d\n" , Max , Min);
        }
    }
}
```

2 Graph Theory

2.1 Tarjan

```
int ncnt , scnt , bel[N] , low[N] , dfn[N];
int f[N];
stack<int> S;
void dfs(int x) {
    int i , y;
    low[x] = dfn[x] = ++ ncnt;
    f[x] = 1 , S.push(x);
    for (i = pre[x] ; ~i ; i = e[i].next) {
        y = e[i].x;
        if (!dfn[y]) {
            dfs(y);
            low[x] = min(low[x] , low[y]);
        } else if (f[y])
            low[x] = min(low[x] , dfn[y]);
    }
    if (low[x] == dfn[x]) {
        val[scnt] = 0;
        do {
            i = S.top() , S.pop() , f[i] = 0;
            bel[i] = scnt , val[scnt] += v[i];
        }while (i != x);
        ++ scnt;
    }
}
/***********************************************/
int dfn[N] , low[N] , ncnt;
stack<int> S;
int bel[M] , tmp[N];
void dfs(int x , int fa) {
    dfn[x] = low[x] = ++ ncnt;
    for (int i = pre[x] ; ~i ; i = e[i].next) {
        int y = e[i].x;
        if (!dfn[y]) {
            S.push(i);
            dfs(y , i ^ 1);
            low[x] = min(low[x] , low[y]);
            if (low[y] > dfn[x]) {}//(x , y) is bridge
            if (low[y] >= dfn[x]) {
                ++ n; int j;
                do {
                    j = S.top() , S.pop();
                    if (tmp[e[j].x] != n)
                        E[m ++] = make_pair(n , e[j].x) , tmp[e[j].x] = n;
                    if (tmp[e[j ^ 1].x] != n)
                        E[m ++] = make_pair(n , e[j ^ 1].x ) , tmp[e[j ^ 1].x] = n;
                    bel[j >> 1] = n;
                } while (j != i);
            }
        } else if (i != fa && dfn[y] < dfn[x])
            S.push(i) , low[x] = min(low[x] , dfn[y]);
    }
}
```

2.2 最小树形图

```
const int INF = 1e9;
const int N = 505;
int n;
int from[N][N + N];
int edge[N][N + N];
int sel[N + N] , f[N + N] , vis[N + N];
int getf(int x){
    return f[x] == x ? x : f[x] = getf(f[x]);
}
void liuzhu(){
    f[1] = 1;
    for (int i = 2 ; i <= n; ++ i) {
        sel[i] = 1;
        f[i] = i;
        for (int j = 1 ; j <= n ; ++ j)
            if (f[j] != i) {
                from[j][i] = i;
                if (edge[sel[i]][i] > edge[j][i])
                    sel[i] = j;
            }
    }
    int limit = n;
    while(1) {
        int prelimit = limit;
        memset(vis , 0 , sizeof(vis));
        for (int i = 2 ; i <= prelimit ; ++ i)
            if (f[i] == i && !vis[i]) {
                int j = i;
                while(j != 1 && !vis[j]) {
                    vis[j] = i;
                    j = getf(sel[j]);
                }
                if(j == 1 || vis[j] != i) continue;
                vector<int> C;
                int k = j;
                do {
                    C.push_back(k);
                    k = getf(sel[k]);
                } while(k != j);
                ++limit;
                for (int i = 1; i <= n; ++ i) {
                    edge[i][limit] = INF;
                    from[i][limit] = limit;
                }
                f[limit] = vis[limit] = limit;
                for (int i = 0 ; i < (int)C.size() ; ++ i) {
                    int x = C[i];
                    f[x] = limit;
                    for (int j = 1; j <= n ; ++ j) {
                        if (edge[j][x] == INF)
                            continue;
                        if (edge[j][limit] > edge[j][x] - edge[sel[x]][x]) {
                            edge[j][limit] = edge[j][x] - edge[sel[x]][x];
                            from[j][limit] = x;
                        }
                    }
                }
                for (int j = 1 ; j <= n ; ++ j)
                    if (getf(j) == limit)
                        edge[j][limit] = INF;
                sel[limit] = 1;
                for (int j = 1 ; j <= n ; ++ j)
                    if (edge[sel[limit]][limit] > edge[j][limit])
                        sel[limit] = j;
            }
        if (prelimit == limit) break;
    }
    for (int i = limit ; i > 1 ; -- i)
        sel[from[sel[i]][i]] = sel[i];
}
```

2.3 全局最小割

```
pair<int , int> find() {
    int s = 0 , t = 0;
    for (int i = 1 ; i <= n ; ++ i) {
        d[i].w = 0;
        d[i].V.clear();
        vis[i] = 0;
    }
    for (int i = 1 ; i <= n ; ++ i) {
        int x = -1;
        for (int j = 1 ; j <= n ; ++ j)
            if (!f[j] && !vis[j] && (!~x || d[x] < d[j]))
                x = j;
        if (!~x) break;
        vis[x] = 1 , s = t , t = x;
        for (int j = 1 ; j <= n ; ++ j)
            if (!f[j] && !vis[j])
                d[j] += g[x][j];
    }
    res = min(res , d[t]);
    return make_pair(s , t);
}

void global_minimum_cut() {
    memset(f , 0 , sizeof(f));
    for (int i = 1 ; i < n ; ++ i) {
        pair<int , int> t = find();
        int x = t.first , y = t.second;
        f[y] = 1;
        for (int i = 1 ; i <= n ; ++ i) {
            g[x][i] += g[y][i];
            g[i][x] += g[i][y];
        }
    }
}
```

2.4 Hopcorft-Karp

```
int mx[N] , my[N];
queue<int> que;
int dx[N] , dy[N];
bool vis[N];

bool find(int x) {
    for (int i = pre[x] ; ~i ; i = e[i].next) {
        int y = e[i].x;
        if (!vis[y] && dy[y] == dx[x] + 1) {
            vis[y] = 1;
            if (!~my[y] || find(my[y])) {
                mx[x] = y , my[y] = x;
                return 1;
            }
        }
    }
    return 0;
}

int matching() {
    memset(mx , -1 , sizeof(mx));
    memset(my , -1 , sizeof(my));
    int ans = 0;
    while (1) {
        bool flag = 0;
        while (!que.empty()) que.pop();
        memset(dx , 0 , sizeof(dx));
        memset(dy , 0 , sizeof(dy));
        for (int i = 0 ; i < n ; ++ i)
            if (!~mx[i]) que.push(i);
        while (!que.empty()) {
            int x = que.front(); que.pop();
            for (int i = pre[x] ; ~i ; i = e[i].next) {
                int y = e[i].x;
                if (!dy[y]) {
                    dy[y] = dx[x] + 1 ;
                    if (~my[y])
                        que.push(my[y]) , dx[my[y]] = dy[y] + 1;
                    else
                        flag = 1;
                }
            }
        }
        if (!flag) break;
        memset(vis , 0 , sizeof(vis));
        for (int i = 0 ; i < n ; ++ i)
            if (!~mx[i] && find(i)) ++ ans;
    }
    return ans;
}
```

2.5 Dinic

```
int pre[N] , mcnt , s , t;
struct arc {
    int x , f , next;
} e[M];
void addarc(int x ,int y ,int z) {
    e[mcnt] = (arc) {y , z , pre[x]} , pre[x] = mcnt ++;
    e[mcnt] = (arc) {x , 0 , pre[y]} , pre[y] = mcnt ++;
}
int d[N] , cur[N] , q[N];
bool BFS() {
    memset(d , -1 , sizeof(d));
    int top = 0 , bot = -1;
    q[++ bot] = t , d[t] = 1;
    while (top != bot + 1) {
        int x = q[top ++];
        for (int i = pre[x] ; ~i ;i = e[i].next) {
            int y = e[i].x;
            if (!~d[y] && e[i ^ 1].f) {
                d[y] = d[x] + 1 , q[++ bot] = y;
                if (y == s) return 1;
            }
        }
    }
    return 0;
}
int DFS(int x , int flow = 1 << 30) {
    if (x == t || !flow) return flow;
    int sum = 0 , u;
    for (int& i = cur[x] ; ~i ; i = e[i].next) {
        int y = e[i].x;
        if (d[x] == d[y] + 1 && (u = DFS(y , min(flow , e[i].f)))) {
            e[i].f -= u , e[i ^ 1].f += u;
            sum += u , flow -= u;
            if (!flow) break;
        }
    }
    if (!sum) d[x] = -1;
    return sum;
}
int dinic() {
    int ans = 0;
    while (BFS()) {
        memcpy(cur , pre , sizeof(cur));
        ans += DFS(s);
    }
    return ans;
}
```

2.6 费用流

```
int S , T , pre[N] , mcnt;
struct arc {
    int x , f , c , next;
} e[M];

void addarc(int x , int y , int z , int c) {
    e[mcnt] = (arc) {y , z , c , pre[x]} , pre[x] = mcnt ++;
    e[mcnt] = (arc) {x , 0 ,-c , pre[y]} , pre[y] = mcnt ++;
}

int maxflow , ans , d[N] , h[N];
bool f[N];
bool Dijkstra() {
    priority_queue< pair<int , int> > Q;
    memset(d , 0x3f , sizeof(d));
    d[T] = 0; Q.push(make_pair(-d[T] , T));
    while (!Q.empty()) {
        int x = Q.top().second , w = -Q.top().first; Q.pop();
        if (w > d[x]) continue;
        if (x == S) {
            for (int i = 0 ; i <= T ; ++ i) {
                h[i] += d[i];
            }
            return 1;
        }
        for (int i = pre[x] ; ~i ; i = e[i].next) {
            int y = e[i].x , z = e[i ^ 1].c + h[x] - h[y];
            if (e[i ^ 1].f && d[x] + z < d[y]) {
                d[y] = d[x] + z;
                Q.push(make_pair(-d[y] , y));
            }
        }
    }
    return 0;
}
int dfs(int x , int flow = 1 << 30) {
    if (x == T) {
        maxflow += flow , ans += h[S] * flow;
        return flow;
    } f[x] = 1; int sum = 0 , u;
    for (int i = pre[x] ; ~i ; i = e[i].next) {
        int y = e[i].x , u;
        if (e[i].f && !f[y] && h[x] == e[i].c + h[y]) {
            u = dfs(y , min(flow , e[i].f));
            e[i].f -= u , e[i ^ 1].f += u;
            flow -= u , sum += u;
            if (!flow) break;
        }
    }
    return sum;
}
void MincostMaxflow() {
    //memset(h , 0 , sizeof(h));
    queue<int> Q;// 无负权边可选
    memset(f , 0 , sizeof(f));
    memset(h , 0x3f , sizeof(h));
    h[T] = 0 , f[T] = 1 , Q.push(T);
    while (!Q.empty()) {
        int x = Q.front(); Q.pop() , f[x] = 0;
        for (int i = pre[x] ; ~i ; i = e[i].next){
            int y = e[i].x , z = e[i ^ 1].c;
            if (e[i ^ 1].f && h[y] > h[x] + z){
                h[y] = h[x] + z;
                if (!f[y]) {
                    Q.push(y);
                    f[y] = 1;
                }
            }
        }
    }
    maxflow = 0 , ans = 0;
    while (Dijkstra()) {
        do {
            memset(f , 0 , sizeof(f));
        } while (dfs(S));
    } // while (Dijkstra());
}
```

2.7 一般图匹配

```
const int N = 505;

int S[N], Q[N], *Top = Q , idx;
int n, m;
int f[N] , pre[N] , nxt[N] , vis[N];
vector<int> e[N];
int getf(int x) {
    return x == f[x] ? x : f[x] = getf(f[x]);
}
int LCA(int x, int y) {
    idx ++;
    x = getf(x);
    y = getf(y);
    while (1) {
        if (x) {
            if (vis[x] == idx)
                return x;
            vis[x] = idx;
            x = getf(pre[nxt[x]]);
        }
        swap(x , y);
    }
}
void blossom(int x, int y, int l) {
    while (getf(x) != l) {
        pre[x] = y;
        if(S[nxt[x]] == 1) {
            *Top ++;
            S[*Top = nxt[x]] = 0;
        }
        f[x] = f[nxt[x]] = l;
        y = nxt[x];
        x = pre[y];
    }
}
void match(int x) {
    for (int i = 1 ; i <= n ; ++ i)
        f[i] = i;
    memset(S, -1, sizeof(S));
    memset(Q, 0, sizeof(Q));
    S[*(Top = Q + 1) = x] = 0;
    for(int *i = Q + 1 ; *i ; *i++) {
        for (auto &g : e[*i]) {
            if (S[g] == -1) {
                pre[g] = *i, S[g] = 1;
                if (!nxt[g]) {
                    for (int u = g, v = *i, lst ; v ; u = lst, v = pre[u])
                        lst = nxt[v], nxt[v] = u, nxt[u] = v;
                    return;
                }
                *Top++, S[*Top = nxt[g]] = 0;
            } else if(!S[g] && getf(g) != getf(*i)) {
                int l = LCA(g, *i);
                blossom(g, *i, l);
                blossom(*i, g, l);
            }
        }
    }
}
int main() {
    scanf("%d%d" , &n , &m);
    for (int i = 0 ; i < m ; ++ i) {
        int x , y;
        scanf("%d%d" , &x , &y);
        e[x].push_back(y);
        e[y].push_back(x);
    }
    for (int i = 1 ; i <= n ; ++ i) {
        if (!nxt[i]) {
            match(i);
        }
    }
    int ans = 0;
    for(int i = 1; i <= n; i++) ans += nxt[i] != 0;
    printf("%d\n", ans / 2);
    for(int i = 1; i <= n; i++) printf("%d ", nxt[i]);
    putchar('\n');
    return 0;
}
```

2.8 Kuhn-Munkras

```
int g[N][N] , lx[N] , ly[N] , match[N] , slack[N];
bool fx[N] , fy[N];
bool find(int x) {
    fx[x] = 1;
    for (int y = 0 ; y < n ; ++ y) {
        if (fy[y]) continue;
        if (lx[x] + ly[y] == g[x][y]) {
            fy[y] = 1;
            if (!~match[y] || find(match[y])) {
                match[y] = x;
                return 1;
            }
        } else {
            slack[y] = min(slack[y] , lx[x] + ly[y] - g[x][y]);
        }
    }
    return 0;
}
void update() {
    int delta = 1 << 30;
    for (int i = 0 ; i < n ; ++ i)
        if (!fy[i])
            delta = min(delta , slack[i]);
    for (int i = 0 ; i < n ; ++ i) {
        if (fx[i]) lx[i] -= delta;
        if (fy[i])
            ly[i] += delta;
        else
            slack[i] -= delta;
    }
}
int Kuhn_Munkras() {
    for (int i = 0 ; i < n ; ++ i) {
        match[i] = -1 , lx[i] = ly[i] = 0;
        for (int j = 0 ; j < n ; ++ j) {
            lx[i] = max(lx[i] , g[i][j]);
        }
    }
    for (int i = 0 ; i < n ; ++ i) {
        for (int j = 0 ; j < n ; ++ j)
            slack[j] = 1 << 30;
        while (1) {
            for (int j = 0 ; j < n ; ++ j)
                fx[j] = fy[j] = 0;
            if (find(i))
                break;
            update();
        }
    }
    int ans = 0;
    for (int i = 0 ; i < n ; ++ i)
        ans += g[match[i]][i];
    return ans;
}
```

2.9 dominator-tree

```
int n , m;
struct edge {
    int x , next;
} e[N << 5];
int mcnt;
int pre[N] , bre[N] , tree[N];
int mstamp , mvis[N];
int *prec, *succ;
vector<int> mord;
vector<int> buf[N];
int buf2[N];
int num[N] , fs[N], mins[N] , fa[N] , dom[N], sem[N];;
void dfs(int u) {
    mvis[u] = mstamp;
    num[u] = mord.size();
    mord.push_back(u);
    for (int i = succ[u] ; ~i ; i = e[i].next) {
        int v = e[i].x;
        if (mvis[v] != mstamp) {
            fa[v] = u;
            dfs(v);
        }
    }
}
int find(int u) {
    if (u != fs[u]) {
        int v = fs[u];
        fs[u] = find(fs[u]);
        if (mins[v] != -1 && num[sem[mins[v]]] < num[sem[mins[u]]]) {
            mins[u] = mins[v];
        }
    }
    return fs[u];
}
void merge(int u, int v) {
    fs[u] = v;
}
void mark(int source) { // prec = bre, succ = pre;
    mord.clear();
    ++mstamp;
    dfs(source);
    for (int i = 0; i < (int)mord.size(); ++i) {
        int u = mord[i];
        fs[u] = u;
        mins[u] = -1;
        buf2[u] = -1;
    }
    for (int i = (int)mord.size() - 1; i > 0; --i) {
        int u = mord[i], p = fa[u];
        sem[u] = p;
        for (int j = prec[u]; ~j ; j = e[j].next) {
            int v = e[j].x;
            if (mvis[v] != mstamp)
                continue;
            if (num[v] > num[u]) {
                find(v);
                v = sem[mins[v]];
            }
            if (num[v] < num[sem[u]]) {
                sem[u] = v;
            }
        }
        buf[sem[u]].push_back(u);
        mins[u] = u;
        merge(u, p);
        while (buf[p].size()) {
            int v = buf[p].back();
            buf[p].pop_back();
            find(v);
            if (sem[v] == sem[mins[v]]) {
                dom[v] = sem[v];
            } else {
                buf2[v] = mins[v];
            }
        }
    }
    dom[mord[0]] = mord[0];
    for (int i = 0; i < (int)mord.size(); ++i) {
        int u = mord[i];
        if (~buf2[u]) {
            dom[u] = dom[buf2[u]];
        }
        if (u != source) {
            //printf("%d dom %d\n" , dom[u] , u);
            //e[mcnt] = (edge) {u , tree[dom[u]]};
            //tree[dom[u]] = mcnt ++;
            res[dom[u]] = 1;
        }
    }
}
```

2.10 Gomory-Hu-tree

```
void divide(int l , int r) {
    if (l == r) return;
    random_shuffle(a + l , a + r + 1);
    for (int i = 0 ; i < mcnt ; i += 2)
        e[i].f = e[i ^ 1].f = (e[i].f + e[i ^ 1].f) / 2;
    s = a[l] , t = a[r];
    E[m ++] = (edge) {s , t , -dinic()};
    int ns = 0 , nt = 0;
    for (int i = l ; i <= r ; ++ i)
        if (!~d[a[i]])
            T[nt ++] = a[i];
        else
            S[ns ++] = a[i];
    for (int i = 0 ; i < ns ; ++ i)
        a[l + i] = S[i];
    for (int i = 0 ; i < nt ; ++ i)
        a[l + ns + i] = T[i];
    divide(l , l + ns - 1);
    divide(l + ns , r);
}
```

2.11 最大团搜索

```
int n , mc[N] , list[N][N] , len[N] , ans;
bool g[N][N] , found;

void dfs(int size) {
    int i , j , k;
    if (!len[size]) {
        if (size > ans)
            ans = size , found = 1;
        return;
    }
    for (k = 0 ; k < len[size] && !found ; ++ k) {
        if (size + len[size] - k <= ans)
            break;
        i = list[size][k];
        if (size + mc[i] <= ans)
            break;
        for (j = k + 1 , len[size + 1] = 0 ; j < len[size] ; ++ j)
            if (g[i][list[size][j]])
                list[size + 1][len[size + 1] ++] = list[size][j];
        dfs(size + 1);
    }
}
void max_cluster() {
    int i , j;
    mc[n] = ans = 1;
    for (i = n - 1 ; i ; -- i) {
        found = 0 , len[1] = 0;
        for (j = i + 1 ; j <= n ; ++ j)
            if (g[i][j])
                list[1][len[1] ++] = j;
        dfs(1);
        mc[i] = ans;
    }
}
void work() {
    for (int i = 1 ; i <= n ; ++ i)
        for (int j = 1 ; j <= n ; ++ j)
            scanf("%d",&g[i][j]);
    max_cluster();
    cout << ans << endl;
}
/**************************************/
// 极大团枚举 O(3^{n/3})
int trail_zero(ULL s) {
    return s ? __builtin_ctzll(s) : 64;
}
bool BronKerbosch(const vector<ULL> &g, ULL cur, ULL allow, ULL forbid) {
    if (allow == 0 && forbid == 0) {
        for (int i = 0; i < n; ++ i) {
            printf("%d" , (int) (cur >> i & 1));
        }
        puts("");
        return false;
    }
    if (allow == 0) return false;
    int pivot = trail_zero(allow | forbid);
    ULL z = allow & ~g[pivot];
    for (size_t u = trail_zero(z); u < g.size(); u += trail_zero(z >> (u + 1)) + 1) {
        if (BronKerbosch(g, cur | (1ULL << u), allow & g[u], forbid & g[u])) return true;
        allow ^= 1ULL << u; forbid |= 1ULL << u;
    }
    return false;
}
//BronKerbosch(g , 0 , (1ULL << n) - 1 , 0);
```

3 Mathematics

3.1 高斯消元

```
int rank = 0;
for (int i = 0 ; i < n ; ++ i) {
    int pivot = rank;
    for (int j = rank + 1 ; j < m ; ++ j)
        if (fabs(a[j][i]) > fabs(a[pivot][i]))
            pivot = j;
    if (fabs(a[pivot][i]) < 1e-10)
        continue;
    for (int j = 0 ; j < n ; ++ j)
        swap(a[rank][j] , a[pivot][j]);
    double tmp = a[rank][i];
    for (int j = 0 ; j < n ; ++ j)
        a[rank][j] /= tmp;
    for (int k = 0 ; k < m ; ++ k) {
        if (k != rank) {
            double times = a[k][i];
            for (int j = 0 ; j < n ; ++ j) {
                a[k][j] -= a[rank][j] * times;
            }
        }
    }
    ++ rank;
}
```

3.2 行列式

```
LL det(LL A[][N]) {
    LL ans = 1;
    for (int i = 1 ; i <= n ; ++ i) {
        for (int j = i + 1 ; j <= n ; ++ j) {
            while (A[j][i]) {
                LL t = A[i][i] / A[j][i];
                for (int k = 1 ; k <= n ; ++ k) {
                    A[i][k] -= A[j][k] * t;
                    swap(A[i][k] , A[j][k]);
                }
                ans = -ans;
            }
        }
        if (!A[i][i])
            return 0;
        ans *= A[i][i];
    }
    return ans;
}
```

3.3 FFT

```
void FFT(Complex P[], int n, int oper) {
    for (int i = 1, j = 0; i < n - 1; i ++) {
        for (int s = n; j ^= s >>= 1, ~j & s;);
        if (i < j) {
            swap(P[i], P[j]);
        }
    }
    for (int d = 0; (1 << d) < n; d++) {
        int m = 1 << d, m2 = m * 2;
        double p0 = pi / m * oper;
        Complex unit_p0(cos(p0) , sin(p0));
        for (int i = 0; i < n; i += m2) {
            Complex unit(1 , 0);
            for (int j = 0; j < m; j++) {
                Complex &P1 = P[i + j + m], &P2 = P[i + j];
                Complex t = unit * P1;
                P1 = P2 - t;
                P2 = P2 + t;
                unit = unit * unit_p0;
            }
        }
    }
}
void NTT(int P[], int n, int oper) {
    for (int i = 1, j = 0; i < n - 1; i++) {
        for (int s = n; j ^= s >>= 1, ~j & s;);
        if (i < j) {
            swap(P[i], P[j]);
        }
    }//998244353 1004535809
    for (int d = 0; (1 << d) < n; d++) {
        int m = 1 << d, m2 = m << 1;
        int unit_p0 = power(G , Q - 1 >> d + 1);
        if (oper == -1)
            unit_p0 = inverse(unit_p0);
        for (int i = 0; i < n; i += m2) {
            int unit = 1;
            for (int j = 0; j < m; j++) {
                int &P1 = P[i + j + m], &P2 = P[i + j];
                int t = (LL)unit * P1 % Q;
                P1 = P2 - t + Q;
                if (P1 >= Q) P1 -= Q;
                P2 = P2 + t;
                if (P2 >= Q) P2 -= Q;
                unit = (LL)unit * unit_p0 % Q;
            }
        }
    }
}
void FFT(int P[] , int n , int oper) {
    if (n == 1) return;
    int m = 0;
    for (int i = 0 ; i < n ; i += 3) tmp[m ++] = P[i];
    for (int i = 1 ; i < n ; i += 3) tmp[m ++] = P[i];
    for (int i = 2 ; i < n ; i += 3) tmp[m ++] = P[i];
    memcpy(P , tmp , n << 2) , m = n / 3;
    FFT(P , m , oper);
    FFT(P + m , m , oper);
    FFT(P + m + m , m , oper);
    int unit_p0 = hash[oper * n];
    int unit = 1;
    for (int i = 0 , j = 0 ; i < n ; ++ i) {
        tmp[i] = P[j] + (LL)unit * (P[m + j] + (LL)unit * P[m + m + j] % Q) % Q;
        tmp[i] %= Q;
        unit = (LL)unit * unit_p0 % Q;
        if (++ j == m)
            j = 0;
    }
    memcpy(P , tmp , n << 2);
}
void FWT(int a[] , int len , int oper) {
    for (int k = 0 ; 1 << k < len ; ++ k) {
        for (int i = 0 ; i < len ; ++ i) {
            if (~i >> k & 1) {
                int j = i ^ (1 << k);
                int x = (a[i] + Q - a[j]) % Q;
                int y = (a[i] + a[j]) % Q;
                if (oper == -1) {
                    x = (Q - x) % Q;
                    swap(x , y);
                }
                a[i] = x;
                a[j] = y;
            }
        }
    }
}
```

3.4 Euler 筛

```
int m , n;
bool f[N];
int prime[N] , tot;
int mu[N] , phi[N];
void init(int n) {
    int i , j , x;
    mu[1] = 1;
    for (i = 2 ; i <= n ; ++ i) {
        if (!f[i]) {
            prime[tot ++] = i ;
            phi[i] = i - 1 , h[i] = 1;
            mu[i] = -1;
        }
        for (j = 0 ; j < tot ; ++ j) {
            x = i * prime[j];
            if (x > n) break;
            f[x] = 1 ;
            if (i % prime[j] == 0) {
                phi[x] = phi[i] * prime[j];
                mu[x] = 0;
                break;
            } else {
                phi[x] = phi[i] * (prime[j] - 1) ;
                mu[x] = -mu[i];
            }
        }
    }
}
LL solve(int n , int m) {
    LL ans = 0;
    if (n > m) swap(n , m);
    for (int i = 1 , x ; i <= n ; i = x + 1) {
        x = min(n / (n / i) , m / (m / i));
        ans += (LL) (sum[x] - sum[i - 1]) * (n / i) * (m / i);
    }
    return ans;
}
```

3.5 自适应 simpson 积分

```
double F(double x) {
    return sqrt(1 + 4 * a * a * x * x);
}
double simpson(double a , double b) {
    double c = (a + b) * 0.5;
    return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
}
double asr(double a , double b , double eps , double A) {
    double c = (a + b) * 0.5;
    double L = simpson(a , c) , R = simpson(c , b);
    if (fabs(L + R - A) <= 15 * eps)
        return L + R + (L + R - A) / 15;
    return asr(a , c , eps / 2 , L) + asr(c , b , eps / 2 , R);
}
double cal(double L , double R) {
    return asr(L , R , 1e-5 , simpson(L , R));
}
```

3.6 离散对数 BSGS

```
//S * P^k = T , m = sqrt(S) , I = P ^-m
map<unsigned int , int> hash;
unsigned int E = S;
for (i = 0 ; i < m ; ++ i) {
    if (E == T) {
        printf("%u\n" , i);
        return;
    }
    if (!hash.count(E))
        hash[E] = i;
    E = E * P;
}
for (i = 1 ; i < m ; ++ i) {
    T = T * I;
    if (hash.count(T)) {
        printf("%u\n" , i * m + hash[T]);
        return;
    }
}
puts("poor sisyphus");
```

3.7 素性测试启发式分解

```
inline LL mod_mul(LL a , LL b , LL Q){
    return (a * b - (LL)((long double)a * b / Q) * Q) % Q;
}
inline LL myrand() {
    return rand() << 30 | rand();
}
LL mod_exp(LL a , LL x , LL n) {
    LL ret = 1;
    while(x) {
        if(x & 1)
            ret = mod_mul(ret , a , n);
        a = mod_mul(a , a , n) , x >>= 1;
    }
    return ret;
}
bool Rabin_Miller(LL n) { //素性测试
    LL k = 0 , i , j , m , a;
    if (n < 2) return 0;
    if (n == 2) return 1;
    if (~n & 1) return 0;
    m = n - 1;
    while(~m & 1)
        m >>= 1 , ++ k;
    for(i = 0 ; i < 20; ++ i) {
        a = myrand() % (n - 2) + 2;
        a = mod_exp(a , m , n);
        if (a == 1)
            continue;
        for (j = 0 ; j < k ; ++ j) {
            if (a == n - 1)
                break;
            a = mod_mul(a , a , n);
        }
        if (j < k)
            continue;
        return 0;
    }
    return 1;
}
inline LL func(LL x , LL n) {
    return (mod_mul(x , x , n) + 1) % n;
}
LL Pollard(LL n) { //启发式分解
    LL i , x , y , p;
    if (Rabin_Miller(n))
        return n;
    if(~n & 1)
        return 2;
    for(i = 1 ; i < 20 ; ++ i) {
        x = i;
        y = func(x , n);
        p = __gcd(y - x , n);
        while(p == 1) {
            x = func(x , n);
            y = func(func(y , n) , n);
            p = __gcd((y - x + n) % n , n) % n;
        }
        if(p == 0 || p == n)
            continue;
        return p;
    }
}
void factor(LL n , vector<int>& ans) {
    LL x = Pollard(n);
    if(x == n) {
        ans.push_back(x);
        return;
    }
    factor(x , ans);
    factor(n / x , ans);
}
```

3.8 线性递推数列

```
LL n ;
int d[N] , c[N] , t[N] , Deg;
struct Poly {
    int a[N];
    Poly() {
        memset(a , 0 , sizeof(a));
    }
    int& operator [] (int x) {
        return a[x];
    }
};
inline void add(int& A , int B) {
    A += B;
    if (A >= Q)
        A -= Q;
}
Poly operator * (Poly& X , Poly& Y) {
    int i , j; Poly ans;
    for (i = 0 ; i < Deg ; ++ i)
        for (j = 0 ; j < Deg ; ++ j)
            add(ans[i + j] , (LL)X[i] * Y[j] % Q);
    for (i = Deg + Deg - 2 ; i >= Deg ; -- i) {
        for (j = 1 ; j <= Deg ; ++ j)
            add(ans[i - j] , (LL)ans[i] * c[j] % Q);
        ans[i] = 0;
    }
    return ans;
}
void work() {
    memset(c , 0 , sizeof(c));
    memset(d , 0 , sizeof(d));
    Deg = 2;
    d[0] = 3 , d[1] = 4;
    c[2] = 1 , c[1] = 2;
    n = 3;
    /*  c 为转移关系 d 为初值 Deg 为阶数*/
    //Fi = 2 * Fi-1 + 1 * Fi - 2
    //F0 = 3 , F1 = 4 , F2 = 11 , F3 = 26...
    Poly ans , P;
    P[1] = 1 , ans[0] = 1;
    while (n) {
        if (n & 1)
            ans = ans * P;
        P = P * P , n >>= 1;
    }
    int res = 0;
    for (int i = 0 ; i < Deg ; ++ i) {
        add(res , (LL)d[i] * ans[i] % Q);
    }
    printf("%d\n" , res);
}
```

3.9 拉格朗日插值

```
for (int i = 0 ; i < n ; ++ i)
    val[i] = cal(i);
for (int i = 0 ; i < n ; ++ i) {
    int cur = 0 , nxt = 1;
    memset(f[cur] , 0 , sizeof(f[cur]));
    f[cur][0] = 1;
    for (int j = 0 ; j < n ; ++ j) {
        if (i != j) {
            x = inverse((i - j + Q) % Q);
            y = (LL)(Q - j) * x % Q;
            //printf("%d %d : %d %d\n" , i , j , x , y);
            memset(f[nxt] , 0 , sizeof(f[nxt]));
            for (int k = 0 ; k < n ; ++ k) {
                if (f[cur][k]) {
                    f[nxt][k] += (LL)f[cur][k] * y % Q;
                    f[nxt][k] %= Q;
                    f[nxt][k + 1] += (LL)f[cur][k] * x % Q
                    ;
                    f[nxt][k + 1] %= Q;
                }
            }
            swap(cur , nxt);
        }
    }
    for (int j = 0 ; j < n ; ++ j)
        L[i][j] = f[cur][j];
}
memset(res , 0 , sizeof(res));
for (int i = 0 ; i < n ; ++ i)
    for (int x = 0 ; x < n ; ++ x) {
        res[x] += (LL)L[i][x] * val[i] % Q;
        res[x] %= Q;
    }
/*******************************************/
// 特殊的，一个K阶多项式已知前K+1项求第n项 O(Klogn)
// 例: 求前n个自然数的K次方和，多项式是K+1阶
scanf("%d%d" , &n , &K);
for (int i = 1 ; i <= K + 1 ; ++ i) {
    f[i] = (f[i - 1] + power(i , K)) % Q;
}
if (n <= K + 1) {
    printf("%d\n" , f[n]);
    return;
}
int A = 1 , B = 1;
for (int i = 0 ; i <= K + 1; ++ i)
    A = (LL)A * (n - i) % Q;
for (int i = 1 ; i <= K + 1; ++ i)
    B = (LL)B * (Q - i) % Q;
int res = 0;
for (int i = 0 ; i <= K + 1; ++ i) {
    int C = (LL)A * inverse((LL)(n - i) * B % Q) % Q;
    res += (LL)f[i] * C % Q , res %= Q;
    if (i == K + 1) break;
    B = (LL)B * (i + 1) % Q * inverse(Q - (K + 1 - i)) % Q
    ;
}
printf("%d\n" , res);
```

4 Geometry

4.1 基础 2D 几何

```
const double eps = 1e-10 , pi = acos(-1.0);
inline int dcmp(double x) {
    return (x > eps) - (x < -eps);
}

struct Point {
    double x , y;
    Point (double x = 0 , double y = 0) : x(x) , y(y) {}
    void input() {
        scanf("%lf%lf",&x,&y);
    }
    bool operator < (const Point& R) const{
        if (dcmp(x - R.x) == 0)
            return dcmp(y - R.y) < 0;
        return dcmp(x - R.x) < 0;
    }
    bool operator == (const Point& R) const{
        return dcmp(x - R.x) == 0 && dcmp(y - R.y) == 0;
    }
    Point operator + (const Point& R) const{
        return Point(x + R.x , y + R.y);
    }
    Point operator - (const Point& R) const{
        return Point(x - R.x , y - R.y);
    }
    Point operator * (const double& R) const{
        return Point(x * R , y * R);
    }
    Point operator / (const double& R) const{
        return Point(x / R , y / R);
    }
    double operator ^ (const Point& R) const{
        return x * R.y - y * R.x;
    }
    double operator % (const Point& R) const{
        return x * R.x + y * R.y;
    }
    double len() {
        return sqrt(*this % *this);
    }
};
// 向量的极角，[-pi,pi)
double Angle(Point V) {
    return atan2(V.y , V.x);
}
// 两个向量的夹角，不分正负[0,pi)
double Angle(Point A , Point B) {
    return acos((A % B) / A.len() / B.len());
}
// 逆时针旋转
Point Rotate(Point A , double rad) {
    double Sin = sin(rad) , Cos = cos(rad);
    return Point(A.x * Cos - A.y * Sin , A.x * Sin + A.y * Cos);
}
// 向量的单位法向量，利用旋转得到
Point Normal(Point A) {
    double L = A.len();
    return Point(-A.y / L , A.x / L);
}
// 直线交点，v和w为两个直线的方向向量，
// 设交点的参数为P+vt,Q+wt,连立方程解t
// 线段，射线对这个t的参数有限制，很好理解。
Point GetLineIntersection(Point P , Point v , Point Q , Point w) {
    Point u = P - Q;
    double t1 = (w ^ u) / (v ^ w);
    return P + v * t1;
}
// 点到直线有向距离，这里直线是用两个点表示的
double DistancePointToLine(Point P , Point A , Point B) {
    Point v = B - A;
    return (v ^ (P - A)) / v.len();
}
// 点到线段距离，就是上面的代码判断一下P在AB上投影的位置。
double DistancePointToSegment(Point P , Point A , Point B) {
    if (A == B) return (P - A).len();
    Point v1 = B - A , v2 = P - A , v3 = P - B;
    if (dcmp(v1 % v2) < 0) return v2.len();
    if (dcmp(v1 % v3) > 0) return v3.len();
    return fabs(v1 ^ v2) / v1.len();
}
// 返回点在直线上的投影
Point GetLineProjection(Point P , Point A , Point B) {
    Point v = B - A;
    return A + v * (v % (P - A) / (v % v));
}
// 判断线段是否相交，没有考虑共线的情况。
bool SegmentProperIntersection(Point a1 , Point a2 , Point b1 , Point b2) {
    double c1 = (a2 - a1) ^ (b1 - a1);
    double c2 = (a2 - a1) ^ (b2 - a1);
    double c3 = (b2 - b1) ^ (a1 - b1);
    double c4 = (b2 - b1) ^ (a2 - b1);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
// 点是否在线段上,判定方式为到两个端点的方向是否不一致。
bool OnSegment(Point P , Point a1 , Point a2) {
    double len = (P - a1).len();
    if (dcmp(len) == 0) return true;
    a1 = a1 - P , a2 = a2 - P;
    return dcmp((a1 ^ a2) / len) == 0 && dcmp(a1 % a2) <= 0;
}
```

4.2 直线与圆

```
struct Line {
    Point P , V; // P + Vt
    double angle;
    Line () {}
    Line (Point A , Point B) {
        P = A , V = B - A;
        angle = atan2(V.y , V.x);
    }
    bool operator < (const Line& R) const {
        return angle < R.angle;
    }
    Point point(double t){
        return P + V * t;
    }
};
struct Circle {
    Point O;
    double r;
    Circle () {}
    Circle (Point _O , double _r) {O = _O , r = _r;}
    Point point(double arc) {
        return Point(O.x + cos(arc) * r , O.y + sin(arc) * r);
    }
    void input() {
        O.input() , scanf("%lf",&r);
    }
    void print() {
        printf("(%f,%f,%f)\n" , O.x + eps , O.y + eps , r + eps);
    }
};

// 判定直线与圆相交
// 方法为连立直线的参数方程与圆的方程，很好理解
// t1,t2为两个参数，sol为点集。有了参数，射线线段什么的也很方便
int getLineCircleIntersection(Line L , Circle C , double& t1 , double& t2 , vector<Point>& sol) {
    double a = L.V.x , b = L.P.x - C.O.x , c = L.V.y , d = L.P.y - C.O.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)) / (e + e);
    t2 = (-f + sqrt(delta)) / (e + e);
    sol.push_back(L.point(t1)) , sol.push_back(L.point(t2));
    return 2;
}
// 判定圆和圆之间的关系
// 内含，内切，相交，重合，外切，相离
int getCircleCircleIntersection(Circle C1 , Circle C2 , vector<Point>& sol) {
    double d = (C1.O - C2.O).len();
    if (dcmp(d) == 0) { //同心
        if (dcmp(C1.r - C2.r) == 0)//重合
            return -1;
        return 0;//内含
    }
    if (dcmp(C1.r + C2.r - d) < 0) return 0;//相离
    if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0;//内含
    double a = Angle(C2.O - C1.O); // acos内可能越界
    double p = (C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d);
    p = max(-1.0 , min(1.0 , p));
    double da = acos(p);
    Point P1 = C1.point(a - da) , P2 = C1.point(a + da);
    sol.push_back(P1);
    if (P1 == P2) return 1; //切
    sol.push_back(P2);
    return 2;
}
// 过点p到圆C的切线。返回切线条数，sol里为方向向量
int getTangents(Point P, Circle C, vector<Point>& sol) {
    Point u = C.O - P;
    double dist = u.len();
    if(dist < C.r) return 0;// 园内
    if(dcmp(dist - C.r) == 0) {// p在圆上，只有一条切线
        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;
    }
}
//两个圆的公切线，对应切点存在ab里面
int getTangents(Circle A , Circle B , Point* a , Point* b) {
    int cnt = 0;
    if (A.r < B.r)
        swap(A , B) , swap(a , b);
    double dist = (A.O - B.O).len() , dr = A.r - B.r , sr = A.r + B.r;
    if (dcmp(dist - dr) < 0) // 内含
        return 0;
    double base = Angle(B.O - A.O);
    if (dcmp(dist) == 0 && dcmp(A.r - B.r) == 0)
        return -1;//重合
    if (dcmp(dist - dr) == 0) {//内切
        a[cnt] = A.point(base);
        b[cnt] = B.point(base);
        return 1;
    }
    double ang = acos(dr / dist);//非上述情况，两条外公切线
    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 (dcmp(dist - sr) == 0) {// 外切，中间一条内公切线
        a[cnt] = A.point(base) , b[cnt] = B.point(pi + base) , ++ cnt;
    } else if (dcmp(dist - sr) > 0) {
        ang = acos(sr / dist);//相离，两条内公切线
        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;
}
// 外接圆，三根中线交点
Circle CircumscribedCircle(Point A , Point B , Point C) {
    Point D = (B + C) / 2 , d = Normal(B - C);
    Point E = (A + C) / 2 , e = Normal(A - C);
    Point P = GetLineIntersection(D , d , E , e);
    return Circle(P , (C - P).len());
}
// 内接圆，黑科技
Circle InscribedCircle(Point A , Point B , Point C) {
    double a = (B - C).len() , b = (A - C).len() , c = (A - B).len();
    Point P = (A * a + B * b + C * c) / (a + b + c);
    return Circle(P , fabs(DistancePointToLine(P , A , B)));
}
```

4.3 点在多边形内判定

```
for (int i = 0 ; i < n ; ++ i)
    if (OnSegment(P , p[i] , p[i + 1]))
        return 0;
int res = 0;
for (int i = 0 ; i < n ; ++ i) {
    Point a = p[i] , b = p[i + 1];
    if (a.y > b.y) swap(a , b);
    if (dcmp((a - P) ^ (b - P)) < 0 && dcmp(a.y - P.y) < 0 && dcmp(b.y - P.y) >= 0)
        res ^= 1;
}
return res;
```

4.4 2D 凸包相关

```
inline LL OnLeft(Point P , Point A , Point B) {
    return (B - A) ^ (P - A);
}
/********* Naive 凸包 2.0 O(n+m) *********/
int top = 0;
for (int i = 0 ; i < n ; ++ i) {
    while (top > 1 && OnLeft(p[i] , s[top - 2] , s[top - 1]) <= 0) {
        -- top;
    }
    s[top ++] = p[i];
}
int tmp = top;
for (int i = n - 2 ; i >= 0 ; -- i) {
    while (top > tmp && OnLeft(p[i] , s[top - 2] , s[top - 1]) <= 0) {
        -- top;
    }
    s[top ++] = p[i];
}
if (n > 1)
    -- top;
/********* Minkowski-Sum O(n+m) *********/
Vec.clear();
Point cur = a[0] + b[0];
for (int i = 0 , j = 0 ; i < n || j < m ; ) {
    if (i < n && (j == m || ((a[i + 1] - a[i]) ^ (b[j + 1] - b[j])) >= 0)) {
        cur = cur + a[i + 1] - a[i];
        ++ i;
    } else {
        cur = cur + b[j + 1] - b[j];
        ++ j;
    }
    Vec.push_back(make_pair(cur , 1));
}
/******* 点在凸多边形内判定 O(logn) *******/
bool InConvex(Point q) {
    if (OnLeft(q , p[0] , p[1]) < 0 || OnLeft(q , p[0] , p[n - 1]) > 0)
        return 0;
    int l = 2 , r = n - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (OnLeft(q , p[0] , p[mid]) <= 0) {
            r = mid;
        } else {
            l = mid + 1;
        }
    }
    return OnLeft(q , p[r - 1] , p[r]) >= 0;
}
/******* 点到凸多边形的切线 O(logn) *******/
#define above(b , c) (OnLeft(b , q , c) > 0)
#define below(b , c) (OnLeft(b , q , c) < 0)
int getRtangent(Point q) { // find max
    int ret = 0;
    int l = 1 , r = n - 1;
    while (l <= r) {
        int dnl = above(p[l] , p[l + 1]);
        int mid = l + r >> 1;
        int dnm = above(p[mid] , p[mid + 1]);
        if (dnm) {
            if (above(p[mid], p[ret])) {
                ret = mid;
            }
        }
        if (dnl) {
            if (above(p[l], p[ret])) {
                ret = l;
            }
            if (dnm && above(p[mid] , p[l])) {
                r = mid - 1;
            } else {
                l = mid + 1;
            }
        } else {
            if (!dnm && above(p[mid] , p[l])) {
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        }
    }
    return ret;
}
```

```
int getLtangent(Point q) { // find min
    int ret = 0;
    int l = 1 , r = n - 1;
    while (l <= r) {
        int dnl = below(p[l] , p[l - 1]);
        int mid = l + r + 1 >> 1;
        int dnm = below(p[mid] , p[mid - 1]);
        if (dnm) {
            if (below(p[mid], p[ret])) {
                ret = mid;
            }
        }
        if (dnl) {
            if (below(p[l], p[ret])) {
                ret = l;
            }
            if (dnm && below(p[mid] , p[l])) {
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        } else {
            if (!dnm && below(p[mid] , p[l])) {
                r = mid - 1;
            } else {
                l = mid + 1;
            }
        }
    }
    return ret;
}
/****** 直线对凸多边形的交点 O(logn) ******/
double arc[N] , sum[N];
void init() {
    for (int i = 0 ; i < n ; ++ i) {
        p[i + n] = p[i];
    } p[n + n] = p[0];
    for (int i = 0 ; i < n + n ; ++ i) {
        sum[i + 1] = sum[i] + (p[i] ^ p[i + 1]);
    }
    for (int i = 0 ; i < n ; ++ i) {
        int j = (i + 1) % n;
        arc[i] = atan2(p[j].y - p[i].y , p[j].x - p[i].x);
        if (i && arc[i] < arc[i - 1]) {
            arc[i] += pi + pi;
        }
    }
}
int getseg(Point P , Point V , int l , int r) {
    -- l;
    while (l < r) {
        int mid = l + r + 1 >> 1;
        if ((V ^ (p[mid] - P)) < 0) {
            l = mid;
        } else {
            r = mid - 1;
        }
    }
    return l;
}
void work(Point A , Point B) {
    if (B < A) {
        swap(A , B);
    }
    double al = atan2(B.y - A.y , B.x - A.x);
    if (al < arc[0]) al += pi + pi;
    int Left = (lower_bound(arc , arc + n , al) - arc) % n ;
    double ar = atan2(A.y - B.y , A.x - B.x);
    if (ar < arc[0]) ar += pi + pi;
    int Right = lower_bound(arc , arc + n , ar) - arc;
    int down = getseg(A , B - A , Left , Right);
    int up = getseg(B , A - B , Right , Left + n);
    if (down < Left || up < Right) {
        puts("0.000000");
    } else {
        Point D = GetLineIntersection(A , B - A , p[down] , p[down + 1] - p[down]);
        Point U = GetLineIntersection(B , A - B , p[up] , p[up + 1] - p[up]);
        //printf("%f %f / %f %f\n" , D.x , D.y , U.x , U.y );
        double area = (D ^ p[down + 1]) + (sum[up] - sum[down + 1]) + (p[up] ^ U) + (U ^ D);
        printf("%.6f\n" , min(sum[n] - area , area) / 2);
    }
}
```

4.5 半平面交

```
typedef vector<Point> Polygon;

//用有向直线AB的左半平面切割 O(n)
Polygon CutPolygon(const Polygon& poly , Point A , Point B ) {
    Polygon newpoly;
    int n = poly.size();
    for (int i = 0 ; i < n ; ++ i) {
        const Point &C = poly[i] , &D = poly[(i + 1) % n];
        if (dcmp((B - A) ^ (C - A)) >= 0)
            newpoly.push_back(C);
        if (dcmp((B - A) ^ (C - D)) != 0) {
            double t = ((B - A) ^ (C - A)) / ((D - C) ^ (B - A));
            if (dcmp(t) > 0 && dcmp(t - 1) < 0)
                newpoly.push_back(C + (D - C) * t);
        }
    }
    return newpoly;
}
/****************************************************/
inline bool Onleft(Line L , Point P) {
    return (L.V ^ (P - L.P)) > 0;
}
Point GetLineIntersection(Line A , Line B) {
    Point u = A.P - B.P;
    double t = (B.V ^ u) / (A.V ^ B.V);
    return A.point(t);
}
Point p[N];
Line q[N];
int HalfPlaneIntersection(Line* L , int n , Point* Poly) {
    sort(L , L + n);
    int top = 0 , bot = 0;
    q[0] = L[0];
    for (int i = 1 ; i < n ; ++ i) {
        while (top < bot && !Onleft(L[i] , p[bot - 1])) -- bot;
        while (top < bot && !Onleft(L[i] , p[top])) ++ top ;
        q[++ bot] = L[i];
        if (dcmp(L[i].V ^ q[bot - 1].V) == 0) {
            -- bot;
            if (Onleft(q[bot] , L[i].P))
                q[bot] = L[i];
        }
        if (top < bot)
            p[bot - 1] = GetLineIntersection(q[bot - 1] , q[bot]);
    }
    while (top < bot && !Onleft(q[top] , p[bot - 1])) -- bot;
    if (bot - top <= 1) return 0;
    p[bot] = GetLineIntersection(q[bot] , q[top]);
    int m = 0;
    for (int i = top ; i <= bot ; ++ i) Poly[m ++] = p[i];
    return m;
}
```

4.6 圆面积相关

```
/******圆和多边形求交****/
double sector_area(Point A , Point B , double R) {
    double theta = Angle(A) - Angle(B);
    while (theta < 0) theta += pi + pi;
    while (theta >= pi + pi) theta -= pi + pi;
    theta = min(theta , pi + pi - theta);
    return R * R * theta;
}//a[n] = a[0]
double cal(double R) {
    double area = 0;
    for (int i = 0 ; i < n ; ++ i) {
        double t1 = 0 , t2 = 0 , delta;
        Line L = Line(a[i] , a[i + 1]);
        int cnt = getLineCircleIntersection(L , Circle(Point(0 , 0) , R) , t1 , t2);
        Point X = L.point(t1) , Y = L.point(t2);
        bool f1 = dcmp(a[i].len() - R) <= 0 , f2 = dcmp(a[i + 1].len() - R) <= 0;
        if (f1 && f2)
            delta = fabs(a[i] ^ a[i + 1]);
        else if (!f1 && f2) {
            delta = sector_area(a[i] , X , R) + fabs(X ^ a[i + 1]);
        } else if (f1 && !f2) {
            delta = fabs(a[i] ^ Y) + sector_area(Y , a[i + 1] , R);
        } else {
            if (cnt > 1 && 0 < t1 && t1 < 1 && 0 < t2 && t2 < 1) {
                delta = sector_area(a[i] , X , R) + sector_area(Y , a[i + 1] , R) + fabs(X ^ Y);
            } else {
                delta = sector_area(a[i] , a[i + 1] , R);
            }
        }
        area += delta * dcmp(a[i] ^ a[i + 1]);
    }
    return area / 2;
}
/*********圆交/并*******/
void getarea() { // 计算圆并的重心，必要的时候可以去除有包含关系的圆
    for (int i = 0 ; i < n ; ++ i) {
        vector< pair<double , int> > Vec;
        int cnt = 1;
        Vec.push_back({0 , 0});
        Vec.push_back({2 * pi , 0});
        for (int j = 0 ; j < n ; ++ j) {
            double dist = (c[j].O - c[i].O).len();
            if (dcmp(dist) == 0 && dcmp(c[i].r - c[j].r) == 0) {
                if (i < j) {
                    ++ cnt;
                }
                continue;
            }
            if (dcmp(dist - c[j].r - c[i].r) >= 0) {
                continue;
            }
            if (dcmp(dist + c[j].r - c[i].r) <= 0) { // j in i
                continue;
            }
            if (dcmp(dist + c[i].r - c[j].r) <= 0) { // i in j
                ++ cnt;
                continue;
            }
            double an = atan2(c[j].O.y - c[i].O.y , c[j].O.x - c[i].O.x);
            double p = (c[i].r * c[i].r + dist * dist - c[j].r * c[j].r) / (2 * c[i].r * dist);
            double da = acos(max(-1.0 , min(1.0 , p)));

            double L = an - da , R = an + da;
            //printf("%d : %f %f\n" , j , L , R);
            if (L < 0) L += 2 * pi;
            if (R < 0) R += 2 * pi;
            if (L >= 2 * pi) L -= 2 * pi;
            if (R >= 2 * pi) R -= 2 * pi;
            if (L < R) {
                Vec.push_back({L , 1});
                Vec.push_back({R , -1});
            } else {
                Vec.push_back({0 , 1});
                Vec.push_back({R , -1});
                Vec.push_back({L , 1});
                Vec.push_back({2 * pi , -1});
            }
        }
        sort(Vec.begin() , Vec.end());
        for (int j = 0 ; j + 1 < Vec.size() ; ++ j) {
            //printf("%d : %d %f\n" , j , cnt , Vec[j].first);
            cnt += Vec[j].second;
            if (cnt == 1) {
                double delta = Vec[j + 1].first - Vec[j].first;
                if (dcmp(delta) <= 0)
                    continue;
                double SIN = sin(delta / 2);
                Point W = Point(0 , 4 * c[i].r * SIN * SIN * SIN / (3 * (delta - sin(delta))));
                W = Rotate(W , (Vec[j + 1].first + Vec[j].first - pi) / 2) + c[i].O;
                double area = c[i].r * c[i].r * (delta - sin(delta));
                sx -= area * W.x;
                sy -= area * W.y;
                s -= area;

                Point A = c[i].point(Vec[j].first) , B = c[i].point(Vec[j + 1].first);
                area = (A ^ B);
                sx -= area * (A.x + B.x) / 3;
                sy -= area * (A.y + B.y) / 3;
                s -= area;
            }
        }
    }
}
```

4.7 平面划分

```
void work() {
    scanf("%d" , &n);
    for (int i = 0 ; i < n ; ++ i) {
        L[i].input();
        P[i] = L[i];
    }
    int m = n;
    for (int i = 0 ; i + 1 < n ; ++ i)
        for (int j = i + 1 ; j + 1 < n ; ++ j) {
            if (dcmp((P[i + 1] - P[i]) ^ (P[j + 1] - P[j])) != 0)
                P[m ++] = GetLineIntersection(P[i] , P[i + 1] - P[i] , P[j] , P[j + 1] - P[j]);
        }
    sort(P , P + m);
    m = unique(P , P + m) - P;
    memset(pre , -1 , sizeof(pre));
    set< pair<int , int> > Hash;
    for (int i = 0 ; i + 1 < n ; ++ i) {
        vector< pair <Point , int> > V;
        for (int j = 0 ; j < m ; ++ j)
            if (OnSegment(P[j] , L[i] , L[i + 1]))
                V.push_back(make_pair(P[j] , j));
        sort(V.begin() , V.end());
        for (int j = 0 ; j + 1 < V.size() ; ++ j) {
            int x = V[j].second , y = V[j + 1].second;
            if (!Hash.count(make_pair(x , y))) {
                Hash.insert(make_pair(x , y));
                e[mcnt] = (edge) {y , pre[x]} , pre[x] = mcnt ++;
            }
            if (!Hash.count(make_pair(y , x))) {
                Hash.insert(make_pair(y , x));
                e[mcnt] = (edge) {x , pre[y]} , pre[y] = mcnt ++;
            }
        }
    }
    for (int x = 0 ; x < m ; ++ x) {
        vector< pair<double , int> > V;
        for (int i = pre[x] ; ~i ; i = e[i].next) {
            int y = e[i].x;
            V.push_back(make_pair((P[y] - P[x]).arg() , i));
        }
        sort(V.begin() , V.end());
        for (int i = 0 ; i < V.size() ; ++ i) {
            int j = (i + 1) % V.size();
            Next[V[j].second ^ 1] = V[i].second;
        }
    }
    double res = 0;
    for (int i = 0 ; i < mcnt ; ++ i) {
        if (!vis[i]) {
            int x = i;
            double area = 0;
            while (!vis[x]) {
                vis[x] = 1;
                area += (P[e[x ^ 1].x] ^ P[e[x].x]);
                x = Next[x];
            }
            if (x == i && dcmp(area) > 0)
                res += area;
        }
    }
    printf("%.8f\n" , res / 2);
}
```

4.8 基础 3D 几何

```
const double eps = 1e—8 , pi = acos(—1.0);
inline int dcmp(double x) {
    return (x > eps) — (x < —eps);
}
struct Point {
    double x , y , z;
    Point () {x = y = z = 0;}
    Point (double _x , double _y , double _z) {
        x = _x , y = _y , z = _z;
    }
    void input() {
        scanf("%lf%lf%lf" , &x , &y , &z);
    }
    bool operator < (const Point &R) const {
        if (dcmp(x — R.x) != 0)
            return x < R.x;
        if (dcmp(y — R.y) != 0)
            return y < R.y;
        return z < R.z;
    }
    bool operator == (const Point &R) const {
        return dcmp(x — R.x) == 0 && dcmp(y — R.y) == 0 && dcmp(z — R.z) == 0;
    }
    Point operator + (const Point& R) const {
        return Point(x + R.x , y + R.y , z + R.z);
    }
    Point operator — (const Point& R) const {
        return Point(x — R.x , y — R.y , z — R.z);
    }
    Point operator * (const double& R) const {
        return Point(x * R , y * R , z * R);
    }
    Point operator / (const double& R) const {
        return Point(x / R , y / R , z / R);
    }
    double operator % (const Point& R) const {
        return x * R.x + y * R.y + z * R.z;
    }
    Point operator ^ (const Point& R) const {
        return Point(y * R.z — z * R.y , z * R.x — x * R.z , x * R.y — y * R.x);
    }
    inline double len() {
        return sqrt(*this % *this);
    }
};
Point GetLinePlaneProjection(Point A , Point P , Point n) {
    double t = (n % (P — A)) / (n % n);
    return A + n * t; // t * n.len() 是距离
} // 直线平面投影
Point GetLinePlaneIntersection(Point A , Point V , Point P , Point n) {
    double t = (n % (P — A)) / (n % V);
    return A + V * t;
} // 直线平面交点
inline double area(Point A , Point B , Point C) {
    return ((B — A) ^ (C — A)).len();
}
bool PointinTri(Point P) {
    double area1 = area(P , a[0] , a[1]);
    double area2 = area(P , a[1] , a[2]);
    double area3 = area(P , a[2] , a[0]);
    return dcmp(area1 + area2 + area3 — area(a[0] , a[1] , a[2])) == 0;
}
double GetLineIntersection(Point P , Point v , Point Q , Point w) {
    //共面时使用
    Point u = P — Q;
    Point delta = v ^ w , cross = w ^ u;
    if (dcmp(delta.z) != 0)
        return cross.z / delta.z;
    else if (dcmp(delta.y) != 0)
        return cross.y / delta.y;
    else if (dcmp(delta.x) != 0)
        return cross.x / delta.x;
    else {
        return 1e60;
    }
}

//a点绕Ob向量逆时针旋转弧度angle. cossin可预先计算
Point Rotate(Point a, Point b, double angle) {
    static Point e1 ,e2 , e3;
    b = b / b.len() , e3 = b;
    double lens = a % e3;
    e1 = a — e3 * lens;
    if (dcmp(e1.len()) > 0)
        e1 = e1 / e1.len();
    else
        return a;
    e2 = e1 ^ e3;
    double x1 = a % e2 , y1 = a % e1 , x2 , y2;
    x2 = x1 * cos(angle) — y1 * sin(angle);
    y2 = x1 * sin(angle) + y1 * cos(angle);
    return e3 * lens + e1 * y2 + e2 * x2;
}
/**
   绕任意轴（过原点）逆时针旋转（注意要把轴向量归一化，不然会在“点在轴上”这个情况下出问题）
   rotate x y z d
   | (1—cos(d))*x*x+cos(d)     (1—cos(d))*x*y+sin(d)*z     (1—cos(d))*x*z—sin(d)*y   0 |
   | (1—cos(d))*y*x—sin(d)*z   (1—cos(d))*y*y+cos(d)       (1—cos(d))*y*z+sin(d)*x   0 |
   | (1—cos(d))*z*x+sin(d)*y   (1—cos(d))*z*y—sin(d)*x     (1—cos(d))*z*z+cos(d)     0 |
   |          0                          0                          0                1 |
**/
```

4.9 凸包 3D

```
double mix(const Point &a, const Point &b, const Point &c) {
    return a % (b ^ c);
}
const int N = 305;
int mark[N][N];
Point info[N];
int n , cnt;

double area(int a, int b, int c) {
    return ((info[b] — info[a]) ^ (info[c] — info[a])).len();
}
double volume(int a, int b, int c, int d) {
    return mix(info[b] — info[a], info[c] — info[a], info[d] — info[a]);
}
struct Face {
    int v[3];
    Face() {}
    Face(int a, int b, int c) {
        v[0] = a , v[1] = b , v[2] = c;
    }
    int& operator [] (int k) {
        return v[k];
    }
};
vector <Face> face;
inline void insert(int a, int b, int c) {
    face.push_back(Face(a, b, c));
}
void add(int v) {
    vector <Face> tmp;
    int a, b, c;
    cnt ++;
    for (int i = 0; i < face.size() ; ++ i) {
        a = face[i][0] , b = face[i][1] , c = face[i][2];
        if (dcmp(volume(v, a, b, c)) < 0)
            mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c][a] = mark[a][c] = cnt;
        else
            tmp.push_back(face[i]);
    }
    face = tmp;
    for (int i = 0; i < tmp.size() ; ++ i) {
        a = face[i][0] , b = face[i][1] , c = face[i][2];
        if (mark[a][b] == cnt) insert(b, a, v);
        if (mark[b][c] == cnt) insert(c, b, v);
        if (mark[c][a] == cnt) insert(a, c, v);
    }
}
int Find() {
    for (int i = 2; i < n; ++ i) {
        Point ndir = (info[0] - info[i]) ^ (info[1] - info[i]);
        if (ndir == Point())
            continue;
        swap(info[i], info[2]);
        for (int j = i + 1; j < n; j++)
            if (dcmp(volume(0, 1, 2, j)) != 0) {
                swap(info[j], info[3]);
                insert(0, 1, 2);
                insert(0, 2, 1);
                return 1;
            }
    }
    return 0;
}
void work() {
    for (int i = 0; i < n; ++ i)
        info[i].input();
    sort(info, info + n);
    n = unique(info, info + n) - info;
    face.clear();
    random_shuffle(info, info + n);
    if (Find()) {
        memset(mark, 0, sizeof(mark));
        cnt = 0;
        for (int i = 3; i < n; ++ i) add(i);
        vector<Point> Ndir;
        for (int i = 0; i < face.size() ; ++i) {
            Point p = (info[face[i][0]] - info[face[i][1]]) ^ (info[face[i][2]] - info[face[i][1]]);
            p = p / p.len();
            Ndir.push_back(p);
        }
        sort(Ndir.begin(), Ndir.end());
        int ans = unique(Ndir.begin(), Ndir.end()) - Ndir.begin();
        printf("%d\n", ans);
    } else {
        printf("1\n");
    }
}
```

5 String

5.1 Ext-KMP

```
f[0] = m;
for (int i = 1 , k = 0 ; i < m ; ++ i) {
    int len = k + f[k], l = f[i - k];
    if (i > 1 && l < len - i)
        f[i] = l;
    else {
        int j = i > 1 ? max(0 , len - i) : 0;
        while(i + j < m && t[j] == t[i + j])
            ++ j;
        f[i] = j , k = i;
    }
}
for (int i = 0 , k = 0 ; i < n ; ++ i) {
    int len = k + L[k] , l = f[i - k];
    if (i > 0 && l < len - i)
        L[i] = l;
    else {
        int j = i > 0 ? max(0 , len - i) : 0;
        while (j < m && t[j] == s[(i + j) % n])
            ++ j;
        L[i] = j , k = i;
    }
}
```

5.2 manacher

```
s[0] = '%';
for (int i = 0 ; str[i] ; ++ i) {
    s[len ++] = '#';
    s[len ++] = str[i];
    s[len ++] = '#';
}s[len] = 0;
int id = 0 , mx = 0;
for (int i = 0 ; i < len ; ++ i) {
    p[i] = mx > i ? min(p[id + id - i], mx - i) : 1;
    while (s[i + p[i]] == s[i - p[i]]) ++ p[i];
    if (i + p[i] > mx)
        mx = i + p[i] , id = i;
}
```

5.3 最小表示法

```
int MinR(char *str) {
    int i = 0 , j = 1 , k = 0 , len = strlen(str);
    while (i < len && j < len && k < len) {
        int cmp = str[(j + k) % len] - str[(i + k) % len];
        if (!cmp)
            ++ k;
        else {
            if (cmp > 0)
                j += k + 1;
            else i += k + 1;
            if (i == j) ++ j;
            k = 0;
        }
    }
    return min(i , j);
}
```

5.4 后缀数组

```
int sa[N] , t1[N] , t2[N] , c[N];
int Rank[N] , height[N];
void buildsa(char *s , int n , int m) {
    int i , k , p , a1 , a2 , *x = t1 , *y = t2;
    memset(c , 0 , m << 2);
    for (i = 0 ; i < n ; ++ i) ++ c[x[i] = s[i]];
    for (i = 1 ; i < m ; ++ i) c[i] += c[i - 1];
    for (i = n - 1 ; i >= 0 ; -- i) sa[-- c[x[i]]] = i;
    for (k = 1 , p = 0; k < n ; k <<= 1 , p = 0) {
        for (i = n - k ; i < n ; ++ i) y[p ++] = i;
        for (i = 0 ; i < n ; ++ i) if (sa[i] >= k) y[p ++] = sa[i] - k;
        memset(c , 0 , m << 2);
        for (i = 0 ; i < n ; ++ i) ++ c[x[y[i]]];
        for (i = 1 ; i < m ; ++ i) c[i] += c[i - 1];
        for (i = n - 1 ; i >= 0 ; -- i) sa[-- c[x[y[i]]]] = y[i];
        swap(x , y) , p = 1 , x[sa[0]] = 0;
        for (i = 1 ; i < n ; ++ i) {
            a1 = sa[i - 1] + k < n ? y[sa[i - 1] + k] : -1;
            a2 = sa[i] + k < n ? y[sa[i] + k] : -1;
            x[sa[i]] = (y[sa[i - 1]] == y[sa[i]] && a1 == a2) ? p - 1 : p ++;
        }
        if (p >= n) break; m = p;
    }
    for (i = 0 ; i < n ; ++ i) Rank[sa[i]] = i;
    for (i = 0 , k = 0; i < n ; ++ i) {
        if (k) -- k; if (!Rank[i]) continue;
        int j = sa[Rank[i] - 1];
        while (s[i + k] == s[j + k]) ++ k;
        height[Rank[i]] = k;
    }
}
/******Suffix array for Trie *****/
int f[18][N] , s[N] , dep[N];
int sa[N] , t1[N] , t2[N] , c[N];
int Rank[18][N] , rnk[N];
inline int LCP(int x , int y) {
    int len = 0;
    for (int i = 17 ; i >= 0 ; -- i)
        if (Rank[i][x] && Rank[i][x] == Rank[i][y]) {
            len += 1 << i;
            x = f[i][x];
            y = f[i][y];
        }
    return len;
}
void buildsa(int *s , int n , int m) {
    int i , j , k , p , a1 , a2;
    int *x = t1 , *y = t2;
    memset(c , 0 , m + 1 << 2);
    for (i = 1 ; i <= n ; ++ i) ++ c[x[i] = s[i]];
    for (i = 1 ; i <= m ; ++ i) c[i] += c[i - 1];
    for (i = n ; i >= 1 ; -- i) sa[c[x[i]] --] = i;
    for (k = 1 , j = 0 , p = 0; k < n ; k <<= 1 , ++ j) {
        memset(c , 0 , m + 1 << 2);
        for (i = 1 ; i <= n ; ++ i) Rank[j][i] = x[i];
        for (i = 1 ; i <= n ; ++ i) ++ c[x[f[j][i]]];
        for (i = 1 ; i <= m ; ++ i) c[i] += c[i - 1];
        for (i = n ; i >= 1 ; -- i) y[c[x[f[j][i]]] --] = i;
        memset(c , 0 , m + 1 << 2);
        for (i = 1 ; i <= n ; ++ i) ++ c[x[y[i]]];
        for (i = 1 ; i <= m ; ++ i) c[i] += c[i - 1];
        for (i = n ; i >= 1 ; -- i) sa[c[x[y[i]]] --] = y[i];
        swap(x , y) , p = 1 , x[sa[1]] = 1;
        for (i = 2 ; i <= n ; ++ i) {
            a1 = y[f[j][sa[i - 1]]];
            a2 = y[f[j][sa[i]]];
            x[sa[i]] = (y[sa[i - 1]] == y[sa[i]] && a1 == a2) ? p : ++ p;
        }
        m = p;
    }
    for (i = 1 ; i <= n ; ++ i) rnk[sa[i]] = i;
}
```

5.5 后缀自动机

```
int root , last , nodecnt;
int u[N << 1][26] , val[N << 1] , f[N << 1];
inline int newnode(int _val) {
    ++ nodecnt;
    memset(u[nodecnt] , 0 , sizeof(u[nodecnt]));
    val[nodecnt] = _val , f[nodecnt] = 0;
    return nodecnt;
}
void extend(int c) {
    int p = last , np = newnode(val[p] + 1);
    while (p && u[p][c] == 0)
        u[p][c] = np , p = f[p];
    if (p == 0)
        f[np] = root;
    else {
        int q = u[p][c];
        if (val[p] + 1 == val[q]) {
            f[np] = q;
        } else {
            int nq = newnode(val[p] + 1);
            memcpy(u[nq] , u[q] , sizeof(u[q]));
            f[nq] = f[q];
            f[q] = f[np] = nq;
            while (p && u[p][c] == q)
                u[p][c] = nq , p = f[p];
        }
    }
    last = np;
}
void work() {
    nodecnt = 0;
    root = last = newnode(0);
}
```

5.6 后缀树

```
const int INF = 1000000000, C = 26, N = 100005;
int pos;
int text[N];
struct Node {
    int l, r;
    Node *suf, *ch[C];
    int dgr;
    Node *fa;

    Node (int l = -1, int r = INF) : l(l), r(r) {
        suf = fa = NULL;
        memset(ch, 0, sizeof(ch));
        dgr = 0;
    }
    Node* addEdge(Node *t) {
        int c = text[t->l];
        dgr += !ch[c];
        ch[c] = t;
        t->fa = this;
        return t;
    }
    int len() {
        return min(r, pos + 1) - l;
    }
};

int top;
Node pool[N << 1];
Node *root, *nxtSuf, *cur;
int remCnt, curP, curLen;
long long size;
queue<Node*> leaves;
void init() {
    top = 0, pos = -1;
    remCnt = 0, curP = 0, curLen = 0;
    nxtSuf = NULL;
    root = cur = new(pool + (top++)) Node(-1, -1);
    size = 0;
    while (leaves.size()) {
        leaves.pop();
    }
}
void link(Node *u) {
    if (nxtSuf) {
        nxtSuf->suf = u;
    }
    nxtSuf = u;
}
bool walk(Node *u) {
    int len = u->len();
    if (curLen >= len) {
        curP += len;
        curLen -= len;
        cur = u;
        return true;
    }
    return false;
}
void extend(int c) {
    text[++pos] = c;
    nxtSuf = NULL;
    ++remCnt;
    while (remCnt) {
        curP = curLen ? curP : pos;
        int curE = text[curP];
        if (!cur->ch[curE]) {
            leaves.push(cur->addEdge(new(pool + (top++)) Node(pos)));
            link(cur);
        } else {
            Node *nxt = cur->ch[curE];
            if (walk(nxt)) {
                continue;
            }
            if (text[nxt->l + curLen] == c) {
                ++curLen;
                link(cur);
                break;
            }
            Node *split = new(pool + (top++)) Node(nxt->l, nxt->l + curLen);
            cur->addEdge(split);
            leaves.push(split->addEdge(new(pool + (top++)) Node(pos)));
            nxt->l += curLen;
            split->addEdge(nxt);
            link(split);
        }
        --remCnt;
        if (cur == root && curLen > 0) {
            curP = pos - (--curLen);
        } else {
            cur = cur->suf ? cur->suf : root;
        }
    }
    size += leaves.size();
}
void finish() {
    nxtSuf = NULL;
    for (int i = 0; i < top; ++i) {
        if (pool[i].r == INF) {
            link(pool + i);
        }
    }
    while (remCnt > 0) {
        if (curLen) {
            int curE = text[curP];
            Node *nxt = cur->ch[curE];
            if (walk(nxt)) {
                continue;
            }
            Node *split = new(pool + (top++)) Node(nxt->l, nxt->l + curLen);
            leaves.push(cur->addEdge(split));
            nxt->l += curLen;
            split->addEdge(nxt);
            link(split);
        } else {
            leaves.push(cur);
            link(cur);
        }
        --remCnt;
        if (cur == root && curLen > 0) {
            --curLen;
            curP = pos - remCnt + 1;
        } else {
            cur = cur->suf ? cur->suf : root;
        }
    }
    if (nxtSuf != root) {
        link(root);
    }
}
void eraseUp(Node *&u) {
    size -= u->len();
    int ch = text[u->l];
    u = u->fa;
    u->ch[ch] = NULL;
    --(u->dgr);
}
void erase() {
    Node *u = leaves.front();
    leaves.pop();
    while (u->dgr == 0 && u != cur) {
        eraseUp(u);
    }
    if (u == cur) {
        if (cur->dgr == 0 && curLen == 0) {
            int len = u->len();
            curLen = len;
            curP = pos - len + 1;
            cur = cur->fa;
            eraseUp(u);
        }
        if (curLen) {
            int curE = text[curP];
            if (!cur->ch[curE]) {
                Node *leaf = new(pool + (top++)) Node(pos - curLen + 1);
                leaves.push(cur->addEdge(leaf));
                size += leaf->len();
                --remCnt;
                if (cur == root && curLen > 0) {
                    curP = pos - (--curLen) + 1;
                } else {
                    cur = cur->suf ? cur->suf : root;
                }
                while (curLen && walk(cur->ch[text[curP]])) {
                    continue;
                }
            }
        }
    }
}
int n;
char s[N], buf[N];
int ord[N], stop, sord[N << 1];
void dfs(Node *u) {
    sord[u - pool] = stop++;
    for (int i = 0; i < C; ++i) {
        if (u->ch[i]) {
            dfs(u->ch[i]);
        }
    }
}
void getOrd() {
    init();
    for (int i = 0; i < n; ++i) {
        extend(s[i] - 'a');
    }
    finish();
    stop = 0;
    dfs(root);
    int i = 0;
    while (leaves.size()) {
        ord[i++] = sord[leaves.front() - pool];
        leaves.pop();
    }
}
long long res[N];
int main() {
    while (scanf("%s", s) == 1) {
        n = strlen(s);
        getOrd();
        int q , l;
        scanf("%d%d", &q , &l);
        long long ans = 0;
        int pos = 0;
        init();
        for (int i = 0; i < n; ++i) {
            extend(s[i] - 'a');
            if (i >= l) {
                erase();
            }
            if (i >= l - 1) {
                res[i - l + 1] = size;
                if (size > ans || (size == ans && ord[i - l + 1] < ord[pos])) {
                    ans = size;
                    pos = i - l + 1;
                }
            }
        }
        while (q --) {
            int x;
            scanf("%d" ,&x);
            printf("%lld\n" , res[-- x]);
        }

    }
    return 0;
}
```

5.7 回文树

```
struct PalinTree {
    char str[N];
    int n;
    int u[N][26];
    int len[N] , f[N] , cnt[N];
    int nodecnt , root;
    void init() {
        scanf("%s" , str);
        n = strlen(str);
        nodecnt = 2;
        len[1] = -1 , len[2] = 0;
        f[1] = 0 , f[2] = 1;
        memset(u[1] , 0 , sizeof(u[1]));
        memset(u[2] , 0 , sizeof(u[2]));
        root = 1;
        for (int i = 0 ; i < n ; ++ i)
            extend(i , str[i] - 'a');
    }
    void extend(int i , int c) {
        int p = root;
        while (str[i - 1 - len[p]] != str[i])
            p = f[p];
        int& pp = u[p][c];
        if (!pp) {
            pp = ++ nodecnt;
            len[pp] = len[p] + 2;
            cnt[pp] = 0;
            memset(u[pp] , 0 , sizeof(u[pp]));
            int q = f[p];
            while (q && str[i - 1 - len[q]] != str[i])
                q = f[q];
            f[pp] = q ? u[q][c] : 2;
        }
        ++ cnt[pp];
        root = pp;
    }
}
```

6 Other

6.1 emacs

```
(add-hook 'c++-mode-hook 'linux-cpp-mode)
(defun linux-cpp-mode()
  (define-key c++-mode-map [return] 'newline-and-indent)
  (interactive)
  (c-set-style "K&R")
  (c-toggle-auto-state)
  (setq c-basic-offset 4)
  (show-paren-mode t)
  (setq show-paren-style 'parentheses)
)
(setq-default indent-tabs-mode  nil)
(global-linum-mode t)
```

6.2 Dancing Links

```
int U[M] , D[M] , L[M] , R[M], col[M] , row[M];
int cnt , p[N] , s[N];

#define FOR(i,A,s) for (int i = A[s]; i != s ; i = A[i])
//由矩阵建立十字链表
cnt = n + 1;
for (i = 0 ; i <= n ;i ++)
    L[i] = i - 1 , R[i] = i + 1;
memset(s , 0 , sizeof(s));
L[0] = n , R[n] = 0;
for (i = 1 ; i <= n ; i ++)
    p[i] = i;
for (i = 1 ; i <= n ; ++ i) {
    x = y = -1;
    for (j = 1 ; j <= n ;j ++)
        if (g[i][j]) {
            if (x == -1)
                x = cnt , y = cnt;
            else
                L[cnt] = y , R[y] = cnt , y = cnt;
            D[p[j]] = cnt , U[cnt] = p[j]  , p[j] = cnt;
            col[cnt] = j , row[cnt] = i, s[j] ++;
            ++ cnt;
        }
    L[x] = y , R[y] = x;
}
for (i = 1 ; i <= n ;i ++)
    D[p[i]] = i , U[i] = p[i];
//可重复覆盖：
void remove(int c) {
    FOR(i,D,c) L[R[i]] = L[i] , R[L[i]] = R[i];
}
void resume(int c) {
    FOR(i,U,c) L[R[i]] = R[L[i]] = i;
}
int H() {
    int val = 0; bool u[N] = {0};
    FOR(i,R,0) if (!u[i]) {
        ++ val;
        FOR (j,D,i) FOR(k,R,j)
            u[col[k]] = 1;
    }
    return val;
}
bool dfs(int d) {
    if (d + H() > K)return 0 ;
    if (R[0] == 0) {
        return 1;
    }
    int c = R[0];
    FOR(i,R,0) if (s[i] < s[c]) c = i;
    FOR(i,D,c) {
        remove(i);
        FOR(j,R,i) remove(j);
        if (dfs(d + 1))
            return 1;
        FOR(j,L,i) resume(j);
        resume(i);
    }
    return 0;
}
//精确覆盖
void remove(int c) {
    L[R[c]] = L[c] , R[L[c]] = R[c];
    FOR(i,D,c) FOR(j,R,i)
        U[D[j]] = U[j] , D[U[j]] = D[j] , -- s[col[j]];
}
void resume(int c) {
    FOR(i,U,c) FOR(j,L,i)
        U[D[j]] = D[U[j]] = j , ++ s[col[j]];
    L[R[c]] = R[L[c]] = c;
}

bool dfs(int d) {
    if (R[0] == 0)
        return 1;
    int c = R[0];
    FOR(i,R,0) if (s[i] < s[c]) c = i;
    remove(c);
    FOR(i,D,c) {
        ans.pb(row[i]);
        FOR(j,R,i) remove(col[j]);
        if (dfs(d + 1)) return 1;
        FOR(j,L,i) resume(col[j]);
        ans.pop_back();
    }
    resume(c);
    return 0;
}
```

6.3 unorderedmap

```
template<typename T1 , typename T2> struct hashmap {
    const static int MOD = 99991;
    const static int Size = 500005;
    int pre[MOD] , mcnt;
    struct node {
        T1 key;
        T2 val ;
        int next;
    } e[Size];
    void clear() {
        memset(pre , -1 , sizeof(pre));
        mcnt = 0;
    }
    void insert(const T1& K , const T2& V) {
        int x = K % MOD;
        e[mcnt] = (node) {K , V , pre[x]};
        pre[x] = mcnt ++;
    }
    int find(const T1 &K) {
        int x = K % MOD;
        for (int i = pre[x] ; ~i ; i = e[i].next)
            if (e[i].key == K)
                return i;
        return -1;
    }
    T2& operator [] (const T1 &x){
        int i = find(x);
        if (!~i){
            insert(x , 0);
            return e[mcnt - 1].val;
        }
        return e[i].val;
    }
};
struct hash_func {
    size_t operator() (const Matrix &p) const {
        return p[0][0] * 2333333 ^ p[0][1] * (145777 ^ p[1][1]);
    }
};
unordered_map<Matrix , int , hash_func>
```

6.4 插头 DP

```
inline int getpos(int x , int k) {
  return x >> k + k & 3;
}
inline int setpos(int x , int k , int v) {
  return (x & ~(3 << k + k)) | (v << k + k);
}

void work() {
  int res = -1 << 30;
  for (int i = 0 ; i < n ; ++ i)
    for (int j = 0 ; j < m ; ++ j) {
      scanf("%d" , &a[i][j]);
      res = max(res , a[i][j]);
    }
  int cur = 0 , nxt = 1;
  f[cur].clear();
  f[cur][0] = 0;
  for (int i = 0 ; i < n ; ++ i) {
    for (int j = 0 ; j < m ; ++ j) {
      f[nxt].clear();

      for (int it = 0 ; it < f[cur].mcnt ; ++ it) {
        int k = f[cur].e[it].key;
        int w = f[cur].e[it].val;
        int L = getpos(k , j);
        int U = getpos(k , j + 1);
        int num = getpos(k , m + 1);
        //printf("%d %d %d %d : %d\n" , i , j , k , num , w);
        if (!L && !U)
          f[nxt][k] = max(f[nxt][k] , w);
        w += a[i][j];
        if (!L && !U) {
          if (j + 1 < m) {
            int K = setpos(k , j , 1);
            K = setpos(K , j + 1 , 2);
            f[nxt][K] = max(f[nxt][K] , w);
          }
          if (num < 2) {
            int K = setpos(k , m + 1 , num + 1);
            K = setpos(K , j , 3);
            f[nxt][K] = max(f[nxt][K] , w);
            if (j + 1 < m) {
              int K = setpos(k , m + 1 , num + 1);
              K = setpos(K , j + 1 , 3);
              f[nxt][K] = max(f[nxt][K] , w);
            }
          }
        } else {
          static int match[N];
          static int S[N];
          int top = 0;
          for (int l = 0 ; l <= m ; ++ l) {
            int x = getpos(k , l);
            if (x == 1)
              S[top ++] = l;
            if (x == 2) {
              -- top;
              match[S[top]] = l;
              match[l] = S[top];
            }
          }
          if (L && !U) {
            if (num < 2) {
              int K = setpos(k , m + 1 , num + 1);
              K = setpos(K , j , 0);
              if (L != 3) K = setpos(K , match[j] , 3);
              f[nxt][K] = max(f[nxt][K] , w);
            }
            if (j + 1 < m) {
              int K = setpos(k , j , U);
              K = setpos(K , j + 1 , L);
              f[nxt][K] = max(f[nxt][K] , w);
            }
            f[nxt][k] = max(f[nxt][k] , w);
          }
          if (!L && U) {
            if (num < 2) {
              int K = setpos(k , m + 1 , num + 1);
              K = setpos(K , j + 1 , 0);
              if (U != 3) K = setpos(K , match[j + 1] , 3);
              f[nxt][K] = max(f[nxt][K] , w);
            }
            if (j + 1 < m) {
              f[nxt][k] = max(f[nxt][k] , w);
            }
            int K = setpos(k , j , U);
            K = setpos(K , j + 1 , L);
            f[nxt][K] = max(f[nxt][K] , w);
          }
          if (L && U) {
            int K = setpos(k , j , 0);
            K = setpos(K , j + 1 , 0);
            if (L == 3 || U == 3) {
              if (L != 3)
                K = setpos(K , match[j] , 3);
              if (U != 3)
                K = setpos(K , match[j + 1] , 3);
              f[nxt][K] = max(f[nxt][K] , w);
            } else {
              if (L == U) {
                if (L == 1 && U == 1) {
                  K = setpos(K , match[j + 1] , 1);
                }
                if (L == 2 && U == 2) {
                  K = setpos(K , match[j] , 2);
                }
                f[nxt][K] = max(f[nxt][K] , w);
              } else if (L == 2 && U == 1) {
                f[nxt][K] = max(f[nxt][K] , w);
              }
            }
          }
        }
      }
      swap(cur , nxt);
    }
    f[nxt].clear();
    for (int it = 0 ; it < f[cur].mcnt ; ++ it) {
      int k = f[cur].e[it].key;
      int w = f[cur].e[it].val;
      int num = getpos(k , m + 1);
      k = setpos(k , m + 1 , 0) << 2;
      f[nxt][setpos(k , m + 1 , num)] = w;
    }
    swap(cur , nxt);
  }
  res = max(res , f[cur][2 << m + m + 2]);
  cout << res << endl;
}
```

6.5 压位 LCS

```
typedef unsigned long long LL;
const int N = 1005;
const int B = 64;
const int M = (N + B - 1) / B + 5;
int n , m;
char s[N] , t[N];
LL c[26][M];
LL f[2][M] , X[M];

void work() {
    n = strlen(s);
    m = strlen(t);
    memset(c , 0 , sizeof(c));
    for (int i = 0 ; i < n ; ++ i)
        c[s[i] - 'a'][i >> 6] |= 1ULL << (i & 63);
    int L = (n + B - 1) / B;
    int cur = 0 , nxt = 1;
    memset(f , 0 , sizeof(f));
    for (int i = 0 ; i < m ; ++ i) {
        int id = t[i] - 'a';
        for (int j = 0 ; j < L ; ++ j)
            X[j] = f[cur][j] | c[id][j];
        for (int j = 0 , x = 1 ; j < L ; ++ j) {
            int y = f[cur][j] >> 63 & 1;
            f[cur][j] <<= 1 , f[cur][j] |= x;
            x = y;
        }
        memcpy(f[nxt] , X , sizeof(X));
        for (int j = 0 , x = 0; j < L ; ++ j) {
            if (f[nxt][j] < x + f[cur][j]) {
                f[nxt][j] -= x + f[cur][j];
                x = 1;
            } else {
                f[nxt][j] -= x + f[cur][j];
                x = 0;
            }
            f[nxt][j] ^= X[j];
            f[nxt][j] &= X[j];
        }
        swap(cur , nxt);
    }
    int ans = 0;
    for (int i = 0 ; i < n ; ++ i)
        if (f[cur][i >> 6] >> (i & 63) & 1)
            ++ ans;
    printf("%d\n" , ans);
}
```

6.6 bitset 区间询问

```
int u[N] , v[N];
struct Range {
    int key[N] , val[N];
    bitset<N> w[N / B + 1][N / B + 1];
    void init() { // u有序，B可以开大一点
        int m = (n + B - 1) / B;
        for (int i = 0 ; i < n ; ++ i) {
            key[i] = u[i];
            val[i] = v[i];
        }
        for (int i = 0 ; i < m ; ++ i) {
            int L = i * B , R = min(n , L + B);
            for (int j = L ; j < R ; ++ j) {
                w[i][i].set(v[j]);
            }
        }
        for (int i = 0 ; i < m ; ++ i) {
            for (int j = i + 1 ; j < m ; ++ j) {
                w[i][j] = w[i][j - 1] | w[j][j];
            }
        }
    } // 提取key在一个区间内的val构成的bitset
    bitset<N> get(int l , int r) {
        l = lower_bound(key , key + n , l) - key;
        r = upper_bound(key , key + n , r) - key;
        int ll = l / B , rr = r / B;
        bitset<N> ret;
        if (ll == rr) {
            for (int i = l ; i < r ; ++ i)
                ret.set(val[i]);
        } else {
            ret |= w[ll + 1][rr - 1];
            int R = (ll + 1) * B , L = rr * B;
            for (int i = l ; i < R ; ++ i)
                ret.set(val[i]);
            for (int i = L ; i < r ; ++ i)
                ret.set(val[i]);
        }
        return ret;
    }
};
```

6.7 转转转

```
bool cmp(const pair<Point , int> &AA , const pair<Point , int> &BB) {
    const Point &A = AA.first;
    const Point &B = BB.first;
    if (A.sign() != B.sign())
        return A.sign() < B.sign();
    return (A ^ B) > 0;
}
int n , D , id[N][2];
Point a[N][2] , d[N];
int f[N];
vector<int> add[N] , del[N];
int getf(int x) {
    return f[x] == x ? x : f[x] = getf(f[x]);
}
Point P , V;
double len;
inline double distance(int i) {
    Point u = P - a[i][0] , w = a[i][1] - a[i][0];
    return len * (w ^ u) / (V ^ w);
}
struct segment {
    bool operator () (const int &x , const int &y) {
        double dx = distance(x);
        double dy = distance(y);
        return dx < dy;
        /*if (fabs(dx - dy) > 1e-6)
            return dx < dy;
            return x < y;*/
    }
};
int main() {
    scanf("%d" , &n);
    for (int i = 0 ; i < n ; ++ i) {
        for (int j = 0 ; j < 2 ; ++ j) {
            scanf("%d%d" , &a[i][j].x , &a[i][j].y);
            d[D ++] = a[i][j];
        }
    }
    sort(d , d + D);
    D = unique(d , d + D) - d;

    for (int i = 0 ; i < n ; ++ i) {
        int x = lower_bound(d , d + D , a[i][0]) - d;
        int y = lower_bound(d , d + D , a[i][1]) - d;
        id[i][0] = x , id[i][1] = y;
        E.push_back(make_pair(0 , make_pair(x , y)));
    }

    for (int i = 0 ; i < D ; ++ i) {
        vector< pair<Point , int> > Vec;
        for (int j = 0 ; j < D ; ++ j) {
            if (i != j)
                Vec.push_back(make_pair(d[j] - d[i] , j));
        }
        for (int j = 0 ; j < D ; ++ j) {
            add[j].clear();
            del[j].clear();
        }
        set<int , segment> Hash;
        sort(Vec.begin() , Vec.end() , cmp);

        P = d[i];
        V = Point(1 , 0);
        len = 1;
        for (int j = 0 ; j < n ; ++ j) {
            int &x = id[j][0] , &y = id[j][1];
            if (x == i || y == i)
                continue;
            if (((d[x] - d[i]) ^ (d[y] - d[i])) < 0) {
                swap(x , y);
                swap(a[j][0] , a[j][1]);
            }
            if (d[y].y >= d[i].y && d[x].y < d[i].y) {
                Hash.insert(j);
                add[x].push_back(j);
                del[y].push_back(j);
            } else {
                add[x].push_back(j);
                del[y].push_back(j);
            }
        }

        for (int j = 0 ; j < (int)Vec.size() ; ++ j) {
            V = Vec[j].first;
            len = sqrt(V.len());
            int x = Vec[j].second;
            for (auto &k : del[x])
                Hash.erase(k);
            bool flag = 0;
            if (!Hash.empty()) {
                int k = *Hash.begin();
                if (((a[k][1] - a[k][0]) ^ (d[x] - a[k][0])) >= 0)
                    flag = 1;
            } else {
                flag = 1;
            }
            if (flag) {
                //E.push_back(make_pair((d[i] - d[x]).len() , make_pair(i , x)));
            }
            for (auto &k : add[x])
                Hash.insert(k);
        }
    }
    return 0;
}
```

6.8 Time-travel

```
int del[N];
void divide(const vector<int> &A) {
    if (A.size() <= 1)
        return;
    // 负数询问，否则偶数插入奇数删除
    vector<int> P , Q;
    int r = A.size() , mid = r / 2;
    for (int i = 0 ; i < mid ; ++ i)
        P.push_back(A[i]);
    divide(P);
    P.clear();
    for (int i = 0 ; i < r ; ++ i)
        if (A[i] > 0 && (A[i] & 1))
            del[A[i] >> 1] = 1;
    for (int i = 0 ; i < mid ; ++ i)
        if (A[i] > 0 && (~A[i] & 1)) {
            if (!del[A[i] >> 1])
                P.push_back(A[i] >> 1);
            else
                del[A[i] >> 1] = 2;
        }
    for (int i = mid ; i < r ; ++ i)
        if (A[i] < 0)
            Q.push_back(-A[i]);
    update(P , Q);
    Q.clear();
    int c1 = 0 , c2 = 0;
    for (int i = r - 1 ; i >= mid ; -- i) {
        if (A[i] > 0 && (A[i] & 1) && del[A[i] >> 1] == 2) {
            Q.push_back(A[i] ^ 1);
            ++ c1;
        } else if (A[i] < 0) {
            Q.push_back(A[i]);
            ++ c2;
        }
    }
    for (int i = 0 ; i < r ; ++ i)
        if (A[i] > 0 && (A[i] & 1))
            del[A[i] >> 1] = 0;
    if (c1 && c2)
        divide(Q);
    P.clear();
    for (int i = mid ; i < r ; ++ i)
        P.push_back(A[i]);
    divide(P);
}
```