

Contents

1 Basic	1
1.1 vimrc	1
1.2 IncreaseStackSize	1
1.3 Default Code	1
2 Data Structure	2
2.1 extc_heap	2
2.2 extc_balance_tree	2
2.3 Heavy Light Decomposition	2
2.4 Disjoint Set	2
2.5 Treap	2
2.6 Persistent Treap	2
2.7 Persistent Segment Tree	3
3 Graph	3
3.1 BCC Edge	3
3.2 BCC Vertex	4
3.3 Strongly Connected Components	4
3.4 Heavy Light Decomposition	4
3.5 Maximum Clique	5
3.6 MinimumMeanCycle	5
4 Flow	5
4.1 Dinic	5
4.2 Cost Flow	6
4.3 Kuhn Munkres	6
4.4 Maximum Simple Graph Matching	6
4.5 Minimum Weight Matching (Clique version)	7
5 Math	8
5.1 Bigint	8
5.2 ax+by=gcd	9
5.3 Linear Prime Sieve	9
5.4 Bsgs	9
5.5 Chinese Remainder	9
5.6 Fast Fourier Transform	9
5.7 Kth Residue	10
5.8 Gauss Elimination	11
5.9 Matrix	11
5.10 NTT	11
5.11 NTT(eddy ver.)	12
5.12 Miller Rabin	12
5.13 Pollard Rho	12
5.14 Algorithms about Primes	12
5.15 Primitive Root	13
5.16 Pseudoinverse of Square matrix	13
5.17 Theorem	13
5.17.1 Lucas' Theorem	13
5.17.2 Sum of Two Squares Thm (Legendre)	13
5.17.3 Difference of D1-D3 Thm	14
5.17.4 Krush-Kuhn-Tucker Conditions	14
5.17.5 Chinese remainder theorem	14
5.17.6 Stirling Numbers(permutation $ P = n$ with k cycles)	14
5.17.7 Stirling Numbers(Partition n elements into k non-empty set)	14
5.17.8 Pick' s Theorem	14
5.17.9 Kirchhoff's theorem	14
6 Geometry	14
6.1 Point operators	14
6.2 Intersection of two circles	14
6.3 Intersection of two lines	14
6.4 Intersection of two segments	14
6.5 Intersection of circle and line	14
6.6 Tangent line of two circles	15
6.7 Circle cover	15
6.8 Half plane intersection	15
6.9 Poly union area	16
6.10 2D Convex hull	16
6.11 Convex hull trick	16
6.12 KDTree (Nearest Point)	17
6.13 Triangle	17
7 Stringology	18
7.1 Suffix Array	18
7.2 Suffix Array (SAIS TWT514)	18
7.3 Aho-Corasick Algorithm	18
7.4 KMP	19
7.5 Z value	19
7.6 Z value (palindrome ver.)	19
7.7 Lexicographically Smallest Rotation	19
7.8 Suffix Automaton	19

1 Basic

1.1 vimrc

```
colo ron
syn on
se ai ar nu rnu
se mouse=a bs=2 ts=4 sw=4 ttm=100
au BufNewFile *.cpp 0r ~/default.cpp | :1,$-7 fo
filetype indent on
```

1.2 IncreaseStackSize

```
//stack resize
asm( "mov %0,%esp\n" ::"g"(mem+10000000) );
//change esp to rsp if 64-bit system

//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
    const rlim_t ks = 64*1024*1024;
    struct rlimit rl;
    int res=getrlimit(RLIMIT_STACK, &rl);
    if(res==0){
        if(rl.rlim_cur<ks){
            rl.rlim_cur=ks;
            res=setrlimit(RLIMIT_STACK, &rl);
        }
    }
}
```

1.3 Default Code

```
// #pragma GCC optimize ("-O2")
// #pragma GCC optimize ("unroll-loops")
#include<bits/stdc++.h>
using namespace std;
#define F first
#define S second
#define pb push_back
#define MP make_pair
#define ALL(x) begin(x),end(x)
#define SZ(x) ((int)(x).size())
typedef long long LL;
typedef double DB;
typedef long double LDB;
typedef pair<int, int> PII;
typedef pair<LL, LL> PLL;
// #define IOS ios_base::sync_with_stdio(0); cin.tie(0)
const int MXN = (int)1e6 + 7;
int main(){
    return 0;
}
```

2 Data Structure

2.1 extc_heap

```
#include <bits/extc++.h>
typedef __gnu_pbds::priority_queue<int> heap_t;
heap_t a,b;

int main() {
    a.clear();
    b.clear();
    a.push(1);
    a.push(3);
    b.push(2);
    b.push(4);
    assert(a.top() == 3);
    assert(b.top() == 4);
    // merge two heap
    a.join(b);
    assert(a.top() == 4);
    assert(b.empty());
}
```

```
    return 0;
}
```

2.2 extc_balance_tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
            tree_order_statistics_node_update> set_t;
typedef cc_hash_table<int, int> umap_t;

int main(){
    // Insert some entries into s.
    set_t s;
    s.insert(12);
    s.insert(505);

    // The order of the keys should be: 12, 505.
    assert(*s.find_by_order(0) == 12);
    assert(s.find_by_order(2) == end(s));

    // The order of the keys should be: 12, 505.
    assert(s.order_of_key(12) == 0);
    assert(s.order_of_key(505) == 1);

    // Erase an entry.
    s.erase(12);

    // The order of the keys should be: 505.
    assert(*s.find_by_order(0) == 505);

    // The order of the keys should be: 505.
    assert(s.order_of_key(505) == 0);
}
```

2.3 Heavy Light Decomposition

```
// TBA
```

2.4 Disjoint Set

```
struct DisjointSet {
    // save() is like recursive
    // undo() is like return
    int n, fa[MXN], sz[MXN];
    vector<pair<int*, int*>> h;
    vector<int> sp;
    void init(int tn) {
        n=tn;
        for (int i=0; i<n; i++) {
            fa[i]=i;
            sz[i]=1;
        }
        sp.clear(); h.clear();
    }
    void assign(int *k, int v) {
        h.PB({k, *k});
        *k=v;
    }
    void save() { sp.PB(SZ(h)); }
    void undo() {
        assert(!sp.empty());
        int last=sp.back(); sp.pop_back();
        while (SZ(h)!=last) {
            auto x=h.back(); h.pop_back();
            *x.F=x.S;
        }
    }
    int f(int x) {
        while (fa[x]!=x) x=fa[x];
        return x;
    }
    void uni(int x, int y) {
        x=f(x); y=f(y);
        if (x==y) return;
        if (sz[x]<sz[y]) swap(x, y);
        assign(&sz[x], sz[x]+sz[y]);
    }
}
```

```
        assign(&fa[y], x);
    }
}djs;
```

2.5 Treap

```
const int MEM = 360004;
struct Treap {
    static Treap mem[MEM], *pmem;
    Treap *l, *r;
    int val;
    int size;
    int pri;
    Treap () : l(NULL), r(NULL), size(0) {}
    Treap (int _val) :
        l(NULL), r(NULL), val(_val), size(1), pri(rand()) {}
} Treap::mem[MEM], *Treap::pmem = Treap::mem;

int size(const Treap *t) {
    return t ? t->size : 0;
}

void pull(Treap *t) {
    if (!t) return;
    t->size = size(t->l) + size(t->r) + 1;
}

Treap *merge(Treap *a, Treap *b) {
    if (!a || !b) return a ? a : b;
    if (a->pri > b->pri) {
        a->r = merge(a->r, b);
        pull(a);
        return a;
    } else {
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}

void split(Treap *t, int k, Treap *&a, Treap *&b) {
    if (!t) a = b = NULL;
    else if (t->val <= k) {
        a = t;
        split(t->r, k, a->r, b);
        pull(a);
    } else {
        b = t;
        split(t->l, k, a, b->l);
        pull(b);
    }
}
```

2.6 Persistent Treap

```
const int MEM = 16000004;
struct Treap {
    static Treap nil, mem[MEM], *pmem;
    Treap *l, *r;
    char val;
    int size;
    Treap () : l(&nil), r(&nil), size(0) {}
    Treap (char _val) :
        l(&nil), r(&nil), val(_val), size(1) {}
} Treap::nil, Treap::mem[MEM], *Treap::pmem = Treap::
    mem;

int size(const Treap *t) { return t->size; }
void pull(Treap *t) {
    if (!size(t)) return;
    t->size = size(t->l) + size(t->r) + 1;
}

Treap* merge(Treap *a, Treap *b) {
    if (!size(a)) return b;
    if (!size(b)) return a;
    Treap *t;
    if (rand() % (size(a) + size(b)) < size(a)) {
        t = new (Treap::pmem++) Treap(*a);
        t->r = merge(a->r, b);
    } else {
        t = new (Treap::pmem++) Treap(*b);
        t->l = merge(a, b->l);
    }
    pull(t);
}
```

```

    return t;
}
void split(Treap *t, int k, Treap *&a, Treap *&b) {
    if (!size(t)) a = b = &Treap::nil;
    else if (size(t->l) + 1 <= k) {
        a = new (Treap::pmem++) Treap(*t);
        split(t->r, k - size(t->l) - 1, a->r, b);
        pull(a);
    } else {
        b = new (Treap::pmem++) Treap(*t);
        split(t->l, k, a, b->l);
        pull(b);
    }
}

int nv;
Treap *rt[50005];

void print(const Treap *t) {
    if (!size(t)) return;
    print(t->l);
    cout << t->val;
    print(t->r);
}

int main(int argc, char** argv) {
    IOS;
    rt[nv=0] = &Treap::nil;
    Treap::pmem = Treap::mem;
    int Q, cmd, p, c, v;
    string s;
    cin >> Q;
    while (Q--) {
        cin >> cmd;
        if (cmd == 1) {
            // insert string s after position p
            cin >> p >> s;
            Treap *tl, *tr;
            split(rt[nv], p, tl, tr);
            for (int i=0; i<SZ(s); i++)
                tl = merge(tl, new (Treap::pmem++) Treap(s[i]));
            rt[++nv] = merge(tl, tr);
        } else if (cmd == 2) {
            // remove c characters starting at position
            Treap *tl, *tm, *tr;
            cin >> p >> c;
            split(rt[nv], p-1, tl, tm);
            split(tm, c, tm, tr);
            rt[++nv] = merge(tl, tr);
        } else if (cmd == 3) {
            // print c characters starting at position p, in
            // version v
            Treap *tl, *tm, *tr;
            cin >> v >> p >> c;
            split(rt[v], p-1, tl, tm);
            split(tm, c, tm, tr);
            print(tm);
            cout << "\n";
        }
    }
    return 0;
}

```

2.7 Persistent Segment Tree

```

int a[MXN], _a[MXN];
struct Seg {
    static Seg mem[20*MXN], *pmem;
    int siz;
    Seg *ls, *rs;
    Seg(){};
    Seg(int l, int r) : siz(0) {
        if (l == r) return;
        int m = (l + r) >> 1;
        ls = new (pmem++) Seg(l, m);
        rs = new (pmem++) Seg(m+1, r);
    }
    Seg *ins(int l, int r, int x) {
        Seg *t = new (pmem++) Seg(*this);
        t->siz++;
        if (l != r) {
            int mid = (l + r) >> 1;
            if (x <= mid) t->ls = t->ls->ins(l, mid, x);

```

```

            else t->rs = t->rs->ins(mid+1, r, x);
        }
        return t;
    }
} Seg::mem[20*MXN], *Seg::pmem = mem;
int ask(Seg *tl, Seg *tr, int l, int r, int k) {
    if (l == r) return l;
    int m = (l + r) >> 1, lsz = tr->ls->siz - tl->ls->siz;
    if (k <= lsz) return ask(tl->ls, tr->ls, l, m, k);
    else return ask(tl->rs, tr->rs, m+1, r, k - lsz);
}
Seg *seg[MXN];
int main() {
    int n, m; scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", a + i);
        _a[i] = a[i];
    }
    sort(_a + 1, _a + n + 1);
    seg[0] = new (Seg::pmem++) Seg(1, n);
    for (int i = 1; i <= n; i++) {
        int x = lower_bound(_a + 1, _a + n + 1, a[i]) - _a;
        seg[i] = seg[i-1]->ins(1, n, x);
    }
    while (m--) {
        int l, r, k; scanf("%d %d %d", &l, &r, &k);
        int x = ask(seg[l-1], seg[r], 1, n, k);
        printf("%d\n", _a[x]);
    }
    return 0;
}

```

3 Graph

3.1 BCC Edge

```

struct BccEdge {
    static const int MXN = 100005;
    struct Edge { int v, eid; };
    int n, m, step, par[MXN], dfn[MXN], low[MXN];
    vector<Edge> E[MXN];
    DisjointSet djs;
    void init(int _n) {
        n = _n; m = 0;
        for (int i=0; i<n; i++) E[i].clear();
        djs.init(n);
    }
    void add_edge(int u, int v) {
        E[u].PB({v, m});
        E[v].PB({u, m});
        m++;
    }
    void DFS(int u, int f, int f_eid) {
        par[u] = f;
        dfn[u] = low[u] = step++;
        for (auto it:E[u]) {
            if (it.eid == f_eid) continue;
            int v = it.v;
            if (dfn[v] == -1) {
                DFS(v, u, it.eid);
                low[u] = min(low[u], low[v]);
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }
    void solve() {
        step = 0;
        memset(dfn, -1, sizeof(int)*n);
        for (int i=0; i<n; i++) {
            if (dfn[i] == -1) DFS(i, i, -1);
        }
        djs.init(n);
        for (int i=0; i<n; i++) {
            if (low[i] < dfn[i]) djs.uni(i, par[i]);
        }
    }
} graph;

```

3.2 BCC Vertex

```

struct BccVertex {
    int n, nBcc, step, root, dfn[MXN], low[MXN];
    vector<int> E[MXN], ap;
    vector<PII> bcc[MXN];
    int top;
    PII stk[MXN];
    void init(int _n) {
        n = _n;
        nBcc = step = 0;
        for (int i=0; i<n; i++) E[i].clear();
    }
    void add_edge(int u, int v) {
        E[u].PB(v);
        E[v].PB(u);
    }
    void DFS(int u, int f) {
        dfn[u] = low[u] = step++;
        int son = 0;
        for (auto v:E[u]) {
            if (v == f) continue;
            if (dfn[v] == -1) {
                son++;
                stk[top++] = {u,v};
                DFS(v,u);
                if (low[v] >= dfn[u]) {
                    if (v != root) ap.PB(v);
                    do {
                        assert(top > 0);
                        bcc[nBcc].PB(stk[--top]);
                    } while (stk[top] != PII(u,v));
                    nBcc++;
                }
                low[u] = min(low[u], low[v]);
            } else {
                if (dfn[v] < dfn[u]) stk[top++] = PII(u,v);
                low[u] = min(low[u], dfn[v]);
            }
        }
        if (u == root && son > 1) ap.PB(u);
    }
    // return the edges of each bcc;
    vector<vector<PII>> solve() {
        vector<vector<PII>> res;
        for (int i=0; i<n; i++) {
            dfn[i] = low[i] = -1;
        }
        ap.clear();
        for (int i=0; i<n; i++) {
            if (dfn[i] == -1) {
                top = 0;
                root = i;
                DFS(i,i);
            }
        }
        for(int i = 0; i < nBcc; i++){
            res.PB(bcc[i]);
            bcc[i].clear();
        }
        return res;
    }
}graph;

```

3.3 Strongly Connected Components

```

struct Scc{
    int n, nScc, vst[MXN], bln[MXN];
    vector<int> E[MXN], rE[MXN], vec;
    void init(int _n){
        n = _n;
        for (int i=0; i<n; i++){
            E[i].clear();
            rE[i].clear();
        }
    }
    void add_edge(int u, int v){
        E[u].PB(v);
        rE[v].PB(u);
    }
    void DFS(int u){
        vst[u]=1;
        for (auto v: E[u])

```

```

            if (!vst[v]) DFS(v);
            vec.PB(u);
        }
    }
    void rDFS(int u){
        vst[u] = 1;
        bln[u] = nScc;
        for (auto v : rE[u])
            if (!vst[v]) rDFS(v);
    }
    void solve(){
        nScc = 0;
        vec.clear();
        for (int i=0; i<n; i++) vst[i] = 0;
        for (int i=0; i<n; i++)
            if (!vst[i]) DFS(i);
        reverse(vec.begin(), vec.end());
        for (int i=0; i<n; i++) vst[i] = 0;
        for (auto v : vec){
            if (!vst[v]){
                rDFS(v);
                nScc++;
            }
        }
    }
}
};

```

3.4 Heavy Light Decomposition

```

struct HLD{
    int n;
    vector<int> edge[MXN];
    int siz[MXN], dep[MXN];
    int cntp, re[MXN], in[MXN], out[MXN];
    int prt[MXN][20], head[MXN];
    void pre(int u, int pa){
        dep[u] = dep[pa] + 1;
        prt[0][u] = pa; siz[u] = 1; head[u] = u;
        for(int v : edge[u]){
            if (v == pa) continue;
            pre(v, u);
            siz[u] += siz[v];
        }
        return ;
    }
    void dfs(int u){
        cntp++;
        in[u] = cntp;
        re[ cntp ] = u;
        sort(ALL(g[u]), [&](int a, int b){ return siz[a] > siz[b] });
        bool f = 1;
        for(int &v : edge[u]) if(v != prt[0][u]){
            if(f) head[v] = head[u], f = 0;
            dfs(v);
        }
        out[u] = cntp;
    }
    void addEdge(int u, int v){
        edge[u].pb(v);
        edge[v].pb(u);
    }
    void init(int _n){
        n = _n;
        rep1(i, 1, n+1) edge[i].clear();
    }
    void solve(){
        pre(1, 0);
        cntp = 0;
        dfs(1);
        rep1(i, 1, 20) rep1(j, 1, n+1){
            prt[i][j] = prt[i-1][ prt[i-1][j] ];
        }
    }
    vector< PII >getpath( int u, int v ){
        vector<PII> res;
        while( in[u] < in[ head[v] ] ){
            res.pb( MP(in[ head[v] ], in[v]) );
            v = prt[ head[v] ][0];
        }
        res.pb( MP(in[u], in[v]) );
        reverse( ALL(res) );
        return res;
    }
}

```

```
}tree;
```

3.5 Maximum Clique

```
class MaxClique {
public:
    static const int MV = 210;

    int V;
    int el[MV][MV/30+1];
    int dp[MV];
    int ans;
    int s[MV][MV/30+1];
    vector<int> sol;

    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }

    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }

    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a ; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {
                    lb = (unsigned int)(lb) >> 1;
                    lg++;
                }
                int u = i*32 + lg;
                if(k + dp[u] <= ans) return 0;
                if(dfs(u, k+1)) {
                    sol.push_back(v);
                    return 1;
                }
            }
        }
        return 0;
    }

    int solve() {
        for(int i=V-1; i>=0; i--) {
            dfs(i, 1);
            dp[i] = ans;
        }
        return ans;
    }
};
```

3.6 MinimumMeanCycle

```
/* minimum mean cycle */
const int MAXE = 1805;
const int MAXN = 35;
const double inf = 1029384756;
const double eps = 1e-6;
struct Edge {
    int v,u;
```

```
double c;
};
int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];
Edge e[MAXE];
vector<int> edgeID, cycle, rho;
double d[MAXN][MAXN];
inline void bellman_ford() {
    for(int i=0; i<n; i++) d[0][i]=0;
    for(int i=0; i<n; i++) {
        fill(d[i+1], d[i+1]+n, inf);
        for(int j=0; j<m; j++) {
            int v = e[j].v, u = e[j].u;
            if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
                d[i+1][u] = d[i][v]+e[j].c;
                prv[i+1][u] = v;
                prve[i+1][u] = j;
            }
        }
    }
}
double karp_mmc() {
    // returns inf if no cycle, mmc otherwise
    double mmc=inf;
    int st = -1;
    bellman_ford();
    for(int i=0; i<n; i++) {
        double avg=-inf;
        for(int k=0; k<n; k++) {
            if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i])/(n-k));
            else avg=max(avg,inf);
        }
        if (avg < mmc) tie(mmc, st) = tie(avg, i);
    }
    for(int i=0; i<n; i++) vst[i] = 0;
    edgeID.clear(); cycle.clear(); rho.clear();
    for (int i=n; !vst[st]; st=prv[i--][st]) {
        vst[st]++;
        edgeID.PB(prve[i][st]);
        rho.PB(st);
    }
    while (vst[st] != 2) {
        int v = rho.back(); rho.pop_back();
        cycle.PB(v);
        vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize(SZ(cycle));
    return mmc;
}
```

4 Flow

4.1 Dinic

```
struct Dinic{
    static const int MXN = 10000;
    struct Edge{ int v,f,re; };
    int n,s,t,level[MXN];
    vector<Edge> E[MXN];
    void init(int _n, int _s, int _t){
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
    }
    void add_edge(int u, int v, int f){
        E[u].PB({v,f,SZ(E[v])});
        E[v].PB({u,0,SZ(E[u])-1});
    }
    bool BFS(){
        for (int i=0; i<n; i++) level[i] = -1;
        queue<int> que;
        que.push(s);
        level[s] = 0;
        while (!que.empty()){
            int u = que.front(); que.pop();
            for (auto it : E[u]){
                if (it.f > 0 && level[it.v] == -1){
                    level[it.v] = level[u]+1;
                    que.push(it.v);
                }
            }
        }
    }
```

```

    return level[t] != -1;
}
int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
        if (it.f > 0 && level[it.v] == level[u]+1){
            int tf = DFS(it.v, min(nf, it.f));
            res += tf; nf -= tf; it.f -= tf;
            E[it.v][it.re].f += tf;
            if (nf == 0) return res;
        }
    }
    if (!res) level[u] = -1;
    return res;
}
int flow(int res=0){
    while ( BFS() )
        res += DFS(s, 2147483647);
    return res;
}
}flow;

```

4.2 Cost Flow

```

typedef pair<long long, long long> pll;
struct CostFlow {
    static const int MXN = 205;
    static const long long INF = 102938475610293847LL;
    struct Edge {
        int v, r;
        long long f, c;
    };
    int n, s, t, prv[MXN], prvl[MXN], inq[MXN];
    long long dis[MXN], fl, cost;
    vector<Edge> E[MXN];
    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
        fl = cost = 0;
    }
    void add_edge(int u, int v, long long f, long long c) {
        E[u].PB({v, SZ(E[v]), f, c});
        E[v].PB({u, SZ(E[u])-1, 0, -c});
    }
    pll flow() {
        while (true) {
            for (int i=0; i<n; i++) {
                dis[i] = INF;
                inq[i] = 0;
            }
            dis[s] = 0;
            queue<int> que;
            que.push(s);
            while (!que.empty()) {
                int u = que.front(); que.pop();
                inq[u] = 0;
                for (int i=0; i<SZ(E[u]); i++) {
                    int v = E[u][i].v;
                    long long w = E[u][i].c;
                    if (E[u][i].f > 0 && dis[v] > dis[u] + w) {
                        prv[v] = u; prvl[v] = i;
                        dis[v] = dis[u] + w;
                        if (!inq[v]) {
                            inq[v] = 1;
                            que.push(v);
                        }
                    }
                }
            }
            if (dis[t] == INF) break;
            long long tf = INF;
            for (int v=t, u, l; v!=s; v=u) {
                u=prv[v]; l=prvl[v];
                tf = min(tf, E[u][l].f);
            }
            for (int v=t, u, l; v!=s; v=u) {
                u=prv[v]; l=prvl[v];
                E[u][l].f -= tf;
                E[v][E[u][l].r].f += tf;
            }
            cost += tf * dis[t];
            fl += tf;
        }
    }
}

```

```

    }
    return {fl, cost};
}
}flow;

```

4.3 Kuhn Munkres

```

struct KM{
    // Maximum Bipartite Weighted Matching (Perfect Match)
    static const int MXN = 650;
    static const int INF = 2147483647; // Long Long
    int n, match[MXN], vx[MXN], vy[MXN];
    int edge[MXN][MXN], lx[MXN], ly[MXN], slack[MXN];
    // ^^^ Long Long
    void init(int _n){
        n = _n;
        for (int i=0; i<n; i++)
            for (int j=0; j<n; j++)
                edge[i][j] = 0;
    }
    void add_edge(int x, int y, int w){ // Long Long
        edge[x][y] = w;
    }
    bool DFS(int x){
        vx[x] = 1;
        for (int y=0; y<n; y++){
            if (vy[y]) continue;
            if (lx[x]+ly[y] > edge[x][y]){
                slack[y] = min(slack[y], lx[x]+ly[y]-edge[x][y]);
            } else {
                vy[y] = 1;
                if (match[y] == -1 || DFS(match[y])){
                    match[y] = x;
                    return true;
                }
            }
        }
        return false;
    }
    int solve(){
        fill(match, match+n, -1);
        fill(lx, lx+n, -INF);
        fill(ly, ly+n, 0);
        for (int i=0; i<n; i++)
            for (int j=0; j<n; j++)
                lx[i] = max(lx[i], edge[i][j]);
        for (int i=0; i<n; i++){
            fill(slack, slack+n, INF);
            while (true){
                fill(vx, vx+n, 0);
                fill(vy, vy+n, 0);
                if ( DFS(i) ) break;
                int d = INF; // Long Long
                for (int j=0; j<n; j++)
                    if (!vy[j]) d = min(d, slack[j]);
                for (int j=0; j<n; j++){
                    if (vx[j]) lx[j] -= d;
                    if (vy[j]) ly[j] += d;
                    else slack[j] -= d;
                }
            }
        }
        int res=0;
        for (int i=0; i<n; i++)
            res += edge[match[i]][i];
        return res;
    }
}graph;

```

4.4 Maximum Simple Graph Matching

```

struct GenMatch { // 1-base
    static const int MAXN = 514;
    int V;
    bool el[MAXN][MAXN];
    int pr[MAXN];
    bool inq[MAXN], inp[MAXN], inb[MAXN];
    queue<int> qe;
    int st, ed;
    int nb;
}

```

```

int bk[MAXN], djs[MAXN];
int ans;
void init(int _V) {
    V = _V;
    for(int i = 0; i <= V; i++) {
        for(int j = 0; j <= V; j++) el[i][j] = 0;
        pr[i] = bk[i] = djs[i] = 0;
        inq[i] = inp[i] = inb[i] = 0;
    }
    ans = 0;
}
void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
}
int lca(int u, int v) {
    for(int i = 0; i <= V; i++) inp[i] = 0;
    while(1) {
        u = djs[u];
        inp[u] = true;
        if(u == st) break;
        u = bk[pr[u]];
    }
    while(1) {
        v = djs[v];
        if(inp[v]) return v;
        v = bk[pr[v]];
    }
    return v;
}
void upd(int u) {
    int v;
    while(djs[u] != nb) {
        v = pr[u];
        inb[djs[u]] = inb[djs[v]] = true;
        u = bk[v];
        if(djs[u] != nb) bk[u] = v;
    }
}
void blo(int u, int v) {
    nb = lca(u, v);
    for (int i=0; i<=V; i++) inb[i] = 0;
    upd(u); upd(v);
    if(djs[u] != nb) bk[u] = v;
    if(djs[v] != nb) bk[v] = u;
    for(int tu = 1; tu <= V; tu++)
        if(inb[djs[tu]]) {
            djs[tu] = nb;
            if(!inq[tu]){
                qe.push(tu);
                inq[tu] = 1;
            }
        }
}
void flow() {
    for(int i = 1; i <= V; i++) {
        inq[i] = 0;
        bk[i] = 0;
        djs[i] = i;
    }

    while(qe.size()) qe.pop();
    qe.push(st);
    inq[st] = 1;
    ed = 0;
    while(qe.size()) {
        int u = qe.front(); qe.pop();
        for(int v = 1; v <= V; v++)
            if(el[u][v] && (djs[u] != djs[v]) && (pr[u] != v)) {
                if((v == st) || ((pr[v] > 0) && bk[pr[v]] > 0))
                    blo(u, v);
                else if(bk[v] == 0) {
                    bk[v] = u;
                    if(pr[v] > 0) {
                        if(!inq[pr[v]]) qe.push(pr[v]);
                    } else {
                        ed = v;
                        return;
                    }
                }
            }
    }
}
void aug() {
    int u, v, w;

```

```

    u = ed;
    while(u > 0) {
        v = bk[u];
        w = pr[v];
        pr[v] = u;
        pr[u] = v;
        u = w;
    }
}
int solve() {
    for(int i = 0; i <= V; i++) pr[i] = 0;
    for(int u = 1; u <= V; u++)
        if(pr[u] == 0) {
            st = u;
            flow();
            if(ed > 0) {
                aug();
                ans ++;
            }
        }
    return ans;
}
}G;
int main() {
    G.init(V);
    for(int i=0; i<E; i++) {
        int u, v;
        cin >> u >> v;
        G.add_edge(u, v);
    }
    cout << G.solve() << endl;
}

```

4.5 Minimum Weight Matching (Clique version)

```

struct Graph {
    // Minimum General Weighted Matching (Perfect Match)
    // 0-base
    static const int MXN = 105;

    int n, edge[MXN][MXN];
    int match[MXN], dis[MXN], onstk[MXN];
    vector<int> stk;

    void init(int _n) {
        n = _n;
        for (int i=0; i<n; i++)
            for (int j=0; j<n; j++)
                edge[i][j] = 0;
    }
    void add_edge(int u, int v, int w) {
        edge[u][v] = edge[v][u] = w;
    }
    bool SPFA(int u){
        if (onstk[u]) return true;
        stk.PB(u);
        onstk[u] = 1;
        for (int v=0; v<n; v++){
            if (u != v && match[u] != v && !onstk[v]){
                int m = match[v];
                if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
                    dis[m] = dis[u] - edge[v][m] + edge[u][v];
                    onstk[v] = 1;
                    stk.PB(v);
                    if (SPFA(m)) return true;
                }
                stk.pop_back();
                onstk[v] = 0;
            }
        }
        onstk[u] = 0;
        stk.pop_back();
        return false;
    }

    int solve() {
        // find a match
        for (int i=0; i<n; i+=2){
            match[i] = i+1;
            match[i+1] = i;
        }
    }
}

```



```

while (true){
    int found = 0;
    for (int i=0; i<n; i++){
        dis[i] = onstk[i] = 0;
        for (int i=0; i<n; i++){
            stk.clear();
            if (!onstk[i] && SPFA(i)){
                found = 1;
                while (SZ(stk)>=2){
                    int u = stk.back(); stk.pop_back();
                    int v = stk.back(); stk.pop_back();
                    match[u] = v;
                    match[v] = u;
                }
            }
        }
        if (!found) break;
    }
    int ret = 0;
    for (int i=0; i<n; i++){
        ret += edge[i][match[i]];
    }
    ret /= 2;
    return ret;
}
}graph;

```

5 Math

5.1 Bigint

```

struct Bigint{
    static const int LEN = 60;
    static const int BIGMOD = 10000;

    int s;
    int vl, v[LEN];
    // vector<int> v;
    Bigint() : s(1) { vl = 0; }
    Bigint(long long a) {
        s = 1; vl = 0;
        if (a < 0) { s = -1; a = -a; }
        while (a) {
            push_back(a % BIGMOD);
            a /= BIGMOD;
        }
    }
    Bigint(string str) {
        s = 1; vl = 0;
        int stPos = 0, num = 0;
        if (!str.empty() && str[0] == '-') {
            stPos = 1;
            s = -1;
        }
        for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
            num += (str[i] - '0') * q;
            if ((q * 10) >= BIGMOD) {
                push_back(num);
                num = 0; q = 1;
            }
        }
        if (num) push_back(num);
        n();
    }

    int len() const {
        return vl;
        // return SZ(v);
    }
    bool empty() const { return len() == 0; }
    void push_back(int x) {
        v[vl++] = x;
        // v.PB(x);
    }
    void pop_back() {
        vl--;
        // v.pop_back();
    }
    int back() const {
        return v[vl-1];
        // return v.back();
    }
    void n() {

```

```

        while (!empty() && !back()) pop_back();
    }
    void resize(int nl) {
        vl = nl;
        fill(v, v+vl, 0);
        // v.resize(nl);
        // fill(ALL(v), 0);
    }

    void print() const {
        if (empty()) { putchar('0'); return; }
        if (s == -1) putchar('-');
        printf("%d", back());
        for (int i=len()-2; i>=0; i--) printf("%.4d", v[i]);
    }

    friend std::ostream& operator << (std::ostream& out,
        const Bigint &a) {
        if (a.empty()) { out << "0"; return out; }
        if (a.s == -1) out << "-";
        out << a.back();
        for (int i=a.len()-2; i>=0; i--) {
            char str[10];
            snprintf(str, 5, "%.4d", a.v[i]);
            out << str;
        }
        return out;
    }

    int cp3(const Bigint &b) const {
        if (s != b.s) return s - b.s;
        if (s == -1) return -(*this).cp3(-b);
        if (len() != b.len()) return len()-b.len(); //int
        for (int i=len()-1; i>=0; i--)
            if (v[i] != b.v[i]) return v[i]-b.v[i];
        return 0;
    }

    bool operator < (const Bigint &b) const { return cp3(b) < 0; }
    bool operator <= (const Bigint &b) const { return cp3(b) <= 0; }
    bool operator == (const Bigint &b) const { return cp3(b) == 0; }
    bool operator != (const Bigint &b) const { return cp3(b) != 0; }
    bool operator > (const Bigint &b) const { return cp3(b) > 0; }
    bool operator >= (const Bigint &b) const { return cp3(b) >= 0; }

    Bigint operator - () const {
        Bigint r = (*this);
        r.s = -r.s;
        return r;
    }
    Bigint operator + (const Bigint &b) const {
        if (s == -1) return -(*this)+(-b);
        if (b.s == -1) return (*this)-(-b);
        Bigint r;
        int nl = max(len(), b.len());
        r.resize(nl + 1);
        for (int i=0; i<nl; i++) {
            if (i < len()) r.v[i] += v[i];
            if (i < b.len()) r.v[i] += b.v[i];
            if (r.v[i] >= BIGMOD) {
                r.v[i+1] += r.v[i] / BIGMOD;
                r.v[i] %= BIGMOD;
            }
        }
        r.n();
        return r;
    }

    Bigint operator - (const Bigint &b) const {
        if (s == -1) return -(*this)-(-b);
        if (b.s == -1) return (*this)+(-b);
        if ((*this) < b) return -(b-(*this));
        Bigint r;
        r.resize(len());
        for (int i=0; i<len(); i++) {
            r.v[i] += v[i];
            if (i < b.len()) r.v[i] -= b.v[i];
            if (r.v[i] < 0) {
                r.v[i] += BIGMOD;
                r.v[i+1]--;
            }
        }
    }

```



```

    r.n();
    return r;
}
Bigint operator * (const Bigint &b) {
    Bigint r;
    r.resize(len() + b.len() + 1);
    r.s = s * b.s;
    for (int i=0; i<len(); i++) {
        for (int j=0; j<b.len(); j++) {
            r.v[i+j] += v[i] * b.v[j];
            if(r.v[i+j] >= BIGMOD) {
                r.v[i+j+1] += r.v[i+j] / BIGMOD;
                r.v[i+j] %= BIGMOD;
            }
        }
    }
    r.n();
    return r;
}
Bigint operator / (const Bigint &b) {
    Bigint r;
    r.resize(max(1, len()-b.len()+1));
    int oriS = s;
    Bigint b2 = b; // b2 = abs(b)
    s = b2.s = r.s = 1;
    for (int i=r.len()-1; i>=0; i--) {
        int d=0, u=BIGMOD-1;
        while(d<u) {
            int m = (d+u+1)>>1;
            r.v[i] = m;
            if((r*b2) > (*this)) u = m-1;
            else d = m;
        }
        r.v[i] = d;
    }
    s = oriS;
    r.s = s * b.s;
    r.n();
    return r;
}
Bigint operator % (const Bigint &b) {
    return (*this)-(*this)/b*b;
}
};

```

5.2 $ax+by=\gcd$

```

PLL ex_gcd(LL a, LL b){
    if(b == 0) return MP(1, 0);
    else{
        LL p = a / b;
        PLL q = ex_gcd(b, a % b);
        return MP(q.S, q.F - q.S * p);
    }
}

```

5.3 Linear Prime Sieve

```

int ck[MXN];
vector<int> pr;
void linear_sieve(){
    for (int i = 2; i < MXN; i++) {
        if(!ck[i]) pr.pb(i);
        for (int j = 0; i*pr[j] < MXN; j++){
            ck[i*pr[j]] = pr[j];
            if(i % pr[j] == 0) break;
        }
    }
}

```

5.4 Bsgs

```

/* solve x for v^x = m mod p
p is prime
O( sqrt(p)log(p) ) */
struct Bsgs{
    LL mypow(LL v, LL t, LL md) {
        LL res = 1;
        while (t) {

```

```

            if (t & 1) res = res*v%md;
            t >>= 1;
            v = v*v%md;
        }
        return res;
    }
map<LL, int>mp;
LL solve(LL p, LL v, LL m){
    mp.clear();
    int h = ceil( sqrt(p + 0.5) );
    LL cv = 1;
    for (int i = 0; i < h; i++) {
        mp[cv] = i;
        cv = cv*v%p;
    }
    cv = mypow(cv, p - 2, p);
    int ok = 0, ans = 0;
    for (int i = 0; i <= h; i++) {
        if (mp.find(m) != mp.end()) {
            ans += mp[m];
            ok = 1; break;
        }
        ans += h;
        m = m*cv%p;
    }
    return ok ? ans : -1;
}
}bsgs;

```

5.5 Chinese Remainder

```

int pfn; // number of distinct prime factors
int pf[MAXNUM]; // prime factor powers
int rem[MAXNUM]; // corresponding remainder
int pm[MAXNUM];
inline void generate_primes() {
    int i,j;
    pnum=1;
    prime[0]=2;
    for(i=3;i<MAXVAL;i+=2) {
        if(nprime[i]) continue;
        prime[pnum++]=i;
        for(j=i*i;j<MAXVAL;j+=i) nprime[j]=1;
    }
}
inline int inverse(int x,int p) {
    int q,tmp,a=x,b=p;
    int a0=1,a1=0,b0=0,b1=1;
    while(b) {
        q=a/b; tmp=b; b=a-b*q; a=tmp;
        tmp=b0; b0=a0-b0*q; a0=tmp;
        tmp=b1; b1=a1-b1*q; a1=tmp;
    }
    return a0;
}
inline void decompose_mod() {
    int i,p,t=mod;
    pfn=0;
    for(i=0;i<pnum&&prime[i]<=t;i++) {
        p=prime[i];
        if(t%p==0) {
            pf[pfn]=1;
            while(t%p==0) {
                t/=p;
                pf[pfn]*=p;
            }
            pfn++;
        }
    }
    if(t>1) pf[pfn++]=t;
}
inline int chinese_remainder() {
    int i,m,s=0;
    for(i=0;i<pfn;i++) {
        m=mod/pf[i];
        pm[i]=(long long)m*inverse(m,pf[i])%mod;
        s=(s+(long long)pm[i]*rem[i])%mod;
    }
    return s;
}

```

5.6 Fast Fourier Transform

```

#define rep(i, a) for (int i = 0; i < a; i++)
#define rep1(i, a, b) for(int i = a; i < b; i++)
struct cp{
    double a,b;
    cp(){};
    cp(double _a, double _b){
        a = _a, b = _b;
    }
    cp operator +(const cp &o){ return cp(a+o.a, b+o.b);
    }
    cp operator -(const cp &o){ return cp(a-o.a, b-o.b);
    }
    cp operator *(const cp &o){ return cp(a*o.a-b*o.b, b*o.a+a*o.b); }
    cp operator *(const double &o){ return cp(a*o, b*o);
    }
    cp operator !(){ return cp(a, -b); }
}w[MXN];
int pos[MXN];
void fft_init(int len){
    int j = 0;
    while((1<<j) < len) j++;
    j--;
    rep(i, len) pos[i] = pos[i>>1]>>1 | ((i&1)<<j);
    return ;
}
void fft(cp *x, int len, int sta){
    rep(i, len) if(i < pos[i]) swap(x[i], x[pos[i]]);
    w[0] = cp(1, 0);
    for(unsigned i = 2; i <= len; i <= 1){
        cp g = cp(cos(2*PI/i), sin(2*PI/i)*sta);
        for(int j = i>>1; j >= 0; j -= 2) w[j] = w[j>>1];
        for(int j = 1; j < (i>>1); j += 2) w[j] = w[j-1]*g;
        for(int j = 0; j < len; j += i){
            cp *a = x+j, *b = a+(i>>1);
            rep(l, i>>1){
                cp o = b[l]*w[l];
                b[l] = a[l]-o;
                a[l] = a[l]+o;
            }
        }
    }
    if(sta == -1) rep(i, len) x[i].a /= len, x[i].b /= len;
    return ;
}
cp xt[MXN], yt[MXN], zt[MXN];
LL st[3][MXN];
void FFT(int a, int b, int l1, int l2, int c){
    int len = 1;
    while(len <= (l1+l2)>>1) len <= 1;
    fft_init(len);
    rep1(i, l1>>1, len) xt[i].a = xt[i].b = 0;
    rep(i, l1) (i&1 ? xt[i>>1].b : xt[i>>1].a) = st[a][i];
    fft(xt, len, 1);

    rep1(i, l2>>1, len) yt[i].a = yt[i].b = 0;
    rep(i, l2) (i&1 ? yt[i>>1].b : yt[i>>1].a) = st[b][i];
    fft(yt, len, 1);

    rep(i, len>>1){
        int j = len - 1&len - i;
        zt[i] = xt[i]*yt[i] - (xt[i]-!xt[j])*(yt[i]-!yt[j])
            *(w[i+cp(1,0))*0.25;
    }
    rep1(i, len>>1, len){
        int j = len - 1&len - i;
        zt[i] = xt[i]*yt[i] - (xt[i]-!xt[j])*(yt[i]-!yt[j])
            *(cp(1,0)-w[i^len>>1])*0.25;
    }
    fft(zt, len, -1);
    rep(i, l1 + l2 - 1){
        if(i&1) st[c][i] = (LL)(zt[i>>1].b+0.5);
        else st[c][i] = (LL)(zt[i>>1].a+0.5);
    }
    return ;
}

```

```

/* find x for x^t = m (mod p) p is prime
* 1. find PrimitiveRoot of p (assume it is v) O( sqrt(n)log(n) )
* 2. v^(at) = v^b
* 3. use Bsgs to find b O( sqrt(n) + m*log(m) )
* 4. use ex_gcd to find a(ax + by = gcd, at + b(p-1) = gcd) O(log(n))
*/
LL mypow(LL v, LL t, LL md = mod) {
    LL res = 1;
    while (t) {
        if (t & 1) res = res*v%md;
        t >>= 1;
        v = v*v%md;
    }
    return res;
}
LL gcd(LL v1, LL v2){
    while (v1) {
        LL tmp = v2 % v1;
        v2 = v1;
        v1 = tmp;
    }
    return v2;
}
struct KthResidue{
    struct PriRoot{
        int a[MXN], cntp;
        LL phi(LL n){
            int h = sqrt(n);
            LL res = n, v = n;
            for (int i = 2; i <= h; i++) {
                if (v % i == 0) {
                    res = res / i * (i - 1);
                    while (v % i == 0) v /= i;
                }
            }
            if (v != 1) res = res / v * (v - 1);
            return res;
        }
    }
    int solve(LL n){
        LL num = phi(n); // if n is prime, num = n - 1
        LL v = num;
        int h = sqrt(num);
        cntp = 0;
        for (int i = 2; i <= h; i++) {
            if (v % i == 0) {
                a[++cntp] = i;
                while (v % i == 0) v /= i;
            }
        }
        if (v != 1) a[++cntp] = v;
        v = num;
        for (int i = 2; i < n; i++) {
            if (gcd(n, i) != 1) continue;
            bool ok = 1;
            for (int j = 1; j <= cntp; j++) {
                if (mypow(i, v / a[j], n) == 1) {
                    ok = 0; break;
                }
            }
            if (ok) return i;
        }
        return -1;
    }
}root;
struct Bsgs{
    map<LL, int>mp;
    LL solve(LL v, LL m, LL p){
        mp.clear();
        int h = ceil( sqrt(p + 0.5) );
        LL cv = 1;
        for (int i = 0; i < h; i++) {
            mp[cv] = i;
            cv = cv*v%p;
        }
        cv = mypow(cv, p - 2, p);
        int ok = 0, ans = 0;
        for (int i = 0; i <= h; i++) {
            if (mp.find(m) != mp.end()) {
                ans += mp[m];
                ok = 1; break;
            }
            ans += h;
            m = m*cv%p;
        }
    }
}

```

5.7 Kth Residue

/*

```

    return ok ? ans : -1;
}
}bsgs;

PLL ex_gcd(LL a, LL b){
    if(b == 0) return MP(1, 0);
    else{
        LL p = a / b;
        PLL q = ex_gcd(b, a % b);
        return MP(q.S, q.F - q.S * p);
    }
}

LL solve(LL t, LL m, LL p){
    LL v = root.solve(p);
    LL gd = bsgs.solve(v, m, p);
    if (gd == -1) return -1;

    PLL res = ex_gcd(t, p-1);
    LL val = (t*res.F + (p-1)*res.S);
    if (gd % val) return -1;
    LL num = (res.F*(gd / val)%(p-1) + p - 1) % (p-1);
    return mypow(v, num, p);
}
}residue;

```

5.8 Gauss Elimination

```

const int MAX = 300;
const double EPS = 1e-8;

double mat[MAX][MAX];
void Gauss(int n) {
    for(int i=0; i<n; i++) {
        bool ok = 0;
        for(int j=i; j<n; j++) {
            if(fabs(mat[j][i]) > EPS) {
                swap(mat[j], mat[i]);
                ok = 1;
                break;
            }
        }
        if(!ok) continue;

        double fs = mat[i][i];
        for(int j=i+1; j<n; j++) {
            double r = mat[j][i] / fs;
            for(int k=i; k<n; k++) {
                mat[j][k] -= mat[i][k] * r;
            }
        }
    }
}

```

5.9 Matrix

```

struct Mat {
    int n, m;
    LL a[MXN][MXN];
#define rep1(i, a, b) for (int i = a; i < b; i++)
    void init(int _n, int _m) {
        n = _n, m = _m;
        rep1(i, 1, n+1) rep1(j, 1, m+1){
            a[i][j] = 0;
        }
    }
    Mat operator *(const Mat &p2) {
        Mat res; res.init(n, p2.m);
        rep1(i, 1, n+1) rep1(j, 1, m+1) rep1(k, 1, p2.m+1) {
            res.a[i][k] = (res.a[i][k] + a[i][j]*p2.a[j][k])%mod;
        }
        return res;
    }
    Mat operator ^(const LL &p2) {
        LL t = p2 - 1;
        Mat res = *this, x = *this;
        while (t) {
            if(t & 1) res = res*x;
            t >>= 1;
            x = x*x;
        }
    }
}

```

```

    }
    return res;
}
};

```

5.10 NTT

```

const int P = 998244353, root = 3, MAXNUM=2097152;
int bigmod(LL v, LL t){
    LL res = 1;
    while(t){
        if(t & 1) res = res*v%mod;
        v = v*v%mod;
        t >>= 1;
    }
    return res;
}
int inv(LL a, LL b){
    if(a == 1) return 1;
    return ( (a - inv(b%a,a))*b + 1) / a %b;
}
std::vector<long long> ps(MAXNUM);
struct poly{
    vector<LL> co;
    int n; //polynomial degree = n
    poly(int d = 0){n = d; co.resize(n,0); }
    void init(int _n, LL _co[]){
        n = _n;
        co.resize(n);
        for(int i = 0; i < n; ++i)
            co[i] = _co[i];
    }
    void trans2(int NN){
        int r = 0;
        while( (1<<r) < (NN>>1) ) ++r;
        for(int N = 2; N <= NN; N <= 1, --r){
            for(int st = 0; st < NN; st += N){
                int ss = st + (N>>1);
                for(int i = (N>>1)-1; i >= 0; --i){
                    LL a = co[st + i], b = ps[i<<r]*co[ss+i]%P;
                    co[st+i] = a + b; if(co[st+i] >= P) co[st + i] -= P;
                    co[ss+i] = a - b; if(co[ss+i] < 0) co[ss + i] += P;
                }
            }
        }
    }
    void trans1(int NN){
        int r = 0;
        for(int N = NN; N > 1; N >>= 1, ++r){
            for(int st = 0; st < NN; st += N){
                int ss = st + (N>>1);
                for(int i = (N>>1)-1; i >= 0; --i){
                    LL a = co[st + i], b = co[ss + i];
                    co[st+i] = a + b; if(co[st+i] >= P) co[st+i] -= P;
                    co[ss+i] = (a + P - b)*ps[i<<r]%P;
                }
            }
        }
    }
    poly operator*(const poly& _b) const{
        poly a = *this, b = _b;
        int k = n + b.n, N = 1;
        while( N <= k ) N <= 1;
        a.co.resize(N,0); b.co.resize(N,0);
        int r = bigmod(root, (P-1)/N), Ni = inv(N, P);
        ps[0] = 1;
        for(int i = 1; i < N; ++i) ps[i] = ps[i-1]*r%P;
        a.trans1(N); b.trans1(N);
        for(int i = 0; i < N; ++i) a.co[i] = a.co[i]*b.co[i]%P;
        r = inv(r, P);
        for(int i = 1; i < N / 2; ++i) swap(ps[i], ps[N-i]);
        a.trans2(N);
        for(int i = 0; i < k; ++i) a.co[i] = a.co[i]*Ni%P;
        a.n = k - 1; return a;
    }
};

```

5.11 NTT(eddy ver.)

```
typedef long long LL;
// Remember coefficient are mod P
/* p=a*2^n+1
   n   2^n       p       a   root
   16   65536     65537    1   3
   20   1048576   7340033  7   3 */
// (must be 2^k)
template<LL P, LL root, int MAXN>
struct NTT{
    static LL bigmod(LL a, LL b) {
        LL res = 1;
        for (LL bs = a; b >= 1; bs = (bs * bs) % P)
            if(b&1) res=(res*bs)%P;
        return res;
    }
    static LL inv(LL a, LL b) {
        if(a==1) return 1;
        return (((LL)(a-inv(b%a,a))*b+1)/a)%b;
    }
    LL omega[MAXN+1];
    NTT() {
        omega[0] = 1;
        LL r = bigmod(root, (P-1)/MAXN);
        for (int i=1; i<=MAXN; i++)
            omega[i] = (omega[i-1]*r)%P;
    }
    // n must be 2^k
    void tran(int n, LL a[], bool inv_ntt=false){
        int basic = MAXN / n, theta = basic;
        for (int m = n; m >= 2; m >= 1) {
            int mh = m >> 1;
            for (int i = 0; i < mh; i++) {
                LL w = omega[i*theta%MAXN];
                for (int j = i; j < n; j += m) {
                    int k = j + mh;
                    LL x = a[j] - a[k];
                    if (x < 0) x += P;
                    a[j] += a[k];
                    if (a[j] > P) a[j] -= P;
                    a[k] = (w * x) % P;
                }
            }
            theta = (theta * 2) % MAXN;
        }
        int i = 0;
        for (int j = 1; j < n - 1; j++) {
            for (int k = n >> 1; k > (i ^ k); k >= 1);
            if (j < i) swap(a[i], a[j]);
        }
        if (inv_ntt) {
            LL ni = inv(n,P);
            reverse(a+1, a+n);
            for (i = 0; i < n; i++)
                a[i] = (a[i] * ni) % P;
        }
    }
};
const LL P=2013265921,root=31;
const int MAXN=4194304;
NTT<P, root, MAXN> ntt;
```

5.12 Miller Rabin

```
// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : pirmses <= 13
// n < 2^64                7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
long long mult(LL a, LL b, LL mod) {
    a %= mod, b %= mod;
    LL res = 0;
    while (b) {
        if(b & 1) res = (res + a) % mod;
        b >>= 1;
        a = (a<<1)%mod;
    }
    return res;
}
long long power(long long x, long long p, long long mod){
```

```
long long s=1,m=x;
while (p) {
    if(p&1) s=mult(s,m,mod);
    p>>=1;
    m=mult(m,m,mod);
}
return s;
}
bool witness(long long a, long long n, long long u, int t)
{
    long long x=power(a,u,n);
    for(int i = 0; i < t; i++) {
        long long nx=mult(x,x,n);
        if(nx==1&&x!=1&&x!=n-1) return 1;
        x=nx;
    }
    return x!=1;
}
bool miller_rabin(long long n, int s=100) {
    // iterate s times of witness on n
    // return 1 if prime, 0 otherwise
    if(n<2) return 0;
    if(!(n&1)) return n==2;
    long long u=n-1;
    int t=0;
    // n-1 = u*2^t
    while(!(u&1)) {
        u>>=1;
        t++;
    }
    while(s--) {
        long long a=randll()%(n-1)+1;
        if(witness(a,n,u,t)) return 0;
    }
    return 1;
}
```

5.13 Pollard Rho

```
// does not work when n is prime
LL f(LL x, LL mod){ return add(mul(x,x,mod),1,mod); }
LL pollard_rho(LL n) {
    if(!(n&1)) return 2;
    while(true){
        LL y=2, x=rand()%(n-1)+1, res=1;
        for(int sz=2; res==1; sz*=2) {
            for(int i=0; i<sz && res<=1; i++) {
                x = f(x, n);
                res = __gcd(abs(x-y), n);
            }
            y = x;
        }
        if (res!=0 && res!=n) return res;
    }
}
```

5.14 Algorithms about Primes

```
/*
* 12721
* 13331
* 14341
* 75577
* 123457
* 222557
* 556679
* 999983
* 1097774749
* 1076767633
* 100102021
* 999997771
* 1001010013
* 1000512343
* 987654361
* 999991231
* 999888733
* 98789101
* 987777733
* 999991921
* 1010101333
* 1010102101
* 100000000039
```

```

* 1000000000000037
* 2305843009213693951
* 4611686018427387847
* 9223372036854775783
* 18446744073709551557
*/

int mu[MX], p_tbl[MX];
vector<int> primes;
void sieve() {
    mu[1] = p_tbl[1] = 1;
    for (int i=2; i<MX; i++) {
        if (!p_tbl[i]) {
            p_tbl[i] = i;
            primes.PB(i);
            mu[i] = -1;
        }
        for (auto p : primes) {
            int x = i*p;
            if (x >= M) break;
            p_tbl[x] = p;
            mu[x] = -mu[i];
            if (i%p==0) {
                mu[x] = 0;
                break;
            }
        }
    }
}

vector<int> factor(int x) {
    vector<int> fac{1};
    while (x > 1) {
        int fn=SZ(fac), p=p_tbl[x], pos=0;
        while (x%p == 0) {
            x /= p;
            for (int i=0; i<fn; i++)
                fac.PB(fac[pos++]*p);
        }
    }
    return fac;
}

```

5.15 Primitive Root

```

/*
* Primitive root exist only if :
* P = 2, 4, prime^x, 2*prime^x
* if Primitive root exist, then number of root = phi(
    phi(P) )
* O( sqrt(P) + m*log(P-1) ) m = number of prime factor
  of phi(P)
*/
struct PriRoot{
    int a[MXN], cntp;
    LL mypow(LL v, LL t, LL md) {
        LL res = 1;
        while (t) {
            if (t & 1) res = res*v%md;
            t >>= 1;
            v = v*v%md;
        }
        return res;
    }
    LL gcd(LL v1, LL v2){
        while (v1) {
            LL tmp = v2 % v1;
            v2 = v1;
            v1 = tmp;
        }
        return v2;
    }
    LL phi(LL n){
        int h = sqrt(n);
        LL res = n, v = n;
        for (int i = 2; i <= h; i++) {
            if (v % i == 0) {
                res = res / i * (i - 1);
                while (v % i == 0) v /= i;
            }
        }
        if (v != 1) res = res / v * (v - 1);
        return res;
    }
}

```

```

int solve(LL n){
    LL num = phi(n); // if n is prime, num = n - 1
    LL v = num;
    int h = sqrt(num);
    cntp = 0;
    for (int i = 2; i <= h; i++) {
        if (v % i == 0) {
            a[++cntp] = i;
            while (v % i == 0) v /= i;
        }
    }
    if (v != 1) a[++cntp] = v;
    v = num;
    for (int i = 2; i < n; i++) {
        if (gcd(n, i) != 1) continue;
        bool ok = 1;
        for (int j = 1; j <= cntp; j++) {
            if (mypow(i, v / a[j], n) == 1) {
                ok = 0; break;
            }
        }
        if (ok) return i;
    }
    return -1;
}
}root;

```

5.16 Pseudoinverse of Square matrix

```

Mat pinv(Mat m){
    Mat res = I;

    FZ(used);
    for(int i=0; i<W; i++){
        int piv = -1;
        for(int j=0; j<W; j++){
            if(used[j]) continue;
            if(abs(m.v[j][i]) > EPS){
                piv = j;
                break;
            }
        }
        if(piv == -1) continue;
        used[i] = true;
        swap(m.v[piv], m.v[i]);
        swap(res.v[piv], res.v[i]);

        ld rat = m.v[i][i];
        for(int j=0; j<W; j++){
            m.v[i][j] /= rat;
            res.v[i][j] /= rat;
        }
        for(int j=0; j<W; j++){
            if(j == i) continue;
            rat = m.v[j][i];
            for(int k=0; k<W; k++){
                m.v[j][k] -= rat * m.v[i][k];
                res.v[j][k] -= rat * res.v[i][k];
            }
        }
    }
    for(int i=0; i<W; i++){
        if(used[i]) continue;
        for(int j=0; j<W; j++){
            res.v[i][j] = 0;
        }
    }
    return res;
}

```

5.17 Theorem

5.17.1 Lucas' Theorem

For non-negative integer n, m and prime p , $\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p}$ where m_i is the i -th digit of m in base p .

5.17.2 Sum of Two Squares Thm (Legendre)

For a given positive integer n , let
 $D_1 = (\# \text{ of positive integers } d \text{ dividing } N \text{ that } 1 \equiv d \pmod{4})$
 $D_3 = (\# \text{ of positive integers } d \text{ dividing } N \text{ that } 3 \equiv d \pmod{4})$

then n can be written as a sum of two squares in exactly $R(n) = 4(D_1 - D_3)$ ways.

5.17.3 Difference of D1-D3 Thm

let $n = 2^t \cdot (p_1^{e_1} \cdot \dots \cdot p_r^{e_r}) \cdot \dots \cdot (q_1^{f_1} \cdot \dots \cdot q_s^{f_s})$
 where p_i, q_i are primes and $1 \equiv p_i \pmod{4}, 3 \equiv q_i \pmod{4}$
 then $D_1 - D_3 = \begin{cases} (e_1 + 1)(e_2 + 1) \dots (e_r + 1), & \text{if } f_i \text{ all even} \\ 0, & \text{if any } f_i \text{ is odd} \end{cases}$

5.17.4 Krush–Kuhn–Tucker Conditions

Stationarity

For maximizing $f(x)$: $\nabla f(x^*) = \sum_{i=1}^m \mu_i \nabla g_i(x^*) + \sum_{j=1}^l \lambda_j \nabla h_j(x^*)$

For minimizing $f(x)$: $-\nabla f(x^*) = \sum_{i=1}^m \mu_i \nabla g_i(x^*) + \sum_{j=1}^l \lambda_j \nabla h_j(x^*)$

Primal feasibility

$g_i(x^*) \leq 0$, for all $i = 1, \dots, m$

$h_j(x^*) = 0$, for all $j = 1, \dots, l$

Dual feasibility

$\mu_i \geq 0$, for all $i = 1, \dots, m$

Complementary slackness

$\mu_i g_i(x^*) = 0$, for all $i = 1, \dots, m$

5.17.5 Chinese remainder theorem

$x \equiv r_i \pmod{p_i}$

$N = \prod p_i$

$N_i = N/p_i$

$x \equiv \sum r_i N_i (N_i)^{-1}_{p_i} \pmod{N}$

5.17.6 Stirling Numbers(permutation $|P| = n$ with k cycles)

$S(n, k) = \text{coefficient of } x^k \text{ in } \Pi_{i=0}^{n-1} (x + i)$

5.17.7 Stirling Numbers(Partition n elements into k non-empty set)

$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$

5.17.8 Pick' s Theorem

$A = I + O/2 - 1$

5.17.9 Kirchhoff's theorem

$A_{ii} = \deg(i), A_{ij} = (i, j) \in E ? - 1 : 0$, Deleting any one row, one column, and cal the $\det(A)$

6 Geometry

6.1 Point operators

```
typedef double type;
typedef pair<type, type> Pt;
typedef pair<Pt, Pt> Line;
typedef pair<Pt, type> Circle;
#define X first
#define Y second
#define O first
#define R second
Pt operator+(const Pt &p1, const Pt &p2) {
    return { p1.X + p2.X, p1.Y + p2.Y };
}
Pt operator-(const Pt &p1, const Pt &p2) {
    return { p1.X - p2.X, p1.Y - p2.Y };
}
Pt operator*(const Pt &p1, const type &p2) {
    return { p1.X*p2, p1.Y*p2 };
}
Pt operator/(const Pt &p1, const type &p2) {
    return { p1.X/p2, p1.Y/p2 };
}
type operator*(const Pt &p1, const Pt &p2) {
```

```
    return p1.X*p2.X + p1.Y*p2.Y ;
}
type operator^(const Pt &p1, const Pt &p2) {
    return p1.X*p2.Y - p1.Y*p2.X ;
}
type norm2(const Pt &p1) {
    return p1*p1;
}
double norm(const Pt &p1) {
    return sqrt(p1*p1);
}
Pt perp(const Pt &p1) {
    return { -p1.Y, p1.X };
}
```

6.2 Intersection of two circles

```
vector<Pt> interCircle(Pt o1, D r1, Pt o2, D r2) {
    if( norm(o1 - o2) > r1 + r2 ) return {};
    if( norm(o1 - o2) < max(r1, r2) - min(r1, r2) )
        return {};
    D d2 = (o1 - o2) * (o1 - o2);
    D d = sqrt(d2);
    if(d > r1 + r2) return {};
    Pt u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
    D A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d));
    Pt v = Pt(o1.Y - o2.Y, -o1.X + o2.X) * A / (2*d2);
    return {u+v, u-v};
}
```

6.3 Intersection of two lines

```
Pt interPnt(Pt p1, Pt p2, Pt q1, Pt q2){
    double f1 = (p2 - p1) ^ (q1 - p1);
    double f2 = (p2 - p1) ^ (p1 - q2);
    double f = (f1 + f2);
    if( fabs( f ) < eps ) return Pt( nan(""), nan("") );
    return q1 * (f2 / f) + q2 * (f1 / f);
}
```

6.4 Intersection of two segments

```
int ori(const Pt &o, const Pt &a, const Pt &b) {
    LL ret = (a - o) ^ (b - o);
    return (ret > 0) - (ret < 0);
}
// p1 == p2 || q1 == q2 need to be handled
bool banana(const Pt &p1, const Pt &p2,
             const Pt &q1, const Pt &q2) {
    if( ( (p2 - p1) ^ (q2 - q1) ) == 0 ) { // parallel
        if( ori(p1, p2, q1) ) return false;
        return ( (p1 - q1) * (p2 - q1) ) <= 0 ||
                ( (p1 - q2) * (p2 - q2) ) <= 0 ||
                ( (q1 - p1) * (q2 - p1) ) <= 0 ||
                ( (q1 - p2) * (q2 - p2) ) <= 0;
    }
    return (ori(p1, p2, q1) * ori(p1, p2, q2) <= 0) &&
           (ori(q1, q2, p1) * ori(q1, q2, p2) <= 0);
}
```

6.5 Intersection of circle and line

```
// p1, p2 should not be zero vector
bool Inter(const Pt &p1, const Pt &p2, Circ &cc) {
    Pt dp = p2 - p1;
    double a = dp * dp;
    double b = 2 * ( dp * (p1 - cc.O) );
    double c = cc.O * cc.O + p1 * p1 - 2 * ( cc.O * p1 )
                - cc.R * cc.R;
    double bb4ac = b * b - 4 * a * c;
    return !( fabs( a ) < eps || bb4ac < 0 );
}
```

6.6 Tangent line of two circles

```
vector<Line> go( const Cir& c1 , const Cir& c2 , int
    sign1 ){
    // sign1 = 1 for outer tang, -1 for inter tang
    vector<Line> ret;
    double d_sq = norm2( c1.O - c2.O );
    if( d_sq < eps ) return ret;
    double d = sqrt( d_sq );
    Pt v = ( c2.O - c1.O ) / d;
    double c = ( c1.R - sign1 * c2.R ) / d;
    if( c * c > 1 ) return ret;
    double h = sqrt( max( 0.0 , 1.0 - c * c ) );
    for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
        Pt n = { v.X * c - sign2 * h * v.Y ,
                v.Y * c + sign2 * h * v.X };
        Pt p1 = c1.O + n * c1.R;
        Pt p2 = c2.O + n * ( c2.R * sign1 );
        if( fabs( p1.X - p2.X ) < eps and
            fabs( p1.Y - p2.Y ) < eps )
            p2 = p1 + perp( c2.O - c1.O );
        ret.push_back( { p1 , p2 } );
    }
    return ret;
}
```

6.7 Circle cover

```
#define N 1021
struct CircleCover{
    int C; Circ c[ N ];
    bool g[ N ][ N ], overlap[ N ][ N ];
    // Area[i] : area covered by at least i circles
    // O(n*nlog(n))
    D Area[ N ];
    void init( int _C ){ C = _C; }
    bool CCinter(Circ &a, Circ &b, Pt &p1, Pt &p2){
        Pt o1 = a.O, o2 = b.O;
        D r1 = a.R , r2 = b.R;
        if (norm(o1 - o2) > r1 + r2) return {};
        if (norm(o1 - o2) < max(r1, r2) - min(r1, r2))
            return {};
        D d2 = (o1 - o2)*(o1 - o2);
        D d = sqrt(d2);
        if( d > r1 + r2 ) return false;
        Pt u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
        D A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d));
        Pt v = Pt(o1.Y - o2.Y, -o1.X + o2.X) * A / (2*d2);
        p1 = u + v; p2 = u - v;
        return true;
    }
    struct Teve {
        Pt p; D ang; int add;
        Teve() {}
        Teve(Pt _a, D _b, int _c) :
            p(_a), ang(_b), add(_c) {}
        bool operator < (const Teve &a) const {
            return ang < a.ang;
        }
    }
    eve[ N * 2 ];
    // strict: x = 0, otherwise x = -1
    bool disjuct( Circ& a, Circ &b, int x ) {
        return sign( norm(a.O - b.O) - a.R - b.R ) > x;
    }
    bool contain( Circ& a, Circ &b, int x ) {
        return sign( a.R - b.R - norm(a.O - b.O) ) > x;
    }
    bool contain(int i, int j){
        /* c[j] is non-strictly in c[i]. */
        return (sign(c[i].R - c[j].R) > 0 ||
                (sign(c[i].R - c[j].R) == 0 && i < j) ) &&
            contain(c[i], c[j], -1);
    }
    void solve(){
        for(int i = 0; i <= C + 1; i++){
            Area[ i ] = 0;
            for(int i = 0; i < C; i++){
                for(int j = 0; j < C; j++){
                    overlap[i][j] = contain(i, j);
                }
            }
            for(int i = 0; i < C; i++){
                for(int j = 0; j < C; j++){

```

```
                    g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                                disjuct(c[i], c[j], -1));
                }
            }
            for(int i = 0; i < C; i++){
                int E = 0, cnt = 1;
                for(int j = 0; j < C; j++){
                    if(j != i && overlap[j][i])
                        cnt++;
                }
                for(int j = 0; j < C; j++){
                    if(i != j && g[i][j]){
                        Pt aa, bb;
                        CCinter(c[i], c[j], aa, bb);
                        D A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
                        D B = atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
                        eve[E++] = Teve(bb, B, 1);
                        eve[E++] = Teve(aa, A, -1);
                        if(B > A) cnt++;
                    }
                }
                if(E == 0) Area[cnt] += pi * c[i].R * c[i].R;
                else{
                    sort(eve, eve + E);
                    eve[E] = eve[0];
                    for(int j = 0; j < E; j++){
                        cnt += eve[j].add;
                        Area[cnt] += (eve[j].p ^ eve[j + 1].p) * 0.5;
                        D theta = eve[j + 1].ang - eve[j].ang;
                        if (theta < 0) theta += 2.0 * PI;
                        Area[cnt] +=
                            (theta - sin(theta)) * c[i].R * c[i].R * 0.5;
                    }
                }
            }
        }
    }
};
```

6.8 Half plane intersection

```
Pt interPnt(Line l1, Line l2, bool &res){
    Pt p1, p2, q1, q2;
    tie(p1, p2) = l1; tie(q1, q2) = l2;
    double f1 = (p2 - p1) ^ (q1 - p1);
    double f2 = (p2 - p1) ^ (p1 - q2);
    double f = (f1 + f2);
    if(fabs(f) < eps){
        res = 0; return {0, 0};
    }
    res = true;
    return q1 * (f2 / f) + q2 * (f1 / f);
}
bool isin(Line l0, Line l1, Line l2){
    // Check inter(l1, l2) in l0
    bool res; Pt p = interPnt(l1, l2, res);
    return ( (l0.S - l0.F) ^ (p - l0.F) ) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (L.S - L.F) ^ (p - L.F) > 0
 */
/* --^-- Line.FI --^-- Line.SE --^-- */
vector<Line> halfPlaneInter(vector<Line> lines){
    int sz = SZ(lines);
    vector<double> ata(sz), ord(sz);
    for(int i = 0; i < sz; i++){
        ord[i] = i;
        Pt d = lines[i].S - lines[i].F;
        ata[i] = atan2(d.Y, d.X);
    }
    sort( ALL(ord), [&](int i, int j) {
        if(fabs(ata[i] - ata[j]) < eps)
            return ( (lines[i].S - lines[i].F) ^
                    (lines[j].S - lines[i].F) ) < 0;
        return ata[i] < ata[j];
    });
    vector<Line> fin;
    for (int i = 0; i < sz; i++){
        if (!i || fabs(ata[ord[i]] - ata[ord[i-1]]) > eps)
            fin.pb(lines[ord[i]]);
        deque<Line> dq;
        for (int i = 0; i < SZ(fin); i++){
            while (SZ(dq) >= 2 && !isin(fin[i], dq[SZ(dq) - 2],
                dq[SZ(dq) - 1]))
                dq.pop_back();
            while (SZ(dq) >= 2 && !isin(fin[i], dq[0], dq[1]))

```



```

    dq.pop_front();
    dq.pb(fin[i]);
}
while (SZ(dq) >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq
[SZ(dq) - 1]))
    dq.pop_back();
while (SZ(dq) >= 3 && !isin(dq[SZ(dq) - 1], dq[0], dq
[1]))
    dq.pop_front();
vector<Line> res(ALL(dq));
return res;
}

```

6.9 Poly union area

```

#define eps 1e-8
class PY{ public:
    int n;
    Pt pt[5];
    Pt& operator[](const int x){ return pt[x]; }
    void input(){
        int n = 4;
        for(int i = 0; i < n; i++)
            scanf("%lf %lf", &pt[i].x, &pt[i].y);
    }
    double getArea(){
        double s = pt[n-1]^pt[0];
        for(int i = 0; i < n-1; i++)
            s += pt[i]^pt[i+1];
        return s/2;
    }
};
PY py[500];
pair<double,int> c[5000];
inline double segP(Pt &p, Pt &p1, Pt &p2) {
    if(SG(p1.x-p2.x) == 0) return (p.y-p1.y)/(p2.y-p1.y);
    return (p.x-p1.x) / (p2.x-p1.x);
}
double polyUnion(int n){
    int i,j,ii,jj,ta,tb,r,d;
    double z,w,s,sum,tc,td;
    for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
    sum=0;
    for(i=0;i<n;i++){
        for(ii=0;ii<py[i].n;ii++){
            r=0;
            c[r++]=make_pair(0.0,0);
            c[r++]=make_pair(1.0,0);
            for(j=0;j<n;j++){
                if(i==j) continue;
                for(jj=0;jj<py[j].n;jj++){
                    ta=SG(tri(py[i][ii],py[i][ii+1],py[j][jj]));
                    tb=SG(tri(py[i][ii],py[i][ii+1],py[j][jj+1]));
                    if(ta==0 && tb==0){
                        if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[i][ii])>0 && j<i){
                            c[r++]=make_pair(segP(py[j][jj],py[i][ii],py[i][ii+1]),1);
                            c[r++]=make_pair(segP(py[j][jj+1],py[i][ii],py[i][ii+1]),-1);
                        }
                    } else if(ta>0 && tb<0){
                        tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
                        td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
                        c[r++]=make_pair(tc/(tc-td),1);
                    } else if(ta<0 && tb>0){
                        tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
                        td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
                        c[r++]=make_pair(tc/(tc-td),-1);
                    }
                }
            }
        }
        sort(c,c+r);
        z=min(max(c[0].first,0.0),1.0);
        d=c[0].second; s=0;
        for(j=1;j<r;j++){
            w=min(max(c[j].first,0.0),1.0);
            if(!d) s+=w-z;
            d+=c[j].second; z=w;
        }
        sum+=(py[i][ii]^py[i][ii+1])*s;
    }
}

```

```

    return sum/2;
}
int main(){
    int n,i,j,k;
    double sum,ds;
    int n; scanf("%d", &n); sum = 0;
    for (int i = 0; i < n; i++) {
        py[i].input();
        ds = py[i].getArea();
        if(ds<0){
            for(j=0,k=py[i].n-1;j<k;j++,k--) swap(py[i][j],
                py[i][k]);
            ds=-ds;
        } sum+=ds;
    } printf("%.9f\n",sum/polyUnion(n));
}

```

6.10 2D Convex hull

```

double cross(Pt o, Pt a, Pt b) {
    return (a - o) ^ (b - o);
}
vector<Pt> convex_hull(vector<Pt> pt) {
    sort( ALL(pt) );
    int top = 0;
    vector<Pt> stk(2*SZ(pt));
    for (int i = 0; i < SZ(pt); i++) {
        while (top >= 2 && cross(stk[top-2], stk[top-1], pt
[i])<= 0)
            top--;
        stk[top++] = pt[i];
    }
    for (int i = SZ(pt) - 2, t = top + 1; i >= 0; i--) {
        while (top >= t && cross(stk[top-2], stk[top-1], pt
[i])<= 0)
            top--;
        stk[top++] = pt[i];
    }
    stk.resize(top-1);
    return stk;
}

```

6.11 Convex hull trick

```

/* Given a convexhull, answer queries in O(\lg N)
CH should not contain identical points, the area should
be > 0, min pair(x, y) should be listed first */
double det( const Pt& p1 , const Pt& p2 )
{ return p1.X * p2.Y - p1.Y * p2.X; }
struct Conv{
    int n;
    vector<Pt> a;
    vector<Pt> upper, lower;
    Conv(vector<Pt> _a) : a(_a){
        n = a.size();
        int ptr = 0;
        for(int i=1; i<n; ++i) if (a[ptr] < a[i]) ptr = i;
        for(int i=0; i<=ptr; ++i) lower.push_back(a[i]);
        for(int i=ptr; i<n; ++i) upper.push_back(a[i]);
        upper.push_back(a[0]);
    }
    int sign( LL x ){ // fixed when changed to double
        return x < 0 ? -1 : x > 0; }
    pair<LL,int> get_tang(vector<Pt> &conv, Pt vec){
        int l = 0, r = (int)conv.size() - 2;
        for( ; l + 1 < r; ){
            int mid = (l + r) / 2;
            if(sign(det(conv[mid+1]-conv[mid],vec))>0)r=mid;
            else l = mid;
        }
        return max(make_pair(det(vec, conv[r]), r),
            make_pair(det(vec, conv[0]), 0));
    }
    void upd_tang(const Pt &p, int id, int &i0, int &i1){
        if(det(a[i0] - p, a[id] - p) > 0) i0 = id;
        if(det(a[i1] - p, a[id] - p) < 0) i1 = id;
    }
    void bi_search(int l, int r, Pt p, int &i0, int &i1){
        if(l == r) return;
        upd_tang(p, l % n, i0, i1);
        int sl=sign(det(a[l % n] - p, a[(l + 1) % n] - p));
        for( ; l + 1 < r; ) {

```

```

    int mid = (l + r) / 2;
    int smid = sign(det(a[mid%n]-p, a[(mid+1)%n]-p));
    if (smid == s1) l = mid;
    else r = mid;
}
upd_tang(p, r % n, i0, i1);
}
int bi_search(Pt u, Pt v, int l, int r) {
    int s1 = sign(det(v - u, a[l % n] - u));
    for( ; l + 1 < r; ) {
        int mid = (l + r) / 2;
        int smid = sign(det(v - u, a[mid % n] - u));
        if (smid == s1) l = mid;
        else r = mid;
    }
    return l % n;
}
// 1. whether a given point is inside the CH
bool contain(Pt p) {
    if (p.X < lower[0].X || p.X > lower.back().X)
        return 0;
    int id = lower_bound(lower.begin(), lower.end(), Pt
        (p.X, -INF)) - lower.begin();
    if (lower[id].X == p.X) {
        if (lower[id].Y > p.Y) return 0;
    } else if (det(lower[id-1]-p, lower[id]-p) < 0) return 0;
    id = lower_bound(upper.begin(), upper.end(), Pt(p.X
        , INF), greater<Pt>()) - upper.begin();
    if (upper[id].X == p.X) {
        if (upper[id].Y < p.Y) return 0;
    } else if (det(upper[id-1]-p, upper[id]-p) < 0) return 0;
    return 1;
}
// 2. Find 2 tang pts on CH of a given outside point
// return true with i0, i1 as index of tangent points
// return false if inside CH
bool get_tang(Pt p, int &i0, int &i1) {
    if (contain(p)) return false;
    i0 = i1 = 0;
    int id = lower_bound(lower.begin(), lower.end(), p)
        - lower.begin();
    bi_search(0, id, p, i0, i1);
    bi_search(id, (int)lower.size(), p, i0, i1);
    id = lower_bound(upper.begin(), upper.end(), p,
        greater<Pt>()) - upper.begin();
    bi_search((int)lower.size() - 1, (int)lower.size()
        - 1 + id, p, i0, i1);
    bi_search((int)lower.size() - 1 + id, (int)lower.
        size() - 1 + (int)upper.size(), p, i0, i1);
    return true;
}
// 3. Find tangent points of a given vector
// ret the idx of vertex has max cross value with vec
int get_tang(Pt vec) {
    pair<LL, int> ret = get_tang(upper, vec);
    ret.second = (ret.second + (int)lower.size() - 1) % n;
    ret = max(ret, get_tang(lower, vec));
    return ret.second;
}
// 4. Find intersection point of a given line
// return 1 and intersection is on edge (i, next(i))
// return 0 if no strictly intersection
bool get_intersection(Pt u, Pt v, int &i0, int &i1) {
    int p0 = get_tang(u - v), p1 = get_tang(v - u);
    if (sign(det(v-u, a[p0]-u)) * sign(det(v-u, a[p1]-u)) < 0) {
        if (p0 > p1) swap(p0, p1);
        i0 = bi_search(u, v, p0, p1);
        i1 = bi_search(u, v, p1, p0 + n);
        return 1;
    }
    return 0;
}
};

```

6.12 KDTree (Nearest Point)

```

struct KDTree {
    static const int MXN = (int)1e5 + 7;
    struct Node {
        int x, y, x1, y1, x2, y2;
        int id, f;
        Node *L, *R;
    } tree[MXN];
    int n;

```

```

    Node *root;
    LL dis2(int x1, int y1, int x2, int y2) {
        LL dx = x1 - x2;
        LL dy = y1 - y2;
        return dx * dx + dy * dy;
    }
    static bool cmpx(Node &a, Node &b) { return a.x < b.x; }
    static bool cmpy(Node &a, Node &b) { return a.y < b.y; }
    void init(vector<PII> ip) {
        n = SZ(ip);
        for (int i = 0; i < n; i++) {
            tree[i].id = i;
            tree[i].x = ip[i].F;
            tree[i].y = ip[i].S;
        }
        root = build_tree(0, n-1, 0);
    }
    Node *build_tree(int L, int R, int dep) {
        if (L > R) return NULL;
        int M = (L + R) >> 1;
        tree[M].f = dep % 2;
        nth_element(tree+L, tree+M, tree+R+1, tree[M].f ?
            cmpy : cmpx);
        tree[M].x1 = tree[M].x2 = tree[M].x;
        tree[M].y1 = tree[M].y2 = tree[M].y;

        tree[M].L = build_tree(L, M-1, dep+1);
        if (tree[M].L) {
            tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
        }
        tree[M].R = build_tree(M+1, R, dep+1);
        if (tree[M].R) {
            tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
        }
        return tree + M;
    }
    int touch(Node *r, int x, int y, LL d2) {
        LL dis = sqrt(d2) + 1;
        if (x < r->x1 - dis || x > r->x2 + dis ||
            y < r->y1 - dis || y > r->y2 + dis)
            return 0;
        return 1;
    }
    void nearest(Node *r, int x, int y,
        int &mID, LL &md2) {
        if (!r || !touch(r, x, y, md2)) return;
        LL d2 = dis2(r->x, r->y, x, y);
        if (d2 < md2 || (d2 == md2 && mID < r->id)) {
            mID = r->id;
            md2 = d2;
        }
        // search order depends on split dim
        if ((r->f == 0 && x < r->x) ||
            (r->f == 1 && y < r->y)) {
            nearest(r->L, x, y, mID, md2);
            nearest(r->R, x, y, mID, md2);
        } else {
            nearest(r->R, x, y, mID, md2);
            nearest(r->L, x, y, mID, md2);
        }
    }
    int query(int x, int y) {
        int id = 1029384756;
        LL d2 = 102938475612345678LL;
        nearest(root, x, y, id, d2);
        return id;
    }
} tree;

```

6.13 Triangle

```

Pt inCenter( Pt &A, Pt &B, Pt &C) { // 内心
    double a = norm(B-C), b = norm(C-A), c = norm(A-B);
    return (A * a + B * b + C * c) / (a + b + c);
}

```

```

Pt circumCenter( Pt &a, Pt &b, Pt &c) { // 外心
    Pt bb = b - a, cc = c - a;
    double db=norm2(bb), dc=norm2(cc), d=2*(bb ^ cc);
    return a-Pt(bb.Y*dc-cc.Y*db, cc.X*db-bb.X*dc) / d;
}
Pt othroCenter( Pt &a, Pt &b, Pt &c) { // 垂心
    Pt ba = b - a, ca = c - a, bc = b - c;
    double Y = ba.Y * ca.Y * bc.Y,
    A = ca.X * ba.Y - ba.X * ca.Y,
    x0 = (Y+ca.X*ba.Y*b.X-ba.X*ca.Y*c.X) / A,
    y0 = -ba.X * (x0 - c.X) / ba.Y + ca.Y;
    return Pt(x0, y0);
}

```

7 Stringology

7.1 Suffix Array

```

const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX], sa[MAX], tsa[MAX], tp[
    MAX][2];

void suffix_array(char *ip){
    int len = strlen(ip);
    int alp = 256;

    memset(ct, 0, sizeof(ct));
    for(int i=0;i<len;i++) ct[ip[i]+1]++;
    for(int i=1;i<alp;i++) ct[i]+=ct[i-1];
    for(int i=0;i<len;i++) rk[i]=ct[ip[i]];

    for(int i=1;i<len;i*=2){
        for(int j=0;j<len;j++){
            if(j+i>len) tp[j][1]=0;
            else tp[j][1]=rk[j+i]+1;

            tp[j][0]=rk[j];
        }
        memset(ct, 0, sizeof(ct));
        for(int j=0;j<len;j++) ct[tp[j][1]+1]++;
        for(int j=1;j<len+2;j++) ct[j]+=ct[j-1];
        for(int j=0;j<len;j++) tsa[ct[tp[j][1]]+j]=j;

        memset(ct, 0, sizeof(ct));
        for(int j=0;j<len;j++) ct[tp[j][0]+1]++;
        for(int j=1;j<len+1;j++) ct[j]+=ct[j-1];
        for(int j=0;j<len;j++) sa[ct[tp[tsa[j]][0]]+j]=tsa[j];

        rk[sa[0]]=0;
        for(int j=1;j<len;j++){
            if( tp[sa[j]][0] == tp[sa[j-1]][0] &&
               tp[sa[j]][1] == tp[sa[j-1]][1] )
                rk[sa[j]] = rk[sa[j-1]];
            else
                rk[sa[j]] = j;
        }

        for(int i=0,h=0;i<len;i++){
            if(rk[i]==0) h=0;
            else{
                int j=sa[rk[i]-1];
                h=max(0,h-1);
                for(;ip[i+h]==ip[j+h];h++);
            }
            he[rk[i]]=h;
        }
    }
}

```

7.2 Suffix Array (SAIS TWT514)

```

struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=a; i<int(b); i++ )
    static const int MXN = 300010;
    bool _t[MXN*2];
    int _s[MXN*2], _sa[MXN*2], _c[MXN*2], x[MXN], _p[
        MXN], _q[MXN*2], hei[MXN], r[MXN];
}

```

```

int operator [] (int i){ return _sa[i]; }
void build(int *s, int n, int m){
    memcpy(_s, s, sizeof(int) * n);
    sais(_s, _sa, _p, _q, _t, _c, n, m);
    mkhei(n);
}
void mkhei(int n){
    REP(i,n) r[_sa[i]] = i;
    hei[0] = 0;
    REP(i,n) if(r[i]) {
        int ans = i>0 ? max(hei[r[i-1]] - 1, 0) :
            0;
        while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans
            ++;
        hei[r[i]] = ans;
    }
}
void sais(int *s, int *sa, int *p, int *q, bool *t,
    int *c, int n, int z){
    bool uniq = t[n-1] = true, neq;
    int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s +
        n, lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
    memcpy(x, c, sizeof(int) * z); \
    XD; \
    memcpy(x + 1, c, sizeof(int) * (z - 1)); \
    REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i]
        ]-1]]++ = sa[i]-1; \
    memcpy(x, c, sizeof(int) * z); \
    for(int i = n - 1; i >= 0; i--) if(sa[i] && t[
        sa[i]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
    MS0(c, z);
    REP(i,n) uniq &= ++c[s[i]] < 2;
    REP(i,z-1) c[i+1] += c[i];
    if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return;
    }
    for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s
        [i+1] ? t[i+1] : s[i]<s[i+1]);
    MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[
        s[i]]]=p[q[i]-nn++]=i);
    REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1])
        {
            neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]
                ]+1]-sa[i])*sizeof(int));
            ns[q[lst=sa[i]]]=nmzx+=neq;
        }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn,
        nmzx + 1);
    MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[s
        [p[nsa[i]]]]] = p[nsa[i]]);
}
}sa;

void suffix_array(int* ip, int len) {
    // should padding a zero in the back
    // s is int array, n is array length
    // s[0..n-1] != 0, and s[n] = 0
    // resulting SA will be length n+1
    ip[len++] = 0;
    sa.build(ip, len, 128);
    // original 1-base
    for (int i=0; i<l; i++) {
        hei[i] = sa.hei[i + 1];
        sa[i] = sa._sa[i + 1];
    }
}

```

7.3 Aho-Corasick Algorithm

```

struct AAutomata{
    struct Node{
        int cnt,dp;
        Node *go[26], *fail;
        Node (){
            cnt = 0;
            dp = -1;
            memset(go,0,sizeof(go));
            fail = 0;
        }
    };
    Node *root, pool[1048576];
    int nMem;
}

```

```

Node* new_Node(){
    pool[nMem] = Node();
    return &pool[nMem++];
}
void init(){
    nMem = 0;
    root = new_Node();
}
void add(const string &str){
    insert(root, str, 0);
}
void insert(Node *cur, const string &str, int pos){
    if (pos >= (int)str.size()){
        cur->cnt++;
        return;
    }
    int c = str[pos] - 'a';
    if (cur->go[c] == 0){
        cur->go[c] = new_Node();
    }
    insert(cur->go[c], str, pos+1);
}
void make_fail(){
    queue<Node*> que;
    que.push(root);
    while (!que.empty()){
        Node* fr=que.front();
        que.pop();
        for (int i=0; i<26; i++){
            if (fr->go[i]){
                Node *ptr = fr->fail;
                while (ptr && !ptr->go[i]) ptr = ptr->fail;
                if (!ptr) fr->go[i]->fail = root;
                else fr->go[i]->fail = ptr->go[i];
                que.push(fr->go[i]);
            }
        }
    }
}
};

```

7.4 KMP

```

#include<bits/stdc++.h>
using namespace std;

void build_fail_function(string B, int *fail) {
    int len = B.length(), pos;
    pos = fail[0] = -1;
    for (int i = 1; i < len; i++) {
        while (pos != -1 and B[pos + 1] != B[i])
            pos = fail[pos];
        if (B[pos + 1] == B[i]) pos++;
        fail[i] = pos;
    }
}

void match(string A, string B, int *fail) {
    int lenA = A.length(), lenB = B.length();
    int pos = -1;
    for (int i = 0; i < lenA; i++) {
        while (pos != -1 and B[pos + 1] != A[i])
            pos = fail[pos];

        if (B[pos + 1] == A[i]) pos++;

        if (pos == lenB - 1) {
            // Match ! A[i - lenB + 1, i] = B
            pos = fail[pos];
        }
    }
}

```

7.5 Z value

```

void Zval(const char *s, int len, int *z) {
    z[0] = 0;
    for (int b=0, i=1; i<len; i++) {
        z[i] = max(min(z[i-b], z[b] + b - i), 0);
        while (s[i + z[i]] == s[z[i]]) z[i]++;
        if (i+z[i] > b+z[b]) b=i;
    }
}

```

```

}
}

```

7.6 Z value (palindrome ver.)

```

void Zpal(const char *s, int len, int *z) {
    // Only odd palindrome len is considered
    // z[i] means that the longest odd palindrom
    // centered at
    // i is [i-z[i] .. i+z[i]]
    z[0] = 0;
    for (int b=0, i=1; i<len; i++) {
        if (z[b] + b >= i) z[i] = min(z[2*b-i], b+z[b]-i);
        else z[i] = 0;
        while (i+z[i]+1 < len and i-z[i]-1 >= 0 and
            s[i+z[i]+1] == s[i-z[i]-1]) z[i]++;
        if (z[i] + i > z[b] + b) b = i;
    }
}

```

7.7 Lexicographically Smallest Rotation

```

string mcp(string s){
    int n = s.length();
    s += s;
    int i=0, j=1;
    while (i<n && j<n){
        int k = 0;
        while (k < n && s[i+k] == s[j+k]) k++;
        if (s[i+k] <= s[j+k]) j += k+1;
        else i += k+1;
        if (i == j) j++;
    }
    int ans = i < n ? i : j;
    return s.substr(ans, n);
}

```

7.8 Suffix Automaton

```

// par : fail link
// val : a topological order ( useful for DP )
// go[x] : automata edge ( x is integer in [0,26) )

struct SAM{
    struct State{
        int par, go[26], val;
        State () : par(0), val(0){ FZ(go); }
        State (int _val) : par(0), val(_val){ FZ(go); }
    };
    vector<State> vec;
    int root, tail;

    void init(int arr[], int len){
        vec.resize(2);
        vec[0] = vec[1] = State(0);
        root = tail = 1;
        for (int i=0; i<len; i++)
            extend(arr[i]);
    }
    void extend(int w){
        int p = tail, np = vec.size();
        vec.PB(State(vec[p].val+1));
        for ( ; p && vec[p].go[w]==0; p=vec[p].par)
            vec[p].go[w] = np;
        if (p == 0){
            vec[np].par = root;
        } else {
            if (vec[vec[p].go[w]].val == vec[p].val+1){
                vec[np].par = vec[p].go[w];
            } else {
                int q = vec[p].go[w], r = vec.size();
                vec.PB(vec[q]);
                vec[r].val = vec[p].val+1;
                vec[q].par = vec[np].par = r;
                for ( ; p && vec[p].go[w] == q; p=vec[p].par)
                    vec[p].go[w] = r;
            }
        }
    }
}

```

```
    tail = np;  
  }  
};
```