

# Contents

1	Basic	1	7	Stringology	21
1.1	vimrc	1	7.1	Hash	21
1.2	Debug Macro	1	7.2	Suffix Array	21
1.3	Increase Stack	1	7.3	Suffix Automaton	22
1.4	Pragma Optimization	2	7.4	KMP	22
1.5	IO Optimization	2	7.5	Z value	22
2	Data Structure	2	7.6	Manacher	22
2.1	Dark Magic	2	7.7	Lexico Smallest Rotation	23
2.2	Link-Cut Tree	2	7.8	BWT	23
2.3	LiChao Segment Tree	2	7.9	Palindromic Tree	23
2.4	Treap	3	8	Misc	23
2.5	Sparse Table	3	8.1	Theorems	23
2.6	Linear Basis	3	8.1.1	Kirchhoff's Theorem	23
2.7	Binary Search On Segment Tree	3	8.1.2	Tutte's Matrix	23
3	Graph	4	8.1.3	Cayley's Formula	23
3.1	BCC Edge	4	8.1.4	Erdős-Gallai theorem	23
3.2	BCC Vertex	4	8.1.5	Havel-Hakimi algorithm	23
3.3	2-SAT (SCC)	4	8.1.6	Hall's marriage theorem	23
3.4	Lowbit Decomposition	5	8.1.7	Euler's planar graph formula	23
3.5	MaxClique	5	8.1.8	Pick's theorem	23
3.6	MaxCliqueDyn	6	8.1.9	Lucas's theorem	23
3.7	Virtual Tree	6	8.1.10	Matroid Intersection	24
3.8	Centroid Decomposition	6	8.2	DP-opt Condition	24
3.9	Tree Hashing	7	8.2.1	totally monotone (concave/convex)	24
3.10	Minimum Mean Cycle	7	8.2.2	monge condition (concave/convex)	24
3.11	Mo's Algorithm on Tree	7	8.3	Convex 1D/1D DP	24
3.12	Minimum Steiner Tree	7	8.4	ConvexHull Optimization	24
3.13	Directed Minimum Spanning Tree	8	8.5	Josephus Problem	24
3.14	Manhattan Minimum Spanning Tree	8	8.6	Cactus Matching	24
3.15	Dominator Tree	8	8.7	DLX	25
3.16	Edge Coloring	9	8.8	Tree Knapsack	25
4	Matching & Flow	9	8.9	N Queens Problem	25
4.1	Kuhn Munkres	9	8.10	Aliens Optimization	25
4.2	Bipartite Matching	9	8.11	Hilbert Curve	26
4.3	General Graph Matching	10	8.12	Binary Search On Fraction	26
4.4	Minimum Weight Matching (Clique version)	10	1	Basic	
4.5	Minimum Cost Circulation	11	1.1	vimrc	
4.6	Flow Models	11		se is nu bs=2 ru mouse=a encoding=utf-8 ls=2	
4.7	Dinic	11		se cin cino+=j1 et sw=4 sts=4 tgc sc hls	
4.8	Minimum Cost Maximum Flow	12		syn on	
4.9	GomoryHu Tree	12		colorscheme desert	
4.10	Global Min-Cut	12		filetype indent on	
5	Math	12		inoremap {<CR> {<CR>}<ESC>O	
5.1	$\lfloor \frac{n}{2} \rfloor$ Enumeration	12		map <F8> <ESC>:w<CR>:!g++ "%" -o "%<" -std=c++17 -	
5.2	ax+by=gcd	12		DKISEKI -Wall -Wextra -Wshadow -Wfatal-errors -	
5.3	Pollard Rho	12		Wconversion -fsanitize=address,undefined -g && echo	
5.4	Pi Count (Linear Sieve)	12		success<CR>	
5.5	Stirling Number	13		map <F9> <ESC>:w<CR>:!g++ "%" -o "%<" -O2 -std=c++17 &&	
5.5.1	First Kind	13		echo success<CR>	
5.5.2	Second Kind	13		map <F10> <ESC>:!. / "%<" <CR>	
5.6	Range Sieve	13	1.2	Debug Macro	
5.7	Miller Rabin	13		#ifdef KISEKI	
5.8	Extended Euler	13		#define safe cerr<<__PRETTY_FUNCTION__\	
5.9	Gauss Elimination	13		<<" line "<<__LINE__<<" safe\n"	
5.10	Fast Fourier Transform	14		#define debug(a...) qwerty(#a, a)	
5.11	Chinese Remainder	14		#define orange(a...) dvorak(#a, a)	
5.12	Berlekamp Massey	14		using std::cerr;	
5.13	NTT	15		template <typename ...T>	
5.14	Polynomial Operations	15		void qwerty(const char *s, T ...a) {	
5.15	FWT	16		cerr << "\e[1;32m(" << s << ")" = (";	
5.16	DiscreteLog	16		int cnt = sizeof...(T);	
5.17	FloorSum	16		(..., (cerr << a << (--cnt ? ", " : ") \e[0m\n"));	
5.18	ExtendedFloorSum	16		}	
5.19	Quadratic residue	17		template <typename Iter>	
5.20	De-Bruijn	17		void dvorak(const char *s, Iter L, Iter R) {	
5.21	Simplex Construction	17		cerr << "\e[1;32m[" << s << "]" = [";	
5.22	Simplex	17		for (int f = 0; L != R; ++L)	
5.23	Characteristic Polynomial	18		cerr << (f++ ? ", " : "") << *L;	
5.24	Partition Number	18		cerr << " ] \e[0m\n";	
6	Geometry	18		}	
6.1	Basic Geometry	18		#else	
6.2	2D Convex Hull	18		#define safe ((void)0)	
6.3	3D Convex Hull	18		#define debug(...) ((void)0)	
6.4	2D Farthest Pair	18		#define orange(...) ((void)0)	
6.5	2D Closest Pair	19		#endif	
6.6	kD Closest Pair (3D ver.)	19	1.3	Increase Stack	
6.7	Simulated Annealing	19		const int size = 256 << 20;	
6.8	Half Plane Intersection	19		register long rsp asm("rsp");	
6.9	Minkowski Sum	19		char *p = (char*)malloc(size)+size, *bak = (char*)rsp;	
6.10	Intersection of line and Circle	20		--asm__("movq %0, %%rsp\n"::"r"(p));	
6.11	Intersection of Polygon and Circle	20		// main	
6.12	Intersection of Two Circle	20		--asm__("movq %0, %%rsp\n"::"r"(bak));	
6.13	Tangent line of Two Circle	20			
6.14	Minimum Covering Circle	21			
6.15	KDTree (Nearest Point)	21			
6.16	Rotating Sweep Line	21			
6.17	Circle Cover	21			

## 1.4 Pragma Optimization

```
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
```

## 1.5 IO Optimization

```
static inline int gc() {
    constexpr int B = 1<<20;
    static char buf[B], *p, *q;
    if(p == q &&
        (q=(p=buf)+fread(buf,1,B,stdin)) == buf)
        return EOF;
    return *p++;
}

template < typename T >
static inline bool gn( T &x ) {
    int c = gc(); T sgn = 1; x = 0;
    while(('0'>c|c>'9') && c!=EOF && c!='-') c = gc();
    if(c == '-') sgn = -1, c = gc();
    if(c == EOF) return false;
    while('0'<=c&&c<='9') x = x*10 + c - '0', c = gc();
    return x *= sgn, true;
}
```

# 2 Data Structure

## 2.1 Dark Magic

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
// heap tags: pairing/binary/binomial/rc_binomial/thin
template<typename T>
using pbds_heap=__gnu_pbds::prioity_queue<T,less<T>, \
    pairing_heap_tag>;

// pbds_heap::point_iterator
// x = pq.push(10); pq.modify(x, 87); a.join(b);
// tree tags: rb_tree_tag/ov_tree_tag/splay_tree_tag
template<typename T>
using ordered_set = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order, order_of_key
// hash tables: cc_hash_table/gp_hash_table
```

## 2.2 Link-Cut Tree

```
struct Node{
    Node *par,*ch[2];
    int xor_sum,v;
    bool is_rev;
    Node(int _v){
        v=xor_sum=_v;is_rev=false;
        par=ch[0]=ch[1]=nullptr;
    }
    inline void set_rev(){is_rev^=1;swap(ch[0],ch[1]);}
    inline void down(){
        if(is_rev){
            if(ch[0]!=nullptr) ch[0]->set_rev();
            if(ch[1]!=nullptr) ch[1]->set_rev();
            is_rev=false;
        }
    }
    inline void up(){
        xor_sum=v;
        if(ch[0]!=nullptr){
            xor_sum^=ch[0]->xor_sum;
            ch[0]->par=this;
        }
        if(ch[1]!=nullptr){
            xor_sum^=ch[1]->xor_sum;
            ch[1]->par=this;
        }
    }
    inline bool is_root(){
        return par==nullptr || \
            (par->ch[0]!=this && par->ch[1]!=this);
    }
    bool is_rch(){return !is_root() && par->ch[1]==this;}
    *node[maxn],*stk[maxn];
    int top;
    void to_child(Node* p,Node* c,bool dir){
        p->ch[dir]=c;
        p->up();
    }
}
```

```

    }
    inline void rotate(Node* node){
        Node* par=node->par;
        Node* par_par=par->par;
        bool dir=node->is_rch();
        bool par_dir=par->is_rch();
        to_child(par,node->ch[!dir],dir);
        to_child(node,par,!dir);
        if(par_par!=nullptr && par_par->ch[par_dir]==par)
            to_child(par_par,node,par_dir);
        else node->par=par_par;
    }
    inline void splay(Node* node){
        Node* tmp=node;
        stk[top++]=node;
        while(!tmp->is_root()){
            tmp=tmp->par;
            stk[top++]=tmp;
        }
        while(top) stk[--top]->down();
        for(Node *fa=node->par;
            !node->is_root();
            rotate(node),fa=node->par)
            if(!fa->is_root())
                rotate(fa->is_rch()==node->is_rch()?fa:node);
    }
    inline void access(Node* node){
        Node* last=nullptr;
        while(node!=nullptr){
            splay(node);
            to_child(node,last,true);
            last=node;
            node=node->par;
        }
    }
    inline void change_root(Node* node){
        access(node);splay(node);node->set_rev();
    }
    inline void link(Node* x,Node* y){
        change_root(x);splay(x);x->par=y;
    }
    inline void split(Node* x,Node* y){
        change_root(x);access(y);splay(x);
        to_child(x,nullptr,true);y->par=nullptr;
    }
    inline void change_val(Node* node,int v){
        access(node);splay(node);node->v=v;node->up();
    }
    inline int query(Node* x,Node* y){
        change_root(x);access(y);splay(y);
        return y->xor_sum;
    }
    inline Node* find_root(Node* node){
        access(node);splay(node);
        Node* last=nullptr;
        while(node!=nullptr){
            node->down();last=node;node=node->ch[0];
        }
        return last;
    }
    set<pii> dic;
    inline void add_edge(int u,int v){
        if(u>v) swap(u,v);
        if(find_root(node[u])==find_root(node[v])) return;
        dic.insert(pii(u,v));
        link(node[u],node[v]);
    }
    inline void del_edge(int u,int v){
        if(u>v) swap(u,v);
        if(dic.find(pii(u,v))==dic.end()) return;
        dic.erase(pii(u,v));
        split(node[u],node[v]);
    }
}
```

## 2.3 LiChao Segment Tree

```
struct Line{
    int m, k, id;
    Line() : id( -1 ) {}
    Line( int a, int b, int c )
        : m( a ), k( b ), id( c ) {}
    int at( int x ) { return m * x + k; }
}
```

```

};
class LiChao {
private:
    int n; vector< Line > nodes;
    inline int lc( int x ) { return 2 * x + 1; }
    inline int rc( int x ) { return 2 * x + 2; }
    void insert( int l, int r, int id, Line ln ) {
        int m = ( l + r ) >> 1;
        if ( nodes[ id ].id == -1 ) {
            nodes[ id ] = ln;
            return;
        }
        bool atLeft = nodes[ id ].at( l ) < ln.at( l );
        if ( nodes[ id ].at( m ) < ln.at( m ) ) {
            atLeft ^= 1; swap( nodes[ id ], ln );
        }
        if ( r - l == 1 ) return;
        if ( atLeft ) insert( l, m, lc( id ), ln );
        else insert( m, r, rc( id ), ln );
    }
    int query( int l, int r, int id, int x ) {
        int ret = 0;
        if ( nodes[ id ].id != -1 )
            ret = nodes[ id ].at( x );
        int m = ( l + r ) >> 1;
        if ( r - l == 1 ) return ret;
        else if ( x < m )
            return max( ret, query( l, m, lc( id ), x ) );
        else
            return max( ret, query( m, r, rc( id ), x ) );
    }
public:
    void build( int n_ ) {
        n = n_; nodes.clear();
        nodes.resize( n << 2, Line() );
    }
    void insert( Line ln ) { insert( 0, n, 0, ln ); }
    int query( int x ) { return query( 0, n, 0, x ); }
} lichao;

```

## 2.4 Treap

```

namespace Treap{
#define sz( x ) ( ( x ) ? ( ( x )->size ) : 0 )
struct node{
    int size;
    uint32_t pri;
    node *lc, *rc;
    node() : size(0), pri(rand()), lc( 0 ), rc( 0 ) {}
    void pull() {
        size = 1;
        if ( lc ) size += lc->size;
        if ( rc ) size += rc->size;
    }
};
node* merge( node* L, node* R ) {
    if ( not L or not R ) return L ? L : R;
    if ( L->pri > R->pri ) {
        L->rc = merge( L->rc, R ); L->pull();
        return L;
    } else {
        R->lc = merge( L, R->lc ); R->pull();
        return R;
    }
}
void split_by_size( node*rt, int k, node*&L, node*&R ) {
    if ( not rt ) L = R = nullptr;
    else if( sz( rt->lc ) + 1 <= k ) {
        L = rt;
        split_by_size( rt->rc, k-sz(rt->lc)-1, L->rc, R );
        L->pull();
    } else {
        R = rt;
        split_by_size( rt->lc, k, L, R->lc );
        R->pull();
    }
}
#undef sz
}

```

## 2.5 Sparse Table

```

template < typename T, typename Cmp_ = less< T > >
class SparseTable {

```

```

private:
    vector< vector< T > > tbl;
    vector< int > lg;
    T cv( T a, T b ) {
        return Cmp_()( a, b ) ? a : b;
    }
public:
    void init( T arr[], int n ) {
        // 0-base
        lg.resize( n + 1 );
        lg[ 0 ] = -1;
        for( int i=1; i<=n; ++i ) lg[i] = lg[i>>1] + 1;
        tbl.resize( lg[n] + 1 );
        tbl[ 0 ].resize( n );
        copy( arr, arr + n, tbl[ 0 ].begin() );
        for ( int i = 1; i <= lg[ n ]; ++i ) {
            int len = 1 << ( i - 1 ), sz = 1 << i;
            tbl[ i ].resize( n - sz + 1 );
            for ( int j = 0; j <= n - sz; ++j )
                tbl[i][j] = cv(tbl[i-1][j], tbl[i-1][j+len]);
        }
        T query( int l, int r ) {
            // 0-base [l, r)
            int wh = lg[ r - l ], len = 1 << wh;
            return cv( tbl[ wh ][ l ], tbl[ wh ][ r - len ] );
        }
};

```

## 2.6 Linear Basis

```

template <int BITS>
struct LinearBasis {
    array<uint64_t, BITS> basis;
    Basis() { basis.fill(0); }
    void add(uint64_t x) {
        for (int i = 0; i < BITS; ++i) if ((x >> i) & 1) {
            if (basis[i] == 0) {
                basis[i] = x;
                return;
            }
            x ^= basis[i];
        }
    }
    bool ok(uint64_t x) {
        for (int i = 0; i < BITS; ++i)
            if ((x >> i) & 1) x ^= basis[i];
        return x == 0;
    }
};

```

## 2.7 Binary Search On Segment Tree

```

// find_first = x -> minimal x s.t. check( [a, x) )
// find_last = x -> maximal x s.t. check( [x, b) )
template <typename C>
int find_first(int l, const C &check) {
    if (l >= n)
        return n;
    l += sz;
    for (int i = height; i > 0; i--)
        propagate(l >> i);
    Monoid sum = identity;
    do {
        while ((l & 1) == 0)
            l >>= 1;
        if (check(f(sum, data[l]))) {
            while (l < sz) {
                propagate(l);
                l <<= 1;
                auto nxt = f(sum, data[l]);
                if (not check(nxt)) {
                    sum = nxt;
                    l++;
                }
            }
            return l + 1 - sz;
        }
        sum = f(sum, data[l++]);
    } while ((l & -l) != 1);
    return n;
}

```

```

template <typename C>

```

```

int find_last(int r, const C &check) {
    if (r <= 0)
        return -1;
    r += sz;
    for (int i = height; i > 0; i--)
        propagate((r - 1) >> i);
    Monoid sum = identity;
    do {
        r--;
        while (r > 1 and (r & 1))
            r >>= 1;
        if (check(f(data[r], sum))) {
            while (r < sz) {
                propagate(r);
                r = (r << 1) + 1;
                auto nxt = f(data[r], sum);
                if (not check(nxt)) {
                    sum = nxt;
                    r--;
                }
            }
            return r - sz;
        }
        sum = f(data[r], sum);
    } while ((r & -r) != r);
    return -1;
}

```

### 3 Graph

#### 3.1 BCC Edge

```

class BCC_Bridge {
private:
    int n, ecnt;
    vector<vector<pair<int,int>>> G;
    vector<int> dfn, low;
    vector<bool> bridge;
    void dfs(int u, int f) {
        dfn[u] = low[u] = dfn[f] + 1;
        for (auto [v, t]: G[u]) {
            if (v == f) continue;
            if (dfn[v]) {
                low[u] = min(low[u], dfn[v]);
                continue;
            }
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u]) bridge[t] = true;
        }
    }
public:
    void init(int n_) {
        G.clear(); G.resize(n = n_);
        low.assign(n, ecnt = 0);
        dfn.assign(n, 0);
    }
    void add_edge(int u, int v) {
        G[u].emplace_back(v, ecnt);
        G[v].emplace_back(u, ecnt++);
    }
    void solve() {
        bridge.assign(ecnt, false);
        for (int i = 0; i < n; ++i)
            if (not dfn[i]) dfs(i, i);
    }
    bool is_bridge(int x) { return bridge[x]; }
} bcc_bridge;

```

#### 3.2 BCC Vertex

```

class BCC_AP {
private:
    int n, ecnt;
    vector<vector<pair<int,int>>> G;
    vector<int> bcc, dfn, low, st;
    vector<bool> ap, ins;
    void dfs(int u, int f) {
        dfn[u] = low[u] = dfn[f] + 1;
        int ch = 0;
        for (auto [v, t]: G[u]) if (v != f) {
            if (not ins[t]) {
                st.push_back(t);

```

```

                ins[t] = true;
            }
            if (dfn[v]) {
                low[u] = min(low[u], dfn[v]);
                continue;
            } ++ch; dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                ap[u] = true;
                while (true) {
                    int eid = st.back(); st.pop_back();
                    bcc[eid] = ecnt;
                    if (eid == t) break;
                }
                ecnt++;
            }
        }
        if (ch == 1 and u == f) ap[u] = false;
    }
public:
    void init(int n_) {
        G.clear(); G.resize(n = n_);
        ecnt = 0; ap.assign(n, false);
        low.assign(n, 0); dfn.assign(n, 0);
    }
    void add_edge(int u, int v) {
        G[u].emplace_back(v, ecnt);
        G[v].emplace_back(u, ecnt++);
    }
    void solve() {
        ins.assign(ecnt, false);
        bcc.resize(ecnt); ecnt = 0;
        for (int i = 0; i < n; ++i)
            if (not dfn[i]) dfs(i, i);
    }
    int get_id(int x) { return bcc[x]; }
    int count() { return ecnt; }
    bool is_ap(int x) { return ap[x]; }
} bcc_ap;

```

#### 3.3 2-SAT (SCC)

```

class TwoSat {
private:
    int n;
    vector<vector<int>> rG, G, sccs;
    vector<int> ord, idx;
    vector<bool> vis, result;
    void dfs(int u) {
        vis[u] = true;
        for (int v: G[u])
            if (!vis[v]) dfs(v);
        ord.push_back(u);
    }
    void rdfs(int u) {
        vis[u] = false; idx[u] = sccs.size() - 1;
        sccs.back().push_back(u);
        for (int v: rG[u])
            if (vis[v]) rdfs(v);
    }
public:
    void init(int n_) {
        n = n_; G.clear(); G.resize(n);
        rG.clear(); rG.resize(n);
        sccs.clear(); ord.clear();
        idx.resize(n); result.resize(n);
    }
    void add_edge(int u, int v) {
        G[u].push_back(v); rG[v].push_back(u);
    }
    void orr(int x, int y) {
        if ((x^y) == 1) return;
        add_edge(x^1, y); add_edge(y^1, x);
    }
    bool solve() {
        vis.clear(); vis.resize(n);
        for (int i = 0; i < n; ++i)
            if (not vis[i]) dfs(i);
        reverse(ord.begin(), ord.end());
        for (int u: ord) {
            if (!vis[u]) continue;
            sccs.push_back(vector<int>());
            rdfs(u);

```

```

    }
    for(int i=0;i<n;i+=2)
        if(idx[i]==idx[i+1])
            return false;
    vector<bool> c(sccs.size());
    for(size_t i=0;i<sccs.size();++i){
        for(size_t j=0;j<sccs[i].size();++j){
            result[sccs[i][j]]=c[i];
            c[idx[sccs[i][j]^1]]!=c[i];
        }
    }
    return true;
}
bool get(int x){return result[x];}
inline int get_id(int x){return idx[x];}
inline int count(){return sccs.size();}
} sat2;

```

### 3.4 Lowbit Decomposition

```

class LowbitDecomp{
private:
    int time_, chain_, LOG_N;
    vector< vector< int > > G, fa;
    vector< int > tl, tr, chain, chain_st;
    // chain_ : number of chain
    // tl, tr[ u ] : subtree interval in the seq. of u
    // chain_st[ u ] : head of the chain contains u
    // chain[ u ] : chain id of the chain u is on
    void predfs( int u, int f ) {
        chain[ u ] = 0;
        for ( int v : G[ u ] ) {
            if ( v == f ) continue;
            predfs( v, u );
            if( lowbit( chain[ u ] ) < lowbit( chain[ v ] ) )
                chain[ u ] = chain[ v ];
        }
        if ( not chain[ u ] )
            chain[ u ] = chain_++;
    }
    void dfschain( int u, int f ) {
        fa[ u ][ 0 ] = f;
        for ( int i = 1 ; i < LOG_N ; ++ i )
            fa[ u ][ i ] = fa[ fa[ u ][ i - 1 ] ][ i - 1 ];
        tl[ u ] = time_++;
        if ( not chain_st[ chain[ u ] ] )
            chain_st[ chain[ u ] ] = u;
        for ( int v : G[ u ] )
            if ( v != f and chain[ v ] == chain[ u ] )
                dfschain( v, u );
        for ( int v : G[ u ] )
            if ( v != f and chain[ v ] != chain[ u ] )
                dfschain( v, u );
        tr[ u ] = time_;
    }
    bool anc( int u, int v ) {
        return tl[ u ] <= tl[ v ] and tr[ v ] <= tr[ u ];
    }
public:
    int lca( int u, int v ) {
        if ( anc( u, v ) ) return u;
        for ( int i = LOG_N - 1 ; i >= 0 ; -- i )
            if ( not anc( fa[ u ][ i ], v ) )
                u = fa[ u ][ i ];
        return fa[ u ][ 0 ];
    }
    void init( int n ) {
        fa.assign( ++n, vector< int >( LOG_N ) );
        for ( LOG_N = 0 ; ( 1 << LOG_N ) < n ; ++ LOG_N );
        G.clear(); G.resize( n );
        tl.assign( n, 0 ); tr.assign( n, 0 );
        chain.assign( n, 0 ); chain_st.assign( n, 0 );
    }
    void add_edge( int u, int v ) {
        // 1-base
        G[ u ].push_back( v );
        G[ v ].push_back( u );
    }
    void decompose(){
        chain_ = 1;
        predfs( 1, 1 );
        time_ = 0;
        dfschain( 1, 1 );
    }

```

```

}
PII get_subtree(int u) { return {tl[ u ], tr[ u ] }; }
vector< PII > get_path( int u, int v ){
    vector< PII > res;
    int g = lca( u, v );
    while ( chain[ u ] != chain[ g ] ) {
        int s = chain_st[ chain[ u ] ];
        res.emplace_back( tl[ s ], tl[ u ] + 1 );
        u = fa[ s ][ 0 ];
    }
    res.emplace_back( tl[ g ], tl[ u ] + 1 );
    while ( chain[ v ] != chain[ g ] ) {
        int s = chain_st[ chain[ v ] ];
        res.emplace_back( tl[ s ], tl[ v ] + 1 );
        v = fa[ s ][ 0 ];
    }
    res.emplace_back( tl[ g ] + 1, tl[ v ] + 1 );
    return res;
}
/* res : list of intervals from u to v
 * ( note only nodes work, not edge )
 * usage :
 * vector< PII > path = tree.get_path( u, v )
 * for( auto [ l, r ] : path ) {
 *     0-base [ l, r )
 * }
 */
} tree;

```

### 3.5 MaxClique

```

// contain aself loop u to u, than u won't in clique
template < size_t MAXN >
class MaxClique{
private:
    using bits = bitset< MAXN >;
    bits popped, G[ MAXN ], ans;
    size_t deg[ MAXN ], deo[ MAXN ], n;
    void sort_by_degree() {
        popped.reset();
        for ( size_t i = 0 ; i < n ; ++ i )
            deg[ i ] = G[ i ].count();
        for ( size_t i = 0 ; i < n ; ++ i ) {
            size_t mi = MAXN, id = 0;
            for ( size_t j = 0 ; j < n ; ++ j )
                if ( not popped[ j ] and deg[ j ] < mi )
                    mi = deg[ id = j ];
            popped[ deo[ i ] = id ] = 1;
            for( size_t u = G[ i ]._Find_first();
                u < n ; u = G[ i ]._Find_next( u ) )
                -- deg[ u ];
        }
    }
    void BK( bits R, bits P, bits X ) {
        if ( R.count()+P.count() <= ans.count() ) return;
        if ( not P.count() and not X.count() ) {
            if ( R.count() > ans.count() ) ans = R;
            return;
        }
        /* greedily choose max degree as pivot
        bits cur = P | X; size_t pivot = 0, sz = 0;
        for ( size_t u = cur._Find_first();
            u < n ; u = cur._Find_next( u ) )
            if ( deg[ u ] > sz ) sz = deg[ pivot = u ];
        cur = P & ( ~G[ pivot ] );
        */ // or simply choose first
        bits cur = P & ( ~G[ ( P | X )._Find_first() ] );
        for ( size_t u = cur._Find_first();
            u < n ; u = cur._Find_next( u ) ) {
            if ( R[ u ] ) continue;
            R[ u ] = 1;
            BK( R, P & G[ u ], X & G[ u ] );
            R[ u ] = P[ u ] = 0, X[ u ] = 1;
        }
    }
public:
    void init( size_t n_ ) {
        n = n_;
        for ( size_t i = 0 ; i < n ; ++ i )
            G[ i ].reset();
        ans.reset();
    }
    void add_edges( int u, bits S ) { G[ u ] = S; }

```



```

void add_edge( int u, int v ) {
    G[ u ][ v ] = G[ v ][ u ] = 1;
}
int solve() {
    sort_by_degree(); // or simply iota( deo... )
    for ( size_t i = 0 ; i < n ; ++ i )
        deg[ i ] = G[ i ].count();
    bits pob, nob = 0; pob.set();
    for (size_t i=n; i<MAXN; ++i) pob[i] = 0;
    for ( size_t i = 0 ; i < n ; ++ i ) {
        size_t v = deo[ i ];
        bits tmp; tmp[ v ] = 1;
        BK( tmp, pob & G[ v ], nob & G[ v ] );
        pob[ v ] = 0, nob[ v ] = 1;
    }
    return static_cast< int >( ans.count() );
}
};

```

### 3.6 MaxCliqueDyn

```

constexpr int kN = 150;
struct MaxClique { // Maximum Clique
    bitset<kN> a[kN], cs[kN];
    int ans, sol[kN], q, cur[kN], d[kN], n;
    void init(int _n) {
        n = _n, ans = q = 0;
        for (int i = 0; i < n; i++) a[i].reset();
    }
    void addEdge(int u, int v) { a[u][v] = a[v][u] = 1; }
    void csort(vector<int> &r, vector<int> &c) {
        int mx = 1, km = max(ans - q + 1, 1), t = 0,
            m = int(r.size());
        cs[1].reset(); cs[2].reset();
        for (int i = 0; i < m; i++) {
            int p = r[i], k = 1;
            while ((cs[k] & a[p]).count()) k++;
            if (k > mx) cs[++mx + 1].reset();
            cs[k][p] = 1;
            if (k < km) r[t++] = p;
        }
        c.resize(m);
        if (t) c[t - 1] = 0;
        for (int k = km; k <= mx; k++) {
            for (int p = int(cs[k]._Find_first());
                p < kN; p = int(cs[k]._Find_next(p))) {
                r[t] = p; c[t++] = k;
            }
        }
    }
    void dfs(vector<int> &r, vector<int> &c, int l,
        bitset<kN> mask) {
        while (!r.empty()) {
            int p = r.back(); r.pop_back();
            mask[p] = 0;
            if (q + c.back() <= ans) return;
            cur[q++] = p;
            vector<int> nr, nc;
            bitset<kN> nmask = mask & a[p];
            for (int i : r)
                if (a[p][i]) nr.push_back(i);
            if (!nr.empty()) {
                if (l < 4) {
                    for (int i : nr)
                        d[i] = int((a[i] & nmask).count());
                    sort(nr.begin(), nr.end(),
                        [&](int x, int y) {
                            return d[x] > d[y];
                        });
                }
                csort(nr, nc); dfs(nr, nc, l + 1, nmask);
            } else if (q > ans) {
                ans = q; copy(cur, cur + q, sol);
            }
            c.pop_back(); q--;
        }
    }
    void solve(bitset<kN> mask) { // vertex mask
        vector<int> r, c;
        for (int i = 0; i < n; i++)
            if (mask[i]) r.push_back(i);
        for (int i = 0; i < n; i++)
            d[i] = int((a[i] & mask).count());
    }
};

```

```

sort(r.begin(), r.end(),
    [&](int i, int j) { return d[i] > d[j]; });
csort(r, c);
dfs(r, c, 1, mask);
return ans; // sol[0 ~ ans-1]
}
} graph;

```

### 3.7 Virtual Tree

```

inline bool cmp(const int &i, const int &j) {
    return dfn[i] < dfn[j];
}
void build(int vectrices[], int k) {
    static int stk[MAX_N];
    sort(vectrices, vectrices + k, cmp);
    stk[sz++] = 0;
    for (int i = 0; i < k; ++i) {
        int u = vectrices[i], lca = LCA(u, stk[sz - 1]);
        if (lca == stk[sz - 1]) stk[sz++] = u;
        else {
            while (sz >= 2 && dep[stk[sz - 2]] >= dep[lca]) {
                addEdge(stk[sz - 2], stk[sz - 1]);
                sz--;
            }
            if (stk[sz - 1] != lca) {
                addEdge(lca, stk[sz - 1]);
                stk[sz++] = lca, vectrices[cnt++] = lca;
            }
            stk[sz++] = u;
        }
    }
    for (int i = 0; i < sz - 1; ++i)
        addEdge(stk[i], stk[i + 1]);
}

```

### 3.8 Centroid Decomposition

```

struct Centroid {
    vector<vector<int64_t>> Dist;
    vector<int> Parent, Depth;
    vector<int64_t> Sub, Sub2;
    vector<int> Sz, Sz2;
    Centroid(vector<vector<pair<int, int>>> g) {
        int N = g.size();
        vector<bool> Vis(N);
        vector<int> sz(N), mx(N);
        vector<int> Path;
        Dist.resize(N);
        Parent.resize(N);
        Depth.resize(N);
        auto DfsSz = [&](auto dfs, int x) -> void {
            Vis[x] = true; sz[x] = 1; mx[x] = 0;
            for (auto [u, w] : g[x]) {
                if (Vis[u]) continue;
                dfs(dfs, u);
                sz[x] += sz[u];
                mx[x] = max(mx[x], sz[u]);
            }
            Path.push_back(x);
        };
        auto DfsDist = [&](auto dfs, int x, int64_t D = 0)
            -> void {
            Dist[x].push_back(D); Vis[x] = true;
            for (auto [u, w] : g[x]) {
                if (Vis[u]) continue;
                dfs(dfs, u, D + w);
            }
        };
        auto Dfs = [&]
            (auto dfs, int x, int D = 0, int p = -1) -> void {
            Path.clear(); DfsSz(DfsSz, x);
            int M = Path.size();
            int C = -1;
            for (int u : Path) {
                if (max(M - sz[u], mx[u]) * 2 <= M) C = u;
                Vis[u] = false;
            }
            DfsDist(DfsDist, C);
            for (int u : Path) Vis[u] = false;
            Parent[C] = p; Vis[C] = true;
            Depth[C] = D;
            for (auto [u, w] : g[C]) {
                if (Vis[u]) continue;
            }
        };
    }
};

```

```

    dfs(dfs, u, D + 1, C);
}
};
Dfs(Dfs, 0); Sub.resize(N); Sub2.resize(N);
Sz.resize(N); Sz2.resize(N);
}
void Mark(int v) {
    int x = v, z = -1;
    for (int i = Depth[v]; i >= 0; --i) {
        Sub[x] += Dist[v][i]; Sz[x]++;
        if (z != -1) {
            Sub2[z] += Dist[v][i];
            Sz2[z]++;
        }
        z = x; x = Parent[x];
    }
}
int64_t Query(int v) {
    int64_t res = 0;
    int x = v, z = -1;
    for (int i = Depth[v]; i >= 0; --i) {
        res += Sub[x] + 1LL * Sz[x] * Dist[v][i];
        if (z != -1) res -= Sub2[z] + 1LL * Sz2[z] * Dist[v][i];
        z = x; x = Parent[x];
    }
    return res;
}
};

```

### 3.9 Tree Hashing

```

uint64_t hsah(int u, int f) {
    uint64_t r = 127;
    for (int v : G[u]) if (v != f) {
        uint64_t hh = hsah(v, u);
        r = (r + (hh * hh) % 1010101333) % 1011820613;
    }
    return r;
}

```

### 3.10 Minimum Mean Cycle

```

/* minimum mean cycle O(VE) */
struct MMC{
#define FZ(n) memset((n),0,sizeof(n))
#define E 101010
#define V 1021
#define inf 1e9
    struct Edge { int v,u; double c; };
    int n, m, prv[V][V], prve[V][V], vst[V];
    Edge e[E];
    vector<int> edgeID, cycle, rho;
    double d[V][V];
    void init( int _n ) { n = _n; m = 0; }
    // WARNING: TYPE matters
    void add_edge( int vi , int ui , double ci )
    { e[ m ++ ] = { vi , ui , ci }; }
    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 solve(){
        // 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);
        }
    }
}

```

```

}
FZ(vst);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;
}
} mmc;

```

### 3.11 Mo's Algorithm on Tree

```

int q; vector< int > G[N];
struct Que{
    int u, v, id;
} que[ N ];
int dfn[N], dfn_, block_id[N], block_, stk[N], stk_;
void dfs( int u, int f ) {
    dfn[ u ] = dfn_++; int saved_rbp = stk_;
    for ( int v : G[ u ] ) {
        if ( v == f ) continue;
        dfs( v, u );
        if ( stk_ - saved_rbp < SQRT_N ) continue;
        for ( ++ block_ ; stk_ != saved_rbp ; )
            block_id[ stk[ -- stk_ ] ] = block_;
    }
    stk[ stk_ ++ ] = u;
}
bool inPath[ N ];
void Diff( int u ) {
    if ( inPath[ u ] ^ 1 ) { /*remove this edge*/ }
    else { /*add this edge*/ }
}
void traverse( int& origin_u, int u ) {
    for ( int g = lca( origin_u, u ) ;
        origin_u != g ; origin_u = parent_of[ origin_u ] )
        Diff( origin_u );
    for ( int v = u; v != origin_u; v = parent_of[v])
        Diff( v );
    origin_u = u;
}
void solve() {
    dfs( 1, 1 );
    while ( stk_ ) block_id[ stk[ -- stk_ ] ] = block_;
    sort( que, que + q, [](const Que& x, const Que& y) {
        return tie( block_id[ x.u ], dfn[ x.v ] )
            < tie( block_id[ y.u ], dfn[ y.v ] );
    } );
    int U = 1, V = 1;
    for ( int i = 0 ; i < q ; ++ i ) {
        pass( U, que[ i ].u );
        pass( V, que[ i ].v );
        // we could get our answer of que[ i ].id
    }
}
/*
Method 2:
dfs u:
    push u
    iterate subtree
    push u
Let P = LCA(u, v), and St(u)<=St(v)
if (P == u) query[St(u), St(v)]
else query[Ed(u), St(v)], query[St(P), St(P)]
*/

```

### 3.12 Minimum Steiner Tree

```

// Minimum Steiner Tree
// O(V 3^AT + V^2 2^AT)
struct SteinerTree{
#define V 33
#define T 8
#define INF 1023456789
    int n, dst[V][V], dp[1 << T][V], tdst[V];
    void init( int _n ){

```

```

n = _n;
for( int i = 0 ; i < n ; i ++ ){
    for( int j = 0 ; j < n ; j ++ )
        dst[ i ][ j ] = INF;
    dst[ i ][ i ] = 0;
}
}

void add_edge( int ui , int vi , int wi ){
    dst[ ui ][ vi ] = min( dst[ ui ][ vi ] , wi );
    dst[ vi ][ ui ] = min( dst[ vi ][ ui ] , wi );
}

void shortest_path(){
    for( int k = 0 ; k < n ; k ++ )
        for( int i = 0 ; i < n ; i ++ )
            for( int j = 0 ; j < n ; j ++ )
                dst[ i ][ j ] = min( dst[ i ][ j ],
                    dst[ i ][ k ] + dst[ k ][ j ] );
}

int solve( const vector<int>& ter ){
    int t = (int)ter.size();
    for( int i = 0 ; i < ( 1 << t ) ; i ++ )
        for( int j = 0 ; j < n ; j ++ )
            dp[ i ][ j ] = INF;
    for( int i = 0 ; i < n ; i ++ )
        dp[ 0 ][ i ] = 0;
    for( int msk = 1 ; msk < ( 1 << t ) ; msk ++ ){
        if( msk == ( msk & (-msk) ) ){
            int who = __lg( msk );
            for( int i = 0 ; i < n ; i ++ )
                dp[ msk ][ i ] = dst[ ter[ who ] ][ i ];
            continue;
        }
        for( int i = 0 ; i < n ; i ++ )
            for( int submsk = ( msk - 1 ) & msk ; submsk ;
                submsk = ( submsk - 1 ) & msk )
                dp[ msk ][ i ] = min( dp[ msk ][ i ],
                    dp[ submsk ][ i ] +
                    dp[ msk ^ submsk ][ i ] );
        for( int i = 0 ; i < n ; i ++ ){
            tdst[ i ] = INF;
            for( int j = 0 ; j < n ; j ++ )
                tdst[ i ] = min( tdst[ i ],
                    dp[ msk ][ j ] + dst[ j ][ i ] );
        }
        for( int i = 0 ; i < n ; i ++ )
            dp[ msk ][ i ] = tdst[ i ];
    }
    int ans = INF;
    for( int i = 0 ; i < n ; i ++ )
        ans = min( ans , dp[ ( 1 << t ) - 1 ][ i ] );
    return ans;
}

} solver;

```

### 3.13 Directed Minimum Spanning Tree

```

struct DirectedMST { // find maximum
    struct Edge {
        int u, v;
        int w;
        Edge(int u, int v, int w) : u(u), v(v), w(w) {}
    };
    vector<Edge> Edges;
    void clear() { Edges.clear(); }
    void addEdge(int a, int b, int w) { Edges.emplace_back
        (a, b, w); }
    int solve(int root, int n) {
        vector<Edge> E = Edges;
        int ans = 0;
        while (true) {
            // find best in edge
            vector<int> in(n, -inf), prv(n, -1);
            for (auto e : E)
                if (e.u != e.v && e.w > in[e.v]) {
                    in[e.v] = e.w;
                    prv[e.v] = e.u;
                }
            in[root] = 0;
            prv[root] = -1;
            for (int i = 0; i < n; i++)
                if (in[i] == -inf)
                    return -inf;
            // find cycle

```

```

int tot = 0;
vector<int> id(n, -1), vis(n, -1);
for (int i = 0; i < n; i++) {
    ans += in[i];
    for (int x = i; x != -1 && id[x] == -1; x = prv[x])
        {
            if (vis[x] == i) {
                for (int y = prv[x]; y != x; y = prv[y])
                    id[y] = tot;
                id[x] = tot++;
                break;
            }
            vis[x] = i;
        }
}
if (!tot)
    return ans;
for (int i = 0; i < n; i++)
    if (id[i] == -1)
        id[i] = tot++;
// shrink
for (auto &e : E) {
    if (id[e.u] != id[e.v])
        e.w -= in[e.v];
    e.u = id[e.u], e.v = id[e.v];
}
n = tot;
root = id[root];
}
assert(false);
}
} DMST;

```

### 3.14 Manhattan Minimum Spanning Tree

```

typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
    vi id(sz(ps));
    iota(all(id), 0);
    vector<array<int, 3>> edges;
    rep(k, 0, 4) {
        sort(all(id), [&](int i, int j) {
            return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y;
        });
        map<int, int> sweep;
        for (int i : id) {
            for (auto it = sweep.lower_bound(-ps[i].y);
                it != sweep.end(); sweep.erase(it++)) {
                int j = it->second;
                P d = ps[i] - ps[j];
                if (d.y > d.x) break;
                edges.push_back({d.y + d.x, i, j});
            }
            sweep[-ps[i].y] = i;
        }
        for (P &p : ps)
            if (k & 1) p.x = -p.x;
            else swap(p.x, p.y);
    }
    return edges; // [{w, i, j}, ...]
}

```

### 3.15 Dominator Tree

```

namespace dominator {
    vector<int> g[maxn], r[maxn], rdom[maxn];
    int dfn[maxn], rev[maxn], fa[maxn], sdom[maxn];
    int dom[maxn], val[maxn], rp[maxn], tk;
    void init(int n) {
        // vertices are numbered from 0 to n - 1
        fill(dfn, dfn + n, -1); fill(rev, rev + n, -1);
        fill(fa, fa + n, -1); fill(val, val + n, -1);
        fill(sdom, sdom + n, -1); fill(rp, rp + n, -1);
        fill(dom, dom + n, -1); tk = 0;
        for (int i = 0; i < n; ++i) {
            g[i].clear(); r[i].clear(); rdom[i].clear();
        }
    }
    void add_edge(int x, int y) { g[x].push_back(y); }
    void dfs(int x) {
        rev[dfn[x]] = tk = x;
        fa[tk] = sdom[tk] = val[tk] = tk; tk++;
        for (int u : g[x]) {
            if (dfn[u] == -1) dfs(u), rp[dfn[u]] = dfn[x];

```



```

    r[dfn[u]].push_back(dfn[x]);
}
}
void merge(int x, int y) { fa[x] = y; }
int find(int x, int c = 0) {
    if (fa[x] == x) return c ? -1 : x;
    int p = find(fa[x], 1);
    if (p == -1) return c ? fa[x] : val[x];
    if (sdom[val[x]] > sdom[val[fa[x]]]) val[x] = val[fa[x]];
    fa[x] = p;
    return c ? p : val[x];
}
vector<int> build(int s, int n) {
    // return the father of each node in the dominator tree
    // p[i] = -2 if i is unreachable from s
    dfs(s);
    for (int i = tk - 1; i >= 0; --i) {
        for (int u: r[i]) sdom[i] = min(sdom[i], sdom[find(u)]);
        if (i) rdom[sdom[i]].push_back(i);
        for (int &u : rdom[i]) {
            int p = find(u);
            if (sdom[p] == i) dom[u] = i;
            else dom[u] = p;
        }
        if (i) merge(i, rp[i]);
    }
    vector<int> p(n, -2); p[s] = -1;
    for (int i = 1; i < tk; ++i)
        if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
    for (int i = 1; i < tk; ++i) p[rev[i]] = rev[dom[i]];
    return p;
}
}

```

### 3.16 Edge Coloring

```

// max(d_u) + 1 edge coloring, time: O(NM)
int C[kN][kN], G[kN][kN]; // 1-based, G: ans
void clear(int N) {
    for (int i = 0; i <= N; i++)
        for (int j = 0; j <= N; j++)
            C[i][j] = G[i][j] = 0;
}
void solve(vector<pair<int, int>> &E, int N) {
    int X[kN] = {}, a;
    auto update = [&](int u) {
        for (X[u] = 1; C[u][X[u]]; X[u]++);
    };
    auto color = [&](int u, int v, int c) {
        int p = G[u][v];
        G[u][v] = G[v][u] = c;
        C[u][c] = v, C[v][c] = u;
        C[u][p] = C[v][p] = 0;
        if (p) X[u] = X[v] = p;
        else update(u), update(v);
        return p;
    };
    auto flip = [&](int u, int c1, int c2) {
        int p = C[u][c1];
        swap(C[u][c1], C[u][c2]);
        if (p) G[u][p] = G[p][u] = c2;
        if (!C[u][c1]) X[u] = c1;
        if (!C[u][c2]) X[u] = c2;
        return p;
    };
    for (int i = 1; i <= N; i++) X[i] = 1;
    for (int t = 0; t < E.size(); t++) {
        auto [u, v] = E[t];
        int v0 = v, c = X[u], c0 = c, d;
        vector<pair<int, int>> L; int vst[kN] = {};
        while (!G[u][v0]) {
            L.emplace_back(v, d = X[v]);
            if (!C[v][c]) for(a=L.size()-1; a>=0; a--)
                c = color(u, L[a].first, c);
            else if(!C[u][d]) for(a=L.size()-1; a>=0; a--)
                color(u, L[a].first, L[a].second);
            else if (vst[d]) break;
            else vst[d] = 1, v = C[u][d];
        }
        if (!G[u][v0]) {
            for (; v; v = flip(v, c, d), swap(c, d));
            if (C[u][c0]) { a = int(L.size()) - 1;
                while (--a >= 0 && L[a].second != c);
                for(; a>=0; a--) color(u, L[a].first, L[a].second);
            }
        }
    }
}

```

```

    } else t--;
}
}
}

```

## 4 Matching & Flow

### 4.1 Kuhn Munkres

```

class KM {
private:
    static constexpr lld INF = 1LL << 60;
    vector<lld> hl, hr, slk;
    vector<int> fl, fr, pre, qu;
    vector<vector<lld>> w;
    vector<bool> vl, vr;
    int n, ql, qr;
    bool check(int x) {
        if (vl[x] = true, fl[x] != -1)
            return vr[qu[qr++]] = fl[x] = true;
        while (x != -1) swap(x, fr[fl[x] = pre[x]]);
        return false;
    }
    void bfs(int s) {
        fill(slk.begin(), slk.end(), INF);
        fill(vl.begin(), vl.end(), false);
        fill(vr.begin(), vr.end(), false);
        ql = qr = 0;
        qu[qr++] = s;
        vr[s] = true;
        while (true) {
            lld d;
            while (ql < qr) {
                for (int x = 0, y = qu[ql++]; x < n; ++x) {
                    if (!vl[x] && slk[x] > (d = hl[x] + hr[y] - w[x][y])) {
                        if (pre[x] = y, d) slk[x] = d;
                        else if (!check(x)) return;
                    }
                }
            }
            d = INF;
            for (int x = 0; x < n; ++x)
                if (!vl[x] && d > slk[x]) d = slk[x];
            for (int x = 0; x < n; ++x) {
                if (vl[x]) hl[x] += d;
                else slk[x] -= d;
                if (vr[x]) hr[x] -= d;
            }
            for (int x = 0; x < n; ++x)
                if (!vl[x] && !slk[x] && !check(x)) return;
        }
    }
public:
    void init(int n_) {
        n = n_; qu.resize(n);
        fl.clear(); fl.resize(n, -1);
        fr.clear(); fr.resize(n, -1);
        hr.clear(); hr.resize(n); hl.resize(n);
        w.clear(); w.resize(n, vector<lld>(n));
        slk.resize(n); pre.resize(n);
        vl.resize(n); vr.resize(n);
    }
    void set_edge(int u, int v, lld x) { w[u][v] = x; }
    lld solve() {
        for (int i = 0; i < n; ++i)
            hl[i] = *max_element(w[i].begin(), w[i].end());
        for (int i = 0; i < n; ++i) bfs(i);
        lld res = 0;
        for (int i = 0; i < n; ++i) res += w[i][fl[i]];
        return res;
    }
} km;

```

### 4.2 Bipartite Matching

```

class BipartiteMatching {
private:
    vector<int> X[N], Y[N];
    int fx[N], fy[N], n;
    bitset<N> walked;
    bool dfs(int x) {
        for (auto i: X[x]) {
            if (walked[i]) continue;

```

```

    walked[i]=1;
    if(fY[i]==-1||dfs(fY[i])){
        fY[i]=x;fX[x]=i;
        return 1;
    }
    return 0;
}
public:
void init(int _n){
    n=_n; walked.reset();
    for(int i=0;i<n;i++){
        X[i].clear();Y[i].clear();
        fX[i]=fY[i]=-1;
    }
}
void add_edge(int x, int y){
    X[x].push_back(y); Y[y].push_back(x);
}
int solve(){
    int cnt = 0;
    for(int i=0;i<n;i++){
        walked.reset();
        if(dfs(i)) cnt++;
    }
    // return how many pair matched
    return cnt;
}
};

```

### 4.3 General Graph Matching

```

namespace matching {
int fa[kN], pre[kN], match[kN], s[kN], v[kN];
vector<int> g[kN];
queue<int> q;
void Init(int n) {
    for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
    for (int i = 0; i < n; ++i) g[i].clear();
}
void AddEdge(int u, int v) {
    g[u].push_back(v);
    g[v].push_back(u);
}
int Find(int u) {
    return u == fa[u] ? u : fa[u] = Find(fa[u]);
}
int LCA(int x, int y, int n) {
    static int tk = 0; tk++;
    x = Find(x), y = Find(y);
    for (; ; swap(x, y)) {
        if (x != n) {
            if (v[x] == tk) return x;
            v[x] = tk;
            x = Find(pre[match[x]]);
        }
    }
}
void Blossom(int x, int y, int l) {
    while (Find(x) != l) {
        pre[x] = y, y = match[x];
        if (s[y] == 1) q.push(y), s[y] = 0;
        if (fa[x] == x) fa[x] = l;
        if (fa[y] == y) fa[y] = l;
        x = pre[y];
    }
}
bool Bfs(int r, int n) {
    for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
    while (!q.empty()) q.pop();
    q.push(r);
    s[r] = 0;
    while (!q.empty()) {
        int x = q.front(); q.pop();
        for (int u : g[x]) {
            if (s[u] == -1) {
                pre[u] = x, s[u] = 1;
                if (match[u] == n) {
                    for (int a = u, b = x, last; b != n; a = last, b = pre[a])
                        last = match[b], match[b] = a, match[a] = b;
                    return true;
                }
            }
        }
    }
}
}

```

```

    q.push(match[u]);
    s[match[u]] = 0;
} else if (!s[u] && Find(u) != Find(x)) {
    int l = LCA(u, x, n);
    Blossom(x, u, l);
    Blossom(u, x, l);
}
}
return false;
}
int Solve(int n) {
    int res = 0;
    for (int x = 0; x < n; ++x) {
        if (match[x] == n) res += Bfs(x, n);
    }
    return res;
}
}

```

### 4.4 Minimum Weight Matching (Clique version)

```

struct Graph {
    // 0-base (Perfect Match)
    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 set_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]];
    return ret>>1;
}
} graph;

```

## 4.5 Minimum Cost Circulation

```

struct Edge { int to, cap, rev, cost; };
vector<Edge> g[kN];
int dist[kN], pv[kN], ed[kN];
bool mark[kN];
int NegativeCycle(int n) {
    memset(mark, false, sizeof(mark));
    memset(dist, 0, sizeof(dist));
    int upd = -1;
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j < n; ++j) {
            int idx = 0;
            for (auto &e : g[j]) {
                if (e.cap > 0 && dist[e.to] > dist[j] + e.cost) {
                    dist[e.to] = dist[j] + e.cost;
                    pv[e.to] = j, ed[e.to] = idx;
                    if (i == n) {
                        upd = j;
                        while (!mark[upd]) mark[upd] = 1, upd = pv[upd];
                        return upd;
                    }
                }
            }
            idx++;
        }
    }
    return -1;
}
int Solve(int n) {
    int rt = -1, ans = 0;
    while ((rt = NegativeCycle(n)) >= 0) {
        memset(mark, false, sizeof(mark));
        vector<pair<int, int>> cyc;
        while (!mark[rt]) {
            cyc.emplace_back(pv[rt], ed[rt]);
            mark[rt] = true;
            rt = pv[rt];
        }
        reverse(cyc.begin(), cyc.end());
        int cap = kInf;
        for (auto &i : cyc) {
            auto &e = g[i.first][i.second];
            cap = min(cap, e.cap);
        }
        for (auto &i : cyc) {
            auto &e = g[i.first][i.second];
            e.cap -= cap;
            g[e.to][e.rev].cap += cap;
            ans += e.cost * cap;
        }
    }
    return ans;
}

```

## 4.6 Flow Models

- Maximum/Minimum flow with lower bound / Circulation problem
  - Construct super source  $S$  and sink  $T$ .
  - For each edge  $(x, y, l, u)$ , connect  $x \rightarrow y$  with capacity  $u - l$ .
  - For each vertex  $v$ , denote by  $in(v)$  the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  - If  $in(v) > 0$ , connect  $S \rightarrow v$  with capacity  $in(v)$ , otherwise, connect  $v \rightarrow T$  with capacity  $-in(v)$ .
    - To maximize, connect  $t \rightarrow s$  with capacity  $\infty$  (skip this in circulation problem), and let  $f$  be the maximum flow from  $S$  to  $T$ . If  $f \neq \sum_{v \in V, in(v) > 0} in(v)$ , there's no solution. Otherwise, the maximum flow from  $s$  to  $t$  is the answer.
    - To minimize, let  $f$  be the maximum flow from  $S$  to  $T$ . Connect  $t \rightarrow s$  with capacity  $\infty$  and let the flow from  $S$  to  $T$  be  $f'$ . If  $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$ , there's no solution. Otherwise,  $f'$  is the answer.
  - The solution of each edge  $e$  is  $l_e + f_e$ , where  $f_e$  corresponds to the flow of edge  $e$  on the graph.
- Construct minimum vertex cover from maximum matching  $M$  on bipartite graph  $(X, Y)$ 
  - Redirect every edge:  $y \rightarrow x$  if  $(x, y) \in M$ ,  $x \rightarrow y$  otherwise.
  - Dfs from unmatched vertices in  $X$ .
  - $x \in X$  is chosen iff  $x$  is unvisited.
  - $y \in Y$  is chosen iff  $y$  is visited.
- Minimum cost cyclic flow

- Construct super source  $S$  and sink  $T$
- For each edge  $(x, y, c)$ , connect  $x \rightarrow y$  with  $(cost, cap) = (c, 1)$  if  $c > 0$ , otherwise connect  $y \rightarrow x$  with  $(cost, cap) = (-c, 1)$
- For each edge with  $c < 0$ , sum these cost as  $K$ , then increase  $d(y)$  by 1, decrease  $d(x)$  by 1
- For each vertex  $v$  with  $d(v) > 0$ , connect  $S \rightarrow v$  with  $(cost, cap) = (0, d(v))$
- For each vertex  $v$  with  $d(v) < 0$ , connect  $v \rightarrow T$  with  $(cost, cap) = (0, -d(v))$
- Flow from  $S$  to  $T$ , the answer is the cost of the flow  $C + K$

- Maximum density induced subgraph
  - Binary search on answer, suppose we're checking answer  $T$
  - Construct a max flow model, let  $K$  be the sum of all weights
  - Connect source  $s \rightarrow v$ ,  $v \in G$  with capacity  $K$
  - For each edge  $(u, v, w)$  in  $G$ , connect  $u \rightarrow v$  and  $v \rightarrow u$  with capacity  $w$
  - For  $v \in G$ , connect it with sink  $v \rightarrow t$  with capacity  $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  - $T$  is a valid answer if the maximum flow  $f < K|V|$
- Minimum weight edge cover
  - For each  $v \in V$  create a copy  $v'$ , and connect  $u' \rightarrow v'$  with weight  $w(u, v)$ .
  - Connect  $v \rightarrow v'$  with weight  $2\mu(v)$ , where  $\mu(v)$  is the cost of the cheapest edge incident to  $v$ .
  - Find the minimum weight perfect matching on  $G'$ .
- Project selection problem
  - If  $p_v > 0$ , create edge  $(s, v)$  with capacity  $p_v$ ; otherwise, create edge  $(v, t)$  with capacity  $-p_v$ .
  - Create edge  $(u, v)$  with capacity  $w$  with  $w$  being the cost of choosing  $u$  without choosing  $v$ .
  - The mincut is equivalent to the maximum profit of a subset of projects.

- 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y}')$$

can be minimized by the mincut of the following graph:

- Create edge  $(x, t)$  with capacity  $c_x$  and create edge  $(s, y)$  with capacity  $c_y$ .
- Create edge  $(x, y)$  with capacity  $c_{xy}$ .
- Create edge  $(x, y)$  and edge  $(x', y')$  with capacity  $c_{xyx'y'}$ .

## 4.7 Dinic

```

template <typename Cap = int64_t>
class Dinic{
private:
    struct E{
        int to, rev;
        Cap cap;
    };
    int n, st, ed;
    vector<vector<E>> G;
    vector<int> lv, idx;
    bool BFS(){
        lv.assign(n, -1);
        queue<int> bfs;
        bfs.push(st); lv[st] = 0;
        while (not bfs.empty()){
            int u = bfs.front(); bfs.pop();
            for (auto e : G[u]) {
                if (e.cap <= 0 or lv[e.to] != -1) continue;
                bfs.push(e.to); lv[e.to] = lv[u] + 1;
            }
        }
        return lv[ed] != -1;
    }
    Cap DFS(int u, Cap f){
        if (u == ed) return f;
        Cap ret = 0;
        for(int &i = idx[u]; i < int(G[u].size()); ++i) {
            auto &e = G[u][i];
            if (e.cap <= 0 or lv[e.to] != lv[u] + 1) continue;
            Cap nf = DFS(e.to, min(f, e.cap));
            ret += nf; e.cap -= nf; f -= nf;
            G[e.to][e.rev].cap += nf;
            if (f == 0) return ret;
        }
        if (ret == 0) lv[u] = -1;
        return ret;
    }
public:
    void init(int n_) { G.assign(n = n_, vector<E>()); }
    void add_edge(int u, int v, Cap c){
        G[u].push_back({v, int(G[v].size()), c});

```

```

    G[v].push_back({u, int(G[u].size()-1, 0)});
}
Cap max_flow(int st_, int ed_){
    st = st_, ed = ed_; Cap ret = 0;
    while (BFS()) {
        idx.assign(n, 0);
        Cap f = DFS(st, numeric_limits<Cap>::max());
        ret += f;
        if (f == 0) break;
    }
    return ret;
}
};

```

#### 4.8 Minimum Cost Maximum Flow

```

class MiniCostMaxiFlow{
    using Cap = int; using Wei = int64_t;
    using PCW = pair<Cap,Wei>;
    static constexpr Cap INF_CAP = 1 << 30;
    static constexpr Wei INF_WEI = 1LL<<60;
private:
    struct Edge{
        int to, back;
        Cap cap; Wei wei;
        Edge() {}
        Edge(int a,int b, Cap c, Wei d):
            to(a),back(b),cap(c),wei(d)
        {}
    };
    int ori, edd;
    vector<vector<Edge>> G;
    vector<int> fa, wh;
    vector<bool> inq;
    vector<Wei> dis;
    PCW SPFA(){
        fill(inq.begin(),inq.end(),false);
        fill(dis.begin(),dis.end(),INF_WEI);
        queue<int> qq; qq.push(ori);
        dis[ori] = 0;
        while(not qq.empty()){
            int u=qq.front();qq.pop();
            inq[u] = false;
            for(int i=0;i<SZ(G[u]);++i){
                Edge e=G[u][i];
                int v=e.to; Wei d=e.wei;
                if(e.cap<=0||dis[v]<=dis[u]+d)
                    continue;
                dis[v] = dis[u] + d;
                fa[v] = u, wh[v] = i;
                if (inq[v]) continue;
                qq.push(v);
                inq[v] = true;
            }
        }
        if(dis[edd]==INF_WEI) return {-1, -1};
        Cap mw=INF_CAP;
        for(int i=edd;i!=ori;i=fa[i])
            mw=min(mw,G[fa[i]][wh[i]].cap);
        for (int i=edd;i!=ori;i=fa[i]){
            auto &eg=G[fa[i]][wh[i]];
            eg.cap -= mw;
            G[eg.to][eg.back].cap+=mw;
        }
        return {mw, dis[edd]};
    }
public:
    void init(int a,int b,int n){
        ori=a,edd=b;
        G.clear();G.resize(n);
        fa.resize(n);wh.resize(n);
        inq.resize(n); dis.resize(n);
    }
    void add_edge(int st, int ed, Cap c, Wei w){
        G[st].emplace_back(ed,SZ(G[ed]),c,w);
        G[ed].emplace_back(st,SZ(G[st])-1,0,-w);
    }
    PCW solve(){
        Cap cc=0; Wei ww=0;
        while(true){
            PCW ret=SPFA();
            if(ret.first==1) break;
            cc+=ret.first;

```

```

            ww+=ret.first * ret.second;
        }
        return {cc,ww};
    }
} mcmf;

```

#### 4.9 GomoryHu Tree

```

int g[maxn];
vector<edge> GomoryHu(int n){
    vector<edge> rt;
    for(int i=1;i<=n;++i)g[i]=1;
    for(int i=2;i<=n;++i){
        int t=g[i];
        flow.reset(); // clear flows on all edge
        rt.push_back({i,t,flow(i,t)});
        flow.walk(i); // bfs points that connected to i (use
            edges not fully flow)
        for(int j=i+1;j<=n;++j){
            if(g[j]==t && flow.connect(j))g[j]=i; // check if i
                can reach j
        }
    }
    return rt;
}

```

#### 4.10 Global Min-Cut

```

const int maxn = 500 + 5;
int w[maxn][maxn], g[maxn];
bool v[maxn], del[maxn];
void add_edge(int x, int y, int c) {
    w[x][y] += c; w[y][x] += c;
}
pair<int, int> phase(int n) {
    memset(v, false, sizeof(v));
    memset(g, 0, sizeof(g));
    int s = -1, t = -1;
    while (true) {
        int c = -1;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            if (c == -1 || g[i] > g[c]) c = i;
        }
        if (c == -1) break;
        v[s = t, t = c] = true;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            g[i] += w[c][i];
        }
    }
    return make_pair(s, t);
}
int mincut(int n) {
    int cut = 1e9;
    memset(del, false, sizeof(del));
    for (int i = 0; i < n - 1; ++i) {
        int s, t; tie(s, t) = phase(n);
        del[t] = true; cut = min(cut, g[t]);
        for (int j = 0; j < n; ++j) {
            w[s][j] += w[t][j]; w[j][s] += w[j][t];
        }
    }
    return cut;
}

```

### 5 Math

#### 5.1 $\lfloor \frac{n}{i} \rfloor$ Enumeration

$$T_0 = 1, T_{i+1} = \lfloor \frac{n}{T_i + 1} \rfloor$$

#### 5.2 ax+by=gcd

```

// ax+ny = 1, ax+ny == ax == 1 (mod n)
void exgcd(lld x, lld y, lld &g, lld &a, lld &b) {
    if (y == 0) g=x, a=1, b=0;
    else exgcd(y, x%y, g, b, a), b-=(x/y)*a;
}

```

#### 5.3 Pollard Rho

```

// does not work when n is prime
// return any non-trivial factor
llu pollard_rho(llu n){
    static auto f=[](llu x,llu k,llu m){

```

```

    return add(k,mul(x,x,m),m);
};
if (!(n&1)) return 2;
mt19937 rnd(120821011);
while(true){
    llu y=2,yy=y,x=rnd()%n,t=1;
    for(llu sz=2;t==1;sz<=1) {
        for(llu i=0;i<sz;++i){
            if(t!=1)break;
            yy=f(yy,x,n);
            t=gcd(yy>y?yy-y:y-yy,n);
        }
        y=yy;
    }
    if(t!=1&&t!=n) return t;
}
}

```

## 5.4 Pi Count (Linear Sieve)

```

static constexpr int N = 1000000 + 5;
lld pi[N];
vector<int> primes;
bool sieved[N];
lld cube_root(lld x){
    lld s=cbrt(x-static_cast<long double>(0.1));
    while(s*s*s <= x) ++s;
    return s-1;
}
lld square_root(lld x){
    lld s=sqrt(x-static_cast<long double>(0.1));
    while(s*s <= x) ++s;
    return s-1;
}
void init(){
    primes.reserve(N);
    primes.push_back(1);
    for(int i=2;i<N;i++) {
        if(!sieved[i]) primes.push_back(i);
        pi[i] = !sieved[i] + pi[i-1];
        for(int p: primes) if(p > 1) {
            if(p * i >= N) break;
            sieved[p * i] = true;
            if(p % i == 0) break;
        }
    }
}
lld phi(lld m, lld n) {
    static constexpr int MM = 80000, NN = 500;
    static lld val[MM][NN];
    if(m<MM&&n<NN&&val[m][n])return val[m][n]-1;
    if(n == 0) return m;
    if(primes[n] >= m) return 1;
    lld ret = phi(m,n-1)-phi(m/primes[n],n-1);
    if(m<MM&&n<NN) val[m][n] = ret+1;
    return ret;
}
lld pi_count(lld);
lld P2(lld m, lld n) {
    lld sm = square_root(m), ret = 0;
    for(lld i = n+1;primes[i]<=sm;i++)
        ret+=pi_count(m/primes[i])-pi_count(primes[i])+1;
    return ret;
}
lld pi_count(lld m) {
    if(m < N) return pi[m];
    lld n = pi_count(cube_root(m));
    return phi(m, n) + n - 1 - P2(m, n);
}

```

## 5.5 Stirling Number

### 5.5.1 First Kind

$S_1(n, k)$  counts the number of permutations of  $n$  elements with  $k$  disjoint cycles.

$$S_1(n, k) = (n-1) \cdot S_1(n-1, k) + S_1(n-1, k-1)$$

$$x(x+1) \dots (x+n-1) = \sum_{k=0}^n S_1(n, k) x^k$$

$$g(x) = x(x+1) \dots (x+n-1) = \sum_{k=0}^n a_k x^k$$

$$\Rightarrow g(x+n) = \sum_{k=0}^n \frac{b_k}{(n-k)!} x^{n-k},$$

$$b_k = \sum_{i=0}^k ((n-i)! a_{n-i}) \cdot \left( \frac{n^{k-i}}{(k-i)!} \right)$$

### 5.5.2 Second Kind

$S_2(n, k)$  counts the number of ways to partition a set of  $n$  elements into  $k$  nonempty sets.

$$S_2(n, k) = S_2(n-1, k-1) + k \cdot S_2(n-1, k)$$

$$S_2(n, k) = \sum_{i=0}^k \binom{k}{i} i^n (-1)^{k-i} = \sum_{i=0}^k \frac{(-1)^i}{i!} \cdot \frac{(k-i)^n}{(k-i)!}$$

## 5.6 Range Sieve

```

const int MAX_SQRT_B = 50000;
const int MAX_L = 200000 + 5;

bool is_prime_small[MAX_SQRT_B];
bool is_prime[MAX_L];

void sieve(lld l, lld r){
    // [l, r)
    for(lld i=2;i<r;i++) is_prime_small[i] = true;
    for(lld i=1;i<r;i++) is_prime[i-1] = true;
    if(l==1) is_prime[0] = false;
    for(lld i=2;i<r;i++){
        if(!is_prime_small[i]) continue;
        for(lld j=i*i;j<r;j+=i) is_prime_small[j]=false;
        for(lld j=std::max(2LL, (l+i-1)/i)*i;j<r;j+=i)
            is_prime[j-1]=false;
    }
}

```

## 5.7 Miller Rabin

```

bool isprime(llu x){
    static llu magic[]={2,325,9375,28178,\
        450775,9780504,1795265022};
    static auto witn=[](llu a,llu u,llu n,int t)
    ->bool{
        if (!(a = mpow(a%n,u,n)))return 0;
        while(t--){
            llu a2=mul(a,a,n);
            if(a2==1 && a!=1 && a!=n-1)
                return 1;
            a = a2;
        }
        return a!=1;
    };
    if(x<2)return 0;
    if(!(x&1))return x==2;
    llu x1=x-1;int t=0;
    while(!(x1&1))x1>>=1,t++;
    for(llu m:magic)if(witn(m,x1,x,t))return 0;
    return 1;
}

```

## 5.8 Extended Euler

$$a^b \equiv \begin{cases} a^b \pmod{\varphi(m)+\varphi(m)} & \text{if } (a, m) \neq 1 \wedge b \geq \varphi(m) \\ a^b \pmod{\varphi(m)} & \text{otherwise} \end{cases} \pmod{m}$$

## 5.9 Gauss Elimination

```

void gauss(vector<vector<double>>& d) {
    int n = d.size(), m = d[0].size();
    for (int i = 0; i < m; ++i) {
        int p = -1;
        for (int j = i; j < n; ++j) {
            if (fabs(d[j][i]) < eps) continue;
            if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) p=j;
        }
        if (p == -1) continue;
        for (int j = 0; j < m; ++j) swap(d[p][j], d[i][j]);
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            double z = d[j][i] / d[i][i];

```



```

    for (int k = 0; k < m; ++k) d[j][k] -= z*d[i][k];
  }
}

```

## 5.10 Fast Fourier Transform

```

const int mod = 1000000007;

const int M1 = 985661441; // G = 3
const int M2 = 998244353;
const int M3 = 1004535809;

int superBigCRT(int64_t A, int64_t B, int64_t C) {
    static_assert(M1 <= M2 && M2 <= M3);
    constexpr int64_t r12 = modpow(M1, M2-2, M2);
    constexpr int64_t r13 = modpow(M1, M3-2, M3);
    constexpr int64_t r23 = modpow(M2, M3-2, M3);
    constexpr int64_t M1M2 = 1LL * M1 * M2 % mod;
    B = (B - A + M2) * r12 % M2;
    C = (C - A + M3) * r13 % M3;
    C = (C - B + M3) * r23 % M3;
    return (A + B * M1 + C * M1M2) % mod;
}

namespace fft {
using VI = vector<int>;
using VL = vector<long long>;
const double pi = acos(-1);
cplx omega[maxn + 1];
void prefft() {
    for (int i = 0; i <= maxn; i++)
        omega[i] = cplx(cos(2 * pi * i / maxn),
            sin(2 * pi * i / maxn));
}
void fft(vector<cplx> &v, int n) {
    int z = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; ++i) {
        int x = 0, j = 0;
        for (; (1 < j) < n; ++j) x ^= (i >> j & 1) << (z - j);
        if (x > i) swap(v[x], v[i]);
    }
    for (int s = 2; s <= n; s <= 1) {
        int z = s >> 1;
        for (int i = 0; i < n; i += s) {
            for (int k = 0; k < z; ++k) {
                cplx x = v[i + z + k] * omega[maxn / s * k];
                v[i + z + k] = v[i + k] - x;
                v[i + k] = v[i + k] + x;
            }
        }
    }
}
void ifft(vector<cplx> &v, int n) {
    fft(v, n);
    reverse(v.begin() + 1, v.end());
    for (int i = 0; i < n; ++i) v[i] = v[i] * cplx(1. / n, 0);
}
VL convolution(const VI &a, const VI &b) {
    // Should be able to handle N <= 10^5, C <= 10^4
    int sz = 1;
    while (sz < a.size() + b.size() - 1) sz <= 1;
    vector<cplx> v(sz);
    for (int i = 0; i < sz; ++i) {
        double re = i < a.size() ? a[i] : 0;
        double im = i < b.size() ? b[i] : 0;
        v[i] = cplx(re, im);
    }
    fft(v, sz);
    for (int i = 0; i <= sz / 2; ++i) {
        int j = (sz - i) & (sz - 1);
        cplx x = (v[i] + v[j].conj()) * (v[i] - v[j].conj())
            * cplx(0, -0.25);
        if (j != i) v[j] = (v[j] + v[i].conj()) * (v[j] - v[i].conj())
            * cplx(0, -0.25);
        v[i] = x;
    }
    ifft(v, sz);
    VL c(sz);
    for (int i = 0; i < sz; ++i) c[i] = round(v[i].re);
    return c;
}
VI convolution_mod(const VI &a, const VI &b, int p) {

```

```

    int sz = 1;
    while (sz + 1 < a.size() + b.size()) sz <= 1;
    vector<cplx> fa(sz), fb(sz);
    for (int i = 0; i < (int)a.size(); ++i)
        fa[i] = cplx(a[i] & ((1 << 15) - 1), a[i] >> 15);
    for (int i = 0; i < (int)b.size(); ++i)
        fb[i] = cplx(b[i] & ((1 << 15) - 1), b[i] >> 15);
    fft(fa, sz), fft(fb, sz);
    double r = 0.25 / sz;
    cplx r2(0, -1), r3(r, 0), r4(0, -r), r5(0, 1);
    for (int i = 0; i <= (sz >> 1); ++i) {
        int j = (sz - i) & (sz - 1);
        cplx a1 = (fa[i] + fa[j].conj());
        cplx a2 = (fa[i] - fa[j].conj()) * r2;
        cplx b1 = (fb[i] + fb[j].conj()) * r3;
        cplx b2 = (fb[i] - fb[j].conj()) * r4;
        if (i != j) {
            cplx c1 = (fa[j] + fa[i].conj());
            cplx c2 = (fa[j] - fa[i].conj()) * r2;
            cplx d1 = (fb[j] + fb[i].conj()) * r3;
            cplx d2 = (fb[j] - fb[i].conj()) * r4;
            fa[i] = c1 * d1 + c2 * d2 * r5;
            fb[i] = c1 * d2 + c2 * d1;
        }
        fa[j] = a1 * b1 + a2 * b2 * r5;
        fb[j] = a1 * b2 + a2 * b1;
    }
    fft(fa, sz), fft(fb, sz);
    vector<int> res(sz);
    for (int i = 0; i < sz; ++i) {
        long long a = round(fa[i].re), b = round(fb[i].re),
            c = round(fa[i].im);
        res[i] = (a + ((b % p) << 15) + ((c % p) << 30)) % p;
    }
    return res;
}
}

```

## 5.11 Chinese Remainder

```

lld crt(lld ans[], lld pri[], int n) {
    lld M = 1, ret = 0;
    for (int i = 0; i < n; ++i) M *= pri[i];
    for (int i = 0; i < n; ++i) {
        lld iv = (gcd(M/pri[i], pri[i]).FF + pri[i]) % pri[i];
        ret += (ans[i] * (M/pri[i]) % M * iv) % M;
        ret %= M;
    }
    return ret;
}
/*
Another:
x = a1 % m1
x = a2 % m2
g = gcd(m1, m2)
assert((a1-a2)%g==0)
[p, q] = exgcd(m2/g, m1/g)
return a2+m2*(p*(a1-a2)/g)
0 <= x < lcm(m1, m2)
*/

```

## 5.12 Berlekamp Massey

```

// x: 1-base, p[]: 0-base
template<size_t N>
vector<llf> BM(llf x[N], size_t n) {
    size_t f[N] = {0}, t = 0; llf d[N];
    vector<llf> p[N];
    for (size_t i = 1, b = 0; i <= n; ++i) {
        for (size_t j = 0; j < p[t].size(); ++j)
            d[i] += x[i-j-1] * p[t][j];
        if (abs(d[i] - x[i]) <= EPS) continue;
        f[t] = i; if (!t) { p[++t].resize(i); continue; }
        vector<llf> cur(i - f[b] - 1);
        llf k = -d[i] / d[f[b]]; cur.PB(-k);
        for (size_t j = 0; j < p[b].size(); ++j)
            cur.PB(p[b][j] * k);
        if (cur.size() < p[t].size()) cur.resize(p[t].size());
        for (size_t j = 0; j < p[t].size(); ++j) cur[j] += p[t][j];
        if (i - f[b] + p[b].size() >= p[t].size()) b = t;
        p[++t] = cur;
    }
    return p[t];
}

```

## 5.13 NTT

```

template <int mod, int G, int maxn>
struct NTT {
    static_assert (maxn == (maxn & -maxn));
    int roots[maxn];
    NTT () {
        int r = modpow(G, (mod - 1) / maxn);
        for (int i = maxn >> 1; i; i >>= 1) {
            roots[i] = 1;
            for (int j = 1; j < i; j++)
                roots[i + j] = modmul(roots[i + j - 1], r);
            r = modmul(r, r);
        }
    }
    // n must be 2^k, and 0 <= F[i] < mod
    void inplace_ntt(int n, int F[], bool inv = false) {
        for (int i = 0, j = 0; i < n; i++) {
            if (i < j) swap(F[i], F[j]);
            for (int k = n >> 1; (j ^= k) < k; k >>= 1);
        }
        for (int s = 1; s < n; s *= 2) {
            for (int i = 0; i < n; i += s * 2) {
                for (int j = 0; j < s; j++) {
                    int a = F[i + j];
                    int b = modmul(F[i + j + s], roots[s + j]);
                    F[i + j] = modadd(a, b); // a + b
                    F[i + j + s] = modsub(a, b); // a - b
                }
            }
        }
        if (inv) {
            int invn = modinv(n);
            for (int i = 0; i < n; i++)
                F[i] = modmul(F[i], invn);
            reverse(F + 1, F + n);
        }
    };
    const int P=2013265921, root=31;
    const int MAXN=1<<20;
    NTT<P, root, MAXN> ntt;
}

```

## 5.14 Polynomial Operations

```

using VL = vector<LL>;
#define fi(s, n) for (int i=int(s); i<int(n); ++i)
#define Fi(s, n) for (int i=int(n); i>int(s); --i)
int n2k(int n) {
    int sz = 1; while (sz < n) sz <= 1;
    return sz;
}
template<int MAXN, LL P, LL RT> // MAXN = 2^k
struct Poly { // coefficients in [0, P)
    static NTT<MAXN, P, RT> ntt;
    VL coef;
    int n() const { return coef.size(); } // n()>=1
    LL *data() { return coef.data(); }
    const LL *data() const { return coef.data(); }
    LL &operator[](size_t i) { return coef[i]; }
    const LL &operator[](size_t i) const { return coef[i]; }
    Poly(initializer_list<LL> a) : coef(a) {}
    explicit Poly(int _n = 1) : coef(_n) {}
    Poly(const LL *arr, int _n) : coef(arr, arr + _n) {}
    Poly(const Poly &p, int _n) : coef(_n) {
        copy_n(p.data(), min(p.n(), _n), data());
    }
    Poly& irev() { return reverse(data(), data() + n()), *this; }
    Poly& isz(int _n) { return coef.resize(_n), *this; }
    Poly& iadd(const Poly &rhs) { // n() == rhs.n()
        fi(0, n()) if ((coef[i] += rhs[i]) >= P) coef[i] -= P;
        return *this;
    }
    Poly& imul(LL k) {
        fi(0, n()) coef[i] = coef[i] * k % P;
        return *this;
    }
    Poly Mul(const Poly &rhs) const {
        const int _n = n2k(n()) + rhs.n() - 1;
        Poly X(*this, _n), Y(rhs, _n);
        ntt(X.data(), _n), ntt(Y.data(), _n);
        fi(0, _n) X[i] = X[i] * Y[i] % P;
        ntt(X.data(), _n, true);
    }
}

```

```

    return X.isz(n() + rhs.n() - 1);
}
Poly Inv() const { // coef[0] != 0
    if (n() == 1) return {ntt.minv(coef[0])};
    const int _n = n2k(n()) * 2;
    Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(_n);
    Poly Y(*this, _n);
    ntt(Xi.data(), _n), ntt(Y.data(), _n);
    fi(0, _n) {
        Xi[i] *= (2 - Xi[i] * Y[i]) % P;
        if ((Xi[i] % = P) < 0) Xi[i] += P;
    }
    ntt(Xi.data(), _n, true);
    return Xi.isz(n());
}
Poly Sqrt() const { // Jacobi(coef[0], P) = 1
    if (n() == 1) return {QuadraticResidue(coef[0], P)};
    Poly X = Poly(*this, (n() + 1) / 2).Sqrt().isz(n());
    return X.iadd(Mul(X.Inv()).isz(n())).imul(P / 2 + 1);
}
pair<Poly, Poly> DivMod(const Poly &rhs) const {
    // (rhs).back() != 0
    if (n() < rhs.n()) return {{0}, *this};
    const int _n = n() - rhs.n() + 1;
    Poly X(rhs); X.irev().isz(_n);
    Poly Y(*this); Y.irev().isz(_n);
    Poly Q = Y.Mul(X.Inv()).isz(_n).irev();
    X = rhs.Mul(Q), Y = *this;
    fi(0, n()) if ((Y[i] -= X[i]) < 0) Y[i] += P;
    return {Q, Y.isz(max(1, rhs.n() - 1))};
}
Poly Dx() const {
    Poly ret(n() - 1);
    fi(0, ret.n()) ret[i] = (i + 1) * coef[i + 1] % P;
    return ret.isz(max(1, ret.n()));
}
Poly Sx() const {
    Poly ret(n() + 1);
    fi(0, n()) ret[i + 1] = ntt.minv(i + 1) * coef[i] % P;
    return ret;
}
Poly _tmul(int nn, const Poly &rhs) const {
    Poly Y = Mul(rhs).isz(n() + nn - 1);
    return Poly(Y.data() + n() - 1, nn);
}
VL _eval(const VL &x, const auto up) const {
    const int _n = (int)x.size();
    if (!_n) return {};
    vector<Poly> down(_n * 2);
    down[1] = DivMod(up[1]).second;
    fi(2, _n * 2) down[i] = down[i / 2].DivMod(up[i]).second;
    /* down[1] = Poly(up[1]).irev().isz(n()).Inv().irev()
       ._tmul(_n, *this);
    fi(2, _n * 2) down[i] = up[i ^ 1]._tmul(up[i].n() -
    1, down[i / 2]); */
    VL y(_n);
    fi(0, _n) y[i] = down[_n + i][0];
    return y;
}
static vector<Poly> _tree1(const VL &x) {
    const int _n = (int)x.size();
    vector<Poly> up(_n * 2);
    fi(0, _n) up[_n + i] = {(x[i] ? P - x[i] : 0), 1};
    Fi(0, _n - 1) up[i] = up[i * 2].Mul(up[i * 2 + 1]);
    return up;
}
VL Eval(const VL &x) const { return _eval(x, _tree1(x)); }
static Poly Interpolate(const VL &x, const VL &y) {
    const int _n = (int)x.size();
    vector<Poly> up = _tree1(x), down(_n * 2);
    VL z = up[1].Dx()._eval(x, up);
    fi(0, _n) z[i] = y[i] * ntt.minv(z[i]) % P;
    fi(0, _n) down[_n + i] = {z[i]};
    Fi(0, _n - 1) down[i] = down[i * 2].Mul(up[i * 2 + 1])
        .iadd(down[i * 2 + 1].Mul(up[i * 2]));
    return down[1];
}
Poly Ln() const { // coef[0] == 1
    return Dx().Mul(Inv()).Sx().isz(n());
}
Poly Exp() const { // coef[0] == 0
    if (n() == 1) return {1};
}

```

```

Poly X = Poly(*this, (n() + 1)/2).Exp().isz(n());
Poly Y = X.Ln(); Y[0] = P - 1;
fi(0, n()) if((Y[i] = coef[i] - Y[i]) < 0)Y[i]+=P;
return X.Mul(Y).isz(n());
}
Poly Pow(const string &K) const {
    int nz = 0;
    while (nz < n() && !coef[nz]) ++nz;
    LL nk = 0, nk2 = 0;
    for (char c : K) {
        nk = (nk * 10 + c - '0') % P;
        nk2 = nk2 * 10 + c - '0';
        if (nk2 * nz >= n()) return Poly(n());
        nk2 %= P - 1;
    }
    if (!nk && !nk2) return Poly({1}, n());
    Poly X(data() + nz, n() - nz * nk2);
    LL x0 = X[0];
    return X.imul(ntt.minv(x0)).Ln().imul(nk).Exp()
        .imul(ntt.mpow(x0, nk2)).irev().isz(n()).irev();
}
Poly InvMod(int L) { // (to evaluate linear recursion)
    Poly R{1, 0}; // *this * R mod x^L = 1 (*this[0] == 1)
    for (int level = 0; (1 << level) < L; ++level) {
        Poly O = R.Mul(Poly(data(), min(2 << level, n())));
        Poly Q(2 << level); Q[0] = 1;
        for (int j = (1 << level); j < (2 << level); ++j)
            Q[j] = (P - O[j]) % P;
        R = R.Mul(Q).isz(4 << level);
    }
    return R.isz(L);
}
static LL LinearRecursion(const VL&a, const VL&c, LL n){
    // a_n = \sum c_j a_{n-j}
    const int k = (int)a.size();
    assert((int)c.size() == k + 1);
    Poly C(k + 1), W({1}, k), M = {0, 1};
    fi(1, k + 1) C[k - i] = c[i] ? P - c[i] : 0;
    C[k] = 1;
    while (n) {
        if (n % 2) W = W.Mul(M).DivMod(C).second;
        n /= 2, M = M.Mul(M).DivMod(C).second;
    }
    LL ret = 0;
    fi(0, k) ret = (ret + W[i] * a[i]) % P;
    return ret;
}
};
#undef fi
#undef Fi
using Poly_t = Poly<131072 * 2, 998244353, 3>;
template<> decltype(Poly_t::ntt) Poly_t::ntt = {};

```

## 5.15 FWT

```

/* xor convolution:
* x = (x0,x1), y = (y0,y1)
* z = ( x0y0 + x1y1, x0y1 + x1y0 )
* ==>
* x' = ( x0+x1, x0-x1 ), y' = ( y0+y1, y0-y1 )
* z' = ( ( x0+x1 )( y0+y1 ), ( x0-x1 )( y0-y1 ) )
* z = (1/2) * z'
* or convolution:
* x = (x0, x0+x1), inv = (x0, x1-x0) w/o final div
* and convolution:
* x = (x0+x1, x1), inv = (x0-x1, x1) w/o final div */
const LL MOD = 1e9+7;
inline void fwt( LL x[ MAXN ], int N, bool inv=0 ) {
    for( int d = 1; d < N; d <= 1 ) {
        int d2 = d<<1;
        for( int s = 0; s < N; s += d2 )
            for( int i = s, j = s+d; i < s+d; i++, j++ ){
                LL ta = x[ i ], tb = x[ j ];
                x[ i ] = ta+tb;
                x[ j ] = ta-tb;
                if( x[ i ] >= MOD ) x[ i ] -= MOD;
                if( x[ j ] < 0 ) x[ j ] += MOD;
            }
    }
    if( inv )
        for( int i = 0; i < N; i++ ) {
            x[ i ] *= inv( N, MOD );

```

```

        x[ i ] %= MOD;
    }
}

```

## 5.16 DiscreteLog

```

template<typename Int>
Int BSGS(Int x, Int y, Int M) {
    // x^? \equiv y (mod M)
    Int t = 1, c = 0, g = 1;
    for (Int M_ = M; M_ > 0; M_ >= 1)
        g = g * x % M;
    for (g = gcd(g, M); t % g != 0; ++c) {
        if (t == y) return c;
        t = t * x % M;
    }
    if (y % g != 0) return -1;
    t /= g, y /= g, M /= g;
    Int h = 0, gs = 1;
    for (; h * h < M; ++h) gs = gs * x % M;
    unordered_map<Int, Int> bs;
    for (Int s = 0; s < h; bs[y] = ++s)
        y = y * x % M;
    for (Int s = 0; s < M; s += h) {
        t = t * gs % M;
        if (bs.count(t)) return c + s + h - bs[t];
    }
    return -1;
}

```

## 5.17 FloorSum

```

// @param n `n < 2^32`
// @param m `1 <= m < 2^32`
// @return sum_{i=0}^{n-1} floor((ai + b)/m) mod 2^64
llu floor_sum_unsigned(llu n, llu m, llu a, llu b) {
    llu ans = 0;
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m); a %= m;
        }
        if (b >= m) {
            ans += n * (b / m); b %= m;
        }
        llu y_max = a * n + b;
        if (y_max < m) break;
        // y_max < m * (n + 1)
        // floor(y_max / m) <= n
        n = (llu)(y_max / m), b = (llu)(y_max % m);
        swap(m, a);
    }
    return ans;
}
lld floor_sum(lld n, lld m, lld a, lld b) {
    llu ans = 0;
    if (a < 0) {
        llu a2 = (a % m + m) % m;
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        llu b2 = (b % m + m) % m;
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    return ans + floor_sum_unsigned(n, m, a, b);
}

```

## 5.18 ExtendedFloorSum

$$g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor \cdot \frac{n(n+1)}{2} \\ + g(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1)) \\ - h(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$

$$h(a, b, c, n) = \sum_{i=0}^n \left\lfloor \frac{ai+b}{c} \right\rfloor^2$$

$$= \begin{cases} \left\lfloor \frac{a}{c} \right\rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \left\lfloor \frac{b}{c} \right\rfloor^2 \cdot (n+1) \\ + \left\lfloor \frac{a}{c} \right\rfloor \cdot \left\lfloor \frac{b}{c} \right\rfloor \cdot n(n+1) \\ + h(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{a}{c} \right\rfloor \cdot g(a \bmod c, b \bmod c, c, n) \\ + 2 \left\lfloor \frac{b}{c} \right\rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) \\ - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

## 5.19 Quadratic residue

```
struct S {
    int MOD, w;
    int64_t x, y;
    S(int m, int w_=-1, int64_t x_=1, int64_t y_=-1)
        : MOD(m), w(w_), x(x_), y(y_) {}
    S operator*(const S &rhs) const {
        int w_ = w;
        if (w_ == -1) w_ = rhs.w;
        assert(w_ != -1 and w_ == rhs.w);
        return { MOD, w_,
            (x * rhs.x + y * rhs.y % MOD * w) % MOD,
            (x * rhs.y + y * rhs.x) % MOD };
    }
};

int get_root(int n, int P) {
    if (P == 2 or n == 0) return n;
    if (qpow(n, (P - 1) / 2, P) != 1) return -1;
    auto check = [&](int x) {
        return qpow(x, (P - 1) / 2, P);
    };
    if (check(n) == P-1) return -1;
    int64_t a; int w; mt19937 rnd(7122);
    do { a = rnd() % P;
        w = ((a * a - n) % P + P) % P;
    } while (check(w) != P - 1);
    return qpow(S(P, w, a, 1), (P + 1) / 2).x;
}
```

## 5.20 De-Bruijn

```
int res[maxn], aux[maxn], sz;
void db(int t, int p, int n, int k) {
    if (t > n) {
        if (n % p == 0)
            for (int i = 1; i <= p; ++i)
                res[sz++] = aux[i];
    } else {
        aux[t] = aux[t - p];
        db(t + 1, p, n, k);
        for (int i = aux[t - p] + 1; i < k; ++i) {
            aux[t] = i;
            db(t + 1, t, n, k);
        }
    }
}

int de_bruijn(int k, int n) {
    // return cyclic string of len k^n s.t. every string
    // of len n using k char appears as a substring.
    if (k == 1) {
        res[0] = 0;
        return 1;
    }
    for (int i = 0; i < k * n; i++) aux[i] = 0;
    sz = 0;
    db(1, 1, n, k);
    return sz;
}
```

## 5.21 Simplex Construction

Standard form: maximize  $\sum_{1 \leq i \leq n} c_i x_i$  such that for all  $1 \leq j \leq m$ ,  $\sum_{1 \leq i \leq n} A_{ji} x_i \leq b_j$  and  $x_i \geq 0$  for all  $1 \leq i \leq n$ .

1. In case of minimization, let  $c'_i = -c_i$
2.  $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j \rightarrow \sum_{1 \leq i \leq n} -A_{ji} x_i \leq -b_j$
3.  $\sum_{1 \leq i \leq n} A_{ji} x_i = b_j$ 
  - $\sum_{1 \leq i \leq n} A_{ji} x_i \leq b_j$
  - $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j$
4. If  $x_i$  has no lower bound, replace  $x_i$  with  $x_i - x'_i$

## 5.22 Simplex

```
namespace simplex {
    // maximize c^T x under Ax <= B
    // return VD(n, -inf) if the solution doesn't exist
    // return VD(n, +inf) if the solution is unbounded
    using VD = vector<double>;
    using VVD = vector<vector<double>>>;
    const double eps = 1e-9;
    const double inf = 1e+9;
    int n, m;
    VVD d;
    vector<int> p, q;
    void pivot(int r, int s) {
        double inv = 1.0 / d[r][s];
        for (int i = 0; i < m + 2; ++i)
            for (int j = 0; j < n + 2; ++j)
                if (i != r && j != s)
                    d[i][j] -= d[r][j] * d[i][s] * inv;
        for (int i = 0; i < m + 2; ++i) if (i != r) d[i][s] *= -inv;
        for (int j = 0; j < n + 2; ++j) if (j != s) d[r][j] *= +inv;
        d[r][s] = inv; swap(p[r], q[s]);
    }
    bool phase(int z) {
        int x = m + z;
        while (true) {
            int s = -1;
            for (int i = 0; i <= n; ++i) {
                if (!z && q[i] == -1) continue;
                if (s == -1 || d[x][i] < d[x][s]) s = i;
            }
            if (d[x][s] > -eps) return true;
            int r = -1;
            for (int i = 0; i < m; ++i) {
                if (d[i][s] < eps) continue;
                if (r == -1 || \
                    d[i][n+1]/d[i][s] < d[r][n+1]/d[r][s]) r = i;
            }
            if (r == -1) return false;
            pivot(r, s);
        }
    }
    VD solve(const VVD &a, const VD &b, const VD &c) {
        m = b.size(), n = c.size();
        d = VVD(m + 2, VD(n + 2));
        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j) d[i][j] = a[i][j];
        p.resize(m), q.resize(n + 1);
        for (int i = 0; i < m; ++i)
            p[i] = n + i, d[i][n] = -1, d[i][n + 1] = b[i];
        for (int i = 0; i < n; ++i) q[i] = i, d[m][i] = -c[i];
        q[n] = -1, d[m + 1][n] = 1;
        int r = 0;
        for (int i = 1; i < m; ++i)
            if (d[i][n + 1] < d[r][n + 1]) r = i;
        if (d[r][n + 1] < -eps) {
            pivot(r, n);
            if (!phase(1) || d[m + 1][n + 1] < -eps)
                return VD(n, -inf);
            for (int i = 0; i < m; ++i) if (p[i] == -1) {
                int s = min_element(d[i].begin(), d[i].end() - 1)
                    - d[i].begin();
                pivot(i, s);
            }
        }
        if (!phase(0)) return VD(n, inf);
        VD x(n);
        for (int i = 0; i < m; ++i)
            if (p[i] < n) x[p[i]] = d[i][n + 1];
        return x;
    }
}
```

## 5.23 Characteristic Polynomial

```
vector<vector<int>>> Hessenberg(const vector<vector<int>>> &A) {
    int N = A.size();
    vector<vector<int>>> H = A;
    for (int i = 0; i < N - 2; ++i) {
        if (!H[i + 1][i]) {
            for (int j = i + 2; j < N; ++j) {
                if (H[j][i]) {
                    for (int k = i; k < N; ++k) swap(H[i + 1][k], H[j][k]);
                }
            }
        }
    }
}
```

```

    for (int k = 0; k < N; ++k) swap(H[k][i + 1], H[k][j]);
    break;
}
}
if (!H[i + 1][i]) continue;
int val = fpow(H[i + 1][i], kP - 2);
for (int j = i + 2; j < N; ++j) {
    int coef = 1LL * val * H[j][i] % kP;
    for (int k = i; k < N; ++k) H[j][k] = (H[j][k] + 1LL * H[i + 1][k] * (kP - coef)) % kP;
    for (int k = 0; k < N; ++k) H[k][i + 1] = (H[k][i + 1] + 1LL * H[k][j] * coef) % kP;
}
}
return H;
}
vector<int> CharacteristicPoly(const vector<vector<int>> &A) {
    int N = A.size();
    auto H = Hessenberg(A);
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) H[i][j] = kP - H[i][j];
    }
    vector<vector<int>> P(N + 1, vector<int>(N + 1));
    P[0][0] = 1;
    for (int i = 1; i <= N; ++i) {
        P[i][0] = 0;
        for (int j = 1; j <= i; ++j) P[i][j] = P[i - 1][j - 1];
        int val = 1;
        for (int j = i - 1; j >= 0; --j) {
            int coef = 1LL * val * H[j][i - 1] % kP;
            for (int k = 0; k <= j; ++k) P[i][k] = (P[i][k] + 1LL * P[j][k] * coef) % kP;
            if (j) val = 1LL * val * (kP - H[j][j - 1]) % kP;
        }
    }
    if (N & 1) {
        for (int i = 0; i <= N; ++i) P[N][i] = kP - P[N][i];
    }
    return P[N];
}

```

## 5.24 Partition Number

```

int b = sqrt(n);
ans[0] = tmp[0] = 1;
for (int i = 1; i <= b; i++) {
    for (int rep = 0; rep < 2; rep++)
        for (int j = i; j <= n - i * i; j++)
            modadd(tmp[j], tmp[j - i]);
    for (int j = i * i; j <= n; j++)
        modadd(ans[j], tmp[j - i * i]);
}

```

# 6 Geometry

## 6.1 Basic Geometry

```

using coord_t = int;
using Real = double;
using Point = std::complex<coord_t>;
int sgn(coord_t x) {
    return (x > 0) - (x < 0);
}
coord_t dot(Point a, Point b) {
    return real(conj(a) * b);
}
coord_t cross(Point a, Point b) {
    return imag(conj(a) * b);
}
int ori(Point a, Point b, Point c) {
    return sgn(cross(b - a, c - a));
}
bool operator<(const Point &a, const Point &b) {
    return real(a) != real(b)
        ? real(a) < real(b) : imag(a) < imag(b);
}
int argCmp(Point a, Point b) {
    // -1 / 0 / 1 <=> < / == / > (atan2)
    int qa = (imag(a) == 0

```

```

        ? (real(a) < 0 ? 3 : 1) : (imag(a) < 0 ? 0 : 2));
    int qb = (imag(b) == 0
        ? (real(b) < 0 ? 3 : 1) : (imag(b) < 0 ? 0 : 2));
    if (qa != qb)
        return sgn(qa - qb);
    return sgn(cross(b, a));
}
template <typename V> Real area(const V &pt) {
    coord_t ret = 0;
    for (int i = 1; i + 1 < (int)pt.size(); i++)
        ret += cross(pt[i] - pt[0], pt[i + 1] - pt[0]);
    return ret / 2.0;
}

```

## 6.2 2D Convex Hull

```

template<typename PT>
vector<PT> buildConvexHull(vector<PT> d) {
    sort(ALL(d), [](const PT& a, const PT& b){
        return tie(a.x, a.y) < tie(b.x, b.y);});
    vector<PT> s(SZ(d) < 1);
    int o = 0;
    for(auto p: d) {
        while(o >= 2 && cross(p - s[o - 2], s[o - 1] - s[o - 2]) <= 0)
            o--;
        s[o++] = p;
    }
    for(int i = SZ(d) - 2, t = o + 1; i >= 0; i--) {
        while(o >= t && cross(d[i] - s[o - 2], s[o - 1] - s[o - 2]) <= 0)
            o--;
        s[o++] = d[i];
    }
    s.resize(o - 1);
    return s;
}

```

## 6.3 3D Convex Hull

```

// return the faces with pt indexes
int flag[MXN][MXN];
struct Point {
    ld x, y, z;
    Point operator * (const ld &b) const {
        return (Point){x * b, y * b, z * b};
    }
    Point operator * (const Point &b) const {
        return (Point){y * b.z - b.y * z, z * b.x - b.z * x, x * b.y - b.x * y};
    }
};
Point ver(Point a, Point b, Point c) {
    return (b - a) * (c - a);
}
vector<Face> convex_hull_3D(const vector<Point> pt) {
    int n = SZ(pt), ftop = 0;
    REP(i, n) REP(j, n) flag[i][j] = 0;
    vector<Face> now;
    now.emplace_back(0, 1, 2);
    now.emplace_back(2, 1, 0);
    for (int i = 3; i < n; i++) {
        ftop++; vector<Face> next;
        REP(j, SZ(now)) {
            Face& f = now[j]; int ff = 0;
            ld d = (pt[i] - pt[f.a]).dot(
                ver(pt[f.a], pt[f.b], pt[f.c]));
            if (d <= 0) next.push_back(f);
            if (d > 0) ff = ftop;
            else if (d < 0) ff = -ftop;
            flag[f.a][f.b] = flag[f.b][f.c] = flag[f.c][f.a] = ff;
        }
        REP(j, SZ(now)) {
            Face& f = now[j];
            if (flag[f.a][f.b] > 0 &&
                flag[f.a][f.b] != flag[f.b][f.a])
                next.emplace_back(f.a, f.b, i);
            if (flag[f.b][f.c] > 0 &&
                flag[f.b][f.c] != flag[f.c][f.b])
                next.emplace_back(f.b, f.c, i);
            if (flag[f.c][f.a] > 0 &&
                flag[f.c][f.a] != flag[f.a][f.c])
                next.emplace_back(f.c, f.a, i);
        }
        now = next;
    }
    return now;
}

```



## 6.4 2D Farthest Pair

```
// stk is from convex hull
n = (int)(stk.size());
int pos = 1, ans = 0; stk.push_back(stk[0]);
for(int i=0; i<n; i++) {
    while(abs(cross(stk[i+1]-stk[i],
        stk[(pos+1)%n]-stk[i])) >
        abs(cross(stk[i+1]-stk[i],
        stk[pos]-stk[i]))) pos = (pos+1)%n;
    ans = max({ans, dis(stk[i], stk[pos]),
        dis(stk[i+1], stk[pos])});
}
```

## 6.5 2D Closest Pair

```
struct cmp_y {
    bool operator()(const P& p, const P& q) const {
        return p.y < q.y;
    }
};
multiset<P, cmp_y> s;
void solve(P a[], int n) {
    sort(a, a + n, [](const P& p, const P& q) {
        return tie(p.x, p.y) < tie(q.x, q.y);
    });
    llf d = INF; int pt = 0;
    for (int i = 0; i < n; ++i) {
        while (pt < i and a[i].x - a[pt].x >= d)
            s.erase(s.find(a[pt]));
        auto it = s.lower_bound(P(a[i].x, a[i].y - d));
        while (it != s.end() and it->y - a[i].y < d)
            d = min(d, dis(*it, a[i]));
        s.insert(a[i]);
    }
}
```

## 6.6 kD Closest Pair (3D ver.)

```
llf solve(vector<P> v) {
    shuffle(v.begin(), v.end(), mt19937());
    unordered_map<lld, unordered_map<lld,
        unordered_map<lld, int>>> m;
    llf d = dis(v[0], v[1]);
    auto Idx = [&d] (llf x) -> lld {
        return round(x * 2 / d) + 0.1; };
    auto rebuild_m = [&m, &v, &Idx] (int k) {
        m.clear();
        for (int i = 0; i < k; ++i)
            m[Idx(v[i].x)][Idx(v[i].y)]
                [Idx(v[i].z)] = i;
        rebuild_m(2);
    };
    for (size_t i = 2; i < v.size(); ++i) {
        const lld kx = Idx(v[i].x), ky = Idx(v[i].y),
            kz = Idx(v[i].z); bool found = false;
        for (int dx = -2; dx <= 2; ++dx) {
            const lld nx = dx + kx;
            if (m.find(nx) == m.end()) continue;
            auto& mm = m[nx];
            for (int dy = -2; dy <= 2; ++dy) {
                const lld ny = dy + ky;
                if (mm.find(ny) == mm.end()) continue;
                auto& mmm = mm[ny];
                for (int dz = -2; dz <= 2; ++dz) {
                    const lld nz = dz + kz;
                    if (mmm.find(nz) == mmm.end()) continue;
                    const int p = mmm[nz];
                    if (dis(v[p], v[i]) < d) {
                        d = dis(v[p], v[i]);
                        found = true;
                    }
                }
            }
        }
        if (found) rebuild_m(i + 1);
        else m[kx][ky][kz] = i;
    }
    return d;
}
```

## 6.7 Simulated Annealing

```
llf anneal() {
    mt19937 rnd_engine( seed );
    uniform_real_distribution< llf > rnd( 0, 1 );
```

```
const llf dT = 0.001;
// Argument p
llf S_cur = calc( p ), S_best = S_cur;
for ( llf T = 2000 ; T > EPS ; T -= dT ) {
    // Modify p to p_prime
    const llf S_prime = calc( p_prime );
    const llf delta_c = S_prime - S_cur;
    llf prob = min( ( llf ) 1, exp( -delta_c / T ) );
    if ( rnd( rnd_engine ) <= prob )
        S_cur = S_prime, p = p_prime;
    if ( S_prime < S_best ) // find min
        S_best = S_prime, p_best = p_prime;
}
return S_best;
}
```

## 6.8 Half Plane Intersection

```
// NOTE: Point is complex<Real>
// cross(pt-line.st, line.dir)<=0 <=> pt in half plane
struct Line {
    Point st, ed;
    Point dir;
    Line (Point _s, Point _e)
        : st(_s), ed(_e), dir(_e - _s) {}
};

bool operator<(const Line &lhs, const Line &rhs) {
    if (int cmp = argCmp(lhs.dir, rhs.dir))
        return cmp == -1;
    return ori(lhs.st, lhs.ed, rhs.st) < 0;
}

Point intersect(const Line &A, const Line &B) {
    Real t = cross(B.st - A.st, B.dir) /
        cross(A.dir, B.dir);
    return A.st + t * A.dir;
}

Real HPI(vector<Line> &lines) {
    sort(lines.begin(), lines.end());
    deque<Line> que;
    deque<Point> pt;
    que.push_back(lines[0]);
    for (int i = 1; i < (int)lines.size(); i++) {
        if (argCmp(lines[i].dir, lines[i-1].dir) == 0)
            continue;
#define POP(L, R) \
        while (pt.size() > 0 \
            && ori(L.st, L.ed, pt.back()) < 0) \
            pt.pop_back(), que.pop_back(); \
        while (pt.size() > 0 \
            && ori(R.st, R.ed, pt.front()) < 0) \
            pt.pop_front(), que.pop_front();
        POP(lines[i], lines[i]);
        pt.push_back(intersect(que.back(), lines[i]));
        que.push_back(lines[i]);
    }
    POP(que.front(), que.back())
    if (que.size() <= 1 ||
        argCmp(que.front().dir, que.back().dir) == 0)
        return 0;
    pt.push_back(intersect(que.front(), que.back()));
    return area(pt);
}
```

## 6.9 Minkowski Sum

```
vector<pll> Minkowski(vector<pll> A, vector<pll> B) {
    hull(A), hull(B);
    vector<pll> C(1, A[0] + B[0]), s1, s2;
    for(int i = 0; i < SZ(A); ++i)
        s1.pb(A[(i + 1) % SZ(A)] - A[i]);
    for(int i = 0; i < SZ(B); ++i)
        s2.pb(B[(i + 1) % SZ(B)] - B[i]);
    for(int p1 = 0, p2 = 0; p1 < SZ(A) || p2 < SZ(B);)
        if (p2 >= SZ(B)
            || (p1 < SZ(A) && cross(s1[p1], s2[p2]) >= 0))
            C.pb(C.back() + s1[p1++]);
        else
            C.pb(C.back() + s2[p2++]);
    return hull(C), C;
}
```

## 6.10 Intersection of line and Circle

```
vector<pdd> line_interCircle(const pdd &p1,
    const pdd &p2, const pdd &c, const double r){
    pdd ft=foot(p1,p2,c), vec=p2-p1;
    double dis=abs(c-ft);
    if(fabs(dis-r)<eps) return vector<pdd>{ft};
    if(dis>r) return {};
    vec=vec*sqrt(r*r-dis*dis)/abs(vec);
    return vector<pdd>{ft+vec, ft-vec};
}
```

## 6.11 Intersection of Polygon and Circle

```
// Divides into multiple triangle, and sum up
// test by HDU2892
const double PI=acos(-1);
double _area(pdd pa, pdd pb, double r){
    if(abs(pa)<abs(pb)) swap(pa, pb);
    if(abs(pb)<eps) return 0;
    double S, h, theta;
    double a=abs(pb), b=abs(pa), c=abs(pb-pa);
    double cosB = dot(pb, pb-pa) / a / c, B = acos(cosB);
    double cosC = dot(pa, pb) / a / b, C = acos(cosC);
    if(a > r){
        S = (C/2)*r*r;
        h = a*b*sin(C)/c;
        if (h < r && B < PI/2)
            S -= (acos(h/r)*r*r - h*sqrt(r*r-h*h));
    }
    else if(b > r){
        theta = PI - B - asin(sin(B)/r*a);
        S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
    }
    else S = .5*sin(C)*a*b;
    return S;
}
double area_poly_circle(const vector<pdd> poly,
    const pdd &o, const double r){
    double S=0;
    for(int i=0; i<SZ(poly); ++i)
        S += _area(poly[i]-o, poly[(i+1)%SZ(poly)]-o, r)
            *ori(0, poly[i], poly[(i+1)%SZ(poly)]);
    return fabs(S);
}
```

## 6.12 Intersection of Two Circle

```
bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
    pdd o1 = a.O, o2 = b.O;
    double r1 = a.R, r2 = b.R, d2 = abs2(o1 - o2),
        d = sqrt(d2);
    if(d < max(r1, r2) - min(r1, r2) || d > r1 + r2)
        return 0;
    pdd u = (o1 + o2) * 0.5
        + (o1 - o2) * ((r2 * r2 - r1 * r1) / (2 * d2));
    double A = sqrt((r1 + r2 + d) * (r1 - r2 + d)
        * (r1 + r2 - d) * (-r1 + r2 + d));
    pdd v = pdd(o1.Y - o2.Y, -o1.X + o2.X) * A
        / (2 * d2);
    p1 = u + v, p2 = u - v;
    return 1;
}
```

## 6.13 Tangent line of Two Circle

```
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;
}
```

## 6.14 Minimum Covering Circle

```
template<typename P>
Circle getCircum(const P &a, const P &b, const P &c){
    Real a1 = a.x-b.x, b1 = a.y-b.y;
    Real c1 = (a.x+b.x)/2 * a1 + (a.y+b.y)/2 * b1;
    Real a2 = a.x-c.x, b2 = a.y-c.y;
    Real c2 = (a.x+c.x)/2 * a2 + (a.y+c.y)/2 * b2;
    Circle cc;
    cc.o.x = (c1*b2-b1*c2)/(a1*b2-b1*a2);
    cc.o.y = (a1*c2-c1*a2)/(a1*b2-b1*a2);
    cc.r = hypot(cc.o.x-a.x, cc.o.y-a.y);
    return cc;
}
```

```
template<typename P>
Circle MinCircleCover(const vector<P>& pts){
    random_shuffle(pts.begin(), pts.end());
    Circle c = { pts[0], 0 };
    for(int i=0; i<(int)pts.size(); i++){
        if (dist(pts[i], c.o) <= c.r) continue;
        c = { pts[i], 0 };
        for (int j = 0; j < i; j++) {
            if (dist(pts[j], c.o) <= c.r) continue;
            c.o = (pts[i] + pts[j]) / 2;
            c.r = dist(pts[i], c.o);
            for (int k = 0; k < j; k++) {
                if (dist(pts[k], c.o) <= c.r) continue;
                c = getCircum(pts[i], pts[j], pts[k]);
            }
        }
    }
    return c;
}
```

## 6.15 KDTree (Nearest Point)

```
const int MXN = 100005;
struct KDTree {
    struct Node {
        int x, y, x1, y1, x2, y2;
        int id, f;
        Node *L, *R;
    } tree[MXN], *root;
    int n;
    LL dis2(int x1, int y1, int x2, int y2) {
        LL dx = x1-x2, 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<pair<int,int>> ip) {
        n = ip.size();
        for (int i=0; i<n; i++) {
            tree[i].id = i;
            tree[i].x = ip[i].first;
            tree[i].y = ip[i].second;
        }
        root = build_tree(0, n-1, 0);
    }
    Node* build_tree(int L, int R, int d) {
        if (L>R) return nullptr;
        int M = (L+R)/2; tree[M].f = d%2;
        nth_element(tree+L, tree+M, tree+R+1, d%2?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, d+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, d+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;

```

## 6.16 Rotating Sweep Line

```

void rotatingSweepLine(pair<int, int> a[], int n) {
    vector<pair<int, int>> l;
    l.reserve(n * (n - 1) / 2);
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            l.emplace_back(i, j);
    sort(l.begin(), l.end(), [&a](auto &u, auto &v){
        lld udx = a[u.first].first - a[u.second].first;
        lld udy = a[u.first].second - a[u.second].second;
        lld vdx = a[v.first].first - a[v.second].first;
        lld vdy = a[v.first].second - a[v.second].second;
        if (udx == 0 || vdx == 0) return not udx == 0;
        int s = sgn(udx * vdx);
        return udy * vdx * s < vdy * udx * s;
    });
    vector<int> idx(n), p(n);
    iota(idx.begin(), idx.end(), 0);
    sort(idx.begin(), idx.end(), [&a](int i, int j){
        return a[i] < a[j]; });
    for (int i = 0; i < n; ++i) p[idx[i]] = i;
    for (auto [i, j]: l) {
        // do here
        swap(p[i], p[j]);
        idx[p[i]] = i, idx[p[j]] = j;
    }
}

```

## 6.17 Circle Cover

```

const int N = 1021;
struct CircleCover {
    int C;
    Cir c[N];
    bool g[N][N], overlap[N][N];
    // Area[i] : area covered by at least i circles
    double Area[N];
    void init(int _C){ C = _C; }
    struct Teve {
        pdd p; double ang; int add;
        Teve() {}
        Teve(pdd _a, double _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 disjunct(Cir &a, Cir &b, int x)

```

```

{
    return sign(abs(a.O - b.O) - a.R - b.R) > x; }
bool contain(Cir &a, Cir &b, int x)
{
    return sign(a.R - b.R - abs(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(){
    fill_n(Area, C + 2, 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]) {
                pdd aa, bb;
                CCinter(c[i], c[j], aa, bb);
                double A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
                double 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] += cross(eve[j].p, eve[j + 1].p) * .5;
                double 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;
            }
        }
    }
}

```

## 7 Stringology

### 7.1 Hash

```

class Hash {
private:
    static constexpr int P = 127, Q = 1051762951;
    vector<int> h, p;
public:
    void init(const string &s){
        h.assign(s.size()+1, 0); p.resize(s.size()+1);
        for (size_t i = 0; i < s.size(); ++i)
            h[i + 1] = add(mul(h[i], P), s[i]);
        generate(p.begin(), p.end(), [x=1, y=1, this]()
            mutable{y=x;x=mul(x,P);return y;});
    }
    int query(int l, int r){ // 1-base (l, r)
        return sub(h[r], mul(h[l], p[r-l])); }
};

```

### 7.2 Suffix Array

```

namespace sfxarray {
bool t[maxn * 2];
int hi[maxn], rev[maxn];
int _s[maxn * 2], sa[maxn * 2], c[maxn * 2];
int x[maxn], p[maxn], q[maxn * 2];
// sa[i]: sa[i]-th suffix is the \
// i-th lexicographically smallest suffix.
// hi[i]: longest common prefix \
// of suffix sa[i] and suffix sa[i - 1].
void pre(int *sa, int *c, int n, int z) {

```

```

memset(sa, 0, sizeof(int) * n);
memcpy(x, c, sizeof(int) * z);
}
void induce(int *sa, int *c, int *s, bool *t, int n, int z){
    memcpy(x + 1, c, sizeof(int) * (z - 1));
    for (int i = 0; i < n; ++i)
        if (sa[i] && !t[sa[i] - 1])
            sa[x[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[sa[i] - 1]] = sa[i] - 1;
}
void sais(int *s, int *sa, int *p, int *q,
    bool *t, int *c, int n, int z) {
    bool uniq = t[n - 1] = true;
    int nn=0, nmzx=-1, *nsa = sa+n, *ns=s+n, last=-1;
    memset(c, 0, sizeof(int) * z);
    for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
    for (int i = 0; i < z - 1; ++i) c[i + 1] += c[i];
    if (uniq) {
        for (int i = 0; i < n; ++i) 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]);
    pre(sa, c, n, z);
    for (int i = 1; i <= n - 1; ++i)
        if (t[i] && !t[i - 1])
            sa[--x[s[i]]] = p[q[i] = nn++] = i;
    induce(sa, c, s, t, n, z);
    for (int i = 0; i < n; ++i) {
        if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
            bool neq = last < 0 || \
                memcmp(s + sa[i], s + last,
                    (p[q[sa[i]] + 1] - sa[i]) * sizeof(int));
            ns[q[last = sa[i]]] = nmzx += neq;
        }
    }
    sais(ns, nsa, p+nn, q+n, t+n, c+z, nn, nmzx+1);
    pre(sa, c, n, z);
    for (int i = nn - 1; i >= 0; --i)
        sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
    induce(sa, c, s, t, n, z);
}
void build(const string &s) {
    for (int i = 0; i < (int)s.size(); ++i) _s[i] = s[i];
    _s[(int)s.size()] = 0; // s shouldn't contain 0
    sais(_s, sa, p, q, t, c, (int)s.size() + 1, 256);
    for (int i = 0; i < (int)s.size(); ++i) sa[i]=sa[i+1];
    for (int i = 0; i < (int)s.size(); ++i) rev[sa[i]]=i;
    int ind = 0; hi[0] = 0;
    for (int i = 0; i < (int)s.size(); ++i) {
        if (!rev[i]) {
            ind = 0;
            continue;
        }
        while (i + ind < (int)s.size() && \
            s[i + ind] == s[sa[rev[i] - 1] + ind]) ++ind;
        hi[rev[i]] = ind ? ind-- : 0;
    }
}

```

### 7.3 Suffix Automaton

```

struct Node{
    Node *green, *edge[26];
    int max_len;
    Node(const int _max_len)
        : green(NULL), max_len(_max_len){
        memset(edge, 0, sizeof(edge));
    }
} *ROOT, *LAST;
void Extend(const int c) {
    Node *cursor = LAST;
    LAST = new Node((LAST->max_len) + 1);
    for (; cursor && cursor->edge[c]; cursor=cursor->green)
        cursor->edge[c] = LAST;
    if (!cursor)
        LAST->green = ROOT;
    else {
        Node *potential_green = cursor->edge[c];
        if ((potential_green->max_len)==(cursor->max_len+1))
            LAST->green = potential_green;
    }
}

```

```

else {
    //assert(potential_green->max_len>(cursor->max_len+1));
    Node *wish = new Node((cursor->max_len) + 1);
    for (; cursor && cursor->edge[c]==potential_green;
        cursor = cursor->green)
        cursor->edge[c] = wish;
    for (int i = 0; i < 26; i++)
        wish->edge[i] = potential_green->edge[i];
    wish->green = potential_green->green;
    potential_green->green = wish;
    LAST->green = wish;
}
}
char S[10000001], A[10000001];
int N;
int main(){
    scanf("%d%s", &N, S);
    ROOT = LAST = new Node(0);
    for (int i = 0; S[i]; i++)
        Extend(S[i] - 'a');
    while (N--){
        scanf("%s", A);
        Node *cursor = ROOT;
        bool ans = true;
        for (int i = 0; A[i]; i++){
            cursor = cursor->edge[A[i] - 'a'];
            if (!cursor) {
                ans = false;
                break;
            }
        }
        puts(ans ? "Yes" : "No");
    }
    return 0;
}

```

### 7.4 KMP

```

vector<int> kmp(const string &s) {
    vector<int> f(s.size(), 0);
    /* f[i] = length of the longest prefix
       (excluding s[0:i]) such that it coincides
       with the suffix of s[0:i] of the same length */
    /* i + 1 - f[i] is the length of the
       smallest recurring period of s[0:i] */
    int k = 0;
    for (int i = 1; i < (int)s.size(); ++i) {
        while (k > 0 && s[i] != s[k]) k = f[k - 1];
        if (s[i] == s[k]) ++k;
        f[i] = k;
    }
    return f;
}
vector<int> search(const string &s, const string &t) {
    // return 0-indexed occurrence of t in s
    vector<int> f = kmp(t), r;
    for (int i = 0, k = 0; i < (int)s.size(); ++i) {
        while (k > 0 && (k==(int)t.size() || s[i]!=t[k]))
            k = f[k - 1];
        if (s[i] == t[k]) ++k;
        if (k == (int)t.size()) r.push_back(i-t.size()+1);
    }
    return res;
}

```

### 7.5 Z value

```

char s[MAXN];
int len, z[MAXN];
void Z_value() {
    int i, j, left, right;
    z[left=right=0]=len;
    for (i=1; i<len; i++) {
        j=max(min(z[i-left], right-i), 0);
        for (; i+j<len && s[i+j]==s[j]; j++);
        if (i+(z[i]=j)>right) right=i+z[i];
    }
}

```

### 7.6 Manacher

```

int z[maxn];
int manacher(const string& s) {

```

```

string t = ".";
for(char c: s) t += c, t += '.';
int l = 0, r = 0, ans = 0;
for (int i = 1; i < t.length(); ++i) {
    z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
    while (i - z[i] >= 0 && i + z[i] < t.length()) {
        if(t[i - z[i]] == t[i + z[i]]) ++z[i];
        else break;
    }
    if (i + z[i] > r) r = i + z[i], l = i;
}
for(int i=1;i<t.length();++i) ans = max(ans, z[i]-1);
return ans;
}

```

## 7.7 Lexico 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 BWT

```

struct BurrowsWheeler{
#define SIGMA 26
#define BASE 'a'
    vector<int> v[ SIGMA ];
    void BWT(char* ori, char* res){
        // make ori -> ori + ori
        // then build suffix array
    }
    void iBWT(char* ori, char* res){
        for( int i = 0 ; i < SIGMA ; i ++ )
            v[ i ].clear();
        int len = strlen( ori );
        for( int i = 0 ; i < len ; i ++ )
            v[ ori[i] - BASE ].push_back( i );
        vector<int> a;
        for( int i = 0 , ptr = 0 ; i < SIGMA ; i ++ )
            for( auto j : v[ i ] ){
                a.push_back( j );
                ori[ ptr ++ ] = BASE + i;
            }
        for( int i = 0 , ptr = 0 ; i < len ; i ++ ){
            res[ i ] = ori[ a[ ptr ] ];
            ptr = a[ ptr ];
        }
        res[ len ] = 0;
    }
} bwt;

```

## 7.9 Palindromic Tree

```

struct palindromic_tree{
    struct node{
        int next[26], f, len;
        int cnt, num, st, ed;
        node(int l=0):f(0), len(1), cnt(0), num(0) {
            memset(next, 0, sizeof(next)); }
    };
    vector<node> st;
    vector<char> s;
    int last, n;
    void init(){
        st.clear(); s.clear(); last=1; n=0;
        st.push_back(0); st.push_back(-1);
        st[0].f=1; s.push_back(-1); }
    int getFail(int x){
        while(s[n-st[x].len-1]!=s[n])x=st[x].f;
        return x; }
    void add(int c){
        s.push_back(c-'a'); ++n;
        int cur=getFail(last);

```

```

        if(!st[cur].next[c]){
            int now=st.size();
            st.push_back(st[cur].len+2);
            st[now].f=st[getFail(st[cur].f)].next[c];
            st[cur].next[c]=now;
            st[now].num=st[st[now].f].num+1;
        }
        last=st[cur].next[c];
        ++st[last].cnt; }
    void dpcnt() {
        for (int i=st.size()-1; i >= 0; i--)
            st[st[i].f].cnt += st[i].cnt;
    }
    int size(){ return st.size()-2; }
} pt;
int main() {
    string s; cin >> s; pt.init();
    for (int i=0; i<SZ(s); i++) {
        int prvsz = pt.size(); pt.add(s[i]);
        if (prvsz != pt.size()) {
            int r = i, l = r - pt.st[pt.last].len + 1;
            // pal @ [l,r]: s.substr(l, r-l+1)
        }
    }
    return 0;
}

```

## 8 Misc

### 8.1 Theorems

#### 8.1.1 Kirchhoff's Theorem

Denote  $L$  be a  $n \times n$  matrix as the Laplacian matrix of graph  $G$ , where  $L_{ii} = d(i)$ ,  $L_{ij} = -c$  where  $c$  is the number of edge  $(i, j)$  in  $G$ .

- The number of undirected spanning in  $G$  is  $|\det(\tilde{L}_{11})|$ .
- The number of directed spanning tree rooted at  $r$  in  $G$  is  $|\det(\tilde{L}_{rr})|$ .

#### 8.1.2 Tutte's Matrix

Let  $D$  be a  $n \times n$  matrix, where  $d_{ij} = x_{ij}$  ( $x_{ij}$  is chosen uniform randomly) if  $i < j$  and  $(i, j) \in E$ , otherwise  $d_{ij} = -d_{ji}$ .  $\frac{\text{rank}(D)}{2}$  is the maximum matching on  $G$ .

#### 8.1.3 Cayley's Formula

- Given a degree sequence  $d_1, d_2, \dots, d_n$  for each labeled vertices, there're  $\frac{(n-2)!}{(d_1-1)!(d_2-1)! \dots (d_n-1)!}$  spanning trees.
- Let  $T_{n,k}$  be the number of labeled forests on  $n$  vertices with  $k$  components, such that vertex  $1, 2, \dots, k$  belong to different components. Then  $T_{n,k} = k n^{n-k-1}$ .

#### 8.1.4 Erdős–Gallai theorem

A sequence of non-negative integers  $d_1 \geq d_2 \geq \dots \geq d_n$  can be represented as the degree sequence of a finite simple graph on  $n$  vertices if and only if  $d_1 + d_2 + \dots + d_n$  is even and

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

holds for all  $1 \leq k \leq n$ .

#### 8.1.5 Havel–Hakimi algorithm

find the vertex who has greatest degree unused, connect it with other greatest vertex.

#### 8.1.6 Hall's marriage theorem

Let  $G$  be a finite bipartite graph with bipartite sets  $X$  and  $Y$ . For a subset  $W$  of  $X$ , let  $N_G(W)$  denote the set of all vertices in  $Y$  adjacent to some element of  $W$ . Then there is an  $X$ -saturating matching iff  $\forall W \subseteq X, |W| \leq |N_G(W)|$

#### 8.1.7 Euler's planar graph formula

$$V - E + F = C + 1, E \leq 3V - 6(?)$$

#### 8.1.8 Pick's theorem

For simple polygon, when points are all integer, we have  $A = \#\{\text{lattice points in the interior}\} + \frac{\#\{\text{lattice points on the boundary}\}}{2} - 1$

#### 8.1.9 Lucas's theorem

$\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p}$ , where  $m = m_k p^k + m_{k-1} p^{k-1} + \dots + m_1 p + m_0$ , and  $n = n_k p^k + n_{k-1} p^{k-1} + \dots + n_1 p + n_0$ .



### 8.1.10 Matroid Intersection

Given matroids  $M_1 = (G, I_1), M_2 = (G, I_2)$ , find maximum  $S \in I_1 \cap I_2$ . For each iteration, build the directed graph and find a shortest path from  $s$  to  $t$ .

- $s \rightarrow x : S \sqcup \{x\} \in I_1$
- $x \rightarrow t : S \sqcup \{x\} \in I_2$
- $y \rightarrow x : S \setminus \{y\} \sqcup \{x\} \in I_1$  ( $y$  is in the unique circuit of  $S \sqcup \{x\}$ )
- $x \rightarrow y : S \setminus \{y\} \sqcup \{x\} \in I_2$  ( $y$  is in the unique circuit of  $S \sqcup \{x\}$ )

Alternate the path, and  $|S|$  will increase by 1. Let  $R = \min(\text{rank}(I_1), \text{rank}(I_2)), N = |G|$ . In each iteration,  $|E| = O(RN)$ . For weighted case, assign weight  $-w(x)$  and  $w(x)$  to  $x \in S$  and  $x \notin S$ , resp. Use Bellman-Ford to find the weighted shortest path. The maximum iteration of Bellman-Ford is  $2R + 1$ .

## 8.2 DP-opt Condition

### 8.2.1 totally monotone (concave/convex)

$$\begin{aligned} \forall i < i', j < j', B[i][j] \leq B[i'][j'] &\implies B[i][j'] \leq B[i'][j'] \\ \forall i < i', j < j', B[i][j] \geq B[i'][j'] &\implies B[i][j'] \geq B[i'][j'] \end{aligned}$$

### 8.2.2 monge condition (concave/convex)

$$\begin{aligned} \forall i < i', j < j', B[i][j] + B[i'][j'] &\geq B[i][j'] + B[i'][j] \\ \forall i < i', j < j', B[i][j] + B[i'][j'] &\leq B[i][j'] + B[i'][j] \end{aligned}$$

## 8.3 Convex 1D/1D DP

```
struct segment {
    int i, l, r;
    segment() {}
    segment(int a, int b, int c): i(a), l(b), r(c) {}
};
inline lld f(int l, int r){return dp[l] + w(l+1, r);}
void solve() {
    dp[0] = 0;
    deque<segment> dq; dq.push_back(segment(0, 1, n));
    for (int i = 1; i <= n; ++i) {
        dp[i] = f(dq.front().i, i);
        while(dq.size() && dq.front().r < i+1) dq.pop_front();
        dq.front().l = i + 1;
        segment seg = segment(i, i + 1, n);
        while (dq.size() &&
            f(i, dq.back().l) < f(dq.back().i, dq.back().l))
            dq.pop_back();
        if (dq.size()) {
            int d = 1 << 20, c = dq.back().l;
            while (d >= 1) if (c + d <= dq.back().r)
                if (f(i, c+d) > f(dq.back().i, c+d)) c += d;
            dq.back().r = c; seg.l = c + 1;
        }
        if (seg.l <= n) dq.push_back(seg);
    }
}
```

## 8.4 ConvexHull Optimization

```
struct Line {
    mutable int64_t a, b, p;
    bool operator<(const Line &rhs) const { return a < rhs.a; }
    bool operator<(int64_t x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
    static const int64_t kInf = 1e18;
    bool Isect(iterator x, iterator y) {
        auto Div = [](int64_t a, int64_t b) {
            return a / b - ((a ^ b) < 0 && a % b);
        };
        if (y == end()) { x->p = kInf; return false; }
        if (x->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
        else x->p = Div(y->b - x->b, x->a - y->a);
        return x->p >= y->p;
    }
    void Insert(int64_t a, int64_t b) {
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (Isect(y, z)) z = erase(z);
        if (x != begin() && Isect(--x, y)) Isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p) Isect(x, erase(y));
    }
    int64_t Query(int64_t x) {
        auto l = *lower_bound(x);
        return l.a * x + l.b;
    }
};
```

## 8.5 Josephus Problem

```
// n people kill m for each turn
int f(int n, int m) {
    int s = 0;
    for (int i = 2; i <= n; i++)
        s = (s + m) % i;
    return s;
}
// died at kth
int kth(int n, int m, int k){
    if (m == 1) return n-1;
    for (k = k*m+m-1; k >= n; k = k-n+(k-n)/(m-1));
    return k;
}
```

## 8.6 Cactus Matching

```
vector<int> init_g[maxn], g[maxn*2];
int n, dfn[maxn], low[maxn], par[maxn], dfs_idx, bcc_id;
void tarjan(int u){
    dfn[u]=low[u]=++dfs_idx;
    for(int i=0; i<(int)init_g[u].size(); i++){
        int v=init_g[u][i];
        if(v==par[u]) continue;
        if(!dfn[v]){
            par[v]=u;
            tarjan(v);
            low[u]=min(low[u], low[v]);
            if(dfn[u]<low[v]){
                g[u].push_back(v);
                g[v].push_back(u);
            }
        }else{
            low[u]=min(low[u], dfn[v]);
            if(dfn[v]<dfn[u]){
                int temp_v=u;
                bcc_id++;
                while(temp_v!=v){
                    g[bcc_id+n].push_back(temp_v);
                    g[temp_v].push_back(bcc_id+n);
                    temp_v=par[temp_v];
                }
                g[bcc_id+n].push_back(v);
                g[v].push_back(bcc_id+n);
                reverse(g[bcc_id+n].begin(), g[bcc_id+n].end());
            }
        }
    }
}
int dp[maxn][2], min_dp[2][2], tmp[2][2], tp[2];
void dfs(int u, int fa){
    if(u==n){
        for(int i=0; i<(int)g[u].size(); i++){
            int v=g[u][i];
            if(v==fa) continue;
            dfs(v, u);
            memset(tp, 0x8f, sizeof tp);
            if(v==n){
                tp[0]=dp[u][0]+max(dp[v][0], dp[v][1]);
                tp[1]=max(
                    dp[u][0]+dp[v][0]+1,
                    dp[u][1]+max(dp[v][0], dp[v][1])
                );
            }else{
                tp[0]=dp[u][0]+dp[v][0];
                tp[1]=max(dp[u][0]+dp[v][1], dp[u][1]+dp[v][0]);
            }
            dp[u][0]=tp[0], dp[u][1]=tp[1];
        }
    }else{
        for(int i=0; i<(int)g[u].size(); i++){
            int v=g[u][i];
            if(v==fa) continue;
            dfs(v, u);
            min_dp[0][0]=0;
            min_dp[1][1]=1;
            min_dp[0][1]=min_dp[1][0]=-0x3f3f3f3f;
            for(int i=0; i<(int)g[u].size(); i++){
                int v=g[u][i];
                if(v==fa) continue;
                memset(tmp, 0x8f, sizeof tmp);
```

```

    tmp[0][0]=max(
        min_dp[0][0]+max(dp[v][0],dp[v][1]),
        min_dp[0][1]+dp[v][0]
    );
    tmp[0][1]=min_dp[0][0]+dp[v][0]+1;
    tmp[1][0]=max(
        min_dp[1][0]+max(dp[v][0],dp[v][1]),
        min_dp[1][1]+dp[v][0]
    );
    tmp[1][1]=min_dp[1][0]+dp[v][0]+1;
    memcpy(min_dp,tmp,sizeof tmp);
}
dp[u][1]=max(min_dp[0][1],min_dp[1][0]);
dp[u][0]=min_dp[0][0];
}
}
int main(){
    int m,a,b;
    scanf("%d%d",&n,&m);
    for(int i=0;i<m;i++){
        scanf("%d",&a,&b);
        init_g[a].push_back(b);
        init_g[b].push_back(a);
    }
    par[1]=-1;
    tarjan(1);
    dfs(1,-1);
    printf("%d\n",max(dp[1][0],dp[1][1]));
    return 0;
}

```

## 8.7 DLX

```

struct DLX {
    const static int maxn=210;
    const static int maxm=210;
    const static int maxnode=210*210;
    int n, m, size, row[maxnode], col[maxnode];
    int U[maxnode], D[maxnode], L[maxnode], R[maxnode];
    int H[maxn], S[maxm], ansd, ans[maxn];
    void init(int _n, int _m) {
        n = _n, m = _m;
        for(int i = 0; i <= m; ++i) {
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1, R[i] = i+1;
        }
        R[L[0] = size = m] = 0;
        for(int i = 1; i <= n; ++i) H[i] = -1;
    }
    void Link(int r, int c) {
        ++S[col[++size] = c];
        row[size] = r; D[size] = D[c];
        U[D[c]] = size; U[size] = c; D[c] = size;
        if(H[r] < 0) H[r] = L[size] = R[size] = size;
        else {
            R[size] = R[H[r]];
            L[R[H[r]]] = size;
            L[size] = H[r];
            R[H[r]] = size;
        }
    }
    void remove(int c) {
        L[R[c]] = L[c]; R[L[c]] = R[c];
        for(int i = D[c]; i != c; i = D[i])
            for(int j = R[i]; j != i; j = R[j]) {
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                --S[col[j]];
            }
    }
    void resume(int c) {
        L[R[c]] = c; R[L[c]] = c;
        for(int i = U[c]; i != c; i = U[i])
            for(int j = L[i]; j != i; j = L[j]) {
                U[D[j]] = j;
                D[U[j]] = j;
                ++S[col[j]];
            }
    }
    void dance(int d) {
        if(d>ansd) return;
        if(R[0] == 0){

```

```

            ansd = d;
            return;
        }
        int c = R[0];
        for(int i = R[0]; i; i = R[i])
            if(S[i] < S[c]) c = i;
        remove(c);
        for(int i = D[c]; i != c; i = D[i]) {
            ans[d] = row[i];
            for(int j = R[i]; j != i; j = R[j])
                remove(col[j]);
            dance(d+1);
            for(int j = L[i]; j != i; j = L[j])
                resume(col[j]);
        }
        resume(c);
    }
} sol;

```

## 8.8 Tree Knapsack

```

int dp[N][K];PII obj[N];
vector<int> G[N];
void dfs(int u, int mx){
    for(int s: G[u]) {
        if(mx < obj[s].first) continue;
        for(int i=0;i<=mx-obj[s].FF;i++){
            dp[s][i] = dp[u][i];
            dfs(s, mx - obj[s].first);
            for(int i=obj[s].FF;i<=mx;i++){
                dp[u][i] = max(dp[u][i],
                    dp[s][i - obj[s].FF] + obj[s].SS);
            }
        }
    }
}
int main(){
    int n, k; cin >> n >> k;
    for(int i=1;i<=n;i++){
        int p; cin >> p;
        G[p].push_back(i);
        cin >> obj[i].FF >> obj[i].SS;
    }
    dfs(0, k); int ans = 0;
    for(int i=0;i<=k;i++) ans = max(ans, dp[0][i]);
    cout << ans << '\n';
    return 0;
}

```

## 8.9 N Queens Problem

```

vector<int> solve( int n ) {
    // no solution when n=2, 3
    vector<int> ret;
    if ( n % 6 == 2 ) {
        for ( int i = 2 ; i <= n ; i += 2 )
            ret.push_back( i );
        ret.push_back( 3 ); ret.push_back( 1 );
        for ( int i = 7 ; i <= n ; i += 2 )
            ret.push_back( i );
        ret.push_back( 5 );
    } else if ( n % 6 == 3 ) {
        for ( int i = 4 ; i <= n ; i += 2 )
            ret.push_back( i );
        ret.push_back( 2 );
        for ( int i = 5 ; i <= n ; i += 2 )
            ret.push_back( i );
        ret.push_back( 1 ); ret.push_back( 3 );
    } else {
        for ( int i = 2 ; i <= n ; i += 2 )
            ret.push_back( i );
        for ( int i = 1 ; i <= n ; i += 2 )
            ret.push_back( i );
    }
    return ret;
}

```

## 8.10 Aliens Optimization

```

long long Alien() {
    long long c = kInf;
    for (int d = 60; d >= 0; --d) {
        // cost can be negative, depending on the problem.
        if (c - (1LL << d) < 0) continue;
        long long ck = c - (1LL << d);
        pair<long long, int> r = check(ck);
    }
}

```

```

    if (r.second == k) return r.first - ck * k;
    if (r.second < k) c = ck;
}
pair<long long, int> r = check(c);
return r.first - c * k;
}

```

### 8.11 Hilbert Curve

```

long long hilbert(int n, int x, int y) {
    long long res = 0;
    for (int s = n / 2; s; s >>= 1) {
        int rx = (x & s) > 0, ry = (y & s) > 0;
        res += s * 111 * s * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) x = s - 1 - x, y = s - 1 - y;
            swap(x, y);
        }
    }
    return res;
}

```

### 8.12 Binary Search On Fraction

```

struct Q {
    ll p, q;
    Q go(Q b, ll d) { return {p + b.p*d, q + b.q*d}; }
};
bool pred(Q);
// returns smallest p/q in [lo, hi] such that
// pred(p/q) is true, and 0 <= p,q <= N
Q frac_bs(ll N) {
    Q lo{0, 1}, hi{1, 0};
    if (pred(lo)) return lo;
    assert(pred(hi));
    bool dir = 1, L = 1, H = 1;
    for (; L || H; dir = !dir) {
        ll len = 0, step = 1;
        for (int t = 0; t < 2 && (t ? step/=2 : step*=2);)
            if (Q mid = hi.go(lo, len + step);
                mid.p > N || mid.q > N || dir ^ pred(mid))
                t++;
            else len += step;
        swap(lo, hi = hi.go(lo, len));
        (dir ? L : H) = !len;
    }
    return dir ? hi : lo;
}

```