

Contents

1 Basic	1
1.1 vimrc	1
1.2 Default Code	1
2 Data Structure	1
2.1 Bigint	1
2.2 unordered_map	2
2.3 extc_heap	3
2.4 extc_balance_tree	3
2.5 Heavy Light Decomposition(PEC ver.)	3
2.6 Disjoint Set	3
2.7 Treap	4
3 Graph	4
3.1 BCC Edge	4
3.2 BCC Vertex	5
3.3 Strongly Connected Components	5
3.4 Heavy Light Decomposition	5
3.5 Maximum Clique	6
3.6 MinimumMeanCycle	6
4 Flow	6
4.1 Dinic	6
4.2 Cost Flow	7
4.3 Kuhn Munkres	7
4.4 Maximum Simple Graph Matching	8
4.5 Minimum Weight Matching (Clique version)	8
5 Math	9
5.1 ax+by=gcd	9
5.2 Segmented Sieve	9
5.3 Fast Fourier Transform	9
5.4 FFT (Pec.ver)	9
5.5 Gauss Elimination	10
5.6 Matrix	10
5.7 Miller Rabin	10
5.8 Pollard Rho	10
5.9 Theorem	11
5.9.1 Lucas' Theorem	11
5.9.2 Sum of Two Squares Thm (Legendre)	11
5.9.3 Difference of D1-D3 Thm	11
5.9.4 Krush-Kuhn-Tucker Conditions	11
5.9.5 Chinese remainder theorem	11
6 Geometry	11
6.1 Point operators	11
6.2 Intersection of two circles	11
6.3 Intersection of two lines	11
6.4 Circle cover	11
6.5 Half Plane Intersection	12
6.6 dao point	13
6.7 dao inter	13
6.8 dao 2D convex hull	13
6.9 Minimum Covering Circle	13
7 Stringology	13
7.1 Suffix Array	13
7.2 KMP	14
7.3 Z value	14
7.4 Lexicographically Smallest Rotation	14

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
se makeprg=g++\ -Wall\ -Wshadow\ -O2\ -std=c++0x\ -o\
%<\ %
au BufNewFile *.cpp 0r ~/default.cpp | :1,$-7 fo
filetype indent on

map <f6> :call CompileRunGpp(<cr>
func! CompileRunGpp()
    exec "w"
    exec "!g++ -std=c++14 % -o %<"
    exec "!./%<"
endfunc
```

1.2 Default Code

```
#include<bits/stdc++.h>
#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())
using namespace std;
typedef long long LL;
typedef double D;
typedef long double LDB;
typedef pair<int, int> PII;
typedef pair<LL, LL> PLL;
#define rep(i, n) for(int i = 0; i < n; i++)
#define rep1(i, a, b) for(int i = a; i < b; i++)
#define per1(i, a, b) for(int i = a; i >= b; i--)
#define IOS ios_base::sync_with_stdio(0); cin.tie(0)

int main(){
}
```

2 Data Structure

2.1 Bigint

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

    int s;
    int v1, v[LEN];
    // vector<int> v;
    Bigint() : s(1) { v1 = 0; }
    Bigint(long long a) {
        s = 1; v1 = 0;
        if (a < 0) { s = -1; a = -a; }
        while (a) {
            push_back(a % BIGMOD);
            a /= BIGMOD;
        }
    }
    Bigint(string str) {
        s = 1; v1 = 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;
}
};

```

2.2 unordered_map

```

struct Key {
    int first, second;
    Key () {}
    Key (int _x, int _y) : first(_x), second(_y) {}
    bool operator == (const Key &b) const {
        return tie(F,S) == tie(b.F,b.S);
    }
};

struct KeyHasher {
    size_t operator()(const Key& k) const {
        return k.first + k.second*100000;
    }
};

typedef unordered_map<Key, int, KeyHasher> map_t;

```

```
int main(int argc, char** argv){
    map_t mp;
    for (int i=0; i<10; i++)
        mp[Key(i,0)] = i+1;
    for (int i=0; i<10; i++)
        printf("%d\n", mp[Key(i,0)]);

    return 0;
}
```

2.3 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;
}
```

2.4 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);
}
```

2.5 Heavy Light Decomposition(PEC ver.)

```
#define REP(i, s, e) for(int i = (s); i <= (e); i++)
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
const int MAXN = 100010;
const int LOG = 19;
struct HLD{
    int n;
    vector<int> g[MAXN];
    int sz[MAXN], dep[MAXN];
```

```
int ts, tid[MAXN], tdi[MAXN], tl[MAXN], tr[MAXN];
// ts : timestamp , useless after yutruli
// tid[ u ] : pos. of node u in the seq.
// tdi[ i ] : node at pos i of the seq.
// tl , tr[ u ] : subtree interval in the seq. of
// node u
int prt[MAXN][LOG], head[MAXN];
// head[ u ] : head of the chain contains u
void dfssz(int u, int p){
    dep[u] = dep[p] + 1;
    prt[u][0] = p; sz[u] = 1; head[u] = u;
    for(int& v:g[u]) if(v != p){
        dep[v] = dep[u] + 1;
        dfssz(v, u);
        sz[u] += sz[v];
    }
}
void dfshl(int u){
    ts++;
    tid[u] = tl[u] = tr[u] = ts;
    tdi[tid[u]] = u;
    sort(ALL(g[u]),
        [&](int a, int b){return sz[a] > sz[b];});
    bool flag = 1;
    for(int& v:g[u]) if(v != prt[u][0]){
        if(flag) head[v] = head[u], flag = 0;
        dfshl(v);
        tr[u] = tr[v];
    }
}
inline int lca(int a, int b){
    if(dep[a] > dep[b]) swap(a, b);
    int diff = dep[b] - dep[a];
    REPD(k, LOG-1, 0) if(diff & (1<<k)){
        b = prt[b][k];
    }
    if(a == b) return a;
    REPD(k, LOG-1, 0) if(prt[a][k] != prt[b][k]){
        a = prt[a][k]; b = prt[b][k];
    }
    return prt[a][0];
}
void init( int _n ){
    n = _n; REP( i , 1 , n ) g[ i ].clear();
}
void addEdge( int u , int v ){
    g[ u ].push_back( v );
    g[ v ].push_back( u );
}
void yutruli(){
    dfssz(1, 0);
    ts = 0;
    dfshl(1);
    REP(k, 1, LOG-1) REP(i, 1, n)
        prt[i][k] = prt[prt[i][k-1]][k-1];
}
vector< PII > getPath( int u , int v ){
    vector< PII > res;
    while( tid[ u ] < tid[ head[ v ] ] ){
        res.push_back( PII(tid[ head[ v ] ] , tid[ v ] ) );
        v = prt[ head[ v ] ][ 0 ];
    }
    res.push_back( PII( tid[ u ] , tid[ v ] ) );
    reverse( ALL( res ) );
    return res;
}
/* res : List of intervals from u to v
 * u must be ancestor of v
 * usage :
 * vector< PII >& path = tree.getPath( u , v )
 * for( PII tp : path ) {
 *     int l , r; tie( l , r ) = tp;
 *     upd( l , r );
 *     uu = tree.tdi[ l ] , vv = tree.tdi[ r ];
 *     uu ~> vv is a heavy path on tree
 * }
 */
} tree;
```

2.6 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;

```

2.7 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;
}

```

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 $ax+by=gcd$

```

typedef pair<int, int> pii;

pii gcd(int a, int b){
    if(b == 0) return make_pair(1, 0);
    else{
        int p = a / b;
        pii q = gcd(b, a % b);
        return make_pair(q.second, q.first - q.second * p);
    }
}

```

5.2 Segmented Sieve

```

bool sieve[MXN];
void linear_sieve(){
    vector<int> prime;
    for(int i=2; i< MXN; ++i){
        if(!sieve[i]) prime.push_back(i);
        for(int j = 0; i*prime[j] < N; ++j){
            sieve[i*prime[j]] = true;
            if(i % prime[j] == 0) break;
        }
    }
}

```

5.3 Fast Fourier Transform

```

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, l1) (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 ;
}

```

5.4 FFT (Pec.ver)

```

// const int MAXN = 262144;
// (must be 2^k)
// before any usage, run pre_fft() first
typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acos(-1);
const cplx I(0, 1);
cplx omega[MAXN+1];
void pre_fft(){

```

```

for(int i=0; i<=MAXN; i++)
    omega[i] = exp(i * 2 * PI / MAXN * I);
}
// n must be 2^k
void fft(int n, cplx a[], bool inv=false){
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
        int mh = m >> 1;
        for (int i = 0; i < mh; i++) {
            cplx w = omega[inv ? MAXN-(i*theta%MAXN)
                        : i*theta%MAXN];
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                cplx x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
        }
        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) for (i = 0; i < n; i++) a[i] /= n;
}

```

5.5 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;
            }
        }
    }
}

```

5.6 Matrix

```

struct Mat{
    int n, m;
    LL a[MXN][MXN];
    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;
}
};

```

5.7 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 : pirms <= 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;
}

```

5.8 Pollard Rho

```

// does not work when n is prime
long long modit(long long x, long long mod) {
    if(x>=mod) x-=mod;
    //if(x<0) x+=mod;
    return x;
}
long long mult(long long x, long long y, long long mod) {
    long long s=0, m=x%mod;
    while(y) {
        if(y&1) s=modit(s+m,mod);
        y>>=1;
        m=modit(m+m,mod);
    }
    return s;
}

```

```

long long f(long long x, long long mod) {
    return modit(mult(x, x, mod) + 1, mod);
}
long long pollard_rho(long long n) {
    if(!(n&1)) return 2;
    while (true) {
        long long 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;
    }
}

```

5.9 Theorem

5.9.1 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 .

5.9.2 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.

5.9.3 Difference of D1-D3 Thm

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

5.9.4 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 $i = 1, \dots, m$

5.9.5 Chinese remainder theorem

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

6 Geometry

6.1 Point operators

```

#define x first
#define y second

#define cpdd const pdd
struct pdd : pair<double, double> {
    using pair<double, double>::pair;

    pdd operator + (cpdd &p) const {
        return {x+p.x, y+p.y};
    }

    pdd operator - ( ) const {

```

```

        return {-x, -y};
    }

    pdd operator - (cpdd &p) const {
        return (*this) + (-p);
    }

    pdd operator * (double f) const {
        return {f*x, f*y};
    }

    double operator * (cpdd &p) const {
        return x*p.x + y*p.y;
    }
};

double abs(cpdd &p) { return hypot(p.x, p.y); }
double arg(cpdd &p) { return atan2(p.y, p.x); }
double cross(cpdd &p, cpdd &q) { return p.x*q.y - p.y*q.x; }
double cross(cpdd &p, cpdd &q, cpdd &o) { return cross(p-o, q-o); }
pdd operator * (double f, cpdd &p) { return p*f; } //
!! Not f*p !!

```

6.2 Intersection of two circles

```

using ld = double;
vector<pdd> interCircle(pdd o1, double r1, pdd o2,
    double r2) {
    ld d2 = (o1 - o2) * (o1 - o2);
    ld d = sqrt(d2);
    if (d < abs(r1-r2)) return {};
    if (d > r1+r2) return {};
    pdd u = 0.5*(o1+o2) + ((r2*r2-r1*r1)/(2*d2))*(o1-o2);
    double A = sqrt((r1+r2+d) * (r1-r2+d) * (r1+r2-d) *
        (-r1+r2+d));
    pdd v = A / (2*d2) * pdd(o1.S-o2.S, -o1.F+o2.F);
    return {u+v, u-v};
}

```

6.3 Intersection of two lines

```

const double EPS = 1e-9;

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2, bool &res)
{
    double f1 = cross(p2, q1, p1);
    double f2 = -cross(p2, q2, p1);
    double f = (f1 + f2);

    if (fabs(f) < EPS) {
        res = false;
        return {};
    }

    res = true;
    return (f2 / f) * q1 + (f1 / f) * q2;
}

```

6.4 Circle cover

```

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& tp, const type& tk ){
    return { tp.X * tk, tp.Y * tk };
}

```

```

Pt operator/( const Pt& tp , const type& tk ){
    return { tp.X / tk , tp.Y / tk };
}
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& tp ){
    return tp * tp;
}
double norm( const Pt& tp ){
    return sqrt( norm2( tp ) );
}
Pt perp( const Pt& tp ){
    return { tp.Y , -tp.X };
}

#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
    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 false;
        if( norm( o1 - o2 ) < max(r1, r2) - min(r1, r2) )
            return true;
        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*d));
        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*d);
        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;
        }
    }
    Teve[ N * 2 ];
    // strict: x = 0, otherwise x = -1
    bool disjunct( 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] ||
                    disjunct(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) * .5;
                    D theta = eve[j + 1].ang - eve[j].ang;
                    if( theta < 0 ) theta += 2. * pi;
                    Area[cnt] +=
                        (theta - sin(theta)) * c[i].R * c[i].R * .5;
                }
            }
        }
    }
};

```

6.5 Half Plane Intersection

```

const double EPS = 1e-9;

pdd interPnt(Line l1, Line l2, bool &res){
    pdd p1, p2, q1, q2;
    tie(p1, p2) = l1;
    tie(q1, q2) = l2;
    double f1 = cross(p2, q1, p1);
    double f2 = -cross(p2, q2, p1);
    double f = (f1 + f2);

    if(fabs(f) < EPS) {
        res = false;
        return {0, 0};
    }

    res = true;
    return (f2 / f) * q1 + (f1 / f) * q2;
}

bool isin(Line l0, Line l1, Line l2) {
    // Check inter(l1, l2) in l0
    bool res;
    pdd p = interPnt(l1, l2, res);
    return cross(l0.S, 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).cross(p - L.F) >
 * 0
 */
vector<Line> halfPlaneInter(vector<Line> lines) {
    int sz = lines.size();
    vector<double> ata(sz), ord(sz);
    for( int i=0; i<sz; i++) {
        ord[i] = i;
        pdd d = lines[i].S - lines[i].F;
        ata[i] = atan2(d.y, d.x);
    }
    sort(ALL(ord), [&](int i, int j) {
        if (abs(ata[i] - ata[j]) < EPS) {
            return cross(lines[i].S, lines[j].S, lines[
                i].F) < 0;
        }
        return ata[i] < ata[j];
    });
    vector<Line> fin;
    for( int i=0; i<sz; i++) {
        if (!i or 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 and
            not isin(fin[i], dq[SZ(dq)-2], dq[SZ(dq)-1])) {
            dq.pop_back();
        }
        while(SZ(dq) >= 2 and
            not isin(fin[i], dq[0], dq[1])) {
            dq.pop_front();
        }
        dq.push_back(fin[i]);
    }
}

```

```

while (SZ(dq) >= 3 and
      not isin(dq[0], dq[SZ(dq)-2], dq[SZ(dq)-1]))
{
    dq.pop_back();
}

while (SZ(dq) >= 3 and
      not isin(dq[SZ(dq)-1], dq[0], dq[1])) {
    dq.pop_front();
}
vector<Line> res(ALL(dq));
return res;
}

```

6.6 dao point

```

typedef double Type;
struct Point{
    Type x, y;
    Point(){};
    Point(Type _x, Type _y){
        x = _x, y = _y;
    }
    void read(){
        scanf("%Lf %Lf", &x, &y);
    }
    Point operator +(const Point & P2){
        return Point(x + P2.x, y + P2.y);
    }
    Point operator -(const Point & P2){
        return Point(x - P2.x, y - P2.y);
    }
    Point operator *(const Type & Len){
        return Point(x*Len, y*Len);
    }
    Type operator *(const Point & P2){
        return x*P2.x + y*P2.y;
    }
    Type operator ^(const Point & P2){
        return x*P2.y - y*P2.x;
    }
    Type dis(){
        return x*x+y*y;
    }
};
struct Line{
    Point s, e;
    Line(){};
    Line(Point _s, Point _e){
        s = _s, e = _e;
    }
    void read(){
        s.read(); e.read();
    }
};

```

6.7 dao inter

```

Point inter(Line l1, Line l2){
    Type v1 = (l1.s - l1.e) ^ (l2.s - l1.e);
    Type v2 = (l1.s - l1.e) ^ (l1.e - l2.e);
    Type v3 = (v1 + v2);
    if(v3 + eps > 0 && v3 - eps < 0) return Point(nan(""),
        , nan(""));
    return l2.s*(v2/v3) + l2.e*(v1/v3);
}

```

6.8 dao 2D convex hull

```

int ori(Point s, Point e, Point P){
    Type val = (s - e)^(P - e);
    if(fabs(val) < eps) return 0;
    else if(val > 0) return 1;
    else return -1;
}
bool cmp(Point a, Point b){
    if(a.x != b.x) return a.x < b.x;
    return a.y < b.y;
}

```

```

}
vector<Point> convex_hull(vector<Point> pt){
    sort(pt.begin(), pt.end(), cmp);
    int top=0;
    vector<Point> stk(2*pt.size());
    for (int i=0; i<(int)pt.size(); i++){
        while (top >= 2 && ori(stk[top-2],stk[top-1],pt[i])
            >= 0)
            top--;
        stk[top++] = pt[i];
    }
    for (int i=pt.size()-2, t=top+1; i>=0; i--){
        while (top >= t && ori(stk[top-2],stk[top-1],pt[i])
            >= 0)
            top--;
        stk[top++] = pt[i];
    }
    stk.resize(top-1);
    return stk;
}

```

6.9 Minimum Covering Circle

```

struct Mcc{
    // return pair of center and r^2
    static const int MAXN = 1000100;
    int n;
    pdd p[MAXN],cen;
    double r2;

    void init(int _n, pdd _p[]){
        n = _n;
        memcpy(p,_p,sizeof(pdd)*n);
    }
    double sqr(double a){ return a*a; }
    double abs2(pdd a){ return a*a; }
    pdd center(pdd p0, pdd p1, pdd p2) {
        pdd a = p1-p0;
        pdd b = p2-p0;
        double c1=abs2(a)*0.5;
        double c2=abs2(b)*0.5;
        double d = a % b;
        double x = p0.x + (c1 * b.y - c2 * a.y) / d;
        double y = p0.y + (a.x * c2 - b.x * c1) / d;
        return pdd(x,y);
    }

    pair<pdd,double> solve(){
        random_shuffle(p,p+n);
        r2=0;
        for (int i=0; i<n; i++){
            if (abs2(cen-p[i]) <= r2) continue;
            cen = p[i];
            r2 = 0;
            for (int j=0; j<i; j++){
                if (abs2(cen-p[j]) <= r2) continue;
                cen = 0.5 * (p[i]+p[j]);
                r2 = abs2(cen-p[j]);
                for (int k=0; k<j; k++){
                    if (abs2(cen-p[k]) <= r2) continue;
                    cen = center(p[i],p[j],p[k]);
                    r2 = abs2(cen-p[k]);
                }
            }
        }
        return {cen,r2};
    }
}mcc;

```

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]]+1] = 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]]+1] = 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;
}
}

```

```

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;
}
}

```

7.4 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);
}

```

7.2 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];
        }
    }
}

```

7.3 Z value

```

void Zval(const char *s, int len, int *z) {
    z[0] = 0;

```