Contents

1 Reminder

1.1 Bug List

- 沒開 long long
- 本地編譯請開-Wall -Wextra -Wshadow -fsanitize=address
- 陣列戳出界 / 開不夠大 / 開太大本地 compile 噴怪 error
- 傳之前先確定選對檔案
- 變數打錯
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- dp[i] 從 dp[i-1] 轉移時忘記特判 i > 0
- std::sort 比較運算子寫成 < 或是讓 = 的情況為 true
- 漏 case / 分 case 要好好想
- 線段樹改值懶標初始值不能設為 0
- 少碰動態開點，能離散化就離散化
- 能不用浮點數運算就不用
- DFS 的時候不小心覆寫到全域變數
- 記得刪 cerr

1.2 OwO

- 可以構造複雜點的測資幫助思考
- 真的卡太久請跳題
- Enjoy The Contest!

2 Basic

2.1 Vimrc

```
set number relativenumber ai t_Co=256 tabstop=4
set mouse=a shiftwidth=4 encoding=utf8
set bs=2 ruler laststatus=2 cmdheight=2
set clipboard=unnamedplus showcmd autoread
set belloff=all
filetype indent on

inoremap ( ()<Esc>i
inoremap " ""<Esc>i
inoremap [ []<Esc>i
inoremap ' ''<Esc>i
inoremap { {<CR>}<Esc>ko

nnoremap <tab> gt
nnoremap <S-tab> gT
inoremap <C-n> <Esc>:tabnew<CR>
nnoremap <C-n> :tabnew<CR>

inoremap <F9> <Esc>:w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>
nnoremap <F9> :w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>

syntax on
colorscheme desert
set filetype=cpp
set background=dark
hi Normal ctermfg=white ctermbg=black
```

2.2 Stress

```
g++ gen.cpp -o gen.out
g++ ac.cpp -o ac.out
g++ wa.cpp -o wa.out
for ((i=0;;i++))
do
    echo "$i"
    ./gen.out > in.txt
    ./ac.out < in.txt > ac.txt
    ./wa.out < in.txt > wa.txt
    if [ “$?” -ne 0 ]; then
      exit 1
    fi
    diff ac.txt wa.txt || break
done
```

2.3 Run Sample

```
prog=$1
shift
g++ -O2 -std=c++20 -fsanitize=address -Wall -Wextra -
    Wshadow ${prog}.cpp -o ${prog}.out
for f in "$@"; do
  out=${prog}_$(basename "$f").out
  echo "input: $f"
  cat "$f"
  echo "output: $out"
  ./${prog}.out < "$f" | tee "$out"
  echo
done
```

2.4 Runcpp.sh

```
#! /bin/bash
clear
echo "Start compiling $1..."
echo
g++ -O2 -std=c++20 -fsanitize=address -Wall -Wextra -
    Wshadow $2/$1 -o $2/out
if [ "$?" -ne 0 ]
then
    exit 1
fi
echo
echo "Done compiling"
echo "=========================="
echo
echo "Input file:"
echo
```

```
cat $2/in.txt
echo
echo "=========================="
echo
declare startTime=`date +%s%N`
$2/out < $2/in.txt > $2/out.txt
declare endTime=`date +%s%N`
delta=`expr $endTime - $startTime`
delta=`expr $delta / 1000000`
cat $2/out.txt
echo
echo "time: $delta ms"
```

2.5 Others

```
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
#pragma GCC target("see,see2,see3,see4,avx2,bmi,bmi2,lzcnt,popcnt,tune=native")
#pragma GCC optimize("trapv")
mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> dis(1, 100);
cout << dis(gen) << endl;
shuffle(v.begin(), v.end(), gen);

struct edge {
    int a, b, w;
    friend istream& operator>>(istream& in, edge& x) { in >> x.a >> x.b >> x.w; }
    friend ostream& operator<<(ostream& out, const edge& x) {
        out << "(" << x.a << "," << x.b << "," << x.w << ")";
        return out;
    }
};
struct cmp {
    bool operator()(const edge& x, const edge& y) const { return x.w < y.w; }
};
set<edge, cmp> st;                              // 遞增
map<edge, long long, cmp> mp;                   // 遞增
priority_queue<edge, vector<edge>, cmp> pq;     // 遞減

#include <bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

// map
tree<int, int, less<>, rb_tree_tag, tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

// set
tree<int, null_type, less<>, rb_tree_tag, tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

gp_hash_table<int, int> ht;
ht.find(element);
ht.insert({key, value});
ht.erase(element);

// priority queue Big First
__gnu_pbds::priority_queue<int, less<int>> big_q;
__gnu_pbds::priority_queue<int, greater<int>> small_q; // Small First
q1.join(q2);  // join
```

3 Data Structure

3.1 BIT

```
struct BIT {
    int n;
    long long bit[N];

    void init(int x, vector<long long> &a) {
        n = x;
        for (int i = 1, j; i <= n; i++) {
            bit[i] += a[i - 1], j = i + (i & -i);
            if (j <= n) bit[j] += bit[i];
        }
    }

    void update(int x, long long dif) {
        while (x <= n) bit[x] += dif, x += x & -x;
    }

    long long query(int l, int r) {
        if (l != 1) return query(1, r) - query(1, l - 1);

        long long ret = 0;
        while (l <= r) ret += bit[r], r -= r & -r;
        return ret;
    }
} bm;
```

3.2 Lazy Propagation Segment Tree

```
struct lazy_propagation{
    // 0-based, [l, r], tg[0]->add, tg[1]->set
    ll seg[N * 4], tg[2][N*4];
    void assign (bool op, ll val, int idx){
        if (op == 0){
            if (tg[1][idx]) tg[1][idx] += val;
            else            tg[0][idx] += val;
        }
        else    seg[idx] = 0, tg[0][idx] = 0, tg[1][idx] = val;
    }
    ll sum (int idx, int len){
        if (tg[1][idx]) return tg[1][idx] * len;
        return tg[0][idx] * len + seg[idx];
    }
    void pull (int idx, int len){
        seg[idx] = sum(2*idx, (len+1)/2) + sum(2*idx+1, len/2);
    }
    void push (int idx){
        if (!tg[0][idx] && !tg[1][idx]) return ;
        if (tg[0][idx]){
            assign(0, tg[0][idx], 2*idx);
            assign(0, tg[0][idx], 2*idx+1);
            tg[0][idx] = 0;
        }
        else{
            assign(1, tg[1][idx], 2*idx);
            assign(1, tg[1][idx], 2*idx+1);
            tg[1][idx] = 0;
        }
    }
    void update (bool op, ll val, int gl, int gr, int l, int r, int idx){
        if (r < l || gr < l || r < gl)  return ;
        if (gl <= l && r <= gr){
            assign(op, val, idx);
            return ;
        }

        int mid = (l + r) / 2;
        push(idx);
        update(op, val, gl, gr, l, mid, 2*idx);
        update(op, val, gl, gr, mid+1, r, 2*idx+1);
        pull(idx, r-l+1);
    }
    ll query (int gl, int gr, int l, int r, int idx){
        if (r < l || gr < l || r < gl)  return 0;
        if (gl <= l && r <= gr) return sum(idx, r-l+1);

        push(idx), pull(idx, r-l+1);
        int mid = (l + r) / 2;
        return query(gl, gr, l, mid, 2*idx) + query(gl, gr, mid+1, r, 2*idx+1);
    }
}bm;
```

3.3 Treap

```cpp
mt19937 rng(random_device{}());
struct Treap {
    Treap *l, *r;
    int val, sum, real, tag, num, pri, rev;
    Treap(int k) {
        l = r = NULL;
        val = sum = k;
        num = 1;
        real = -1;
        tag = 0;
        rev = 0;
        pri = rng();
    }
};
int siz(Treap *now) { return now ? now->num : 0ll; }
int sum(Treap *now) {
    if (!now) return 0;
    if (now->real != -1) return (now->real + now->tag)
         * now->num;
    return now->sum + now->tag * now->num;
}
void pull(Treap *&now) {
    now->num = siz(now->l) + siz(now->r) + 1ll;
    now->sum = sum(now->l) + sum(now->r) + now->val +
         now->tag;
}
void push(Treap *&now) {
    if (now->rev) {
        swap(now->l, now->r);
        now->l->rev ^= 1;
        now->r->rev ^= 1;
        now->rev = 0;
    }
    if (now->real != -1) {
        now->real += now->tag;
        if (now->l) {
            now->l->tag = 0;
            now->l->real = now->real;
            now->l->val = now->real;
        }
        if (now->r) {
            now->r->tag = 0;
            now->r->real = now->real;
            now->r->val = now->real;
        }
        now->val = now->real;
        now->sum = now->real * now->num;
        now->real = -1;
        now->tag = 0;
    } else {
        if (now->l) now->l->tag += now->tag;
        if (now->r) now->r->tag += now->tag;
        now->sum += sum(now);
        now->val += now->tag;
        now->tag = 0;
    }
}
Treap *merge(Treap *a, Treap *b) {
    if (!a || !b) return a ? a : b;
    else if (a->pri > b->pri) {
        push(a);
        a->r = merge(a->r, b);
        pull(a);
        return a;
    } else {
        push(b);
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}
void split_size(Treap *rt, Treap *&a, Treap *&b, int
     val) {
    if (!rt) {
        a = b = NULL;
        return;
    }
    push(rt);
    if (siz(rt->l) + 1 > val) {
        b = rt;
        split_size(rt->l, a, b->l, val);
        pull(b);
    } else {
        a = rt;
        split_size(rt->r, a->r, b, val - siz(a->l) - 1)
            ;
        pull(a);
    }
}
void split_val(Treap *rt, Treap *&a, Treap *&b, int val
    ) {
    if (!rt) {
        a = b = NULL;
        return;
    }
    push(rt);
    if (rt->val <= val) {
        a = rt;
        split_val(rt->r, a->r, b, val);
        pull(a);
    } else {
        b = rt;
        split_val(rt->l, a, b->l, val);
        pull(b);
    }
}
```

3.4 Persistent Treap

```cpp
struct node {
    node *l, *r;
    char c;
    int v, sz;
    node(char x = '$') : c(x), v(mt()), sz(1) {
        l = r = nullptr;
    }
    node(node* p) { *this = *p; }
    void pull() {
        sz = 1;
        for (auto i : {l, r})
            if (i) sz += i->sz;
    }
} arr[maxn], *ptr = arr;
inline int size(node* p) { return p ? p->sz : 0; }
node* merge(node* a, node* b) {
    if (!a || !b) return a ?: b;
    if (a->v < b->v) {
        node* ret = new (ptr++) node(a);
        ret->r = merge(ret->r, b), ret->pull();
        return ret;
    } else {
        node* ret = new (ptr++) node(b);
        ret->l = merge(a, ret->l), ret->pull();
        return ret;
    }
}
P<node*> split(node* p, int k) {
    if (!p) return {nullptr, nullptr};
    if (k >= size(p->l) + 1) {
        auto [a, b] = split(p->r, k - size(p->l) - 1);
        node* ret = new (ptr++) node(p);
        ret->r = a, ret->pull();
        return {ret, b};
    } else {
        auto [a, b] = split(p->l, k);
        node* ret = new (ptr++) node(p);
        ret->l = b, ret->pull();
        return {a, ret};
    }
}
```

3.5 Li Chao Tree

```cpp
const int eps=1e-9;
struct line
{
     double m,k;
}p[N];
int seg[N<<2],cnt;
double cal(int id,int x){return 1.0*p[id].m*x+p[id].k;}
void add(int x0,int y0,int x1,int y1)
{
    cnt++;
    if(x0==x1)p[cnt].m=0,p[cnt].k=max(y0,y1);
    else p[cnt].m=1.0*(y1-y0)/(x1-x0),p[cnt].k=y0-p[cnt].m*x0;
}
void update(int x,int l,int r,int ql,int qr,int u)
{
    int v=seg[x],mid=(l+r)>>1;
    double resu=cal(u,mid),resv=cal(v,mid);
    if(qr<l||r<ql)return;
    if(ql<=l&&r<=qr)
    {
        if(l==r)
        {
            if(resu>resv)seg[x]=u;
            return;
        }
        if(p[u].m-p[v].m>eps)
        {
            if(resu-resv>eps)
            {
                seg[x]=u;
                update(x<<1,l,mid,ql,qr,v);
            }
            else update(x<<1|1,mid+1,r,ql,qr,u);
        }
        else if(p[v].m-p[u].m>eps)
        {
            if(resu-resv>eps)
            {
                seg[x]=u;
                update(x<<1|1,mid+1,r,ql,qr,v);
            }
            else update(x<<1,l,mid,ql,qr,u);
        }
        else if(p[u].k-p[v].k>eps)seg[x]=u;
        return;
    }
    update(x<<1,l,mid,ql,qr,u);
    update(x<<1|1,mid+1,r,ql,qr,u);
}
double ask(int x,int l,int r,int qx)
{
    if(r<qx||qx<l)return{0,0};
    int mid=(l+r)>>1;
    double res=cal(seg[x],qx);
    if(l==r)return res;
    return max({res,seg[x],ask(x<<1,l,mid,qx),ask(x<<1|1,mid+1,r,qx)});
}
```

3.6 LineContainer

```cpp
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }

};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

3.7 Sparse Table

```cpp
int a[N];
int st[N][30];
void pre(int n)
{
    FOR(i,1,n+1)st[i][0]=a[i];
    for(int j=1;(1<<j)<=n+1;j++)
        for(int i=0;i+(1<<j)<=n+1;i++)
            st[i][j]=min(st[i][j-1],st[i+(1<<(j-1))][j-1]);
}
int ask(int l,int r)
{
    int k=__lg(r-l+1);
    return min(st[l][k],st[r-(1<<k)+1][k]);
}
```

3.8 Time Segment Tree

```cpp
constexpr int maxn = 1e5 + 5;
V<P<int>> arr[(maxn + 1) << 2];
V<int> dsu, sz;
V<tuple<int, int, int>> his;
int cnt, q;
int find(int x) {
    return x == dsu[x] ? x : find(dsu[x]);
};
inline bool merge(int x, int y) {
    int a = find(x), b = find(y);
    if (a == b) return false;
    if (sz[a] > sz[b]) swap(a, b);
    his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] += sz[a];
    return true;
};
inline void undo() {
    auto [a, b, s] = his.back();
    his.pop_back();
    dsu[a] = a, sz[b] = s;
}
#define m ((l + r) >> 1)
void insert(int ql, int qr, P<int> x, int i = 1, int l = 0, int r = q) {
    // debug(ql, qr, x); return;
    if (qr <= l || r <= ql) return;
    if (ql <= l && r <= qr) {
        arr[i].push_back(x);
        return;
    }
    if (qr <= m)
        insert(ql, qr, x, i << 1, l, m);
    else if (m <= ql)
        insert(ql, qr, x, i << 1 | 1, m, r);
    else {
        insert(ql, qr, x, i << 1, l, m);
        insert(ql, qr, x, i << 1 | 1, m, r);
    }
}
void traversal(V<int>& ans, int i = 1, int l = 0, int r = q) {
    int opcnt = 0;
    // debug(i, l, r);
    for (auto [a, b] : arr[i])
        if (merge(a, b))
            opcnt++, cnt--;
    if (r - l == 1)
        ans[l] = cnt;
    else {
        traversal(ans, i << 1, l, m);
        traversal(ans, i << 1 | 1, m, r);
    }
    while (opcnt--)
        undo(), cnt++;
    arr[i].clear();
}
```

```cpp
#undef m
inline void solve() {
    int n, m;
    cin >> n >> m >> q, q++;
    dsu.resize(cnt = n), sz.assign(n, 1);
    iota(dsu.begin(), dsu.end(), 0);
    // a, b, time, operation
    unordered_map<ll, V<int>> s;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        if (a > b) swap(a, b);
        s[((ll)a << 32) | b].emplace_back(0);
    }
    for (int i = 1; i < q; i++) {
        int op, a, b;
        cin >> op >> a >> b;
        if (a > b) swap(a, b);
        switch (op) {
            case 1:
                s[((ll)a << 32) | b].push_back(i);
                break;
            case 2:
                auto tmp = s[((ll)a << 32) | b].back();
                s[((ll)a << 32) | b].pop_back();
                insert(tmp, i, P<int>{a, b});
        }
    }
    for (auto [p, v] : s) {
        int a = p >> 32, b = p & -1;
        while (v.size()) {
            insert(v.back(), q, P<int>{a, b});
            v.pop_back();
        }
    }
    V<int> ans(q);
    traversal(ans);
    for (auto i : ans)
        cout << i << ' ';
    cout << endl;
}
```

3.9 Dynamic Median

```cpp
struct Dynamic_Median {
    multiset<long long> lo, hi;
    long long slo = 0, shi = 0;
    void rebalance() {
        // keep sz(lo) >= sz(hi) and sz(lo) - sz(hi) <=
            1
        while((int)lo.size() > (int)hi.size() + 1) {
            auto it = prev(lo.end());
            long long x = *it;
            lo.erase(it); slo -= x;
            hi.insert(x); shi += x;
        }
        while((int)lo.size() < (int)hi.size()) {
            auto it = hi.begin();
            long long x = *it;
            hi.erase(it); shi -= x;
            lo.insert(x); slo += x;
        }
    }
    void add(long long x) {
        if(lo.empty() || x <= *prev(lo.end())) {
            lo.insert(x); slo += x;
        }
        else {
            hi.insert(x); shi += x;
        }
        rebalance();
    }
    void remove_one(long long x) {
        if(!lo.empty() && x <= *prev(lo.end())) {
            auto it = lo.find(x);
            if(it != lo.end()) {
                lo.erase(it); slo -= x;
            }
            else {
                auto it2 = hi.find(x);
                hi.erase(it2); shi -= x;
            }
        }
        else {
            auto it = hi.find(x);
            if(it != hi.end()) {
                hi.erase(it); shi -= x;
            }
            else {
                auto it2 = lo.find(x);
                lo.erase(it2); slo -= x;
            }
        }
        rebalance();
    }
};
```

3.10 SOS DP

```cpp
for (int mask = 0; mask < (1 << n); mask++) {
    for (int submask = mask; submask != 0; submask = (
        submask - 1) & mask) {
        int subset = mask ^ submask;
}   }
```

4 Flow / Matching

4.1 Dinic

```cpp
using namespace std;
const int N = 2000 + 5;
int n, m, s, t, level[N], iter[N];
struct edge {int to, cap, rev;};
vector<edge> path[N];
void add(int a, int b, int c) {
    path[a].pb({b, c, sz(path[b])});
    path[b].pb({a, 0, sz(path[a]) - 1});
}
void bfs() {
    memset(level, -1, sizeof(level));
    level[s] = 0;
    queue<int> q;
    q.push(s);
    while (q.size()) {
        int now = q.front();q.pop();
        for (edge e : path[now]) if (e.cap > 0 && level
            [e.to] == -1) {
                level[e.to] = level[now] + 1;
                q.push(e.to);
        }
    }
}
int dfs(int now, int flow) {
    if (now == t) return flow;
    for (int &i = iter[now]; i < sz(path[now]); i++) {
        edge &e = path[now][i];
        if (e.cap > 0 && level[e.to] == level[now] + 1)
             {
            int res = dfs(e.to, min(flow, e.cap));
            if (res > 0) {
                e.cap -= res;
                path[e.to][e.rev].cap += res;
                return res;
            }
        }
    }
    return 0;
}
int dinic() {
    int res = 0;
    while (true) {
        bfs();
        if (level[t] == -1) break;
        memset(iter, 0, sizeof(iter));
        int now = 0;
        while ((now = dfs(s, INF)) > 0) res += now;
    }
    return res;
}
```

4.2 MCMF

```cpp
struct MCMF {
    int n, s, t, par[N + 5], p_i[N + 5], dis[N + 5], vis[N + 5];
    struct edge {
        int to, cap, rev, cost;
    };
    vector<edge> path[N];
    void init(int _n, int _s, int _t) {
        n = _n, s = _s, t = _t;
        FOR(i, 0, 2 * n + 5)
        par[i] = p_i[i] = vis[i] = 0;
    }
    void add(int a, int b, int c, int d) {
        path[a].pb({b, c, sz(path[b]), d});
        path[b].pb({a, 0, sz(path[a]) - 1, -d});
    }
    void spfa() {
        FOR(i, 0, n * 2 + 5)
        dis[i] = INF,
        vis[i] = 0;
        dis[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            vis[now] = 0;
            for (int i = 0; i < sz(path[now]); i++) {
                edge e = path[now][i];
                if (e.cap > 0 && dis[e.to] > dis[now] + e.cost) {
                    dis[e.to] = dis[now] + e.cost;
                    par[e.to] = now;
                    p_i[e.to] = i;
                    if (vis[e.to] == 0) {
                        vis[e.to] = 1;
                        q.push(e.to);
                    }
                }
            }
        }
    }
    pii flow() {
        int flow = 0, cost = 0;
        while (true) {
            spfa();
            if (dis[t] == INF)
                break;
            int mn = INF;
            for (int i = t; i != s; i = par[i])
                mn = min(mn, path[par[i]][p_i[i]].cap);
            flow += mn;
            cost += dis[t] * mn;
            for (int i = t; i != s; i = par[i]) {
                edge &now = path[par[i]][p_i[i]];
                now.cap -= mn;
                path[i][now.rev].cap += mn;
            }
        }
        return mp(flow, cost);
    }
};
```

4.3 KM

```
struct KM {
    int n, mx[1005], my[1005], pa[1005];
    int g[1005][1005], lx[1005], ly[1005], sy[1005];
    bool vx[1005], vy[1005];
    void init(int _n) {
        n = _n;
        FOR(i, 1, n + 1)
        fill(g[i], g[i] + 1 + n, 0);
    }
    void add(int a, int b, int c) { g[a][b] = c; }
    void augment(int y) {
        for (int x, z; y; y = z)
            x = pa[y], z = mx[x], my[y] = x, mx[x] = y;
    }
    void bfs(int st) {
        FOR(i, 1, n + 1)
        sy[i] = INF,
        vx[i] = vy[i] = 0;
        queue<int> q;
        q.push(st);
        for (;;) {
            while (!q.empty()) {
                int x = q.front();
                q.pop();
                vx[x] = 1;
                FOR(y, 1, n + 1)
                if (!vy[y]) {
                    int t = lx[x] + ly[y] - g[x][y];
                    if (t == 0) {
                        pa[y] = x;
                        if (!my[y]) {
                            augment(y);
                            return;
                        }
                        vy[y] = 1, q.push(my[y]);
                    } else if (sy[y] > t)
                        pa[y] = x, sy[y] = t;
                }
            }
            int cut = INF;
            FOR(y, 1, n + 1)
            if (!vy[y] && cut > sy[y]) cut = sy[y];
            FOR(j, 1, n + 1) {
                if (vx[j]) lx[j] -= cut;
                if (vy[j])
                    ly[j] += cut;
                else
                    sy[j] -= cut;
            }
            FOR(y, 1, n + 1) {
                if (!vy[y] && sy[y] == 0) {
                    if (!my[y]) {
                        augment(y);
                        return;
                    }
                    vy[y] = 1;
                    q.push(my[y]);
                }
            }
        }
    }
    int solve() {
        fill(mx, mx + n + 1, 0);
        fill(my, my + n + 1, 0);
        fill(ly, ly + n + 1, 0);
        fill(lx, lx + n + 1, 0);
        FOR(x, 1, n + 1)
        FOR(y, 1, n + 1)
        lx[x] = max(lx[x], g[x][y]);
        FOR(x, 1, n + 1)
        bfs(x);
        int ans = 0;
        FOR(y, 1, n + 1)
        ans += g[my[y]][y];
        return ans;
    }
};
```

4.4 Hopcroft-Karp

```
struct HopcroftKarp {
    // id: X = [1, nx], Y = [nx+1, nx+ny]
    int n, nx, ny, m, MXCNT;
    vector<vector<int> > g;
    vector<int> mx, my, dis, vis;
    void init(int nnx, int nny, int mm) {
        nx = nnx, ny = nny, m = mm;
        n = nx + ny + 1;
        g.clear();
        g.resize(n);
    }
    void add(int x, int y) {
        g[x].emplace_back(y);
        g[y].emplace_back(x);
    }
    bool dfs(int x) {
        vis[x] = true;
        Each(y, g[x]) {
            int px = my[y];
            if (px == -1 ||
                (dis[px] == dis[x] + 1 &&
                 !vis[px] && dfs(px))) {
                mx[x] = y;
                my[y] = x;
                return true;
            }
        }
        return false;
    }
    void get() {
        mx.clear();
        mx.resize(n, -1);
        my.clear();
        my.resize(n, -1);

        while (true) {
            queue<int> q;
            dis.clear();
            dis.resize(n, -1);
            for (int x = 1; x <= nx; x++) {
                if (mx[x] == -1) {
                    dis[x] = 0;
                    q.push(x);
                }
            }
            while (!q.empty()) {
                int x = q.front();
                q.pop();
                Each(y, g[x]) {
                    if (my[y] != -1 && dis[my[y]] ==
                        -1) {
                        dis[my[y]] = dis[x] + 1;
                        q.push(my[y]);
                    }
                }
            }

            bool brk = true;
            vis.clear();
            vis.resize(n, 0);
            for (int x = 1; x <= nx; x++)
                if (mx[x] == -1 && dfs(x))
                    brk = false;

            if (brk) break;
        }
        MXCNT = 0;
        for (int x = 1; x <= nx; x++)
            if (mx[x] != -1) MXCNT++;
    }
} hk;
```

4.5 Blossom

```
const int N=5e2+10;
struct Graph{
    int to[N],bro[N],head[N],e;
    int lnk[N],vis[N],stp,n;
    void init(int _n){
        stp=0;e=1;n=_n;
        FOR(i,0,n+1)head[i]=lnk[i]=vis[i]=0;
    }
    void add(int u,int v){
        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(i,1,n+1){
            if(!lnk[i]){
                stp++;
                ans+=dfs(i);
            }
        }
        return ans;
    }
    void print_matching(){
        FOR(i,1,n+1)
            if(i<graph.lnk[i])
                cout<<i<<" "<<graph.lnk[i]<<endl;
    }
};
```

4.6 Cover / Independent Set

```
V(E) Cover: choose some V(E) to cover all E(V)
V(E) Independ: set of V(E) not adj to each other

M = Max Matching
Cv = Min V Cover
Ce = Min E Cover
Iv = Max V Ind
Ie = Max E Ind (equiv to M)

M = Cv (Konig Theorem)
Iv = V \ Cv
Ce = V - M

Construct Cv:
1. Run Dinic
2. Find s-t min cut
3. Cv = {X in T} + {Y in S}
```

4.7 Hungarian Algorithm

```
const int N = 2e3;
int match[N];
bool vis[N];
int n;
vector<int> ed[N];
int match_cnt;
bool dfs(int u) {
    vis[u] = 1;
    for(int i : ed[u]) {
        if(match[i] == 0 || !vis[match[i]] && dfs(match
            [i])) {
            match[i] = u;
            return true;
        }
    }
    return false;
}
void hungary() {
    memset(match, 0, sizeof(match));
    match_cnt = 0;
    for(int i = 1; i <= n; i++) {
        memset(vis, 0, sizeof(vis));
        if(dfs(i)) match_cnt++;
    }
}
```

5 Graph

5.1 Heavy-Light Decomposition

```
const int N = 2e5 + 5;
int n, dfn[N], son[N], top[N], num[N], dep[N], p[N];
vector<int> path[N];
struct node {
    int mx, sum;
} seg[N << 2];
void update(int x, int l, int r, int qx, int val) {
    if (l == r) {
        seg[x].mx = seg[x].sum = val;
        return;
    }
    int mid = (l + r) >> 1;
    if (qx <= mid)update(x << 1, l, mid, qx, val);
    else update(x << 1 | 1, mid + 1, r, qx, val);
    seg[x].mx = max(seg[x << 1].mx, seg[x << 1 | 1].mx)
        ;
    seg[x].sum = seg[x << 1].sum + seg[x << 1 | 1].sum;
}
int big(int x, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) return seg[x].mx;
    int mid = (l + r) >> 1;
    int res = -INF;
    if (ql <= mid) res = max(res, big(x << 1, l, mid,
        ql, qr));
    if (mid < qr) res = max(res, big(x << 1 | 1, mid +
        1, r, ql, qr));
    return res;
}
int ask(int x, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) return seg[x].sum;
    int mid = (l + r) >> 1;
    int res = 0;
    if (ql <= mid) res += ask(x << 1, l, mid, ql, qr);
    if (mid < qr) res += ask(x << 1 | 1, mid + 1, r, ql
        , qr);
    return res;
}
void dfs1(int now) {
    son[now] = -1;
    num[now] = 1;
    for (auto i : path[now]) {
        if (!dep[i]) {
            dep[i] = dep[now] + 1;
            p[i] = now;
            dfs1(i);
            num[now] += num[i];
            if (son[now] == -1 || num[i] > num[son[now
                ]]) son[now] = i;
        }
    }
}
int cnt;
void dfs2(int now, int t) {
    top[now] = t;
    cnt++;
    dfn[now] = cnt;
    if (son[now] == -1) return;
    dfs2(son[now], t);
    for (auto i : path[now])
        if (i != p[now] && i != son[now])dfs2(i, i);
}
int path_big(int x, int y) {
    int res = -INF;
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        res = max(res, big(1, 1, n, dfn[top[x]], dfn[x
            ]));
        x = p[top[x]];
    }
    if (dfn[x] > dfn[y]) swap(x, y);
    res = max(res, big(1, 1, n, dfn[x], dfn[y]));
    return res;
}
int path_sum(int x, int y) {
    int res = 0;
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        res += ask(1, 1, n, dfn[top[x]], dfn[x]);
        x = p[top[x]];
    }
    if (dfn[x] > dfn[y]) swap(x, y);
    res += ask(1, 1, n, dfn[x], dfn[y]);
    return res;
}
void buildTree() {
    FOR(i, 0, n - 1) {
        int a, b;
        cin >> a >> b;
        path[a].pb(b);
        path[b].pb(a);
    }
}
void buildHLD(int root) {
    dep[root] = 1;
    dfs1(root);
    dfs2(root, root);
    FOR(i, 1, n + 1) {
        int now;
        cin >> now;
        update(1, 1, n, dfn[i], now);
    }
}
```

5.2 Centroid Decomposition

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
vector<int> a[N];
int sz[N], lv[N];
bool used[N];
int f_sz(int x, int p) {
    sz[x] = 1;
    for (int i : a[x])
        if (i != p && !used[i])
            sz[x] += f_sz(i, x);
    return sz[x];
}
int f_cen(int x, int p, int total) {
    for (int i : a[x]) {
        if (i != p && !used[i] && 2 * sz[i] > total)
            return f_cen(i, x, total);
    }
    return x;
}
void cd(int x, int p) {
    int total = f_sz(x, p);
    int cen = f_cen(x, p, total);
    lv[cen] = lv[p] + 1;
    used[cen] = 1;
    // cout << "cd: " << x << " " << p << " " << cen <<
        "\n";
    for (int i : a[cen]) {
        if (!used[i])
            cd(i, cen);
    }
}
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int n;
    cin >> n;
    for (int i = 0, x, y; i < n - 1; i++) {
        cin >> x >> y;
        a[x].push_back(y);
        a[y].push_back(x);
    }
    cd(1, 0);
    for (int i = 1; i <= n; i++)
        cout << (char)('A' + lv[i] - 1) << " ";
    cout << "\n";
}
```

5.3 Bellman-Ford + SPFA

```
int n, m;

// Graph
vector<vector<pair<int, ll> > > g;
vector<ll> dis;
vector<bool> negCycle;

// SPFA
vector<int> rlx;
queue<int> q;
vector<bool> inq;
vector<int> pa;
void SPFA(vector<int>& src) {
    dis.assign(n + 1, LINF);
    negCycle.assign(n + 1, false);
    rlx.assign(n + 1, 0);
    while (!q.empty()) q.pop();
```

```cpp
    inq.assign(n + 1, false);
    pa.assign(n + 1, -1);

    for (auto& s : src) {
        dis[s] = 0;
        q.push(s);
        inq[s] = true;
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        if (rlx[u] >= n) {
            negCycle[u] = true;
        } else
            for (auto& e : g[u]) {
                int v = e.first;
                ll w = e.second;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    rlx[v] = rlx[u] + 1;
                    pa[v] = u;
                    if (!inq[v]) {
                        q.push(v);
                        inq[v] = true;
                    }
                }
            }
    }
}

// Bellman-Ford
queue<int> q;
vector<int> pa;
void BellmanFord(vector<int>& src) {
    dis.assign(n + 1, LINF);
    negCycle.assign(n + 1, false);
    pa.assign(n + 1, -1);

    for (auto& s : src) dis[s] = 0;

    for (int rlx = 1; rlx <= n; rlx++) {
        for (int u = 1; u <= n; u++) {
            if (dis[u] == LINF) continue;  // Important
                !!
            for (auto& e : g[u]) {
                int v = e.first;
                ll w = e.second;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    pa[v] = u;
                    if (rlx == n) negCycle[v] = true;
                }
            }
        }
    }
}

// Negative Cycle Detection
void NegCycleDetect() {
    /* No Neg Cycle: NO
    Exist Any Neg Cycle:
    YES
    v0 v1 v2 ... vk v0 */

    vector<int> src;
    for (int i = 1; i <= n; i++)
        src.emplace_back(i);

    SPFA(src);
    // BellmanFord(src);

    int ptr = -1;
    for (int i = 1; i <= n; i++)
        if (negCycle[i]) {
            ptr = i;
            break;
        }

    if (ptr == -1) {
        return cout << "NO" << endl, void();
    }

    cout << "YES\n";
    vector<int> ans;
    vector<bool> vis(n + 1, false);

    while (true) {
        ans.emplace_back(ptr);
        if (vis[ptr]) break;
        vis[ptr] = true;
        ptr = pa[ptr];
    }
    reverse(ans.begin(), ans.end());

    vis.assign(n + 1, false);
    for (auto& x : ans) {
        cout << x << ' ';
        if (vis[x]) break;
        vis[x] = true;
    }
    cout << endl;
}

// Distance Calculation
void calcDis(int s) {
    vector<int> src;
    src.emplace_back(s);
    SPFA(src);
    // BellmanFord(src);

    while (!q.empty()) q.pop();
    for (int i = 1; i <= n; i++)
        if (negCycle[i]) q.push(i);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto& e : g[u]) {
            int v = e.first;
            if (!negCycle[v]) {
                q.push(v);
                negCycle[v] = true;
            }
        }
    }
}
```

5.4 BCC - AP

```cpp
int n, m;
int low[maxn], dfn[maxn], instp;
vector<int> E, g[maxn];
bitset<maxn> isap;
bitset<maxm> vis;
stack<int> stk;
int bccnt;
vector<int> bcc[maxn];
inline void popout(int u) {
    bccnt++;
    bcc[bccnt].emplace_back(u);
    while (!stk.empty()) {
        int v = stk.top();
        if (u == v) break;
        stk.pop();
        bcc[bccnt].emplace_back(v);
    }
}
void dfs(int u, bool rt = 0) {
    stk.push(u);
    low[u] = dfn[u] = ++instp;
    int kid = 0;
    Each(e, g[u]) {
        if (vis[e]) continue;
        vis[e] = true;
        int v = E[e] ^ u;
        if (!dfn[v]) {
            // tree edge
            kid++;
            dfs(v);
            low[u] = min(low[u], low[v]);
            if (!rt && low[v] >= dfn[u]) {
                // bcc found: u is ap
                isap[u] = true;
                popout(u);
            }
        } else {
            // back edge
            low[u] = min(low[u], dfn[v]);
        }
    }
    // special case: root
    if (rt) {
        if (kid > 1) isap[u] = true;
        popout(u);
    }
}
void init() {
    cin >> n >> m;
    fill(low, low + maxn, INF);
    REP(i, m) {
        int u, v;
        cin >> u >> v;
        g[u].emplace_back(i);
        g[v].emplace_back(i);
        E.emplace_back(u ^ v);
    }
}
void solve() {
    FOR(i, 1, n + 1, 1) {
        if (!dfn[i]) dfs(i, true);
    }
    vector<int> ans;
    int cnt = 0;
    FOR(i, 1, n + 1, 1) {
        if (isap[i]) cnt++, ans.emplace_back(i);
    }
    cout << cnt << endl;
    Each(i, ans) cout << i << ' ';
    cout << endl;
}
```

5.5 BCC - Bridge

```
int n, m;
vector<int> g[maxn], E;
int low[maxn], dfn[maxn], instp;
int bccnt, bccid[maxn];
stack<int> stk;
bitset<maxm> vis, isbrg;
void init() {
    cin >> n >> m;
    REP(i, m) {
        int u, v;
        cin >> u >> v;
        E.emplace_back(u ^ v);
        g[u].emplace_back(i);
        g[v].emplace_back(i);
    }
    fill(low, low + maxn, INF);
}
void popout(int u) {
    bccnt++;
    while (!stk.empty()) {
        int v = stk.top();
        if (v == u) break;
        stk.pop();
        bccid[v] = bccnt;
    }
}
void dfs(int u) {
    stk.push(u);
    low[u] = dfn[u] = ++instp;

    Each(e, g[u]) {
        if (vis[e]) continue;
        vis[e] = true;

        int v = E[e] ^ u;
        if (dfn[v]) {
            // back edge
            low[u] = min(low[u], dfn[v]);
        } else {
            // tree edge
            dfs(v);
            low[u] = min(low[u], low[v]);
            if (low[v] == dfn[v]) {
                isbrg[e] = true;
                popout(u);
            }
        }
    }
}
void solve() {
    FOR(i, 1, n + 1, 1) {
        if (!dfn[i]) dfs(i);
    }
    vector<pii> ans;
    vis.reset();
    FOR(u, 1, n + 1, 1) {
        Each(e, g[u]) {
            if (!isbrg[e] || vis[e]) continue;
            vis[e] = true;
            int v = E[e] ^ u;
            ans.emplace_back(mp(u, v));
        }
    }
    cout << (int)ans.size() << endl;
    Each(e, ans) cout << e.F << ' ' << e.S << endl;
}
```

5.6 SCC - Tarjan

```
// 2-SAT
vector<int> E, g[maxn];  // 1~n, n+1~2n
int low[maxn], in[maxn], instp;
int sccnt, sccid[maxn];
stack<int> stk;
bitset<maxn> ins, vis;
int n, m;
void init() {
    cin >> m >> n;
    E.clear();
    fill(g, g + maxn, vector<int>());
    fill(low, low + maxn, INF);
    memset(in, 0, sizeof(in));
    instp = 1;
    sccnt = 0;
    memset(sccid, 0, sizeof(sccid));
    ins.reset();
    vis.reset();
}
inline int no(int u) {
    return (u > n ? u - n : u + n);
}
int ecnt = 0;
inline void clause(int u, int v) {
    E.eb(no(u) ^ v);
    g[no(u)].eb(ecnt++);
    E.eb(no(v) ^ u);
    g[no(v)].eb(ecnt++);
}
void dfs(int u) {
    in[u] = instp++;
    low[u] = in[u];
    stk.push(u);
    ins[u] = true;

    Each(e, g[u]) {
        if (vis[e]) continue;
        vis[e] = true;

        int v = E[e] ^ u;
        if (ins[v])
            low[u] = min(low[u], in[v]);
        else if (!in[v]) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
    }
    if (low[u] == in[u]) {
        sccnt++;
        while (!stk.empty()) {
            int v = stk.top();
            stk.pop();
            ins[v] = false;
            sccid[v] = sccnt;
```

```cpp
            if (u == v) break;
        }
    }
}
int main() {
    init();
    REP(i, m) {
        char su, sv;
        int u, v;
        cin >> su >> u >> sv >> v;
        if (su == '-') u = no(u);
        if (sv == '-') v = no(v);
        clause(u, v);
    }
    FOR(i, 1, 2 * n + 1, 1) {
        if (!in[i]) dfs(i);
    }
    FOR(u, 1, n + 1, 1) {
        int du = no(u);
        if (sccid[u] == sccid[du]) {
            return cout << "IMPOSSIBLE\n", 0;
        }
    }
    FOR(u, 1, n + 1, 1) {
        int du = no(u);
        cout << (sccid[u] < sccid[du] ? '+' : '-') << ' 
            ';
    }
    cout << endl;
}
```

5.7 SCC - Kosaraju

```cpp
const int N = 1e5 + 10;
vector<int> ed[N], ed_b[N];  // 反邊
vector<int> SCC(N);          // 最後SCC的分組
bitset<N> vis;
int SCC_cnt;
int n, m;
vector<int> pre;  // 後序遍歷

void dfs(int x) {
    vis[x] = 1;
    for (int i : ed[x]) {
        if (vis[i]) continue;
        dfs(i);
    }
    pre.push_back(x);
}

void dfs2(int x) {
    vis[x] = 1;
    SCC[x] = SCC_cnt;
    for (int i : ed_b[x]) {
        if (vis[i]) continue;
        dfs2(i);
    }
}

void kosaraju() {
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            dfs(i);
        }
    }
    SCC_cnt = 0;
    vis = 0;
    for (int i = n - 1; i >= 0; i--) {
        if (!vis[pre[i]]) {
            SCC_cnt++;
            dfs2(pre[i]);
        }
    }
}
```

5.8 Eulerian Path - Undir

```cpp
// from 1 to n
#define gg return cout << "IMPOSSIBLE\n", void();

int n, m;
vector<int> g[maxn];
bitset<maxn> inodd;

void init() {
    cin >> n >> m;
    inodd.reset();
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        inodd[u] = inodd[u] ^ true;
        inodd[v] = inodd[v] ^ true;
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
}
stack<int> stk;
void dfs(int u) {
    while (!g[u].empty()) {
        int v = g[u].back();
        g[u].pop_back();
        dfs(v);
    }
    stk.push(u);
}
```

5.9 Eulerian Path - Dir

```cpp
// from node 1 to node n
#define gg return cout << "IMPOSSIBLE\n", 0

int n, m;
vector<int> g[maxn];
stack<int> stk;
int in[maxn], out[maxn];

void init() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].emplace_back(v);
        out[u]++, in[v]++;
    }
    for (int i = 1; i <= n; i++) {
        if (i == 1 && out[i] - in[i] != 1) gg;
        if (i == n && in[i] - out[i] != 1) gg;
        if (i != 1 && i != n && in[i] != out[i]) gg;
    }
}
void dfs(int u) {
    while (!g[u].empty()) {
        int v = g[u].back();
        g[u].pop_back();
        dfs(v);
    }
    stk.push(u);
}
void solve() {
    dfs(1) for (int i = 1; i <= n; i++) if ((int)g[i].
        size()) gg;
    while (!stk.empty()) {
        int u = stk.top();
        stk.pop();
        cout << u << ' ';
    }
}
```

5.10 Hamilton Path

```cpp
// top down DP
// Be Aware Of Multiple Edges
int n, m;
ll dp[maxn][1<<maxn];
int adj[maxn][maxn];

void init() {
    cin >> n >> m;
    fill(dp[0], dp[maxn-1]+(1<<maxn), -1);
}

void DP(int i, int msk) {
    if (dp[i][msk] != -1) return;
    dp[i][msk] = 0;
    REP(j, n) if (j != i && (msk & (1<<j)) && adj[j][i
        ]) {
        int sub = msk ^ (1<<i);
        if (dp[j][sub] == -1) DP(j, sub);
        dp[i][msk] += dp[j][sub] * adj[j][i];
        if (dp[i][msk] >= MOD) dp[i][msk] %= MOD;
    }
}


int main() {
    WiwiHorz
    init();

    REP(i, m) {
        int u, v;
        cin >> u >> v;
        if (u == v) continue;
        adj[--u][--v]++;
    }

    dp[0][1] = 1;
    FOR(i, 1, n, 1) {
        dp[i][1] = 0;
        dp[i][1|(1<<i)] = adj[0][i];
    }
    FOR(msk, 1, (1<<n), 1) {
        if (msk == 1) continue;
        dp[0][msk] = 0;
    }


    DP(n-1, (1<<n)-1);
    cout << dp[n-1][(1<<n)-1] << endl;

    return 0;
}
```

5.11 Kth Shortest Path

```cpp
// time: O(|E| \lg |E|+|V| \lg |V|+K)
// memory: O(|E| \lg |E|+|V|)
struct KSP {  // 1-base
    struct nd {
        int u, v;
        ll d;
        nd(int ui = 0, int vi = 0, ll di = INF) {
            u = ui;
            v = vi;
            d = di;
        }
    };
    struct heap {
        nd* edge;
        int dep;
        heap* chd[4];
    };
    static int cmp(heap* a, heap* b) { return a->edge->
        d > b->edge->d; }
    struct node {
        int v;
        ll d;
        heap* H;
        nd* E;
        node() {}
        node(ll _d, int _v, nd* _E) {
            d = _d;
            v = _v;
            E = _E;
        }
        node(heap* _H, ll _d) {
            H = _H;
            d = _d;
        }
        friend bool operator<(node a, node b) { return
            a.d > b.d; }
    };
    int n, k, s, t, dst[N];
    nd* nxt[N];
    vector<nd*> g[N], rg[N];
    heap *nullNd, *head[N];
    void init(int _n, int _k, int _s, int _t) {
        n = _n;
        k = _k;
        s = _s;
        t = _t;
        for (int i = 1; i <= n; i++) {
            g[i].clear();
            rg[i].clear();
            nxt[i] = NULL;
            head[i] = NULL;
            dst[i] = -1;
        }
    }
    void addEdge(int ui, int vi, ll di) {
        nd* e = new nd(ui, vi, di);
        g[ui].push_back(e);
        rg[vi].push_back(e);
    }
    queue<int> dfsQ;
    void dijkstra() {
        while (dfsQ.size()) dfsQ.pop();
        priority_queue<node> Q;
        Q.push(node(0, t, NULL));
        while (!Q.empty()) {
            node p = Q.top();
            Q.pop();
            if (dst[p.v] != -1) continue;
            dst[p.v] = p.d;
            nxt[p.v] = p.E;
            dfsQ.push(p.v);
            for (auto e : rg[p.v]) Q.push(node(p.d + e
                ->d, e->u, e));
        }
    }
    heap* merge(heap* curNd, heap* newNd) {
        if (curNd == nullNd) return newNd;
        heap* root = new heap;
        memcpy(root, curNd, sizeof(heap));
        if (newNd->edge->d < curNd->edge->d) {
            root->edge = newNd->edge;
            root->chd[2] = newNd->chd[2];
            root->chd[3] = newNd->chd[3];
            newNd->edge = curNd->edge;
            newNd->chd[2] = curNd->chd[2];
            newNd->chd[3] = curNd->chd[3];
        }
        if (root->chd[0]->dep < root->chd[1]->dep)
            root->chd[0] = merge(root->chd[0], newNd);
        else
            root->chd[1] = merge(root->chd[1], newNd);
        root->dep = max(root->chd[0]->dep,
                        root->chd[1]->dep) +
                    1;
        return root;
    }
    vector<heap*> V;
    void build() {
        nullNd = new heap;
        nullNd->dep = 0;
        nullNd->edge = new nd;
        fill(nullNd->chd, nullNd->chd + 4, nullNd);
        while (not dfsQ.empty()) {
            int u = dfsQ.front();
            dfsQ.pop();
            if (!nxt[u])
                head[u] = nullNd;
            else
                head[u] = head[nxt[u]->v];
            V.clear();
            for (auto&& e : g[u]) {
                int v = e->v;
                if (dst[v] == -1) continue;
                e->d += dst[v] - dst[u];
                if (nxt[u] != e) {
                    heap* p = new heap;
                    fill(p->chd, p->chd + 4, nullNd);
                    p->dep = 1;
                    p->edge = e;
                    V.push_back(p);
                }
            }
            if (V.empty()) continue;
            make_heap(V.begin(), V.end(), cmp);
#define L(X) ((X << 1) + 1)
#define R(X) ((X << 1) + 2)
            for (size_t i = 0; i < V.size(); i++) {
                if (L(i) < V.size())
                    V[i]->chd[2] = V[L(i)];
                else
                    V[i]->chd[2] = nullNd;
                if (R(i) < V.size())
                    V[i]->chd[3] = V[R(i)];
                else
                    V[i]->chd[3] = nullNd;
            }
            head[u] = merge(head[u], V.front());
        }
    }
    vector<ll> ans;
    void first_K() {
        ans.clear();
        priority_queue<node> Q;
        if (dst[s] == -1) return;
        ans.push_back(dst[s]);
        if (head[s] != nullNd)
            Q.push(node(head[s], dst[s] + head[s]->edge->d));
        for (int _ = 1; _ < k and not Q.empty(); _++) {
            node p = Q.top(), q;
            Q.pop();
            ans.push_back(p.d);
            if (head[p.H->edge->v] != nullNd) {
                q.H = head[p.H->edge->v];
                q.d = p.d + q.H->edge->d;
                Q.push(q);
            }
            for (int i = 0; i < 4; i++)
                if (p.H->chd[i] != nullNd) {
                    q.H = p.H->chd[i];
                    q.d = p.d - p.H->edge->d + p.H->chd[i]->edge->d;
                    Q.push(q);
                }
        }
    }
    void solve() {  // ans[i] stores the i-th shortest path
        dijkstra();
        build();
        first_K();  // ans.size() might less than k
    }
} solver;
```

5.12 System of Difference Constraints

```cpp
vector<vector<pair<int, ll>>> G;
void add(int u, int v, ll w) {
    G[u].emplace_back(make_pair(v, w));
}
```

- $x_u - x_v \le c \Rightarrow$ `add(v, u, c)`
- $x_u - x_v \ge c \Rightarrow$ `add(u, v, -c)`
- $x_u - x_v = c \Rightarrow$ `add(v, u, c), add(u, v -c)`
- $x_u \ge c \Rightarrow$ add super vertex $x_0 = 0$, then $x_u - x_0 \ge c \Rightarrow$ `add(u, 0, -c)`
- Don't for get non-negative constraints for every variable if specified implicitly.
- Interval sum $\Rightarrow$ Use prefix sum to transform into differential constraints. Don't for get $S_{i+1} - S_i \ge 0$ if x_i needs to be non-negative.
- $\frac{x_u}{x_v} \le c \Rightarrow \log x_u - \log x_v \le \log c$

6 String

6.1 Aho Corasick

```cpp
struct ACautomata {
    struct Node {
        int cnt; // 停在此節點的數量
        Node *go[26], *fail, *dic;
        // 子節點 fail指標 最近的模式結尾
        Node() {
            cnt = 0;
            fail = 0;
            dic = 0;
            memset(go, 0, sizeof(go));
        }
    } pool[1048576], *root;
    int nMem;
    Node *new_Node() {
        pool[nMem] = Node();
        return &pool[nMem++];
    }
    void init() {
        nMem = 0;
        root = new_Node();
    }
    void add(const string &str) { insert(root, str, 0); }
    void insert(Node *cur, const string &str, int pos) {
        for (int i = pos; i < str.size(); i++) {
            if (!cur->go[str[i] - 'a'])
                cur->go[str[i] - 'a'] = new_Node();
            cur = cur->go[str[i] - 'a'];
        }
        cur->cnt++;
    }
    void make_fail() { // 全部 add 完做
        queue<Node *> que;
        que.push(root);
        while (!que.empty()) {
            Node *fr = que.front();
            que.pop();
            for (int i = 0; i < 26; i++) {
                if (fr->go[i]) {
                    Node *ptr = fr->fail;
                    while (ptr && !ptr->go[i]) ptr = ptr->fail;
                    fr->go[i]->fail = ptr = (ptr ? ptr->go[i] : root);
                    fr->go[i]->dic = (ptr->cnt ? ptr : ptr->dic);
                    que.push(fr->go[i]);
                }
            }
        }
    }
    // 出現過不同string的總數
    int query_unique(const string& text) {
        Node* p = root;
        int ans = 0;
        for(char ch : text) {
            int i = ch - 'a';
            while(p && !p->go[i]) p = p ->fail;
            p = p ? p->go[i] : root;
            if(p->cnt) {ans += p->cnt, p->cnt = 0;}
            for(Node* t = p->dic; t; t = t->dic) if(t->cnt) {
                ans += t->cnt; t->cnt = 0;
            }
        }
        return ans;
    }
} AC;
```

6.2 KMP

```cpp
vector<int> f;
// 沒匹配到可以退回哪裡
void buildFailFunction(string &s) {
    f.resize(s.size(), -1);
    for (int i = 1; i < s.size(); i++) {
        int now = f[i - 1];
        while (now != -1 and s[now + 1] != s[i]) now = f[now];
        if (s[now + 1] == s[i]) f[i] = now + 1;
    }
}

void KMPmatching(string &a, string &b) {
    for (int i = 0, now = -1; i < a.size(); i++) {
        while (a[i] != b[now + 1] and now != -1) now = f[now];
        if (a[i] == b[now + 1]) now++;
        if (now + 1 == b.size()) {
            cout << "found a match start at position " << i - now << endl;
            now = f[now];
        }
    }
}
```

6.3 Z Value

```cpp
string is, it, s;
// is: 被搜尋 it: 要找的
int n;
vector<int> z;
// 計算每個位置 i 開始的字串，和 s 的共農前綴長度
void init() {
    cin >> is >> it;
    s = it + '0' + is;
    n = (int)s.size();
    z.resize(n, 0);
}
void solve() {
    int ans = 0;
    z[0] = n;
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r) z[i] = min(z[i - l], r - i + 1);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
        if (z[i] == (int)it.size()) ans++;
    }
    cout << ans << endl;
}
```

6.4 Manacher

```cpp
// 找最長回文
int n;
string S, s;
vector<int> m;
void manacher() {
    s.clear();
    s.resize(2 * n + 1, '.');
    for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
    m.clear();
    m.resize(2 * n + 1, 0);
    // m[i] := max k such that s[i-k, i+k] is palindrome
    int mx = 0, mxk = 0;
    for (int i = 1; i < 2 * n + 1; i++) {
        if (mx - (i - mx) >= 0) m[i] = min(m[mx - (i - mx)], mx + mxk - i);
        while (0 <= i - m[i] - 1 && i + m[i] + 1 < 2 * n + 1 &&
               s[i - m[i] - 1] == s[i + m[i] + 1]) m[i]++;
        if (i + m[i] > mx + mxk) mx = i, mxk = m[i];
    }
}
void init() {
    cin >> S;
    n = (int)S.size();
}
void solve() {
    manacher();
    int mx = 0, ptr = 0;
    for (int i = 0; i < 2 * n + 1; i++)
        if (mx < m[i]) {
            mx = m[i];
            ptr = i;
        }
    for (int i = ptr - mx; i <= ptr + mx; i++)
        if (s[i] != '.') cout << s[i];
    cout << endl;
}
```

6.5 Suffix Array

```cpp
#define F first
#define S second
struct SuffixArray {  // don't forget s += "$";
    int n;
    string s;
    vector<int> suf, lcp, rk;
    // 後綴陣列：suf[i] = 第 i 小的後綴起點
    // LCP 陣列：lcp[i] = suf[i] 與 suf[i-1] 的最長共同前綴長度
    // rank 陣列：rk[i] = 起點在 i 的後綴的名次
    vector<int> cnt, pos;
    vector<pair<pair<int, int>, int> > buc[2];
    void init(string _s) {
        s = _s;
        n = (int)s.size();
        // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
        suf.assign(n, 0);
        rk.assign(n, 0);
        lcp.assign(n, 0);
        cnt.assign(n, 0);
        pos.assign(n, 0);
        buc[0].assign(n, {{0,0},0});
        buc[1].assign(n, {{0,0},0});
    }
    void radix_sort() {
        for (int t : {0, 1}) {
            fill(cnt.begin(), cnt.end(), 0);
            for (auto& i : buc[t]) cnt[(t ? i.F.F : i.F.S)]++;
            for (int i = 0; i < n; i++)
                pos[i] = (!i ? 0 : pos[i - 1] + cnt[i - 1]);
            for (auto& i : buc[t])
                buc[t ^ 1][pos[(t ? i.F.F : i.F.S)]++] = i;
        }
    }
    bool fill_suf() {
        bool end = true;
        for (int i = 0; i < n; i++) suf[i] = buc[0][i].S;
        rk[suf[0]] = 0;
        for (int i = 1; i < n; i++) {
            int dif = (buc[0][i].F != buc[0][i - 1].F);
            end &= dif;
            rk[suf[i]] = rk[suf[i - 1]] + dif;
        }
        return end;
    }
    void sa() {
        for (int i = 0; i < n; i++)
            buc[0][i] = make_pair(make_pair(s[i], s[i]), i);
        sort(buc[0].begin(), buc[0].end());
        if (fill_suf()) return;
        for (int k = 0; (1 << k) < n; k++) {
            for (int i = 0; i < n; i++)
                buc[0][i] = make_pair(make_pair(rk[i], rk[(i + (1 << k)) % n]), i);
            radix_sort();
            if (fill_suf()) return;
        }
    }
    void LCP() {
        int k = 0;
        for (int i = 0; i < n - 1; i++) {
            if (rk[i] == 0) continue;
            int pi = rk[i];
            int j = suf[pi - 1];
            while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
            lcp[pi] = k;
            k = max(k - 1, 0);
        }
    }
};
SuffixArray suffixarray;
```

6.6 Suffix Automaton

```
struct SAM {
    struct State {
        int next[26];
        int link, len;
        // suffix link，指向最長真後綴所對應的狀態
        // 該狀態代表的字串集合中的最長字串長度
        State() : link(-1), len(0) { memset(next, -1,
            sizeof next); }
    };
    vector<State> st;
    int last;
    vector<long long> occ; // 每個狀態的出現次數 (
        endpos 個數)
    vector<int> first_bkpos; // 出現在哪裡
    SAM(int maxlen = 0) {
        st.reserve(2 * maxlen + 5); st.push_back(State
            ()); last = 0;
        occ.reserve(2 * maxlen + 5); occ.push_back(0);
        first_bkpos.push_back(-1);
    }
    void extend(int c) {
        int cur = (int)st.size();
        st.push_back(State());
        occ.push_back(0);
        first_bkpos.push_back(0);
        st[cur].len = st[last].len + 1;
        first_bkpos[cur] = st[cur].len - 1;
        int p = last;
        while (p != -1 && st[p].next[c] == -1) {
            st[p].next[c] = cur;
            p = st[p].link;
        }
        if (p == -1) {
            st[cur].link = 0;
        } else {
            int q = st[p].next[c];
            if (st[p].len + 1 == st[q].len) {
                st[cur].link = q;
            } else {
                int clone = (int)st.size();
                st.push_back(st[q]);
                first_bkpos.push_back(first_bkpos[q]);
                occ.push_back(0);
                st[clone].len = st[p].len + 1;
                while (p != -1 && st[p].next[c] == q) {
                    st[p].next[c] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
        last = cur;
        occ[cur] += 1;
    }
    void finalize_occ() {
        int m = (int)st.size();
        vector<int> order(m);
        iota(order.begin(), order.end(), 0);
        sort(order.begin(), order.end(), [&](int a, int
             b){ return st[a].len > st[b].len; });
        for (int v : order) {
            int p = st[v].link;
            if (p != -1) occ[p] += occ[v];
        }
    }
};
```

6.7 Minimum Rotation

```
// rotate(begin(s), begin(s)+minRotation(s), end(s))
// 找出字串的最小字典序旋轉
int minRotation(string s) {
    int a = 0, n = s.size();
    s += s;
    for (int b = 0; b < n; b++)
        for (int k = 0; k < n; k++) {
            if (a + k == b || s[a + k] < s[b + k]) {
                b += max(0, k - 1);
                break;
            }
            if (s[a + k] > s[b + k]) {
                a = b;
                break;
            }
        }
    return a;
}
```

6.8 Lyndon Factorization

```
// Duval: 將字串唯一分解為字典序非遞增的 Lyndon 子字串
vector<string> duval(string const& s) {
    int n = s.size();
    int i = 0;
    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k))
                ;
            i += j - k;
        }
    }
    return factorization;  // O(n)
}
```

6.9 Rolling Hash

```
const ll C = 27;
inline int id(char c) { return c - 'a' + 1; }
struct RollingHash {
    string s;
    int n;
    ll mod;
    vector<ll> Cexp, hs;
    RollingHash(string& _s, ll _mod) : s(_s), n((int)_s
        .size()), mod(_mod) {
        Cexp.assign(n, 0);
        hs.assign(n, 0);
        Cexp[0] = 1;
        for (int i = 1; i < n; i++) {
            Cexp[i] = Cexp[i - 1] * C;
            if (Cexp[i] >= mod) Cexp[i] %= mod;
        }
        hs[0] = id(s[0]);
        for (int i = 1; i < n; i++) {
            hs[i] = hs[i - 1] * C + id(s[i]);
            if (hs[i] >= mod) hs[i] %= mod;
        }
    }
    inline ll query(int l, int r) {
        ll res = hs[r] - (l ? hs[l - 1] * Cexp[r - l +
            1] : 0);
        res = (res % mod + mod) % mod;
        return res;
    }
};
```

6.10 Trie

```
pii a[N][26];

void build(string &s) {
    static int idx = 0;
    int n = s.size();
    for (int i = 0, v = 0; i < n; i++) {
```

```cpp
        pii &now = a[v][s[i] - 'a'];
        if (now.first != -1)
            v = now.first;
        else
            v = now.first = ++idx;
        if (i == n - 1)
            now.second++;
    }
}
```

7 Geometry

7.1 Basic Operations

```cpp
// typedef long long T;
typedef long double T;
const long double eps = 1e-12;

short sgn(T x) {
    if (abs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}

struct Pt {
    T x, y;
    Pt(T _x = 0, T _y = 0) : x(_x), y(_y) {}
    Pt operator+(Pt a) { return Pt(x + a.x, y + a.y); }
    Pt operator-(Pt a) { return Pt(x - a.x, y - a.y); }
    Pt operator*(T a) { return Pt(x * a, y * a); }
    Pt operator/(T a) { return Pt(x / a, y / a); }
    T operator*(Pt a) { return x * a.x + y * a.y; }
    T operator^(Pt a) { return x * a.y - y * a.x; }
    bool operator<(Pt a) { return x < a.x || (x == a.x
         && y < a.y); }
    // return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn
         (y-a.y) < 0); }
    bool operator==(Pt a) { return sgn(x - a.x) == 0 &&
          sgn(y - a.y) == 0; }
};

Pt mv(Pt a, Pt b) { return b - a; }
T len2(Pt a) { return a * a; }
T dis2(Pt a, Pt b) { return len2(b - a); }
Pt rotate(Pt u) { return {-u.y, u.x}; }
Pt unit(Pt x) { return x / sqrtl(x * x); }
short ori(Pt a, Pt b) { return ((a ^ b) > 0) - ((a ^ b)
      < 0); }
bool onseg(Pt p, Pt l1, Pt l2) {
    Pt a = mv(p, l1), b = mv(p, l2);
    return ((a ^ b) == 0) && ((a * b) <= 0);
}
inline T cross(const Pt &a, const Pt &b, const Pt &c) {
    return (b.x - a.x) * (c.y - a.y)
        - (b.y - a.y) * (c.x - a.x);
}

long double polar_angle(Pt ori, Pt pt){
    return atan2(pt.y - ori.y, pt.x - ori.x);
}
// slope to degree atan(Slope) * 180.0 / acos(-1.0);
bool argcmp(Pt u, Pt v) {
    auto half = [](const Pt& p) {
        return p.y > 0 || (p.y == 0 && p.x >= 0);
    };
    if (half(u) != half(v)) return half(u) < half(v);
    return sgn(u ^ v) > 0;
}
int ori(Pt& o, Pt& a, Pt& b) {
    return sgn((a - o) ^ (b - o));
}
struct Line {
    Pt a, b;
    Pt dir() { return b - a; }
};
int PtSide(Pt p, Line L) {
    return sgn(ori(L.a, L.b, p)); // for int
    return sgn(ori(L.a, L.b, p) / sqrt(len2(L.a - L.b))
        );
}
bool PtOnSeg(Pt p, Line L) {
    return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L
        .b)) <= 0;
}
Pt proj(Pt& p, Line& l) {
    Pt d = l.b - l.a;
    T d2 = len2(d);
    if (sgn(d2) == 0) return l.a;
    T t = ((p - l.a) * d) / d2;
    return l.a + d * t;
}
struct Cir {
    Pt o;
    T r;
};
bool disjunct(Cir a, Cir b) {
    return sgn(sqrtl(len2(a.o - b.o)) - a.r - b.r) >=
        0;
}
bool contain(Cir a, Cir b) {
    return sgn(a.r - b.r - sqrtl(len2(a.o - b.o))) >=
        0;
}
```

7.2 Sort by Angle

```cpp
int ud(Pt a) {  // up or down half plane
    if (a.y > 0) return 0;
    if (a.y < 0) return 1;
    return (a.x >= 0 ? 0 : 1);
}
sort(pts.begin(), pts.end(), [&](const Pt& a, const Pt&
     b) {
    if (ud(a) != ud(b)) return ud(a) < ud(b);
    return (a ^ b) > 0;
});
```

7.3 Intersection

```cpp
bool line_intersect_check(Pt p1, Pt p2, Pt q1, Pt q2) {
    if (onseg(p1, q1, q2) || onseg(p2, q1, q2) || onseg
        (q1, p1, p2) || onseg(q2, p1, p2)) return true;
    Pt p = mv(p1, p2), q = mv(q1, q2);
    return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) <
        0) && (ori(q, mv(q1, p1)) * ori(q, mv(q1, p2))
        < 0);
}
// long double
Pt line_intersect(Pt a1, Pt a2, Pt b1, Pt b2) {
    Pt da = mv(a1, a2), db = mv(b1, b2);
    T det = da ^ db;
    if (sgn(det) == 0) { // parallel
        // return Pt(NAN, NAN);
    }
    T t = ((b1 - a1) ^ db) / det;
    return a1 + da * t;
}
vector<Pt> CircleInter(Cir a, Cir b) {
    double d2 = len2(a.o - b.o), d = sqrt(d2);
    if (d < max(a.r, b.r) - min(a.r, b.r) || d > a.r +
        b.r) return {};
    Pt u = (a.o + b.o) / 2 + (a.o - b.o) * ((b.r * b.r
        - a.r * a.r) / (2 * d2));
    double A = sqrt((a.r + b.r + d) * (a.r - b.r + d) *
         (a.r + b.r - d) * (-a.r + b.r + d));
    Pt v = rotate(b.o - a.o) * A / (2 * d2);
    if (sgn(v.x) == 0 and sgn(v.y) == 0) return {u};
    return {u - v, u + v}; // counter clockwise of a
}
vector<Pt> CircleLineInter(Cir c, Line l) {
    Pt H = proj(c.o, l);
    Pt dir = unit(l.b - l.a);
    T h = sqrtl(len2(H - c.o));
    if (sgn(h - c.r) > 0) return {};
    T d = sqrtl(max((T)0, c.r * c.r - h * h));
    if (sgn(d) == 0) return {H};
    return {H - dir * d, H + dir * d};
}
```

7.4 Polygon Area

```cpp
// 2 * area
```

```cpp
T dbPoly_area(vector<Pt>& e) {
    T res = 0;
    int sz = e.size();
    for (int i = 0; i < sz; i++) {
        res += e[i] ^ e[(i + 1) % sz];
    }
    return abs(res);
}
```

7.5 Convex Hull

```cpp
vector<Pt> convexHull(vector<Pt> pts) {
    vector<Pt> hull;
    sort(pts.begin(), pts.end());
    for (int i = 0; i < 2; i++) {
        int b = hull.size();
        for (auto ei : pts) {
            while (hull.size() - b >= 2 && ori(mv(hull[
                hull.size() - 2], hull.back()), mv(hull
                [hull.size() - 2], ei)) == -1) {
                hull.pop_back();
            }
            hull.emplace_back(ei);
        }
        hull.pop_back();
        reverse(pts.begin(), pts.end());
    }
    return hull;
}
```

7.6 Point In Convex

```cpp
bool point_in_convex(const vector<Pt> &C, Pt p, bool
    strict = true) {
    // only works when no three point are collinear
    int n = C.size();
    int a = 1, b = n - 1, r = !strict;
    if (n == 0) return false;
    if (n < 3) return r && onseg(p, C[0], C.back());
    if (ori(mv(C[0], C[a]), mv(C[0], C[b])) > 0) swap(a
        , b);
    if (ori(mv(C[0], C[a]), mv(C[0], p)) >= r || ori(mv
        (C[0], C[b]), mv(C[0], p)) <= -r) return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        if (ori(mv(C[0], C[c]), mv(C[0], p)) > 0) b = c
            ;
        else a = c;
    }
    return ori(mv(C[a], C[b]), mv(C[a], p)) < r;
}
```

7.7 Point Segment Distance

```cpp
double point_segment_dist(Pt q0, Pt q1, Pt p) {
    if (q0 == q1) {
        double dx = double(p.x - q0.x);
        double dy = double(p.y - q0.y);
        return sqrt(dx * dx + dy * dy);
    }
    T d1 = (q1 - q0) * (p - q0);
    T d2 = (q0 - q1) * (p - q1);
    if (d1 >= 0 && d2 >= 0) {
        double area = fabs(double((q1 - q0) ^ (p - q0))
            );
        double base = sqrt(double(dis2(q0, q1)));
        return area / base;
    }
    double dx0 = double(p.x - q0.x), dy0 = double(p.y -
         q0.y);
    double dx1 = double(p.x - q1.x), dy1 = double(p.y -
         q1.y);
    return min(sqrt(dx0 * dx0 + dy0 * dy0), sqrt(dx1 *
        dx1 + dy1 * dy1));
}
```

7.8 Point in Polygon

```cpp
short inPoly(vector<Pt>& pts, Pt p) {
    // 0=Bound 1=In -1=Out
    int n = pts.size();
    for (int i = 0; i < pts.size(); i++) if (onseg(p,
        pts[i], pts[(i + 1) % n])) return 0;
    int cnt = 0;
    for (int i = 0; i < pts.size(); i++) if (
        line_intersect_check(p, Pt(p.x + 1, p.y + 2e9),
         pts[i], pts[(i + 1) % n])) cnt ^= 1;
    return (cnt ? 1 : -1);
}
```

7.9 Minimum Euclidean Distance

```cpp
long long Min_Euclidean_Dist(vector<Pt> &pts) {
    sort(pts.begin(), pts.end());
    set<pair<long long, long long>> s;
    s.insert({pts[0].y, pts[0].x});
    long long l = 0, best = LLONG_MAX;
    for (int i = 1; i < (int)pts.size(); i++) {
        Pt now = pts[i];
        long long lim = (long long)ceil(sqrtl((long
            double)best));
        while (now.x - pts[l].x > lim) {
            s.erase({pts[l].y, pts[l].x}); l++;
}
        auto low = s.lower_bound({now.y - lim,
            LLONG_MIN});
        auto high = s.upper_bound({now.y + lim,
            LLONG_MAX});
        for (auto it = low; it != high; it++) {
            long long dy = it->first - now.y;
            long long dx = it->second - now.x;
            best = min(best, dx * dx + dy * dy);
        }
        s.insert({now.y, now.x});
    }
    return best;
}
```

7.10 Minkowski Sum

```cpp
void reorder(vector <Pt> &P) {
  rotate(P.begin(), min_element(P.begin(), P.end(),
      [&](Pt a, Pt b) { return make_pair(a.y, a.x) <
      make_pair(b.y, b.x); }), P.end());
}
vector <Pt> Minkowski(vector <Pt> P, vector <Pt> Q) {
  // P, Q: convex polygon
  reorder(P), reorder(Q);
  int n = P.size(), m = Q.size();
  P.push_back(P[0]), P.push_back(P[1]), Q.push_back(Q
      [0]), Q.push_back(Q[1]);
  vector <Pt> ans;
  for (int i = 0, j = 0; i < n || j < m; ) {
    ans.push_back(P[i] + Q[j]);
    auto val = (P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]);
    if (val >= 0) i++;
    if (val <= 0) j++;
  }
  return ans;
}
```

7.11 Lower Concave Hull

```cpp
struct Line {
  mutable ll m, b, p;
  bool operator<(const Line& o) const { return m < o.m;
       }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
    else x->p = div(y->b - x->b, x->m - y->m);
    return x->p >= y->p;
  }
  void add(ll m, ll b) {
    auto z = insert({m, b, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.m * x + l.b;
  }
};
```

7.12 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let i = number of points inside the polygon.

Let b = number of points on the boundary of the polygon.

Then we have the following formula:

$$Area = i + \frac{b}{2} - 1$$

7.13 Rotating SweepLine

```cpp
double cross(const Pt &a, const Pt &b) {
    return a.x*b.y - a.y*b.x;
}
int rotatingCalipers(const vector<Pt>& hull) {
    int m = hull.size();
    if (m < 2) return 0;
    int j = 1;
    T maxd = 0;
    for (int i = 0; i < m; ++i) {
        int ni = (i + 1) % m;
        while (abs(cross({hull[ni].x - hull[i].x, hull[ni].y - hull[i].y}, {hull[(j+1)%m].x - hull[i].x, hull[(j+1)%m].y - hull[i].y})) > abs(cross({hull[ni].x - hull[i].x, hull[ni].y - hull[i].y}, {hull[j].x - hull[i].x, hull[j].y - hull[i].y}))) {
            j = (j + 1) % m;
        }
        maxd = max(maxd, dis2(hull[i], hull[j]));
        maxd = max(maxd, dis2(hull[ni], hull[j]));
    }
    return maxd; // TODO
}
```

7.14 Half Plane Intersection

```cpp
bool cover(Line& L, Line& P, Line& Q) {
    long double u = (Q.a - P.a) ^ Q.dir();
    long double v = P.dir() ^ Q.dir();
    long double x = P.dir().x * u + (P.a - L.a).x * v;
    long double y = P.dir().y * u + (P.a - L.a).y * v;
    return sgn(x * L.dir().y - y * L.dir().x) * sgn(v) >= 0;
}
vector<Line> HPI(vector<Line> P) {
    sort(P.begin(), P.end(), [&](Line& l, Line& m) {
        if (argcmp(l.dir(), m.dir())) return true;
        if (argcmp(m.dir(), l.dir())) return false;
        return ori(m.a, m.b, l.a) > 0;
    });

    int l = 0, r = -1;
    for (size_t i = 0; i < P.size(); ++i) {
        if (i && !argcmp(P[i - 1].dir(), P[i].dir())) continue;
        while (l < r && cover(P[i], P[r - 1], P[r])) --r;
        while (l < r && cover(P[i], P[l], P[l + 1])) ++l;
        P[++r] = P[i];
    }
    while (l < r && cover(P[l], P[r - 1], P[r])) --r;
    while (l < r && cover(P[r], P[l], P[l + 1])) ++l;

    if (r - l <= 1 || !argcmp(P[l].dir(), P[r].dir())) return {};
    if (cover(P[l + 1], P[l], P[r])) return {};

    return vector<Line>(P.begin() + l, P.begin() + r + 1);
}
```

7.15 Minimum Enclosing Circle

```cpp
const int INF = 1e9;
Pt circumcenter(Pt A, Pt B, Pt C) {
    // a1(x-A.x) + b1(y-A.y) = c1
    // a2(x-A.x) + b2(y-A.y) = c2
    // solve using Cramer's rule
    T a1 = B.x - A.x, b1 = B.y - A.y, c1 = dis2(A, B) / 2.0;
    T a2 = C.x - A.x, b2 = C.y - A.y, c2 = dis2(A, C) / 2.0;
    T D = Pt(a1, b1) ^ Pt(a2, b2);
    T Dx = Pt(c1, b1) ^ Pt(c2, b2);
    T Dy = Pt(a1, c1) ^ Pt(a2, c2);
    if (D == 0) return Pt(-INF, -INF);
    return A + Pt(Dx / D, Dy / D);
}
Pt center;
T r2;
void minEncloseCircle(vector<Pt> pts) {
    mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
    shuffle(pts.begin(), pts.end(), gen);
    center = pts[0], r2 = 0;

    for (int i = 0; i < pts.size(); i++) {
        if (dis2(center, pts[i]) <= r2) continue;
        center = pts[i], r2 = 0;
        for (int j = 0; j < i; j++) {
            if (dis2(center, pts[j]) <= r2) continue;
            center = (pts[i] + pts[j]) / 2.0;
            r2 = dis2(center, pts[i]);
            for (int k = 0; k < j; k++) {
                if (dis2(center, pts[k]) <= r2) continue;
                center = circumcenter(pts[i], pts[j], pts[k]);
                r2 = dis2(center, pts[i]);
            }
        }
    }
}
```

7.16 Union of Circles

```cpp
// Area[i] : area covered by at least i circle
vector<T> CircleUnion(const vector<Cir> &C) {
    const int n = C.size();
    vector<T> Area(n + 1);
    auto check = [&](int i, int j) {
        if (!contain(C[i], C[j]))
            return false;
        return sgn(C[i].r - C[j].r) > 0 or (sgn(C[i].r - C[j].r) == 0 and i < j);
    };
    struct Teve {
        double ang; int add; Pt p;
        bool operator<(const Teve &b) { return ang < b.ang; }
    };
    auto ang = [&](Pt p) { return atan2(p.y, p.x); };
    for (int i = 0; i < n; i++) {
        int cov = 1;
        vector<Teve> event;
        for (int j = 0; j < n; j++) if (i != j) {
            if (check(j, i)) cov++;
            else if (!check(i, j) and !disjunct(C[i], C[j])) {
                auto I = CircleInter(C[i], C[j]);
                assert(I.size() == 2);
                double a1 = ang(I[0] - C[i].o), a2 = ang(I[1] - C[i].o);
                event.push_back({a1, 1, I[0]});
                event.push_back({a2, -1, I[1]});
                if (a1 > a2) cov++;
            }
        }
        if (event.empty()) {
            Area[cov] += acos(-1) * C[i].r * C[i].r;
            continue;
        }
        sort(event.begin(), event.end());
        event.push_back(event[0]);
        for (int j = 0; j + 1 < event.size(); j++) {
            cov += event[j].add;
            Area[cov] += (event[j].p ^ event[j + 1].p) / 2.;
            double theta = event[j + 1].ang - event[j].ang;
            if (theta < 0) theta += 2 * acos(-1);
            Area[cov] += (theta - sin(theta)) * C[i].r * C[i].r / 2.;
        }
    }
    return Area;
}
```

7.17 Area Of Circle Polygon

```
double AreaOfCirclePoly(Cir C, vector<Pt> &P) {
    auto arg = [&](Pt p, Pt q) { return atan2l(p ^ q, p * q); };
    double r2 = (double)(C.r * C.r / 2);
    auto tri = [&](Pt p, Pt q) {
        Pt d = q - p;
        T a = (d * p) / (d * d);
        T b = ((p * p) - C.r * C.r) / (d * d);
        T det = a * a - b;
        if (det <= 0) return (double)(arg(p, q) * r2);
        T s = max((T)0.0L, -a - sqrtl(det));
        T t = min((T)1.0L, -a + sqrtl(det));
        if (t < 0 || 1 <= s) return (double)(arg(p, q) * r2);
        Pt u = p + d * s, v = p + d * t;
        return (double)(arg(p, u) * r2 + (u ^ v) / 2 + arg(v, q) * r2);
    };
    long double sum = 0.0L;
    for (int i = 0; i < (int)P.size(); i++)
        sum += tri(P[i] - C.o, P[(i + 1) % P.size()] - C.o);
    return (double)fabsl(sum);
}
```

7.18 3D Point

```
struct Pt {
  double x, y, z;
  Pt(double _x = 0, double _y = 0, double _z = 0): x(_x), y(_y), z(_z){}
  Pt operator + (const Pt &o) const
  { return Pt(x + o.x, y + o.y, z + o.z); }
  Pt operator - (const Pt &o) const
  { return Pt(x - o.x, y - o.y, z - o.z); }
  Pt operator * (const double &k) const
  { return Pt(x * k, y * k, z * k); }
  Pt operator / (const double &k) const
  { return Pt(x / k, y / k, z / k); }
  double operator * (const Pt &o) const
  { return x * o.x + y * o.y + z * o.z; }
  Pt operator ^ (const Pt &o) const
  { return {Pt(y * o.z - z * o.y, z * o.x - x * o.z, x * o.y - y * o.x)}; }
};
double abs2(Pt o) { return o * o; }
double abs(Pt o) { return sqrt(abs2(o)); }
Pt cross3(Pt a, Pt b, Pt c)
{ return (b - a) ^ (c - a); }
double area(Pt a, Pt b, Pt c)
{ return abs(cross3(a, b, c)); }
double volume(Pt a, Pt b, Pt c, Pt d)
{ return cross3(a, b, c) * (d - a); }
bool coplaner(Pt a, Pt b, Pt c, Pt d)
{ return sign(volume(a, b, c, d)) == 0; }
Pt proj(Pt o, Pt a, Pt b, Pt c) // o proj to plane abc
{ Pt n = cross3(a, b, c);
  return o - n * ((o - a) * (n / abs2(n)));}
Pt line_plane_intersect(Pt u, Pt v, Pt a, Pt b, Pt c) {
  // intersection of line uv and plane abc
  Pt n = cross3(a, b, c);
  double s = n * (u - v);
  if (sign(s) == 0) return {-1, -1, -1}; // not found
  return v + (u - v) * ((n * (a - v)) / s); }
Pt rotateAroundAxis(Pt v, Pt axis, double theta) {
    axis = axis / abs(axis); // axis must be unit vector
    double cosT = cos(theta);
    double sinT = sin(theta);
    Pt term1 = v * cosT;
    Pt term2 = (axis ^ v) * sinT;
    Pt term3 = axis * ((axis * v) * (1 - cosT));
    return term1 + term2 + term3;
}
```

8 Number Theory

8.1 FFT

```
typedef complex<double> cp;

const double pi = acos(-1);
const int NN = 131072;

struct FastFourierTransform {
    /*
            Iterative Fast Fourier Transform
            How this works? Look at this
            0th recursion 0(000)   1(001)   2(010)   3(011)   4(100)   5(101)   6(110)   7(111)
            1th recursion 0(000)   2(010)   4(100)   6(110) | 1(011)   3(011)   5(101)   7(111)
            2th recursion 0(000)   4(100) | 2(010)   6(110) | 1(011)   5(101) | 3(011)   7(111)
            3th recursion 0(000) | 4(100) | 2(010) | 6(110) | 1(011) | 5(101) | 3(011) | 7(111)
            All the bits are reversed => We can save the reverse of the numbers in an array!
    */
    int n, rev[NN];
    cp omega[NN], iomega[NN];
    void init(int n_) {
        n = n_;
        for (int i = 0; i < n_; i++) {
            // Calculate the nth roots of unity
            omega[i] = cp(cos(2 * pi * i / n_), sin(2 * pi * i / n_));
            iomega[i] = conj(omega[i]);
        }
        int k = __lg(n_);
        for (int i = 0; i < n_; i++) {
            int t = 0;
            for (int j = 0; j < k; j++) {
                if (i & (1 << j)) t |= (1 << (k - j - 1));
            }
            rev[i] = t;
        }
    }

    void transform(vector<cp> &a, cp *xomega) {
        for (int i = 0; i < n; i++)
            if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int len = 2; len <= n; len <<= 1) {
            int mid = len >> 1;
            int r = n / len;
            for (int j = 0; j < n; j += len)
                for (int i = 0; i < mid; i++) {
                    cp tmp = xomega[r * i] * a[j + mid + i];
                    a[j + mid + i] = a[j + i] - tmp;
                    a[j + i] = a[j + i] + tmp;
                }
        }
    }

    void fft(vector<cp> &a) { transform(a, omega); }
    void ifft(vector<cp> &a) {
        transform(a, iomega);
        for (int i = 0; i < n; i++) a[i] /= n;
    }
} FFT;

const int MAXN = 262144;
// (must be 2^k)
// 262144, 524288, 1048576, 2097152, 4194304
// before any usage, run pre_fft() first
typedef long double ld;
typedef complex<ld> cplx;  // real() ,imag()
const ld PI = acosl(-1);
const cplx I(0, 1);
cplx omega[MAXN + 1];
void pre_fft() {
    for (int i = 0; i <= MAXN; i++) {
        omega[i] = exp(i * 2 * PI / MAXN * I);
    }
}
// n must be 2^k
void fft(int n, cplx a[], bool inv = false) {
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
        int mh = m >> 1;
        for (int i = 0; i < mh; i++) {
            cplx w = omega[inv ? MAXN - (i * theta % MAXN) : i * theta % MAXN];
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                cplx x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
        }
        theta = (theta * 2) % MAXN;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++) {
        for (int k = n >> 1; k > (i ^= k); k >>= 1);
        if (j < i) swap(a[i], a[j]);
    }
    if (inv) {
        for (i = 0; i < n; i++) a[i] /= n;
    }
}
cplx arr[MAXN + 1];
inline void mul(int _n, long long a[], int _m, long long b[], long long ans[]) {
    int n = 1, sum = _n + _m - 1;
    while (n < sum) n <<= 1;
    for (int i = 0; i < n; i++) {
        double x = (i < _n ? a[i] : 0), y = (i < _m ? b[i] : 0);
        arr[i] = complex<double>(x + y, x - y);
    }
    fft(n, arr);
    for (int i = 0; i < n; i++) arr[i] = arr[i] * arr[i];
    fft(n, arr, true);
    for (int i = 0; i < sum; i++) ans[i] = (long long int)(arr[i].real() / 4 + 0.5);
}

long long a[MAXN];
long long b[MAXN];
long long ans[MAXN];
int a_length;
int b_length;
```

8.2 Pollard's rho

```
ll add(ll x, ll y, ll p) {
    return (x + y) % p;
}
ll qMul(ll x, ll y, ll mod) {
    ll ret = x * y - (ll)((long double)x / mod * y) * mod;
    return ret < 0 ? ret + mod : ret;
}
ll f(ll x, ll mod) { return add(qMul(x, x, mod), 1, mod); }
ll pollard_rho(ll n) {
    if (!(n & 1)) return 2;
    while (true) {
        ll y = 2, x = rand() % (n - 1) + 1, res = 1;
        for (int sz = 2; res == 1; sz *= 2) {
            for (int i = 0; i < sz && res <= 1; i++) {
                x = f(x, n);
                res = __gcd(llabs(x - y), n);
            }
            y = x;
        }
        if (res != 0 && res != n) return res;
    }
}
vector<ll> ret;
void fact(ll x) {
    if (miller_rabin(x)) {
        ret.push_back(x);
        return;
    }
    ll f = pollard_rho(x);
    fact(f);
    fact(x / f);
}
```

8.3 Miller Rabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool witness(ll a, ll n, ll u, int t) {
    if (!(a %= n)) return 0;
    ll x = mypow(a, u, n);
    for (int i = 0; i < t; i++) {
        ll nx = mul(x, x, n);
        if (nx == 1 && x != 1 && x != n - 1) return 1;
        x = nx;
    }
    return x != 1;
}
bool miller_rabin(ll n, int s = 100) {
    // iterate s times of witness on n
    // return 1 if prime, 0 otherwise
    if (n < 2) return 0;
    if (!(n & 1)) return n == 2;
    ll u = n - 1;
    int t = 0;
    while (!(u & 1)) u >>= 1, t++;
    while (s--) {
        ll a = randll() % (n - 1) + 1;
        if (witness(a, n, u, t)) return 0;
    }
    return 1;
}
```

8.4 Fast Power

Note: $a^n \equiv a^{(n\ mod\ (p-1))} (mod\ p)$

8.5 Extend GCD

```
ll GCD;
pll extgcd(ll a, ll b) {
    if (b == 0) {
        GCD = a;
        return pll{1, 0};
    }
    pll ans = extgcd(b, a % b);
    return pll{ans.S, ans.F - a / b * ans.S};
}
pll bezout(ll a, ll b, ll c) {
    bool negx = (a < 0), negy = (b < 0);
    pll ans = extgcd(abs(a), abs(b));
    if (c % GCD != 0) return pll{-LLINF, -LLINF};
    return pll{ans.F * c / GCD * (negx ? -1 : 1),
               ans.S * c / GCD * (negy ? -1 : 1)};
}
ll inv(ll a, ll p) {
    if (p == 1) return -1;
    pll ans = bezout(a % p, -p, 1);
    if (ans == pll{-LLINF, -LLINF}) return -1;
    return (ans.F % p + p) % p;
}
```

8.6 Mu + Phi

```
const int maxn = 1e6 + 5;
ll f[maxn];
vector<int> lpf, prime;
void build() {
    lpf.clear();
    lpf.resize(maxn, 1);
    prime.clear();
    f[1] = ...; /* mu[1] = 1, phi[1] = 1 */
    for (int i = 2; i < maxn; i++) {
        if (lpf[i] == 1) {
            lpf[i] = i;
            prime.emplace_back(i);
            f[i] = ...; /* mu[i] = 1, phi[i] = i-1 */
        }
        for (auto& j : prime) {
            if (i * j >= maxn) break;
            lpf[i * j] = j;
            if (i % j == 0)
                f[i * j] = ...; /* 0, phi[i]*j */
            else
                f[i * j] = ...; /* -mu[i], phi[i]*phi[j
                    ] */
            if (j >= lpf[i]) break;
        }
    }
}
```

8.7 Discrete Log

```
long long mod_pow(long long a, long long e, long long p
    ){
    long long r = 1 % p;
    while(e){
        if(e & 1) r = (__int128)r * a % p;
        a = (__int128)a * a % p;
        e >>= 1;
    }
    return r;
}
long long mod_inv(long long a, long long p){
    return mod_pow((a%p+p)%p, p-2, p);
}
// BSGS: solve a^x = y (mod p), gcd(a,p)=1, p prime,
    return minimal x>=0, or -1 if no solution
long long bsgs(long long a, long long y, long long p){
    a%=p; y%=p;
    if(y==1%p) return 0;          // x=0
    long long m = (long long)ceil(sqrt((long double)p))
        ;
    // baby steps: a^j
    unordered_map<long long,long long> table;
    table.reserve(m*2);
    long long cur = 1%p;
    for(long long j=0;j<m;++j){
        if(!table.count(cur)) table[cur]=j;
        cur = (__int128)cur * a % p;
    }
    long long am = mod_pow(a, m, p);
    long long am_inv = mod_inv(am, p);
    long long gamma = y % p;
    for(long long i=0;i<=m;++i){
        auto it = table.find(gamma);
        if(it != table.end()){
            long long x = i*m + it->second;
            return x;
        }
        gamma = (__int128)gamma * am_inv % p;
    }
    return -1;
}
```

8.8 sqrt mod

```
// the Jacobi symbol is a generalization of the
    Legendre symbol,
// such that the bottom doesn't need to be prime.
// (n|p) -> same as legendre
// (n|ab) = (n|a)(n|b)
// work with long long
int Jacobi(int a, int m) {
    int s = 1;
    for (; m > 1; ) {
        a %= m;
        if (a == 0) return 0;
        const int r = __builtin_ctz(a);
        if ((r & 1) && ((m + 2) & 4)) s = -s;
        a >>= r;
        if (a & m & 2) s = -s;
        swap(a, m);
    }
    return s;
}
// solve x^2 = a (mod p)
// 0: a == 0
// -1: a isn't a quad res of p
// else: return X with X^2 % p == a
// doesn't work with long long
int QuadraticResidue(int a, int p) {
    if (p == 2) return a & 1;
    if (int jc = Jacobi(a, p); jc <= 0) return jc;
    int b, d;
    for (; ; ) {
        b = rand() % p;
        d = (1LL * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (1LL + p) >> 1; e; e >>= 1) {
        if (e & 1) {
            tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1
                * f1 % p)) % p;
            g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1
             % p)) % p;
        f1 = (2LL * f0 * f1) % p;
        f0 = tmp;
    }
    return g0;
}
```

8.9 Primitive Root

```
unsigned long long primitiveRoot(ull p) {
    auto fac = factor(p - 1);
    sort(all(fac));
    fac.erase(unique(all(fac)), fac.end());
    auto test = [p, fac](ull x) {
        for(ull d : fac)
        if (modpow(x, (p - 1) / d, p) == 1)
            return false;
        return true;
    };
    uniform_int_distribution<unsigned long long> unif
        (1, p - 1);
    unsigned long long root;
    while(!test(root = unif(rng)));
    return root;
}
```

8.10 LinearSieve

```
const int C = 1e7 + 2;
int mo[C], lp[C], phi[C], isp[C];
vector<int> prime;
void sieve() {
    mo[1] = phi[1] = 1;
    for(int i = 1; i < C; i++) lp[i] = 1;
    for(int i = 2; i < C; i++) {
        if(lp[i] == 1) {
            lp[i] = i;
            prime.push_back(i);
            isp[i] = 1;
            mo[i] = -1;
            phi[i] = i - 1;
        }
        for(int p : prime) {
            if(i * p >= C) break;
            lp[i * p] = p;
            if(i % p == 0) {
                phi[p * i] = phi[i] * p;
                break;
            }
            phi[i * p] = phi[i] * (p - 1);
            mo[i * p] = mo[i] * mo[p];
        }
    }
}
```

8.11 Other Formulas

- Inversion:
  $aa^{-1} \equiv 1 \pmod{m}$. a^{-1} exists iff $\gcd(a, m) = 1$.

- Linear inversion:
  $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$

- Fermat's little theorem:
  $a^p \equiv a \pmod{p}$ if p is prime.

- Euler function:
  $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$

- Euler theorem:
  $a^{\phi(n)} \equiv 1 \pmod{n}$ if $\gcd(a, n) = 1$.

- Extended Euclidean algorithm:
  $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$

- Divisor function:
  $\sigma_x(n) = \sum_{d|n} d^x$. $n = \prod_{i=1}^{r} p_i^{a_i}$.
  $\sigma_x(n) = \prod_{i=1}^{r} \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^{r}(a_i + 1)$.

- Chinese remainder theorem (Coprime Moduli):
  $x \equiv a_i \pmod{m_i}$.
  $M = \prod m_i$. $M_i = M/m_i$. $t_i = M_i^{-1}$.
  $x = kM + \sum a_i t_i M_i$, $k \in \mathbb{Z}$.

- Chinese remainder theorem:
  $x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 = m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$
  Solve for (p, q) using ExtGCD.
  $x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{lcm(m_1, m_2)}$

- Avoiding Overflow: $ca \mod cb = c(a \mod b)$

- Dirichlet Convolution: $(f * g)(n) = \sum_{d|n} f(n) g(n/d)$

- Important Multiplicative Functions + Proterties:

  1. $\epsilon(n) = [n = 1]$
  2. $1(n) = 1$
  3. $id(n) = n$
  4. $\mu(n) = 0$ if n has squared prime factor
  5. $\mu(n) = (-1)^k$ if $n = p_1 p_2 \cdots p_k$
  6. $\epsilon = \mu * 1$
  7. $\phi = \mu * id$
  8. $[n = 1] = \sum_{d|n} \mu(d)$
  9. $[gcd = 1] = \sum_{d|gcd} \mu(d)$

- Möbius inversion: $f = g * 1 \Leftrightarrow g = f * \mu$

8.12 Polynomial

```
const int maxk = 20;
const int maxn = 1<<maxk;
const ll LINF = 1e18;

/* P = r*2^k + 1
P                   r   k   g
998244353           119 23  3
1004535809          479 21  3

P                   r   k   g
3                   1   1   2
5                   1   2   2
17                  1   4   3
97                  3   5   5
193                 3   6   5
257                 1   8   3
7681                15  9   17
12289               3   12  11
40961               5   13  3
65537               1   16  3
786433              3   18  10
5767169             11  19  3
7340033             7   20  3
23068673            11  21  3
104857601           25  22  3
167772161           5   25  3
469762049           7   26  3
1004535809          479 21  3
2013265921          15  27  31
2281701377          17  27  3
3221225473          3   30  5
75161927681         35  31  3
77309411329         9   33  7
206158430209        3   36  22
2061584302081       15  37  7
2748779069441       5   39  3
6597069766657       3   41  5
39582418599937      9   42  5
79164837199873      9   43  5
263882790666241     15  44  7
1231453023109121    35  45  3
1337006139375617    19  46  3
3799912185593857    27  47  5
4222124650659841    15  48  19
7881299347898369    7   50  6
31525197391593473   7   52  3
180143985094819841  5   55  6
1945555039024054273 27  56  5
4179340454199820289 29  57  3
9097271247288401921 505 54  6 */

const int g = 3;
const ll MOD = 998244353;

ll pw(ll a, ll n) { /* fast pow */ }

#define siz(x) (int)x.size()

template<typename T>
vector<T>& operator+=(vector<T>& a, const vector<T>& b)
     {
    if (siz(a) < siz(b)) a.resize(siz(b));
    for (int i = 0; i < min(siz(a), siz(b)); i++) {
        a[i] += b[i];
        a[i] -= a[i] >= MOD ? MOD : 0;
    }
    return a;
}

template<typename T>
vector<T>& operator-=(vector<T>& a, const vector<T>& b)
     {
    if (siz(a) < siz(b)) a.resize(siz(b));
    for (int i = 0; i < min(siz(a), siz(b)); i++) {
        a[i] -= b[i];
        a[i] += a[i] < 0 ? MOD : 0;
    }
    return a;
}
```

```cpp
template<typename T>
vector<T> operator-(const vector<T>& a) {
    vector<T> ret(siz(a));
    for (int i = 0; i < siz(a); i++) {
        ret[i] = -a[i] < 0 ? -a[i] + MOD : -a[i];
    }
    return ret;
}

vector<ll> X, iX;
vector<int> rev;

void init_ntt() {
    X.clear(); X.resize(maxn, 1);  // x1 = g^((p-1)/n)
    iX.clear(); iX.resize(maxn, 1);

    ll u = pw(g, (MOD-1)/maxn);
    ll iu = pw(u, MOD-2);

    for (int i = 1; i < maxn; i++) {
        X[i] = X[i-1] * u;
        iX[i] = iX[i-1] * iu;
        if (X[i] >= MOD) X[i] %= MOD;
        if (iX[i] >= MOD) iX[i] %= MOD;
    }

    rev.clear(); rev.resize(maxn, 0);
    for (int i = 1, hb = -1; i < maxn; i++) {
        if (!(i & (i-1))) hb++;
        rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
} }

template<typename T>
void NTT(vector<T>& a, bool inv=false) {

    int _n = (int)a.size();
    int k = __lg(_n) + ((1<<__lg(_n)) != _n);
    int n = 1<<k;
    a.resize(n, 0);

    short shift = maxk-k;
    for (int i = 0; i < n; i++)
        if (i > (rev[i]>>shift))
            swap(a[i], a[rev[i]>>shift]);

    for (int len = 2, half = 1, div = maxn>>1; len <= n
        ; len<<=1, half<<=1, div>>=1) {
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < half; j++) {
                T u = a[i+j];
                T v = a[i+j+half] * (inv ? iX[j*div] :
                    X[j*div]) % MOD;
                a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
                a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v)
                    ;
    } } }

    if (inv) {
        T dn = pw(n, MOD-2);
        for (auto& x : a) {
            x *= dn;
            if (x >= MOD) x %= MOD;
} } }

template<typename T>
inline void resize(vector<T>& a) {
    int cnt = (int)a.size();
    for (; cnt > 0; cnt--) if (a[cnt-1]) break;
    a.resize(max(cnt, 1));
}

template<typename T>
vector<T>& operator*=(vector<T>& a, vector<T> b) {
    int na = (int)a.size();
    int nb = (int)b.size();
    a.resize(na + nb - 1, 0);
    b.resize(na + nb - 1, 0);

    NTT(a); NTT(b);
    for (int i = 0; i < (int)a.size(); i++) {
        a[i] *= b[i];
        if (a[i] >= MOD) a[i] %= MOD;
    }
    NTT(a, true);

    resize(a);
    return a;
}

template<typename T>
void inv(vector<T>& ia, int N) {
    vector<T> _a(move(ia));
    ia.resize(1, pw(_a[0], MOD-2));
    vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));

    for (int n = 1; n < N; n<<=1) {
        // n -> 2*n
        // ia' = ia(2-a*ia);

        for (int i = n; i < min(siz(_a), (n<<1)); i++)
            a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD :
                0));

        vector<T> tmp = ia;
        ia *= a;
        ia.resize(n<<1);
        ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia
            [0] + 2;
        ia *= tmp;
        ia.resize(n<<1);
    }
    ia.resize(N);
}

template<typename T>
void mod(vector<T>& a, vector<T>& b) {
    int n = (int)a.size()-1, m = (int)b.size()-1;
    if (n < m) return;

    vector<T> ra = a, rb = b;
    reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n
        -m+1));
    reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n
        -m+1));

    inv(rb, n-m+1);

    vector<T> q = move(ra);
    q *= rb;
    q.resize(n-m+1);
    reverse(q.begin(), q.end());

    q *= b;
    a -= q;
    resize(a);
}

/* Kitamasa Method (Fast Linear Recurrence):
Find a[K] (Given a[j] = c[0]a[j-N] + ... + c[N-1]a[j
    -1])
Let B(x) = x^N - c[N-1]x^(N-1) - ... - c[1]x^1 - c[0]
Let R(x) = x^K mod B(x)   (get x^K using fast pow and
    use poly mod to get R(x))
Let r[i] = the coefficient of x^i in R(x)
=> a[K] = a[0]r[0] + a[1]r[1] + ... + a[N-1]r[N-1] */
```

9 Linear Algebra

9.1 Gaussian-Jordan Elimination

```cpp
int n;
vector<vector<ll>> v;
void gauss(vector<vector<ll>>& v) {
    int r = 0;
    for (int i = 0; i < n; i++) {
        bool ok = false;
        for (int j = r; j < n; j++) {
            if (v[j][i] == 0) continue;
            swap(v[j], v[r]);
            ok = true;
            break;
        }
        if (!ok) continue;
        ll div = inv(v[r][i]);
        for (int j = 0; j < n + 1; j++) {
            v[r][j] *= div;
            if (v[r][j] >= MOD) v[r][j] %= MOD;
        }
        for (int j = 0; j < n; j++) {
            if (j == r) continue;
            ll t = v[j][i];
            for (int k = 0; k < n + 1; k++) {
                v[j][k] -= v[r][k] * t % MOD;
                if (v[j][k] < 0) v[j][k] += MOD;
            }
        }
        r++;
    }
}
```

9.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then det = 0, otherwise det = product of diagonal elements.

2. Properties of det:

   - Transpose: Unchanged
   - Row Operation 1 - Swap 2 rows: $-det$
   - Row Operation 2 - $k\vec{r_i}$: $k \times det$
   - Row Operation 3 - $k\vec{r_i}$ add to $\vec{r_j}$: Unchaged

10 Combinatorics

10.1 Catalan Number

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, C_n = C_n^{2n} - C_{n-1}^{2n}$$

0	1	1	2	5
4	14	42	132	429
8	1430	4862	16796	58786
12	208012	742900	2674440	9694845

10.2 Burnside's Lemma

Let X be the original set.
Let G be the group of operations acting on X.
Let X^g be the set of x not affected by g.
Let X/G be the set of orbits.
Then the following equation holds:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$