

Contents

| | | | |
|---|-----------|--|-----------|
| 1 Basic | 1 | 8 Misc | 23 |
| 1.1 vimrc | 1 | 8.1 Theorems | 23 |
| 1.2 IncreaseStackSize | 1 | 8.1.1 Kirchhoff's Theorem | 23 |
| 1.3 Pragma optimization | 1 | 8.1.2 Tutte's Matrix | 23 |
| 1.4 Debugger | 1 | 8.1.3 Cayley's Formula | 23 |
| 1.5 Quick Random | 2 | 8.1.4 Erdős-Gallai theorem | 23 |
| 1.6 IO Optimization | 2 | 8.1.5 Havel-Hakimi algorithm | 23 |
| 2 Data Structure | 2 | 8.1.6 Hall's marriage theorem | 23 |
| 2.1 Bigint | 2 | 8.1.7 Euler's planar graph formula | 23 |
| 2.2 Dark Magic | 3 | 8.1.8 Pick's theorem | 23 |
| 2.3 SkewHeap | 3 | 8.2 MaximumEmptyRect | 23 |
| 2.4 Disjoint Set | 3 | 8.3 DP-opt Condition | 23 |
| 2.5 Link-Cut Tree | 4 | 8.3.1 totally monotone (concave/convex) | 23 |
| 2.6 LiChao Segment Tree | 4 | 8.3.2 monge condition (concave/convex) | 23 |
| 2.7 Treap | 5 | 8.4 Convex 1D/1D DP | 23 |
| 2.8 SparseTable | 5 | 8.5 ConvexHull Optimization | 23 |
| 2.9 Linear Basis | 5 | 8.6 Josephus Problem | 24 |
| 3 Graph | 5 | 8.7 Cactus Matching | 24 |
| 3.1 Euler Circuit | 5 | 8.8 DLX | 24 |
| 3.2 BCC Edge | 6 | 8.9 Tree Knapsack | 25 |
| 3.3 BCC Vertex | 6 | 8.10N Queens Problem | 25 |
| 3.4 2-SAT (SCC) | 6 | 1 Basic | |
| 3.5 Lowbit Decomposition | 7 | 1.1 vimrc | |
| 3.6 MaxClique | 7 | se is nu rnu bs=2 ru mouse=a encoding=utf-8 | |
| 3.7 Virtual Tree | 8 | se cin et ts=4 sw=4 sts=4 t_Co=256 | |
| 3.8 Tree Hashing | 8 | syn on | |
| 3.9 Minimum Mean Cycle | 8 | colorscheme ron | |
| 3.10Mo's Algorithm on Tree | 8 | filetype indent on | |
| 3.11Minimum Steiner Tree | 9 | 1.2 IncreaseStackSize | |
| 3.12Directed Minimum Spanning Tree | 9 | //stack resize(change esp to rsp if 64-bit system) | |
| 4 Matching & Flow | 9 | asm("mov %0,%esp\n" :: "g"(mem+10000000)); | |
| 4.1 Kuhn Munkres | 9 | // craziest way | |
| 4.2 Bipartite Matching | 10 | static void run_stack_sz(void(*func>(),size_t stsize){ | |
| 4.3 General Graph Matching | 10 | char *stack, *send; | |
| 4.4 Minimum Weight Matching (Clique version) | 10 | stack=(char *)malloc(stsize); | |
| 4.5 Flow Models | 11 | send=stack+stsize-16; | |
| 4.6 Dinic | 11 | send=(char *)((uintptr_t)send/16*16); | |
| 4.7 Minimum Cost Maximum Flow | 12 | asm volatile(| |
| 4.8 Global Min-Cut | 12 | "mov %%rsp, (%0)\n" | |
| 5 Math | 12 | "mov %0, %%rsp\n" | |
| 5.1 Prime Table | 12 | : "r" (send)); | |
| 5.2 $\lfloor \frac{n}{i} \rfloor$ Enumeration | 12 | func(); | |
| 5.3 ax+by=gcd | 13 | asm volatile(| |
| 5.4 Pollard Rho | 13 | "mov (%0), %%rsp\n" | |
| 5.5 Pi Count (Linear Sieve) | 13 | : "r" (send)); | |
| 5.6 Range Sieve | 13 | free(stack); | |
| 5.7 Miller Rabin | 13 | } | |
| 5.8 Inverse Element | 13 | 1.3 Pragma optimization | |
| 5.9 Euler Phi Function | 13 | #pragma GCC optimize("Ofast,no-stack-protector") | |
| 5.10Gauss Elimination | 14 | #pragma GCC optimize("no-math-errno,unroll-loops") | |
| 5.11Fast Fourier Transform | 14 | #pragma GCC target("sse,sse2,sse3,ssse3,sse4") | |
| 5.12High Speed Linear Recurrence | 14 | #pragma GCC target("popcnt,abm,mmx,avx,tune=native") | |
| 5.13Chinese Remainder | 15 | 1.4 Debugger | |
| 5.14Berlekamp Massey | 15 | #!/usr/bin/env python3 | |
| 5.15NTT | 15 | import subprocess as sp | |
| 5.16Polynomial Sqrt | 15 | os_name = __import__('platform').system() | |
| 5.17Polynomial Division | 16 | cmd,prefix = [],"" | |
| 5.18FWT | 16 | if os_name == 'Windows': | |
| 5.19Discretelog | 16 | cmd=["cmd", "/C"] | |
| 5.20Quadratic residue | 16 | else: | |
| 5.21De-Bruijn | 17 | cmd = ["bash", "-c"] | |
| 5.22Simplex Construction | 17 | prefix = "./" | |
| 5.23Simplex | 17 | def GetTestData(exe): | |
| 6 Geometry | 18 | myout=sp.check_output(cmd+["%s%s"%(prefix, exe)]) | |
| 6.1 Point Class | 18 | return myout.decode("utf8") | |
| 6.2 Circle Class | 18 | def Judge(a,b,testdata): | |
| 6.3 Triangle Circumcentre | 18 | f = open("test.in", "w+") | |
| 6.4 2D Convex Hull | 18 | f.write(testdata) | |
| 6.5 2D Farthest Pair | 19 | f.close() | |
| 6.6 2D Closest Pair | 19 | c=sp.check_output(cmd+["%s%s<test.in"%(prefix, a)]) | |
| 6.7 SimulateAnnealing | 19 | d=sp.check_output(cmd+["%s%s<test.in"%(prefix, b)]) | |
| 6.8 Half Plane Intersection | 19 | if not c == d: | |
| 6.9 Ternary Search on Integer | 19 | | |
| 6.10Minimum Covering Circle | 20 | | |
| 6.11KDTree (Nearest Point) | 20 | | |
| 7 Stringology | 20 | | |
| 7.1 Hash | 20 | | |
| 7.2 Suffix Array | 20 | | |
| 7.3 Aho-Corasick Algorithm | 21 | | |
| 7.4 Suffix Automaton | 21 | | |
| 7.5 KMP | 22 | | |
| 7.6 Z value | 22 | | |
| 7.7 Manacher | 22 | | |
| 7.8 Lexicographically Smallest Rotation | 22 | | |
| 7.9 BWI | 22 | | |
| 7.10Palindromic Tree | 22 | | |

```

    print("answer: %s"%c.decode("utf8"),end="")
    print("output: %s"%d.decode("utf8"),end="")
    print("WA!")
    return False
return True
if __name__ == '__main__':
    cnt = 0
    isOK = True
    while isOK:
        cnt += 1
        print(cnt)
        isOK=Judge("sol", "mysol", GetTestData("gen"))

```

1.5 Quick Random

```

template<class T,T x1,T x2,T x3,int y1,int y2,int y3>
struct PRNG {
    using S = typename std::make_signed<T>::type;
    T s;
    PRNG(T _s = 0) : s(_s) {}
    T next() {
        T z = (s += x1);
        z = (z ^ (z >> y1)) * x2;
        z = (z ^ (z >> y2)) * x3;
        return z ^ (z >> y3);
    }
    T next(T n) { return next() % n; }
    S next(S l, S r){return l+next(r-l+1);}
    T operator()() { return next(); }
    T operator()(T n) { return next(n); }
    S operator()(S l, S r) { return next(l, r); }
    static T gen(T s) { return PRNG(s)(); }
    template<class U>
    void shuffle(U first,U last){
        size_t n=last-first;
        for(size_t i=0;i<n;i++)
            swap(first[i],first[next(i+1)]);
    }
};
using R32=PRNG<uint32_t,0x9E3779B1,0x85EBCA6B,
0xC2B2AE35,16,13,16>;
R32 r32;
using R64=PRNG<uint64_t,0x9E3779B97F4A7C15,
0xBF58476D1CE4E5B9,0x94D049BB133111EB,30,27,31>;
R64 r64;

```

1.6 IO Optimization

```

static inline int gc() {
    static char buf[ 1 << 20 ], *p = buf, *end = buf;
    if ( p == end ) {
        end = buf + fread( buf, 1, 1 << 20, stdin );
        if ( end == buf ) return EOF;
        p = buf;
    }
    return *p++;
}
template < typename T >
static inline bool gn( T &_ ) {
    register int c = gc(); register T __ = 1; _ = 0;
    while(( '0'<c||c>'9' ) && c!=EOF && c!='-') c = gc();
    if(c == '-') { __ = -1; c = gc(); }
    if(c == EOF) return false;
    while( '0'<=c&&c<='9' ) _ = _ * 10 + c - '0', c = gc();
    _ *= __;
    return true;
}
template < typename T, typename ...Args >
static inline bool gn( T &x, Args &...args )
{ return gn(x) && gn(args...); }

```

2 Data Structure

2.1 Bigint

```

class BigInt{
private:
    using lld = int_fast64_t;
    #define PRINTF_ARG PRIdFAST64
    #define LOG_BASE_STR "9"

```

```

static constexpr lld BASE = 1000000000;
static constexpr int LOG_BASE = 9;
vector<lld> dig;
bool neg;
inline int len() const { return (int) dig.size(); }
inline int cmp_minus(const BigInt& a) const {
    if(len() == 0 && a.len() == 0) return 0;
    if(neg ^ a.neg)return (int)a.neg*2 - 1;
    if(len()!=a.len())
        return neg?a.len()-len():len()-a.len();
    for(int i=len()-1;i>=0;i--) if(dig[i]!=a.dig[i])
        return neg?a.dig[i]-dig[i]:dig[i]-a.dig[i];
    return 0;
}
inline void trim(){
    while(!dig.empty()&&!dig.back())dig.pop_back();
    if(dig.empty()) neg = false;
}
public:
    BigInt(): dig(vector<lld>()), neg(false){}
    BigInt(lld a): dig(vector<lld>()){
        neg = a<0; dig.push_back(abs(a));
        trim();
    }
    BigInt(const string& a): dig(vector<lld>()){
        assert(!a.empty()); neg = (a[0]=='-');
        for(int i=((int)a.size()-1;i>=neg;i-=LOG_BASE){
            lld cur = 0;
            for(int j=min(LOG_BASE-1,i-neg);j>=0;j--){
                cur = cur*10+a[i-j]-'0';
                dig.push_back(cur);
            } trim();
        }
    }
    inline bool operator<(const BigInt& a)const {
        return cmp_minus(a)<0;
    }
    inline bool operator<=(const BigInt& a)const {
        return cmp_minus(a)<=0;
    }
    inline bool operator==(const BigInt& a)const {
        return cmp_minus(a)==0;
    }
    inline bool operator!=(const BigInt& a)const {
        return cmp_minus(a)!=0;
    }
    inline bool operator>(const BigInt& a)const {
        return cmp_minus(a)>0;
    }
    inline bool operator>=(const BigInt& a)const {
        return cmp_minus(a)>=0;
    }
    BigInt operator-() const {
        BigInt ret = *this;
        ret.neg ^= 1;
        return ret;
    }
    BigInt operator+(const BigInt& a) const {
        if(neg) return -(-( *this )+(-a));
        if(a.neg) return ( *this )-(-a);
        int n = max(a.len(), len());
        BigInt ret; ret.dig.resize(n);
        lld pro = 0;
        for(int i=0;i<n;i++) {
            ret.dig[i] = pro;
            if(i < a.len()) ret.dig[i] += a.dig[i];
            if(i < len()) ret.dig[i] += dig[i];
            pro = 0;
            if(ret.dig[i] >= BASE) pro = ret.dig[i]/BASE;
            ret.dig[i] -= BASE*pro;
        }
        if(pro != 0) ret.dig.push_back(pro);
        return ret;
    }
    BigInt operator-(const BigInt& a) const {
        if(neg) return -(-( *this ) - (-a));
        if(a.neg) return ( *this ) + (-a);
        int diff = cmp_minus(a);
        if(diff < 0) return -(a - ( *this ));
        if(diff == 0) return 0;
        BigInt ret; ret.dig.resize(len(), 0);
        for(int i=0;i<len();i++) {
            ret.dig[i] += dig[i];
            if(i < a.len()) ret.dig[i] -= a.dig[i];
            if(ret.dig[i] < 0){
                ret.dig[i] += BASE;
                ret.dig[i+1]--;
            }
        }
    }
}

```

```

    ret.trim();
    return ret;
}
BigInt operator*(const BigInt& a) const {
    if(!len()||!a.len()) return 0;
    BigInt ret; ret.dig.resize(len()+a.len()+1);
    ret.neg = neg ^ a.neg;
    for(int i=0;i<len();i++){
        for(int j=0;j<a.len();j++){
            ret.dig[i+j] += dig[i] * a.dig[j];
            if(ret.dig[i+j] >= BASE) {
                lld x = ret.dig[i+j] / BASE;
                ret.dig[i+j+1] += x;
                ret.dig[i+j] -= x * BASE;
            }
        }
    }
    ret.trim();
    return ret;
}
BigInt operator/(const BigInt& a) const {
    assert(a.len());
    if(len() < a.len()) return 0;
    BigInt ret; ret.dig.resize(len()-a.len()+1);
    ret.neg = a.neg;
    for(int i=len()-a.len();i>=0;i--){
        lld l = 0, r = BASE;
        while(r-l > 1){
            lld mid = (l+r)>>1;
            ret.dig[i] = mid;
            if(ret*a<=(neg?-(this):(this))) l = mid;
            else r = mid;
        }
        ret.dig[i] = l;
    }
    ret.neg ^= neg; ret.trim();
    return ret;
}
BigInt operator%(const BigInt& a) const {
    return (this) - (this) / a * a;
}
friend BigInt abs(BigInt a){
    a.neg = 1; return a;
}
friend void swap(BigInt& a, BigInt& b){
    swap(a.dig, b.dig); swap(a.neg, b.neg);
}
friend istream& operator>>(istream& ss, BigInt& a){
    string s; ss >> s; a = s;
    return ss;
}
friend ostream& operator<<(ostream& o, const BigInt& a){
    if(a.len() == 0) return o << '0';
    if(a.neg) o << '-';
    ss << o.dig.back();
    for(int i=a.len()-2;i>=0;i--){
        o<<setw(LOG_BASE)<<setfill('0')<<a.dig[i];
        return o;
    }
}
inline void print() const {
    if(len() == 0){putchar('0');return;}
    if(neg) putchar('-');
    printf("%" PRINTF_ARG, dig.back());
    for(int i=len()-2;i>=0;i--){
        printf("%0" LOG_BASE_STR PRINTF_ARG, dig[i]);
    }
}
#undef PRINTF_ARG
#undef LOG_BASE_STR
};

```

2.2 Dark Magic

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using __gnu_pbds::pairing_heap_tag;
using __gnu_pbds::binary_heap_tag;
using __gnu_pbds::binomial_heap_tag;
using __gnu_pbds::rc_binomial_heap_tag;
using __gnu_pbds::thin_heap_tag;
template<typename T>
using pbds_heap=__gnu_pbds::prioity_queue<T,less<T>,\
    pairing_heap_tag>;

```

```

using __gnu_pbds::rb_tree_tag;
using __gnu_pbds::ov_tree_tag;
using __gnu_pbds::splay_tree_tag;
template<typename T>
using ordered_set = __gnu_pbds::tree<T,\
    __gnu_pbds::null_type,less<T>,rb_tree_tag,\
    __gnu_pbds::tree_order_statistics_node_update>;
template<typename A,typename B>
using hTable1=__gnu_pbds::cc_hash_table<A,B>;
template<typename A,typename B>
using hTable2=__gnu_pbds::gp_hash_table<A,B>;
int main(){
    ordered_set<int> ss;
    ss.insert(1); ss.insert(5);
    assert(*ss.find_by_order(0)==1);
    assert(ss.order_of_key(-1)==0);
    pbds_heap pq1, pq2;
    pq1.push(1); pq2.push(2);
    pq1.join(pq2);
    assert(pq2.size()==0);
    auto it = pq1.push(87);
    pq1.modify(it, 19);
    return 0;
}

```

2.3 SkewHeap

```

template < typename T, typename cmp = less< T > >
class SkewHeap{
private:
    struct SkewNode{
        T x;
        SkewNode *lc, *rc;
        SkewNode( T a = 0 ) : x( a ), lc( 0 ), rc( 0 ) {}
    } *root;
    cmp CMP_;
    size_t count;
    SkewNode* Merge( SkewNode* a, SkewNode* b ) {
        if ( !a or !b ) return a ? a : b;
        if ( CMP_( a->x, b->x ) ) swap( a, b );
        a -> rc = Merge( a->rc, b );
        swap( a -> lc, a->rc );
        return a;
    }
public:
    SkewHeap(): root( 0 ), count( 0 ) {}
    size_t size() { return count; }
    bool empty() { return count == 0; }
    T top() { return root->x; }
    void clear(){ root = 0; count = 0; }
    void push ( const T& x ) {
        SkewNode* a = new SkewNode( x );
        count += 1; root = Merge( root, a );
    }
    void join( SkewHeap& a ) {
        count += a.count; a.count = 0;
        root = Merge( root, a.root );
    }
    void pop() {
        count--; root = Merge( root->lc, root->rc );
    }
    friend void swap( SkewHeap& a, SkewHeap& b ) {
        swap( a.root, b.root ); swap( a.count, b.count );
    }
};

```

2.4 Disjoint Set

```

class DJS{
private:
    vector< int > fa, sz, sv;
    vector< pair< int*, int > > opt;
    inline void assign( int *k, int v ) {
        opt.emplace_back( k, *k );
        *k = v;
    }
public:
    inline void init( int n ) {
        fa.resize( n ); iota( fa.begin(), fa.end(), 0 );
        sz.resize( n ); fill( sz.begin(), sz.end(), 1 );
        opt.clear();
    }
};

```

```

}
int query( int x ) {
    return ( fa[ x ] == x ) ? x : query( fa[ x ] );
}
inline void merge( int a, int b ) {
    int af = query( a ), bf = query( b );
    if( af == bf ) return;
    if( sz[ af ] < sz[ bf ] ) swap( af, bf );
    assign( &fa[ bf ], fa[ af ] );
    assign( &sz[ af ], sz[ af ] + sz[ bf ] );
}
inline void save() {sv.push_back( (int)opt.size() );}
inline void undo() {
    int ls = sv.back(); sv.pop_back();
    while ( ( int ) opt.size() > ls ) {
        pair< int*, int > cur = opt.back();
        *cur.first = cur.second;
        opt.pop_back();
    }
}
};

```

2.5 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]);
}
}

```

2.6 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;

```

2.7 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
}

```

2.8 SparseTable

```

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 ] );
    }
};

```

2.9 Linear Basis

```

struct LinearBasis {
private:
    int n, sz;
    vector< ll_u > B;
    inline ll_u two( int x ){ return ( ( ll_u ) 1 ) << x; }
public:
    void init( int n_ ) {
        n = n_; B.clear(); B.resize( n ); sz = 0;
    }
    void insert( ll_u x ) {
        // add x into B
        for ( int i = n-1; i >= 0; --i ) if( two(i) & x ){
            if ( B[ i ] ) x ^= B[ i ];
            else {
                B[ i ] = x; sz++;
                for ( int j = i - 1; j >= 0; --j )
                    if( B[ j ] && ( two( j ) & B[ i ] ) )
                        B[ i ] ^= B[ j ];
                for ( int j = i + 1; j < n; ++j )
                    if ( two( i ) & B[ j ] )
                        B[ j ] ^= B[ i ];
                break;
            }
        }
    }
    inline int size() { return sz; }
    bool check( ll_u x ) {
        // is x in span(B) ?
        for ( int i = n-1; i >= 0; --i ) if( two(i) & x )
            if( B[ i ] ) x ^= B[ i ];
        else return false;
        return true;
    }
    ll_u kth_small(ll_u k) {
        // 1-base would always > 0
        // should check it
        // if we choose at least one element
        // but size(B)(vectors in B)=N(original elements)
        // then we can't get 0
        ll_u ret = 0;
        for ( int i = 0; i < n; ++i ) if( B[ i ] ) {
            if( k & 1 ) ret ^= B[ i ];
            k >>= 1;
        }
        return ret;
    }
} base;

```

3 Graph

3.1 Euler Circuit

```

bool vis[ N ]; size_t la[ K ];
void dfs( int u, vector< int >& vec ) {
    while ( la[ u ] < G[ u ].size() ) {
        if ( vis[ G[ u ][ la[ u ] ].second ] ) {
            ++ la[ u ];
            continue;
        }
        int v = G[ u ][ la[ u ] ].first;
        vis[ G[ u ][ la[ u ] ].second ] = true;
        ++ la[ u ]; dfs( v, vec );
        vec.push_back( v );
    }
}

```

3.2 BCC Edge

```
class BCC{
private:
    vector< int > low, dfn;
    int cnt;
    vector< bool > bridge;
    vector< vector< PII > > G;
    void dfs( int w, int f ) {
        dfn[ w ] = cnt++;
        low[ w ] = dfn[ w ];
        for ( auto [ u, t ] : G[ w ] ) {
            if ( u == f ) continue;
            if ( dfn[ u ] != 0 ) {
                low[ w ] = min( low[ w ], dfn[ u ] );
            } else {
                dfs( u, w );
                low[ w ] = min( low[ w ], low[ u ] );
                if ( low[ u ] > dfn[ w ] ) bridge[ t ] = true;
            }
        }
    }
public:
    void init( int n, int m ) {
        G.resize( n );
        fill( G.begin(), G.end(), vector< PII >() );
        bridge.clear(); bridge.resize( m );
        low.clear(); low.resize( n );
        dfn.clear(); dfn.resize( n );
        cnt = 0;
    }
    void add_edge( int u, int v ) {
        // should check for multiple edge
        G[ u ].emplace_back( v, cnt );
        G[ v ].emplace_back( u, cnt ++ );
    }
    void solve(){ cnt = 1; dfs( 0, 0 ); }
    // the id will be same as insert order, 0-base
    bool is_bridge( int x ) { return bridge[ x ]; }
} bcc;
```

3.3 BCC Vertex

```
class BCC{
private:
    int n, ecnt;
    vector< vector< pair< int, int > > > G;
    vector< int > low, dfn, id;
    vector< bool > vis, ap;
    void dfs( int u, int f, int d ) {
        int child = 0;
        dfn[ u ] = low[ u ] = d; vis[ u ] = true;
        for ( auto e : G[ u ] ) if ( e.first != f ) {
            if ( vis[ e.first ] ) {
                low[ u ] = min( low[ u ], dfn[ e.first ] );
            } else {
                dfs( e.first, u, d + 1 ); child ++;
                low[ u ] = min( low[ u ], low[ e.first ] );
                if ( low[ e.first ] >= d ) ap[ u ] = true;
            }
        }
        if ( u == f and child <= 1 ) ap[ u ] = false;
    }
    void mark( int u, int idd ) {
        // really????????
        if ( ap[ u ] ) return;
        for ( auto e : G[ u ] )
            if ( id[ e.second ] != -1 ) {
                id[ e.second ] = idd;
                mark( e.first, idd );
            }
    }
public:
    void init( int n_ ) {
        ecnt = 0, n = n_;
        G.clear(); G.resize( n );
        low.resize( n ); dfn.resize( n );
        ap.clear(); ap.resize( n );
        vis.clear(); vis.resize( n );
    }
    void add_edge( int u, int v ) {
```

```
G[ u ].emplace_back( v, ecnt );
G[ v ].emplace_back( u, ecnt ++ );
}
void solve() {
    for ( int i = 0 ; i < n ; ++ i )
        if ( not vis[ i ] ) dfs( i, i, 0 );
    id.resize( ecnt );
    fill( id.begin(), id.end(), -1 );
    ecnt = 0;
    for ( int i = 0 ; i < n ; ++ i )
        if ( ap[ i ] ) for ( auto e : G[ i ] )
            if ( id[ e.second ] != -1 ) {
                id[ e.second ] = ecnt;
                mark( e.first, ecnt ++ );
            }
}
int get_id( int x ) { return id[ x ]; }
int count() { return ecnt; }
bool is_ap( int u ) { return ap[ u ]; }
} bcc;
```

3.4 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;
```


3.5 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
    inline int lowbit( int x ) {
        return x & ( -x );
    }
    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_;
    }
    inline bool anc( int u, int v ) {
        return tl[ u ] <= tl[ v ] \
            and tr[ v ] <= tr[ u ];
    }
public:
    inline 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 ) {
        n ++;
        for ( LOG_N = 0 ; ( 1 << LOG_N ) < n ; ++ LOG_N );
        fa.clear();
        fa.resize( n, vector< int >( LOG_N ) );
        G.clear(); G.resize( n );
        tl.clear(); tl.resize( n );
        tr.clear(); tr.resize( n );
        chain.clear(); chain.resize( n );
        chain_st.clear(); chain_st.resize( n );
    }
    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_inter( 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;

```

3.6 MaxClique

```

#define N 111
struct MaxClique{ // 0-base
    typedef bitset< N > Int;
    Int linkto[ N ], v[ N ];
    int n;
    void init( int _n ){
        n = _n;
        for( int i = 0 ; i < n ; i ++ ){
            linkto[ i ].reset();
            v[ i ].reset();
        }
    }
    void add_edge( int a , int b ){
        v[ a ][ b ] = v[ b ][ a ] = 1;
    }
    int popcount(const Int& val)
    { return val.count(); }
    int lowbit(const Int& val)
    { return val._Find_first(); }
    int ans , stk[ N ];
    int id[ N ] , di[ N ] , deg[ N ];
    Int cans;
    void maxclique(int elem_num, Int candi){
        if(elem_num > ans){
            ans = elem_num;
            cans.reset();
            for( int i = 0 ; i < elem_num ; i ++ )
                cans[ id[ stk[ i ] ] ] = 1;
        }
        int potential = elem_num + popcount(candi);
        if(potential <= ans) return;
        int pivot = lowbit(candi);
        Int smaller_candi = candi & (~linkto[pivot]);
        while(smaller_candi.count() && potential > ans){
            int next = lowbit(smaller_candi);
            candi[ next ] = !candi[ next ];
            smaller_candi[next] = !smaller_candi[next];
            potential --;
            if(next != pivot
                &&!(smaller_candi & linkto[next]).count())
                continue;
            stk[elem_num] = next;
            maxclique(elem_num+1, candi & linkto[next]);
        }
    }
    int solve(){
        for( int i = 0 ; i < n ; i ++ ){
            id[ i ] = i;
            deg[ i ] = v[ i ].count();
        }
        sort( id , id + n , [&](int id1, int id2){
            return deg[id1] > deg[id2]; } );
        for( int i = 0 ; i < n ; i ++ )
            di[ id[ i ] ] = i;
        for( int i = 0 ; i < n ; i ++ )
            for( int j = 0 ; j < n ; j ++ )
                if( v[ i ][ j ] )
                    linkto[ di[ i ] ][ di[ j ] ] = 1;
        Int cand; cand.reset();
    }

```

```

    for( int i = 0 ; i < n ; i ++ )
        cand[ i ] = 1;
    ans = 1;
    cans.reset(); cans[ 0 ] = 1;
    maxclique(0, cand);
    return ans;
}
} solver;

```

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 Tree Hashing

```

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

```

3.9 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+1][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;

```

3.10 Mo's Algorithm on Tree

```

int n, q, nxt[ N ], to[ N ], hd[ N ];
struct Que{
    int u, v, id;
} que[ N ];
void init() {
    cin >> n >> q;
    for ( int i = 1 ; i < n ; ++ i ) {
        int u, v; cin >> u >> v;
        nxt[ i << 1 | 0 ] = hd[ u ];
        to[ i << 1 | 0 ] = v;
        hd[ u ] = i << 1 | 0;
        nxt[ i << 1 | 1 ] = hd[ v ];
        to[ i << 1 | 1 ] = u;
        hd[ v ] = i << 1 | 1;
    }
    for ( int i = 0 ; i < q ; ++ i ) {
        cin >> que[ i ].u >> que[ i ].v; que[ i ].id = i;
    }
}
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_ = hd[ u ] ; v_ ; v_ = nxt[ v_ ] ) {
        if ( to[ v_ ] == f ) continue;
        dfs( to[ v_ ], u );
        if ( stk_ - saved_rbp < SQRT_N ) continue;
        for ( ++ block_ ; stk_ != saved_rbp ; )
            block_id[ 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) ]
*/

```

3.11 Minimum Steiner Tree

```

// Minimum Steiner Tree
// O(V^3 T + V^2 2^T)
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;
            dp[ 0 ][ 0 ] = 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;
}

```

3.12 Directed Minimum Spanning Tree

```

template <typename T> struct DMST {
    T g[maxn][maxn], fw[maxn];
    int n, fr[maxn];
    bool vis[maxn], inc[maxn];
    void clear() {
        for( int i = 0 ; i < maxn ; ++i ) {
            for( int j = 0 ; j < maxn ; ++j ) g[i][j] = inf;
            vis[i] = inc[i] = false;
        }
    }
    void addEdge( int u, int v, T w ) { g[u][v] = min( g[u][v], w ); }
    T operator()( int root, int _n ) {
        n = _n; T ans = 0;
        if ( dfs( root ) != n ) return -1;
        while ( true ) {
            for( int i = 1 ; i <= n ; ++i ) fw[i] = inf, fr[i] = i;
            for( int i = 1 ; i <= n ; ++i ) if ( !inc[i] ) {
                for( int j = 1 ; j <= n ; ++j ) {
                    if ( !inc[j] && i != j && g[j][i] < fw[i] ) {
                        fw[i] = g[j][i]; fr[i] = j;
                    }
                }
            }
            int x = -1;
            for( int i = 1 ; i <= n ; ++i ) if ( i != root && !inc[i] ) {
                int j = i, c = 0;
                while ( j != root && fr[j] != i && c <= n ) ++c, j = fr[j];
                if ( j == root || c > n ) continue;
                else { x = i; break; }
            }
            if ( !~x ) {
                for( int i = 1 ; i <= n ; ++i )
                    if ( i != root && !inc[i] ) ans += fw[i];
                return ans;
            }
            int y = x;
            for( int i = 1 ; i <= n ; ++i ) vis[i] = false;
            do {
                ans += fw[y]; y = fr[y]; vis[y] = inc[y] = true;
            } while ( y != x );
            inc[x] = false;
            for( int k = 1 ; k <= n ; ++k ) if ( vis[k] ) {
                for( int j = 1 ; j <= n ; ++j ) if ( !vis[j] ) {
                    if ( g[x][j] > g[k][j] ) g[x][j] = g[k][j];
                    if ( g[j][k] < inf && g[j][k] - fw[k] < g[j][x] )
                        g[j][x] = g[j][k] - fw[k];
                }
            }
        }
        return ans;
    }
    int dfs( int now ) {
        int r = 1; vis[now] = true;
        for( int i = 1 ; i <= n ; ++i )
            if ( g[now][i] < inf && !vis[i] ) r += dfs(i);
        return r;
    }
};

```

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_;
    for (int i = 0; i < n; i++) {
        X[i].clear();
        Y[i].clear();
        fX[i] = fY[i] = -1;
    }
    walked.reset();
}
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

```

const int N = 514, E = (2e5) * 2;
struct Graph {
    int to[E], bro[E], head[N], e;
    int lnk[N], vis[N], stp, n;
    void init(int n_) {
        stp = 0; e = 1; n = n_;
        for (int i = 0; i <= n; i++)
            head[i] = lnk[i] = vis[i] = 0;
    }
    void add_edge(int u, int v) {
        // 1-base
        to[e] = v, bro[e] = head[u], head[u] = e++;
        to[e] = u, bro[e] = head[v], head[v] = e++;
    }
    bool dfs(int x) {
        vis[x] = stp;
        for (int i = head[x]; i; i = bro[i]) {
            int v = to[i];
            if (!lnk[v]) {
                lnk[x] = v, lnk[v] = x;
                return true;
            } else if (vis[lnk[v]] < stp) {
                int w = lnk[v];
                lnk[x] = v, lnk[v] = x, lnk[w] = 0;
                if (dfs(w)) return true;
                lnk[w] = v, lnk[v] = w, lnk[x] = 0;
            }
        }
        return false;
    }
    int solve() {
        int ans = 0;
        for (int i = 1; i <= n; i++)
            if (!lnk[i])
                stp++; ans += dfs(i);
        return ans;
    }
} graph;

```

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 Flow Models

- Maximum/Minimum flow with lower/upper bound from s to t

- 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 $in(v)$ as 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 ∞ , 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 ∞ 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 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 $f < K|V|$

4.6 Dinic

```

class Dinic{
private:
    using CapT = int64_t;
    struct Edge{
        int to, rev;
        CapT cap;
    };
    int n, st, ed;
    vector<vector<Edge>> G;
    vector<int> lv;
    bool BFS(){
        fill(lv.begin(), lv.end(), -1);
        queue<int> bfs;
        bfs.push(st);
        lv[st] = 0;
        while(!bfs.empty()){
            int u = bfs.front(); bfs.pop();
            for(auto e: G[u]){
                if(e.cap <= 0 or lv[e.to] != -1) continue;
                lv[e.to] = lv[u] + 1;
                bfs.push(e.to);
            }
        }
        return (lv[ed] != -1);
    }
    CapT DFS(int u, CapT f){
        if(u == ed) return f;
        CapT ret = 0;
        for(auto& e: G[u]){
            if(e.cap <= 0 or lv[e.to] != lv[u]+1) continue;
            CapT 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_, int st_, int ed_){
        n = n_, st = st_, ed = ed_;
        G.resize(n); lv.resize(n);
        fill(G.begin(), G.end(), vector<Edge>());
    }
    void add_edge(int u, int v, CapT c){
        G[u].push_back({v, (int)G[v].size(), c});
        G[v].push_back({u, ((int)G[u].size()-1), 0});
    }
    CapT max_flow(){
        CapT ret = 0;
        while(BFS()){

```

```

    CapT f = DFS(st, numeric_limits<CapT>::max());
    ret += f;
    if(f == 0) break;
}
return ret;
}
} flow;

```

4.7 Minimum Cost Maximum Flow

```

class MiniCostMaxiFlow{
    using CapT = int;
    using WeiT = int64_t;
    using PCW = pair<CapT,WeiT>;
    static constexpr CapT INF_CAP = 1 << 30;
    static constexpr WeiT INF_WEI = 1LL<<60;
private:
    struct Edge{
        int to, back;
        WeiT wei;
        CapT cap;
        Edge() {}
        Edge(int a,int b,WeiT c,CapT d):
            to(a),back(b),wei(c),cap(d)
        {}
    };
    int ori, edd;
    vector<vector<Edge>> G;
    vector<int> fa, wh;
    vector<bool> inq;
    vector<WeiT> 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(!qq.empty()){
            int u=qq.front();qq.pop();
            inq[u] = 0;
            for(int i=0;i<SZ(G[u]);++i){
                Edge e=G[u][i];
                int v=e.to;
                WeiT 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]=1;
            }
        }
        if(dis[edd]==INF_WEI)
            return {-1,-1};
        CapT 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,WeiT w,CapT c){
        G[st].emplace_back(ed,SZ(G[ed]),w,c);
        G[ed].emplace_back(st,SZ(G[st])-1,-w,0);
    }
    PCW solve(){
        /* might modify to
        cc += ret.first * ret.second
        or
        ww += ret.first * ret.second
        */

```

```

    CapT cc=0; WeiT ww=0;
    while(true){
        PCW ret=SPFA();
        if(ret.first==-1) break;
        cc+=ret.first;
        ww+=ret.second;
    }
    return {cc,ww};
}
} mcmf;

```

4.8 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[c] = true;
        s = t, t = c;
        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 Prime Table

```

1002939109, 1020288887, 1028798297, 1038684299,
1041211027, 1051762951, 1058585963, 1063020809,
1147930723, 1172520109, 1183835981, 1187659051,
1241251303, 1247184097, 1255940849, 1272759031,
1287027493, 1288511629, 1294632499, 1312650799,
1868732623, 1884198443, 1884616807, 1885059541,
1909942399, 1914471137, 1923951707, 1925453197,
1979612177, 1980446837, 1989761941, 2007826547,
2008033571, 2011186739, 2039465081, 2039728567,
2093735719, 2116097521, 2123852629, 2140170259,
3148478261, 3153064147, 3176351071, 3187523093,
3196772239, 3201312913, 3203063977, 3204840059,
3210224309, 3213032591, 3217689851, 3218469083,
3219857533, 3231880427, 3235951699, 3273767923,
3276188869, 3277183181, 3282463507, 3285553889,
3319309027, 3327005333, 3327574903, 3341387953,
3373293941, 3380077549, 3380892997, 3381118801

```

5.2 $\lfloor \frac{n}{i} \rfloor$ Enumeration

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

5.3 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;
}
```

5.4 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;
    }
}
```

5.5 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);
}
```

5.6 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*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*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){
        a = mpow(a, u, n);
        if (!a) 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 Inverse Element

```
// x's inverse mod k
long long GetInv(long long x, long long k){
    // k is prime: euler_(k)=k-1
    return qPow(x, euler_phi(k)-1);
}
// if you need [1, x] (most use: [1, k-1])
void solve(int x, long long k){
    inv[1] = 1;
    for(int i=2; i<x; i++){
        inv[i] = ((long long)(k - k/i) * inv[k % i]) % k;
    }
}
```

5.9 Euler Phi Function

```
/*
    extended euler:
    a^b mod p
    if gcd(a, p)==1: a^(b%phi(p))
    elif b < phi(p): a^b mod p
    else a^(b%phi(p) + phi(p))
*/
lld euler_phi(int x){
    lld r=1;
    for(int i=2; i*i<=x; ++i){
        if(x%i==0){
            x/=i;

```

```

    r*=(i-1);
    while(x%i==0){
        x/=i;
        r*=i;
    }
}
if(x>1) r*=x-1;
return r;
}

```

```

vector<int> primes;
bool notprime[N];
lld phi[N];
void euler_sieve(int n){
    for(int i=2;i<n;i++){
        if(!notprime[i]){
            primes.push_back(i);
            phi[i] = i-1;
        }
        for(auto j: primes){
            if(i*j >= n) break;
            notprime[i*j] = true;
            phi[i*j] = phi[i] * phi[j];
            if(i % j == 0){
                phi[i*j] = phi[i] * j;
                break;
            }
        }
    }
}

```

5.10 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];
        }
    }
}

```

5.11 Fast Fourier Transform

```

/*
    polynomial multiply:
    DFT(a, len); DFT(b, len);
    for(int i=0;i<len;i++) c[i] = a[i]*b[i];
    iDFT(c, len);
    (len must be 2^k and >= 2*(max(a, b)))
    Hand written Cplx would be 2x faster
*/
Cplx omega[2][N];
void init_omega(int n) {
    static constexpr llf PI=acos(-1);
    const llf arg=(PI/PI)/n;
    for(int i=0;i<n;++i)
        omega[0][i]={cos(arg*i),sin(arg*i)};
    for(int i=0;i<n;++i)
        omega[1][i]=conj(omega[0][i]);
}
void tran(Cplx arr[],int n,Cplx omg[]) {
    for(int i=0,j=0;i<n;++i){
        if(i>j)swap(arr[i],arr[j]);
        for(int l=n>>1;(j^=1)<l;l>>=1);
    }
    for (int l=2;l<=n;l<=1){
        int m=l>>1;
        for(auto p=arr;p!=arr+n;p+=l){
            for(int i=0;i<m;++i){
                Cplx t=omg[n/l*i]*p[m+i];

```

```

                p[m+i]=p[i]-t;
                p[i]+=t;
            }
        }
    }
}
void DFT(Cplx arr[],int n){
    tran(arr,n,omega[0]);
}
void iDFT(Cplx arr[],int n){
    tran(arr,n,omega[1]);
    for(int i=0;i<n;++i)arr[i]/=n;
}

```

5.12 High Speed Linear Recurrence

```

#define mod 998244353
const int N=1000010;
int n,k,m,f[N],h[N],a[N],b[N],ib[N];
int pw(int x,int y){
    int re=1;
    if(y<0)y+=mod-1;
    while(y){
        if(y&1)re=(ll)re*x%mod;
        y>>=1;x=(ll)x*x%mod;
    }
    return re;
}
void inc(int&x,int y){x+=y;if(x>=mod)x-=mod;}
namespace poly{
    const int G=3;
    int rev[N],L;
    void ntt(int*A,int len,int f){
        for(L=0;(1<L)<len;++L);
        for(int i=0;i<len;++i){
            rev[i]=(rev[i>>1]>>1)|((i&1)<<(L-1));
            if(i<rev[i])swap(A[i],A[rev[i]]);
        }
        for(int i=1;i<len;i<=1){
            int wn=pw(G,f*(mod-1)/((i<1)));
            for(int j=0;j<len;j+=i<1){
                int w=1;
                for(int k=0;k<i;++k,w=(ll)w*wn%mod){
                    int x=A[j+k],y=(ll)w*A[j+k+i]%mod;
                    A[j+k]=(x+y)%mod,A[j+k+i]=(x-y+mod)%mod;
                }
            }
        }
        if(!~f){
            int iv=pw(len,mod-2);
            for(int i=0;i<len;++i)A[i]=(ll)A[i]*iv%mod;
        }
    }
    void cls(int*A,int l,int r){
        for(int i=l;i<r;++i)A[i]=0;
    }
    void cpy(int*A,int*B,int l){
        for(int i=0;i<l;++i)A[i]=B[i];
    }
    void inv(int*A,int*B,int l){
        if(l==1){B[0]=pw(A[0],mod-2);return;}
        static int t[N];
        int len=l<<1;
        inv(A,B,l>>1);
        cpy(t,A,l);cls(t,l,len);
        ntt(t,len,1);ntt(B,len,1);
        for(int i=0;i<len;++i)
            B[i]=(ll)B[i]*(2-(ll)t[i]*B[i]%mod+mod)%mod;
        ntt(B,len,-1);cls(B,l,len);
    }
    void pmod(int*A){
        static int t[N];
        int l=k+1,len=1;while(len<=(k<<1))len<<=1;
        cpy(t,A,(k<<1)+1);
        reverse(t,t+(k<<1)+1);
        cls(t,l,len);
        ntt(t,len,1);
        for(int i=0;i<len;++i)t[i]=(ll)t[i]*ib[i]%mod;
        ntt(t,len,-1);
        cls(t,l,len);
        reverse(t,t+1);
        ntt(t,len,1);
        for(int i=0;i<len;++i)t[i]=(ll)t[i]*b[i]%mod;
    }
}

```



```

    ntt(t, len, -1);
    cls(t, 1, len);
    for(int i=0; i<k; ++i) A[i] = (A[i] - t[i] + mod) % mod;
    cls(A, k, len);
}
void pow(int* A, int n){
    if(n==1){cls(A, 0, k+1); A[1]=1; return;}
    pow(A, n>>1);
    int len=1; while(len<=(k<<1)) len<<=1;
    ntt(A, len, 1);
    for(int i=0; i<len; ++i) A[i] = (1ll)A[i]*A[i]%mod;
    ntt(A, len, -1);
    pmod(A);
    if(n&1){
        for(int i=k; i; --i) A[i] = A[i-1]; A[0]=0;
        pmod(A);
    }
}
int main(){
    n=rd(); k=rd();
    for(int i=1; i<=k; ++i) f[i] = (mod+rd())%mod;
    for(int i=0; i<k; ++i) h[i] = (mod+rd())%mod;
    for(int i=a[k]=b[k]=1; i<=k; ++i)
        a[k-i]=b[k-i]=(mod-f[i])%mod;
    int len=1; while(len<=(k<<1)) len<<=1;
    reverse(a, a+k+1);
    poly::inv(a, ib, len);
    poly::cls(ib, k+1, len);
    poly::ntt(b, len, 1);
    poly::ntt(ib, len, 1);
    poly::pow(a, n);
    int ans=0;
    for(int i=0; i<k; ++i) inc(ans, (1ll)a[i]*h[i]%mod);
    printf("%d\n", ans);
    return 0;
}

```

5.13 Chinese Remainder

```

lld crt(lld ans[], lld pri[], int n){
    lld M = 1;
    for(int i=0; i<n; ++i) M *= pri[i];
    lld ret = 0;
    for(int i=0; i<n; ++i){
        lld inv = (gcd(M/pri[i], pri[i]).first + pri[i])%
            pri[i];
        ret += (ans[i]*(M/pri[i])%M * inv)%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)
*/

```

5.14 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];
}

```

5.15 NTT

```

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

```

5.16 Polynomial Sqrt

```

const int mod = (119 << 23) + 1;
int inv_temp[400010];
void poly_inv(int *f, int *inv, int len){
    int *inv_t = inv_temp, *g = inv;
    g[0] = get_inv(f[0]);
    for(int l = 2; l <= len; l <= 1, swap(g, inv_t)){
        for(int i = 0; i < l; ++i){
            inv_t[i] = f[i];
            g[i+1] = inv_t[i+1] = 0;
        }
        exec_ntt(inv_t, 1 << 1, 1);
    }
}

```

```

    exec_ntt(g, 1 << 1, 1);
    for (int i = 0; i < 2 * l; i++)
        inv_t[i] = (1l)inv_t[i] * g[i] % mod;
    for (int i = 0; i < 2 * l; i++) {
        if (inv_t[i])
            inv_t[i] = mod - inv_t[i];
        inv_t[i] += 2, inv_t[i] %= mod;
    }
    for (int i = 0; i < 2 * l; i++)
        inv_t[i] = (1l)inv_t[i] * g[i] % mod;
    exec_ntt(inv_t, 1 << 1, -1);
    for (int i = 0; i < l; i++)
        inv_t[i + l] = 0;
}
for (int i = 0; i < len; i++)
    inv[i] = g[i];
}
int sqrt_temp[400010], inv_t[400010];
void poly_sqrt(int *f, int *sqrt_pol, int len) {
    int *g = sqrt_pol, *t = sqrt_temp, inv2 = get_inv(2);
    g[0] = 1;
    for (int l = 2; l <= len; l <= 1, swap(g, t)) {
        for (int i = 0; i < l; i++)
            t[i] = f[i], t[i + l] = g[i + l] = inv_t[i] = 0;
        poly_inv(g, inv_t, l);
        for (int i = l; i < 2 * l; i++)
            inv_t[i] = 0;
        exec_ntt(g, 1 << 1, 1);
        exec_ntt(inv_t, 1 << 1, 1);
        exec_ntt(t, 1 << 1, 1);
        for (int i = 0; i < (l << 1); i++)
            t[i] = (1l)inv2*(g[i] + (1l)t[i]*inv_t[i] % mod) % mod;
        exec_ntt(t, 1 << 1, -1);
        for (int i = 0; i < l; i++)
            t[i + l] = 0;
    }
    for (int i = 0; i < len; i++)
        sqrt_pol[i] = g[i];
}
int c[400010], inv[400010], sqrt_pol[400010];
int main() {
    int n, m, x;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        if (x <= m)
            c[x] = mod - 4;
    }
    c[0]++, c[0] %= mod;
    int len = 1;
    while (len <= m) len <= 1;
    poly_sqrt(c, sqrt_pol, len);
    sqrt_pol[0]++, sqrt_pol[0] %= mod;
    poly_inv(sqrt_pol, inv, len);
    for (int i = 1; i <= m; i++)
        printf("%d\n", (inv[i] + inv[i]) % mod);
    puts("");
    return 0;
}

```

5.17 Polynomial Division

```

VI inverse(const VI &v, int n) {
    VI q(1, fpow(v[0], mod - 2));
    for (int i = 2; i <= n; i <= 1) {
        VI fv(v.begin(), v.begin() + i);
        VI fq(q.begin(), q.end());
        fv.resize(2 * i), fq.resize(2 * i);
        ntt(fq, 2 * i), ntt(fv, 2 * i);
        for (int j = 0; j < 2 * i; ++j)
            fv[j] = fv[j] * 1ll * fq[j] % mod * fq[j] % mod;
        intt(fv, 2 * i);
        VI res(i);
        for (int j = 0; j < i; ++j) {
            res[j] = mod - fv[j];
            if (j < (i > 1)) (res[j] += 2 * q[j] % mod) %= mod;
        }
        q = res;
    }
    return q;
}

```

```

}
VI divide(const VI &a, const VI &b) {
    // Leading zero should be trimmed
    int n = (int)a.size(), m = (int)b.size();
    int k = 2;
    while (k < n - m + 1) k <= 1;
    VI ra(k), rb(k);
    for (int i = 0; i < min(n, k); ++i) ra[i] = a[n - i - 1];
    for (int i = 0; i < min(m, k); ++i) rb[i] = b[m - i - 1];
    VI rbi = inverse(rb, k);
    VI res = convolution(rbi, ra);
    res.resize(n - m + 1);
    reverse(res.begin(), res.end());
    return res;
}

```

5.18 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;
        }
}

```

5.19 DiscreteLog

```

// Baby-step Giant-step Algorithm
lll BSGS(lll P, lll B, lll N) {
    // find B^L = N mod P
    unordered_map<lll, int> R;
    lll sq = (1ll)sqrt(P);
    lll t = 1;
    for (int i = 0; i < sq; i++) {
        if (t == N) return i;
        if (!R.count(t)) R[t] = i;
        t = (t * B) % P;
    }
    lll f = inverse(t, P);
    for(int i=0;i<sq+1;i++) {
        if (R.count(N))
            return i * sq + R[N];
        N = (N * f) % P;
    }
    return -1;
}

```

5.20 Quadratic residue

```

struct Status{
    ll x,y;
};
ll w;
Status mult(const Status& a,const Status& b,ll mod){
    Status res;
    res.x=(a.x*b.x+a.y*b.y%mod*w)%mod;
}

```

```

    res.y=(a.x*b.y+a.y*b.x)%mod;
    return res;
}
inline Status qpow(Status _base,ll _pow,ll _mod){
    Status res;
    res.x=1,res.y=0;
    while(_pow>0){
        if(_pow&1) res=mult(res,_base,_mod);
        _base=mult(_base,_base,_mod);
        _pow>>=1;
    }
    return res;
}
inline ll check(ll x,ll p){
    return qpow_mod(x,(p-1)>>1,p);
}
inline ll get_root(ll n,ll p){
    if(p==2) return 1;
    if(check(n,p)==p-1) return -1;
    ll a;
    while(true){
        a=rand()%p;
        w=((a*a-n)%p+p)%p;
        if(check(w,p)==p-1) break;
    }
    Status res;
    res.x=a;
    res.y=1;
    res=qpow(res,(p+1)>>1,p);
    return res.x;
}

```

5.21 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;
}

```

5.22 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 $x_i - x'_i$

5.23 Simplex

```

namespace simplex {
    // maximize c^T x under Ax <= B
    // return vector<double>(n, -inf) if the solution doesn't exist
    // return vector<double>(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;
    }
}

```

6 Geometry

6.1 Point Class

```
template<typename T>
struct Point{
    typedef long double llf;
    static constexpr llf EPS = 1e-8;
    T x, y;
    Point(T _=0, T __=0): x(_), y(__){}
    template<typename T2>
        Point(const Point<T2>& a): x(a.x), y(a.y){}
    inline llf theta() const {
        return atan2((llf)y, (llf)x);
    }
    inline llf dis() const {
        return hypot((llf)x, (llf)y);
    }
    inline llf dis(const Point& o) const {
        return hypot((llf)(x-o.x), (llf)(y-o.y));
    }
    Point operator-(const Point& o) const {
        return Point(x-o.x, y-o.y);
    }
    Point operator+=(const Point& o){
        x+=o.x, y+=o.y;
        return *this;
    }
    Point operator+(const Point& o) const {
        return Point(x+o.x, y+o.y);
    }
    Point operator+=(const Point& o){
        x+=o.x, y+=o.y;
        return *this;
    }
    Point operator*(const T& k) const {
        return Point(x*k, y*k);
    }
    Point operator*=(const T& k){
        x*=k, y*=k;
        return *this;
    }
    Point operator/(const T& k) const {
        return Point(x/k, y/k);
    }
    Point operator/=(const T& k){
        x/=k, y/=k;
        return *this;
    }
    Point operator-() const {
        return Point(-x, -y);
    }
    Point rot90() const {
        return Point(-y, x);
    }
    template<typename T2>
    bool in(const Circle<T2>& a) const {
        /* Add struct Circle at top */
        return a.o.dis(*this)+EPS <= a.r;
    }
    bool equal(const Point& o, true_type) const {
        return fabs(x-o.x) < EPS and fabs(y-o.y) < EPS;
    }
    bool equal(const Point& o, false_type) const {
        return tie(x, y) == tie(o.x, o.y);
    }
    bool operator==(const Point& o) const {
        return equal(o, is_floating_point<T>());
    }
    bool operator!=(const Point& o) const {
        return !(*this == o);
    }
    bool operator<(const Point& o) const {
        return theta() < o.theta();
        // sort Like what pairs did
        // if(is_floating_point<T>()) return fabs(x-o.x)<
        // EPS?y<o.y:x<o.x;
        // else return tie(x, y) < tie(o.x, o.y);
    }
    friend inline T cross(const Point& a, const Point& b)
    {
        return a.x*b.y - b.x*a.y;
    }
};
```

```

    }
    friend inline T dot(const Point& a, const Point &b){
        return a.x*b.x + a.y*b.y;
    }
    friend ostream& operator<<(ostream& ss, const Point&
        o){
        ss<<"("<<o.x<<"", "<<o.y<<"")";
        return ss;
    }
};
```

6.2 Circle Class

```
template<typename T>
struct Circle{
    static constexpr llf EPS = 1e-8;
    Point<T> o; T r;
    vector<Point<llf>> operator&(const Circle& aa)const{
        // https://www.cnblogs.com/wangzming/p/8338142.html
        llf d=o.dis(aa.o);
        if(d > r+aa.r+EPS or d < fabs(r-aa.r)-EPS) return
            {};
        llf dt = (r*r - aa.r*aa.r)/d, d1 = (d+dt)/2;
        Point<llf> dir = (aa.o-o); dir /= d;
        Point<llf> pcrs = dir*d1 + o;
        dt=sqrt(max(0.0L, r*r - d1*d1)), dir=dir.rot90();
        return {pcrs + dir*dt, pcrs - dir*dt};
    }
};
```

6.3 Triangle Circumcentre

```
template<typename T>
Circle<llf> get_circum(const Point<T>& a, const Point<T>
    &b, const Point<T>& c){
    llf a1 = a.x-b.x;
    llf b1 = a.y-b.y;
    llf c1 = (a.x+b.x)/2 * a1 + (a.y+b.y)/2 * b1;
    llf a2 = a.x-c.x;
    llf b2 = a.y-c.y;
    llf c2 = (a.x+c.x)/2 * a2 + (a.y+c.y)/2 * b2;

    Circle<llf> 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;
}
```

6.4 2D Convex Hull

```
template<typename T>
class ConvexHull_2D{
private:
    typedef Point<T> PT;
    vector<PT> dots;
    struct myhash{
        uint64_t operator()(const PT& a) const {
            uint64_t xx=0, yy=0;
            memcpy(&xx, &a.x, sizeof(a.x));
            memcpy(&yy, &a.y, sizeof(a.y));
            uint64_t ret = xx*17+yy*31;
            ret = (ret ^ (ret >> 16))*0x9E3779B1;
            ret = (ret ^ (ret >> 13))*0xC2B2AE35;
            ret = ret ^ xx;
            return (ret ^ (ret << 3)) * yy;
        }
    };
    unordered_set<PT, myhash> in_hull;
public:
    inline void init(){in_hull.clear();dots.clear();}
    void insert(const PT& x){dots.pb(x);}
    void solve(){
        sort(ALL(dots), [](const PT& a, const PT& b){
            return tie(a.x, a.y) < tie(b.x, b.y);
        });
        vector<PT> stk(SZ(dots)<<1);
        int top = 0;
        for(auto p: dots){
```

```

    while(top >= 2 and cross(p-stk[top-2], stk[top-1]-stk[top-2]) <= 0)
        top --;
    stk[top++] = p;
}
for(int i=SZ(dots)-2, t = top+1;i>=0;i--){
    while(top >= t and cross(dots[i]-stk[top-2], stk[top-1]-stk[top-2]) <= 0)
        top --;
    stk[top++] = dots[i];
}
stk.resize(top-1);
swap(stk, dots);
for(auto i: stk) in_hull.insert(i);
}
vector<PT> get(){return dots;}
inline bool in_it(const PT& x){
    return in_hull.find(x)!=in_hull.end();
}
};

```

6.5 2D Farthest Pair

```

// stk is from convex hull
n = (int)(stk.size());
int pos = 1, ans = 0; stk.push_back(arr[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])});
}

```

6.6 2D Closest Pair

```

struct Point{
    llf x, y;
    llf dis;
} arr[N];

inline llf get_dis(Point a, Point b){
    return hypot(a.x-b.x, a.y-b.y);
}

llf solve(){
    int cur = rand()%n;
    for(int i=0;i<n;i++) arr[i].dis = get_dis(arr[cur], arr[i]);
    sort(arr, arr+n, [](Point a, Point b){return a.dis < b.dis;});
    llf ans = 1e50;
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(arr[j].dis - arr[i].dis > ans) break;
            ans = min(ans, get_dis(arr[i], arr[j]));
        }
    }
    return ans;
}

```

6.7 SimulateAnnealing

```

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 ) {

```

```

        S_best = S_prime;
        p_best = p_prime;
    }
}
return S_best;
}

```

6.8 Half Plane Intersection

```

inline int dcmp ( double x ) {
    if( fabs( x ) < eps ) return 0;
    return x > 0 ? 1 : -1;
}

struct Line {
    Point st, ed;
    double ang;
    Line( Point _st=Point(), Point _ed=Point() ):
        st(_st),ed(_ed),ang(atan2(_ed.y-_st.y,_ed.x-_st.x)) {}
    inline bool operator< ( const Line& rhs ) const {
        if(dcmp( ang - rhs.ang ) != 0) return ang < rhs.ang;
        return dcmp( cross( st, ed, rhs.st ) ) < 0;
    }
};

// cross(pt, line.ed-line.st) >= 0 <-> pt in half plane
vector< Line > lines;
deque< Line > que;
deque< Point > pt;
double HPI(){
    sort( lines.begin(), lines.end() );
    que.clear();
    pt.clear();
    que.push_back( lines[ 0 ] );
    for ( int i = 1 ; i < (int)lines.size() ; i ++ ) {
        if(!dcmp(lines[i].ang - lines[i - 1].ang)) continue;
        while ( pt.size() > 0 &&
            dcmp(cross(lines[i].st,lines[i].ed,pt.back()))<0){
            pt.pop_back();
            que.pop_back();
        }
        while ( pt.size() > 0 &&
            dcmp(cross(lines[i].st,lines[i].ed,pt.front()))<0)
            {
                pt.pop_front();
                que.pop_front();
            }
        pt.push_back( get_point( que.back(), lines[ i ] ) );
        que.push_back( lines[ i ] );
    }
    while ( pt.size() > 0 &&
        dcmp(cross(que[0].st, que[0].ed, pt.back()))<0){
        que.pop_back();
        pt.pop_back();
    }
    while ( pt.size() > 0 &&
        dcmp(cross(que.back().st, que.back().ed, pt[0]))<0)
        {
            que.pop_front();
            pt.pop_front();
        }
    pt.push_back( get_point( que.front(), que.back() ) );
    vector< Point > conv;
    for ( int i = 0 ; i < (int)pt.size() ; i ++ )
        conv.push_back( pt[ i ] );
    double ret = 0;
    for ( int i = 1 ; i + 1 < (int)conv.size() ; i ++ )
        ret += abs(cross(conv[0], conv[i], conv[i + 1]));
    return ret / 2.0;
}

```

6.9 Ternary Search on Integer

```

int TernarySearch(int l, int r) {
    // (l, r]
    while (r - l > 1){
        int mid = (l + r)>>1;
        if (f(mid) > f(mid + 1)) r = mid;
        else l = mid;
    }
}

```

```

    }
    return l+1;
}

```

6.10 Minimum Covering Circle

```

template<typename T>
Circle<llf> MinCircleCover(const vector<Point<T>>& pts)
{
    random_shuffle(ALL(pts));
    Circle<llf> c = {pts[0], 0};
    int n = SZ(pts);
    for(int i=0; i<n; i++){
        if(pts[i].in(c)) continue;
        c = {pts[i], 0};
        for(int j=0; j<i; j++){
            if(pts[j].in(c)) continue;
            c.o = (pts[i] + pts[j]) / 2;
            c.r = pts[i].dis(c.o);
            for(int k=0; k<j; k++){
                if(pts[k].in(c)) continue;
                c = get_circum(pts[i], pts[j], pts[k]);
            }
        }
    }
    return c;
}

```

6.11 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];
    int n;
    Node *root;
    LL dis2(int x1, int y1, int x2, int y2) {
        LL dx = x1-x2;
        LL dy = y1-y2;
        return dx*dx+dy*dy;
    }
    static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
    static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
    void init(vector<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 dep) {
        if (L>R) return nullptr;
        int M = (L+R)/2;
        tree[M].f = dep%2;
        nth_element(tree+L, tree+M, tree+R+1, tree[M].f ?
            cmpy : cmpx);
        tree[M].x1 = tree[M].x2 = tree[M].x;
        tree[M].y1 = tree[M].y2 = tree[M].y;

        tree[M].L = build_tree(L, M-1, dep+1);
        if (tree[M].L) {
            tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
        }
        tree[M].R = build_tree(M+1, R, dep+1);
        if (tree[M].R) {
            tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
        }
        return tree+M;
    }
    int touch(Node* r, int x, int y, LL d2){

```

```

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

```

7 Stringology

7.1 Hash

```

class Hash{
private:
    const int p = 127, q = 1208220623;
    int sz, prefix[N], power[N];
    inline int add(int x, int y){return x+y>=q?x+y-q:x+y;}
    inline int sub(int x, int y){return x-y<0?x-y+q:x-y;}
    inline int mul(int x, int y){return 1LL*x*y%q;}
public:
    void init(const string &x){
        sz = x.size();prefix[0]=0;power[0]=1;
        for(int i=1;i<=sz;i++)
            prefix[i]=add(mul(prefix[i-1], p), x[i-1]);
        for(int i=1;i<=sz;i++) power[i]=mul(power[i-1], p);
    }
    int query(int l, int r){
        // 1-base (l, r)
        return sub(prefix[r], mul(prefix[l], power[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[s[sa[i] - 1]]++] = sa[i] - 1;
        memcpy(x, c, sizeof(int) * z);
        for (int i = n - 1; i >= 0; --i)
            if (sa[i] && t[sa[i] - 1])

```



```

    sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
}
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 Aho-Corasick Algorithm

```

class AhoCorasick{
private:
    static constexpr int Z = 26;
    struct node{
        node *nxt[ Z ], *fail;
        vector< int > data;
        node(): fail( nullptr ) {
            memset( nxt, 0, sizeof( nxt ) );
            data.clear();
        }
    } *rt;
    inline int Idx( char c ) { return c - 'a'; }
public:
    void init() { rt = new node(); }
    void add( const string& s, int d ) {
        node* cur = rt;
        for ( auto c : s ) {
            if ( not cur->nxt[ Idx( c ) ] )
                cur->nxt[ Idx( c ) ] = new node();
            cur = cur->nxt[ Idx( c ) ];
        }
        cur->data.push_back( d );
    }
    void compile() {
        vector< node* > bfs;
        size_t ptr = 0;
        for ( int i = 0; i < Z; ++i ) {
            if ( not rt->nxt[ i ] )

```

```

                continue;
            rt->nxt[ i ]->fail = rt;
            bfs.push_back( rt->nxt[ i ] );
        }
        while ( ptr < bfs.size() ) {
            node* u = bfs[ ptr++ ];
            for ( int i = 0; i < Z; ++i ) {
                if ( not u->nxt[ i ] )
                    continue;
                node* u_f = u->fail;
                while ( u_f ) {
                    if ( not u_f->nxt[ i ] ) {
                        u_f = u_f->fail; continue;
                    }
                    u->nxt[ i ]->fail = u_f->nxt[ i ];
                    break;
                }
                if ( not u_f ) u->nxt[ i ]->fail = rt;
                bfs.push_back( u->nxt[ i ] );
            }
        }
    }
    void match( const string& s, vector< int >& ret ) {
        node* u = rt;
        for ( auto c : s ) {
            while ( u != rt and not u->nxt[ Idx( c ) ] )
                u = u->fail;
            u = u->nxt[ Idx( c ) ];
            if ( not u ) u = rt;
            node* tmp = u;
            while ( tmp != rt ) {
                for ( auto d : tmp->data )
                    ret.push_back( d );
                tmp = tmp->fail;
            }
        }
    }
} ac;

```

7.4 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; i < 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;
}

```

7.5 KMP

```

int F[N<<1];
void KMP(char s1[], char s2[], int n, int m){
    // make F[] for s1+'\0'+s2;
    char ss[N<<1];
    int len = n+m+1;
    for(int i=0;i<n;i++) ss[i] = s1[i];
    ss[n] = '\0';
    for(int i=0;i<m;i++) ss[i+1+n] = s2[i];
    F[0] = F[1] = 0;
    for(int i=1;i<len;i++){
        int j = F[i];
        while(j > 0 and ss[i]!=ss[j]) j = F[j];
        F[i+1] = (ss[i]==ss[j]?j+1:0);
    }
    // just find (F[len2+i] == len2)
    // i from 1 to len+1 for matching
}
/*
[0, i]是個循環字串，且循環節為i-f[i]:
if(f[i]>0 and i%(i-f[i])==0)
cout << i << " " << i/(i-f[i]) << '\n';
*/

```

7.6 Z value

```

char s[MAXN];
int len,z[MAXN];
void Z_value() {
    int i,j,left,right;
    left=right=0; z[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++);
        z[i]=j;
        if(i+z[i]>right) {
            right=i+z[i];
            left=i;
        }
    }
}

```

7.7 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;
}

```

7.8 Lexicographically Smallest Rotation

```

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

```

7.9 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;

```

7.10 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> state;
    vector<char> s;
    int last,n;
    void init(){
        state.clear();s.clear();last=1; n=0;
        state.push_back(0);state.push_back(-1);
        state[0].f=1;s.push_back(-1);
    }
    int getFail(int x){
        while(s[n-state[x].len-1]!=s[n])x=state[x].f;
        return x;
    }
    void add(int c){
        s.push_back(c-'a'); ++n;
        int cur=getFail(last);
        if(!state[cur].next[c]){
            int now=state.size();
            state.push_back(state[cur].len+2);
            state[now].f=state[getFail(state[cur].f)].next[c];
            state[cur].next[c]=now;
            state[now].num=state[state[now].f].num+1;
        }
        last=state[cur].next[c];
    }
}

```

```

    ++state[last].cnt;
}
int size(){
    return state.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.state[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 $T_{n,k} = kn^{n-k-1}$.

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

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

8.1.8 Pick's theorem

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

8.2 MaximumEmptyRect

```

int max_empty_rect(int n, int m, bool blocked[N][N]){
    static int mxu[2][N], me=0, he=1, ans=0;
    for(int i=0; i<m; i++) mxu[he][i]=0;
    for(int i=0; i<n; i++){
        stack<PII>, vector<PII>> stk;
        for(int j=0; j<m; ++j){
            if(blocked[i][j]) mxu[me][j]=0;
            else mxu[me][j]=mxu[he][j]+1;
        }
    }
}

```

```

int la = j;
while(!stk.empty() && stk.top().FF > mxu[me][j]){
    int x1 = i - stk.top().FF, x2 = i;
    int y1 = stk.top().SS, y2 = j;
    la = stk.top().SS; stk.pop();
    ans = max(ans, (x2-x1)*(y2-y1));
}
if(stk.empty() || stk.top().FF < mxu[me][j])
    stk.push({mxu[me][j], la});
}
while(!stk.empty()){
    int x1 = i - stk.top().FF, x2 = i;
    int y1 = stk.top().SS-1, y2 = m-1;
    stk.pop();
    ans = max(ans, (x2-x1)*(y2-y1));
}
swap(me, he);
}
return ans;
}

```

8.3 DP-opt Condition

8.3.1 totally monotone (concave/convex)

$$\forall i < i', j < j', B[i][j] \leq B[i'][j] \implies B[i][j'] \leq B[i'][j']$$

$$\forall i < i', j < j', B[i][j] \geq B[i'][j] \implies B[i][j'] \geq B[i'][j']$$

8.3.2 monge condition (concave/convex)

$$\forall i < i', j < j', B[i][j] + B[i'][j'] \geq B[i][j'] + B[i'][j]$$

$$\forall i < i', j < j', B[i][j] + B[i'][j'] \leq B[i][j'] + B[i'][j]$$

8.4 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> deq; deq.push_back(segment(0, 1, n));
    for (int i = 1; i <= n; ++i) {
        dp[i] = f(deq.front().i, i);
        while(deq.size() && deq.front().r < i+1) deq.pop_front();
        deq.front().l = i+1;
        segment seg = segment(i, i+1, n);
        while (deq.size() && f(i, deq.back().l) < f(deq.back().i, deq.back().l))
            deq.pop_back();
        if (deq.size()) {
            int d = 1048576, c = deq.back().l;
            while (d >= 1) if (c + d <= deq.back().r)
                if (f(i, c+d) > f(deq.back().i, c+d)) c += d;
            deq.back().r = c; seg.l = c+1;
        }
        if (seg.l <= n) deq.push_back(seg);
    }
}

```

8.5 ConvexHull Optimization

```

inline lld DivCeil(lld n, lld d) { // ceil(n/d)
    return n / d + ((n < 0) != (d > 0)) && (n % d);
}
struct Line {
    static bool flag;
    lld a, b, l, r; // y=ax+b in [l, r]
    lld operator()(lld x) const { return a * x + b; }
    bool operator<(const Line& i) const {
        return flag ? tie(a, b) < tie(i.a, i.b) : l < i.l;
    }
    lld operator&(const Line& i) const {
        return DivCeil(b - i.b, i.a - a);
    }
};
bool Line::flag = true;
class ConvexHullMax {
}

```

```

set<Line> L;
public:
ConvexHullMax() { Line::flag = true; }
void InsertLine(lld a, lld b) { // add y = ax + b
    Line now = {a, b, -INF, INF};
    if (L.empty()) {
        L.insert(now);
        return;
    }
    Line::flag = true;
    auto it = L.lower_bound(now);
    auto prv = it == L.begin() ? it : prev(it);
    if (it != L.end() && ((it != L.begin() &&
        (*it)(it->l) >= now(it->l) &&
        (*prv)(prv->r - 1) >= now(prv->r - 1)) ||
        (it == L.begin() && it->a == now.a))) return;
    if (it != L.begin()) {
        while (prv != L.begin() &&
            (*prv)(prv->l) <= now(prv->l))
            prv = --L.erase(prv);
        if (prv == L.begin() && now.a == prv->a)
            L.erase(prv);
    }
    if (it != L.end())
        while (it != --L.end() &&
            (*it)(it->r) <= now(it->r))
            it = L.erase(it);
    if (it != L.begin()) {
        prv = prev(it);
        const_cast<Line*>(&*prv)->r=now.l((*prv)&now);
    }
    if (it != L.end())
        const_cast<Line*>(&*it)->l=now.r((*it)&now);
    L.insert(it, now);
}
lld Query(lld a) const { // query max at x=a
    if (L.empty()) return -INF;
    Line::flag = false;
    auto it = --L.upper_bound({0, 0, a, 0});
    return (*it)(a);
}
};

```

8.6 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;
}

```

8.7 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%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;
}

```

8.8 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;

```

```

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;
}

```

8.10 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;
}

```

8.9 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);
            }
        }
    }
}

```