

Contents

1 Data Structures & Algorithms	2	6.5 Diophantine	12
1.1 Sparse Table	2	6.6 Chinese Remainder Theorem	12
1.2 Fenwick Tree	2	6.7 Rabin-Miller primality test	12
1.3 Segment Tree	2	6.8 Pollard's Rho prime factorization	13
1.4 Sigma Tree	2	6.9 Multiplicative Function (Sieve)	14
1.5 Persistent Segment Tree	2	6.10 Multiplicative Function	14
1.6 Hash Map	3	6.11 Discrete Logarithm	14
1.7 Parallel Binary Search	3	6.12 More	15
1.8 Ternary Search	3	7 Geometry	15
2 String	4	7.1 Dot Product	15
2.1 Trie 1	4	7.2 Angle	15
2.2 Trie 2	4	7.3 Cross Product	15
2.3 Hash	4	7.4 Distance from a Point to a line	15
2.4 KMP	4	7.5 Template 1	16
2.5 Manacher	4	7.6 Intersection of 2 lines and bla bla bla (I have no time bro)	16
2.6 Aho - Corasick	4	7.7 Circle passing through 3 points	16
2.7 Aho - Corasick (BFS)	5	7.8 Symmetry	16
3 SQRT Decomposition	5	7.9 Rotation	17
3.1 MO	5	7.10 Area of a Polygon	17
4 Graph	5	7.11 Relative position of a point to a poly- gon $O(N)$	17
4.1 Joint and Bridge	5	7.12 Relative position of a point to a poly- gon $O(\log N)$	17
4.2 SCC	5	7.13 Pick Theorem	17
4.3 Topology Sort 1	5	7.14 Convex Hull (Graham scan)	17
4.4 Topology Sort 2	6	7.15 Convex Hull (Monotone chain algorithm)	18
4.5 Max Flow	6	7.16 Find Fermat Point	18
4.6 Bipartite Matching	6	7.17 More	19
4.7 Matching 2	6	8 Algebra	19
4.8 HLD	7	8.1 Matrix Multiplication	19
4.9 DSU on tree	7	8.2 Fast Fourier Transform	20
4.10 Centroid Decomposition	7	8.3 Number Theory Transform	20
4.11 Centroid Tree (CT)	8	9 Combinatoric	21
4.12 Virtual Tree	8	9.1 Formula	21
4.13 Steiner Tree	9	9.2 Catalan	21
5 DP	9	9.3 Derangement	21
5.1 Digit DP	9	9.4 Classical Sums	21
5.2 SOS DP	9	9.5 Useful Identities	21
5.3 DP DNC	10	9.6 Stirling Numbers	21
5.4 Convex Hull Trick	10	10 Others	21
5.5 1D1D Optimization	10	10.1 Test?	21
5.6 Knapsack on Tree	11	10.2 Big Int	22
5.7 DP on Broken Profile	11	10.3 CODE::BLOCK Set Up	23
6 Number Theory	11	10.4 Template	23
6.1 Euler's totient function	11	10.5 CODE::BLOCK shortcuts	23
6.2 Euler's totient function from 1 to N	11	10.6 Multiplication Function	24
6.3 Modular Inverse	11	10.7 Random Function	24
6.4 Extended Euclidean Algorithm	12		

Data Structures & Algorithms

Sparse Table

```
int a[N], st[LG + 1][N];
void preprocess() {
    for (int i = 1; i <= n; ++i) st[0][i] = a[i];
    for (int j = 1; j <= LG; ++j)
        for (int i = 1; i + (1 << j) - 1 <= n; ++i)
            st[j][i] = min(st[j - 1][i], st[j - 1][i +
                (1 << (j - 1))]);
}

int query(int l, int r) {
    int k = __lg(r - l + 1);
    return min(st[k][l], st[k][r - (1 << k) + 1]);
}

//query sum:
int querySum(int l, int r) {
    int len = r - l + 1;
    int sum = 0;
    for (int j = 0; (1 << j) <= len; ++j)
        if (len >> j & 1) {
            sum = sum + st[j][l];
            l = l + (1 << j);
        }
    return sum;
}
```

```
void update(int id, int l, int r, int u, int v, int
    x){
    if (r < u || v < l) return;
    if (u <= l && r <= v){
        apply(id, x);
        return;
    }

    down(id, l, r);

    int mid = (l + r) >> 1;
    update(id << 1, l, mid, u, v, x);
    update(id << 1 | 1, mid + 1, r, u, v, x);

    st[id] = merge(st[id << 1], st[id << 1 | 1]);
}

int get(int id, int l, int r, int u, int v){
    if (r < u || v < l) return -INF;
    if (u <= l && r <= v) return st[id];

    down(id, l, r);

    int mid = (l + r) >> 1;
    return merge(get(id << 1, l, mid, u, v), get(id
        << 1 | 1, mid + 1, r, u, v));
}
} ST;
```

Fenwick Tree

```
void update(int i, int val){
    for (; i <= n; i += i & -i) bit[i] += val;
}

int get(int i){
    int res = 0;
    for (; i; i -= i & -i) res += bit[i];
    return res;
}
```

Sigma Tree

```
struct Sigma_Tree{
    int st[2 * N];

    void init(){
        For(i, 1, n) st[i + n - 1] = a[i];
        ForD(i, n - 1, 1) st[i] = merge(st[i << 1],
            st[i << 1 | 1]);
    }

    void update(int p, int val){
        p += n - 1;
        st[p] = val;
        for (; p > 1; p >>= 1) st[p >> 1] = merge(st[p],
            st[p ^ 1]);
    }

    int get(int l, int r){
        int res = 0;
        for (l += n - 1, r += n - 1; l <= r; l >>= 1,
            r >>= 1){
            if (l & 1) res = merge(res, st[l++]);
            if (!(r & 1)) res = merge(res, st[r--]);
        }
        return res;
    }
} ST;
```

Segment Tree

```
struct Segment_tree{
    int st[4 * N], lazy[4 * N];

    void apply(int id, int c){
        update(st[id], c);
        update(lazy[id], c);
    }

    void down(int id, int l, int r){
        int c = lazy[id]; lazy[id] = 0;
        apply(id << 1, c); apply (id << 1 | 1, c);
    }

    void build(int id, int l, int r){
        if (l == r){
            st[id] = a[l];
            return;
        }

        int mid = (l + r) >> 1;
        build(id << 1, l, mid);
        build(id << 1 | 1, mid + 1, r);

        st[id] = merge(st[id << 1], st[id << 1 | 1]);
    }
}
```

Persistent Segment Tree

```
struct Node {
    int left, right; // ID of left child & right
        child
    long long ln; // Max value of node
    Node() {}
    Node(long long ln, int left, int right) : ln(ln),
        left(left), right(right) {}
}
```

```

} it[N]; // Each node has a position in this array,
        called ID
int nNode;

int ver[N]; // ID of root in each version

// Update max value of a node
inline void refine(int cur) {
    it[cur].ln = max(it[it[cur].left].ln, it[it[cur].
        right].ln);
}

// Update a range, and return new ID of node
int update(int l, int r, int u, int x, int oldId) {
    if (l == r) {
        ++nNode;
        it[nNode] = Node(x, 0, 0);
        return nNode;
    }

    int mid = (l + r) >> 1;
    int cur = ++nNode;

    if (u <= mid) {
        it[cur].left = update(l, mid, u, x, it[oldId]
            .left);
        it[cur].right = it[oldId].right;
        refine(cur);
    }
    else {
        it[cur].left = it[oldId].left;
        it[cur].right = update(mid+1, r, u, x, it[
            oldId].right);
        refine(cur);
    }

    return cur;
}

// Get max of range. Same as usual IT
int get(int nodeId, int l, int r, int u, int v) {
    if (v < l || r < u) return -1;
    if (u <= l && r <= v) return it[nodeId].ln;

    int mid = (l + r) >> 1;
    return max(get(it[nodeId].left, l, mid, u, v),
        get(it[nodeId].right, mid+1, r, u, v));
}

// When update:
++nVer;
ver[nVer] = update(1, n, u, x, ver[nVer-1]);

// When query:
res = get(ver[t], 1, n, u, v);

```

Hash Map

```

//faster than unordered_map
struct hash_map {
    const static int SZ = 2e4 + 9;
    int nxt[SZ >> 3], val[SZ >> 3];
    int key[SZ >> 3];
    int h[SZ + 5], cnt;
    vector<int>vec;

    void clear(){
        for (int i : vec) h[i] = 0;
    }

```

```

        for (int i = 1; i <= cnt; i++)
            val[i] = nxt[i] = 0, key[i] = 0;

        vec.clear();
        cnt = 0;
    }

    int hash(int u) {
        return u % SZ;
    }

    int &operator[](int u) {
        int x = hash(u);

        for (int i = h[x]; i; i = nxt[i])
            if (key[i] == u) return val[i];

        if (!h[x]) vec.push_back(x);

        ++cnt;
        key[cnt] = u;
        val[cnt] = 0;
        nxt[cnt] = h[x];
        h[x] = cnt;
        return val[cnt];
    }

    int qry(int u) {
        int x = hash(u);
        for (int i = h[x]; i; i = nxt[i])
            if (key[i] == u) return val[i];

        return 0;
    }
} hs;

```

Parallel Binary Search

```

while (1){
    bool ok = 1;

    For(i, 1, q){
        if (l[i] > r[i]) continue;
        ok = 1;
        queries[(l[i] + r[i]) >> 1].push_back(i);
    }

    reset();
    For(t, 1, n){
        //update(t,...);
        for (int i: queries[t]){
            if (check(i)) r[i] = t - 1;
            else l[i] = t + 1;
        }
    }
}

```

Ternary Search

Find the maximum Point in a Graph like: \wedge

```

double max_f(double left, double right) {
    int N_ITER = 100;
    for (int i = 0; i < N_ITER; i++) {
        double x1 = left + (right - left) / 3.0;
        double x2 = right - (right - left) / 3.0;
        if (f(x1) > f(x2)) right = x2;
        else left = x1;
    }
}

```

```
    return f(left);
}
```

String

Trie 1

```
struct node{
    node *g[26];
    node(){
        rep(i, 26) g[i] = NULL;
    }
} *root = new node();

void Insert(string s){
    node *p = root;
    for (char t: s){
        if (p->g[t - 'a'] == NULL)
            p->g[t - 'a'] = new node();

        p = p->g[t - 'a'];
    }
}
```

Trie 2

```
int nNode = 0;
int g[N][26];

void Insert(string s){
    int p = 0;
    for (char t: s){
        if (!g[p][t - 'a']) g[p][t - 'a'] = ++nNode;
        p = g[p][t - 'a'];
    }
}
```

Hash

```
ll getHashT(int i, int j) {
    return (hashT[j] - mul(hashT[i - 1], POW[j - i + 1]) + MOD) % MOD;
}

// Precalculate base^i
for (int i = 1; i <= lenT; i++)
    POW[i] = (POW[i - 1] * base) % MOD;

// Calculate hash value of T[1..i]
for (int i = 1; i <= lenT; i++)
    hashT[i] = (hashT[i - 1] * base + (T[i] - 'a' + 1)) % MOD;
```

KMP

```
//prefix function: length of the longest prefix of
//the substring s[1..i] that is also a suffix of
//this same substring
int k = 0;
for(i, 2, n){ //1-indexed
    while (k && s[k + 1] != s[i]) k = kmp[k];
    kmp[i] = (s[k + 1] == s[i]) ? ++k : 0;
}
```

Manacher

```
vector<int> manacher_odd(string s) {
    int n = s.size();
```

```
    s = "$" + s + "~";
    vector<int> p(n + 2);
    int l = 0, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = min(r - i, p[l + (r - i)]);
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}

vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}
```

Aho - Corasick

```
namespace Trie{
    struct Node{
        int child[26], p = -1, cnt = 0;
        char pch;
        int link = -1, go[26];
        Node(int p = -1, char ch = '#'): p(p), pch(ch){
            fill(begin(child), end(child), -1);
            fill(begin(go), end(go), -1);
        }
    };

    vector<Node> g(1);

    void add(string s){
        int v = 0;
        for (char t: s){
            int c = t - 'a';
            if (g[v].child[c] == -1){
                g[v].child[c] = g.size();
                g.emplace_back(v, t);
            }
            v = g[v].child[c];
        }
        g[v].cnt++;
    }

    int go(int v, char c);

    int get_link(int v){
        if (g[v].link == -1){
            if (!v || !g[v].p) g[v].link = 0;
            else g[v].link = go(get_link(g[v].p), g[v].pch);
        }
        return g[v].link;
    }

    int go(int v, char t){
        int c = t - 'a';
        if (g[v].go[c] == -1){
            if (g[v].child[c] != -1) g[v].go[c] = g[v].
                child[c];
```

```

        else g[v].go[c] = (v == 0) ? 0 : go(get_link(v)
            , t);
    }
    return g[v].go[c];
}
}

```

Aho - Corasick (BFS)

```

struct trie{
    struct Node{
        Node *child[26], *link;
        int cnt = 0;
        Node(){
            cnt = 0;
            rep(i, 26) child[i] = NULL;
            link = NULL;
        }
    } *root = new Node();

    void add(string &s){
        Node* p = root;
        for (char &t: s){
            int c = t - 'a';
            if (p->child[c] == NULL) p->child[c] = new Node
                ();
            p = p->child[c];
        }
        p->cnt++;
    }

    void AhoCorasick(){
        root->link = root;
        queue<Node*> q; q.push(root);
        while (!q.empty()){
            Node* p = q.front(); q.pop();
            rep(i, 26) if (p->child[i]){
                Node* k = p->link;
                while (k != root && k->child[i] == NULL) k =
                    k->link;

                if (k->child[i] && k != p) p->child[i]->link
                    = k->child[i];
                else p->child[i]->link = root;

                p->child[i]->cnt += p->child[i]->link->cnt;
                q.push(p->child[i]);
            }
        }
    }
};

```

SQRT Decomposition

MO

```

struct query{
    int l, r, id;
}

bool cmp(const query &a, const query &b){
    if(a.l / S != b.l / S) return a.l < b.l;

    if((a.l / S) & 1)
        return a.r < b.r;
    else
        return a.r > b.r
}

```

```

long long res = 0;

void update(...);

////////////////////

sort(q + 1, q + Q + 1, cmp);

int l = 1, r = 0;
for(int i = 1; i <= Q; i++){
    while(l < q[i].l) update(a[l++], ...);
    while(l > q[i].l) update(a[--l], ...);
    while(r < q[i].r) update(a[++r], ...);
    while(r > q[i].r) update(a[r--], ...);
    ans[q[i].id] = res;
}

```

Graph

Joint and Bridge

```

void dfs(int u, int pre) {
    int child = 0;
    num[u] = low[u] = ++timer;
    for (int v: g[u]) {
        if (v == pre) continue;
        if (!num[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] == num[v]) bridge++;
            child++;
            if (u == pre){
                if (child > 1) joint[u] = true;
            }
            else if (low[v] >= num[u]) joint[u] = true
                ;
        }
        else low[u] = min(low[u], num[v]);
    }
}

```

SCC

```

void dfs(int u) {
    num[u] = low[u] = ++timer;
    st.push(u);
    for (int v : g[u]) {
        if (!num[v]){
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else low[u] = min(low[u], num[v]);
    }
    if (low[u] == num[u]) {
        scc++;
        int v;
        do {
            v = st.top();
            st.pop();
            num[v] = INF;
        }
        while (v != u);
    }
}

```

Topology Sort 1

```

//u -> v

```

```
//++deg[v]
for (int u = 1; u <= n; ++u)
    if (!deg[u]) q.push(u);

while (!q.empty()) {
    int u = q.front();
    q.pop();
    topo.push_back(u);
    for (auto v : g[u]) {
        deg[v]--;
        if (!deg[v]) q.push(v);
    }
}
```

Topology Sort 2

```
void dfs(int u) {
    visit[u] = 1;
    for (auto v : g[u]) {
        assert(visit[v] != 1);
        //graph contains a cycle
        if (!visit[v]) dfs(v);
    }
    topo.push(u);
    visit[u] = 2;
}
```

Max Flow

```
struct edge{
    int to, rev, flow, cap;
};

void add_edge(int u, int v, int cap){
    edge e1 = {v, sz(g[v]), 0, cap};
    edge e2 = {u, sz(g[u]), 0, 0};
    g[u].pb(e1); g[v].pb(e2);
}

bool bfs(){
    memset(dist, 0x3f, sizeof dist);
    queue<int> q;
    q.push(source); dist[source] = 0;
    while (!q.empty()){
        int u = q.front(); q.pop();
        for (edge e: g[u]){
            int v = e.to, flow = e.flow, cap = e.cap;
            if (flow < cap && minimize(dist[v], dist[u] + 1))
                q.push(v);
        }
    }
    return dist[sink] < INF;
}

int dfs(int u, int mn){
    if (u == sink) return mn;
    for (int &i = lazy[u]; i < sz(g[u]); ++i){
        auto &[v, rev, flow, cap] = g[u][i];
        if (dist[v] == dist[u] + 1 && flow < cap){
            int cur = dfs(v, min(mn, cap - flow));
            if (cur > 0){
                flow += cur;
                g[v][rev].flow -= cur;
                return cur;
            }
        }
    }
}
```

```
return 0;
}

int main(){
    //...
    int res = 0;
    while (bfs()){
        memset(lazy, 0, sizeof lazy);
        while (int del = dfs(source, INF))
            res += del;
    }

    cout << res;
    return 0;
}
```

Bipartite Matching

```
bool dfs(int u){
    if (seen[u]) return 0;
    seen[u] = 1;

    for (int v: g[u])
        if (!mt[v] || dfs(mt[v]))
            return mt[v] = u, 1;

    return 0;
}
```

```
//memset(mt, 0, sizeof mt);
//For(i, 1, n){
    //memset(seen, 0, sizeof seen);
    //dfs(i);
//}
```

Matching 2

```
bool bfs(){
    bool res = 0;
    queue<int> q;
    For(i, 1, n){
        if (!mx[i]){
            q.push(i), dist[i] = 0;
        }
        else{
            dist[i] = -1;
        }
    }

    while (!q.empty()){
        int u = q.front(); q.pop();
        for (int v: g[u]){
            if (!my[v]) res = 1;
            else if (dist[my[v]] < 0){
                dist[my[v]] = dist[u] + 1;
                q.push(my[v]);
            }
        }
    }

    return res;
}

bool dfs(int u){
    for (int v: g[u]){
        if (!my[v]){
            mx[u] = v; my[v] = u;

```

```

        return 1;
    }
    else if (dist[my[v]] == dist[u] + 1 && dfs(my[v]))
    ){
        mx[u] = v; my[v] = u;
        return 1;
    }
}
return 0;
}
}

////////////////////////////////////
while (bfs()){
    For(i, 1, n) if (!mx[i]){
        dfs(i);
    }
}
}

```

HLD

```

void dfs(int u){
    sz[u] = 1;
    for (int v: g[u]) if (v != par[u]){
        par[v] = u;
        dfs(v);
        sz[u] += sz[v];
    }
}

void hld(int u){
    if (!Head[nChain]) Head[nChain] = u;
    idChain[u] = nChain;

    pos[u] = ++timer;
    node[timer] = u;

    int bigC = 0;
    for (int v: g[u]) if (v != par[u])
        if (!bigC || sz[v] > sz[bigC])
            bigC = v;

    if (bigC) hld(bigC);
    for (int v: g[u]) if (v != par[u] && v != bigC){
        ++nChain;
        hld(v);
    }
}

//LCA
int LCA(int u, int v){
    while (idChain[u] != idChain[v]){
        if (idChain[u] > idChain[v])
            u = par[Head[idChain[u]]];
        else
            v = par[Head[idChain[v]]];
    }

    if (h[u] < h[v]) return u;
    return v;
}

int get(int u, int v){
    int res = 0;
    while (idChain[u] != idChain[v]){
        if (idChain[u] > idChain[v]){
            maximize(res, ST.get(pos[Head[idChain[u]]],
                                pos[u]));
            u = par[Head[idChain[u]]];
        }
    }
}

```

```

    }
    else{
        maximize(res, ST.get(pos[Head[idChain[v]]],
                             pos[v]));
        v = par[Head[idChain[v]]];
    }
}

if (pos[u] < pos[v])
    maximize(res, ST.get(pos[u], pos[v]));
else
    maximize(res, ST.get(pos[v], pos[u]));

return res;
}

```

DSU on tree

```

void dfs(int u, int prev = -1){
    in[u] = ++timer; node[timer] = u;
    for (int v: g[u]) if (v != prev)
        dfs(v, u);
    out[u] = timer;
}

#define sz(u) out[u] - in[u]

void calc(int u, int prev = -1){
    int bigC = 0;
    for (int v: g[u]) if (v != prev)
        if (sz(v) > sz(bigC))
            bigC = v;

    for (int v: g[u]) if (v != prev && v != bigC){
        calc(v, u);
        //reset(v)...
    }

    if (bigC) calc(bigC, u);

    for (int v: g[u]) if (v != prev && v != bigC){
        For(t, in[v], out[v]){
            int x = node[t];
            //...
        }
    }
}

```

Centroid Decomposition

```

int size(int u, int prev){
    sz[u] = 1;
    for (int v: g[u]) if (!del[v] && v != prev)
        sz[u] += size(v, u);
    return sz[u];
}

int centroid(int u, int prev){
    for (int v: g[u]) if (!del[v] && v != prev)
        if (sz[v] > n/2)
            return centroid(v, u);
    return u;
}

void dfs(int u, int prev){
    in[u] = ++timer; node[timer] = u;
    for (int v: g[u]) if (!del[v] && v != prev){
        dfs(v, u);
        //...
    }
}

```

```

    }
    out[u] = timer;
}

void calc(int u){
    n = size(u, 0);
    u = centroid(u, 0);

    timer = 0;
    dfs(u, 0);

    for (int v: g[u]) if (!del[v]){
        //subtree v...
    }
    //reset
    del[u] = 1;
    for (auto [v, c]: g[u]) if (!del[v])
        calc(v);
}

```

Centroid Tree (CT)

Centroid Tree properties:

- Centroid tree height $\leq \log(n)$
- $LCA(u, v)$ in CT lies on the path from u to v in the original tree

```

int size(int u, int prev){
    sz[u] = 1;
    for (int v: g[u]) if (v != prev && !del[v]){
        sz[u] += size(v, u);
    }
    return sz[u];
}

int centroid(int u, int prev, int m){
    for (int v: g[u]) if (v != prev && !del[v])
        if (sz[v] > m/2)
            return centroid(v, u, m);
    return u;
}

int cd(int u){
    int m = size(u);
    u = centroid(u, 0, m);
    del[u] = 1;
    for (int v: g[u]) if (!del[v]){
        v = cd(v);
        par[v] = u;
    }
    return u;
}

//example problems:
void solve(){
    dfs(1, 0); init(); //to calculate the dist(u, v)
    from the original tree
    cd(1);

    memset(d, 0x3f, sizeof d);

    c[1] = 1; //color
    int pp = 1;
    while (pp){
        minimize(d[pp], dist(pp, 1));
        pp = par[pp];
    }
}

```

```

}

while (q--){
    int t; cin >> t;
    if (t == 1){
        int u; cin >> u;
        c[u] = 1;

        int p = u;
        while (p){
            minimize(d[p], dist(p, u));
            p = par[p];
        }
    }
    else{
        int u; cin >> u;
        if (c[u]) {
            cout << 0 << endl; continue;
        }

        int p = u, res = INF;
        while(p){
            minimize(res, dist(u, p) + d[p]);
            p = par[p];
        }

        cout << res << endl;
    }
}
}
}

```

Virtual Tree

```

void dfs(int u){
    in[u] = ++timer;
    for (int v: g[u]) if (v != up[u][0]){
        up[v][0] = u;
        For(j, 1, 17) up[v][j] = up[up[v][j - 1]][j - 1];
        dfs(v);
    }
    out[u] = timer;
}

bool is_anc(int u, int v){
    if (!u) return 1;
    return in[u] <= in[v] && in[v] <= out[u];
}

//short LCA
int lca(int u, int v){
    if (is_anc(u, v)) return u;
    ForD(j, 17, 0){
        if (!is_anc(up[u][j], v)){
            u = up[u][j];
        }
    }
    return up[u][0];
}

bool cmp(int u, int v){
    return in[u] < in[v];
}

void query(){
    cin >> k;
    For(i, 1, k) cin >> a[i], sz[a[i]] = 1;

    sort(a + 1, a + k + 1, cmp);
    For(i, 1, k - 1) a[i + k] = lca(a[i], a[i + 1]);
}

```



```

sort(a + 1, a + k + k, cmp);
k = unique(a + 1, a + k + k) - a - 1;

stack<int> st; st.push(a[1]);
For(i, 2, k){
    while (!is_anc(st.top(), a[i])) st.pop();
    g[st.top()].pb(a[i]);
    st.push(a[i]);
}

res = 0; calc(a[1]);
cout << res << endl;

For(i, 1, k) sz[a[i]] = 0, g[a[i]].clear();
}

void solve(){
    //...
    dfs(1);
    For(i, 1, n) g[i].clear();
    while (q--) query();
}

```

Steiner Tree

```

int main() {
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<Edge>> g(n);
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        --u; --v;
        g[u].push_back({v, w});
        g[v].push_back({u, w});
    }

    vector<int> term(k);
    for (int i = 0; i < k; i++) {
        cin >> term[i];
        --term[i];
    }

    int FULL = 1 << k;
    vector<vector<int>> dp(FULL, vector<int>(n, INF))
        ;

    for (int i = 0; i < k; i++)
        dp[1 << i][term[i]] = 0;

    for (int mask = 1; mask < FULL; mask++) {
        for (int sub = (mask - 1) & mask; sub =
            (sub - 1) & mask) {
            for (int v = 0; v < n; v++)
                dp[mask][v] = min(dp[mask][v], dp[sub
                    ][v] + dp[mask ^ sub][v]);
        }

        priority_queue<pair<int,int>, vector<pair<int
            ,int>>, greater<>> pq;
        for (int v = 0; v < n; v++)
            if (dp[mask][v] < INF) pq.push({dp[mask][v
                ], v});

        while (!pq.empty()) {
            auto [d, v] = pq.top(); pq.pop();
            if (d != dp[mask][v]) continue;
            for (auto &e : g[v]) {

```

```

                if (dp[mask][e.to] > d + e.w) {
                    dp[mask][e.to] = d + e.w;
                    pq.push({dp[mask][e.to], e.to});
                }
            }
        }

        int ans = INF;
        for (int v = 0; v < n; v++)
            ans = min(ans, dp[FULL - 1][v]);

        cout << ans << "\n";
    }

    //trace:
    void trace(int mask, int v, set<pair<int,int>> &edges
        ) {
        int sub = from_mask[mask][v];
        int u = parent[mask][v];

        if (sub != -1) {
            trace(sub, v, edges);
            trace(mask ^ sub, v, edges);
            return;
        }

        if (u != -1) {
            edges.insert({min(u, v), max(u, v)});
            trace(mask, u, edges);
            return;
        }
    }
}

```

DP

Digit DP

```

int f(int id, bool sml, ...){
    if (id < 0) return ...;
    if (!sml && dp[id][...] != -1) return dp[id
        ][...];

    int lim = sml ? a[id] : 9;
    int res = ...;
    For(c, 0, lim){
        update(res, f(id - 1, sml && c == lim, ...));
    }
    if (!sml) dp[id][...] = res;
    return res;
}

int get(int x){
    int n = 0;
    while (x){
        a[n++] = x % 10;
        x /= 10;
    }
    return f(n - 1, 1, ...);
}

```

SOS DP

```

for (int k = 0; k < n; k++)
    for (int mask = 0; mask < (1 << n); mask++)
        if (mask & (1 << k))
            dp[mask] += dp[mask ^ (1 << k)];

```

DP DNC

Solving problems that have the form:

$$dp(i, j) = \min_{0 \leq k \leq j} (dp(i-1, k-1) + C(k, j))$$

The cost function has to satisfy the quadrangle inequality: $C(a, c) + C(b, d) \leq C(a, d) + C(b, c)$ for all $a \leq b \leq c \leq d$.

Example:

$C(j, i) = f(i - j)$ where f is convex.

$C(j, i) = (i - j)$

$C(j, i) = (i - j)^2$

```
int m, n;
vector<int> dp_before(n + 1), dp_cur(n + 1);

// cost function
int C(int l, int r);

// calculate dp_cur[l], ..., dp_cur[r]
void compute(int l, int r, int optl, int optptr) {
    if (l > r) return;

    int mid = (l + r) >> 1;
    pair<int, int> best = {INT_MAX, -1};
    //calculate dp_cur[mid] & opt[i][mid] depend on
    dp_before and cost func
    for (int k = optl; k <= min(mid, optptr); ++k) {
        minimize(best, {dp_before[k] + C(k, mid), k})
    }
    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optptr);
}

int solve() {
    for (int i = 0; i <= n; ++i)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; ++i) {
        compute(0, n, 0, n);
        dp_before = dp_cur;
    }

    return dp_before[n];
}
```

Convex Hull Trick

Adding lines $y = kx + m$ and querying minimum values at integer x .

```
struct Line {
    int k, m;
    mutable int p;

    int eval(int x){
        return k * x + m;
    }

    bool operator < (const Line& l) const {
        return k < l.k;
    }
    bool operator < (const int &x) const {
        return p < x;
    }
}
```

```

    }
};

struct ConvexHull : multiset<Line, less<>> {
    int div(int a, int b) {
        return a / b - ((a ^ b) < 0 && a % b);
    }

    bool bad(iterator x, iterator y) {
        if(y == end()) {
            x->p = LINF;
            return 0;
        }

        if(x->k == y->k) x->p = x->m > y->m ? LINF : -LINF;
        else x->p = div(y->m - x->m, x->k - y->k);

        return x->p >= y->p;
    }

    void add(int k, int m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (bad(y, z)) z = erase(z);

        if(x != begin() && bad(--x, y)) bad(x, y = erase(y));
        while((y = x) != begin() && (--x)->p >= y->p)
            bad(x, erase(y));
    }

    int query(int x) {
        assert(!empty());
        Line l = *lower_bound(x);
        return l.eval(x);
    }
} CH;

//If you want maximum, just flip signs:
CH.add(-k, -m);
int res = -CH.query(x);
```

1D1D Optimization

Solving problems that have the form:

$$dp(i) = \min_{0 \leq j < i} (dp(j) + C(j, i)),$$

```
struct item {
    int l, r, p;
};

long long w(int j, int i) {
    //cost function
}

void solve() {
    deque<item> dq;
    dq.push_back({1, n, 0});
    for (int i = 1; i <= n; ++i) {
        f[i] = f[dq.front().p] + w(dq.front().p, i);
        ++dq.front().l;

        if (dq.front().l > dq.front().r) {
            dq.pop_front();
        }

        while (!dq.empty()) {

```

```

    auto [l, r, p] = dq.back();
    if (f[i] + w(i, l) < f[p] + w(p, l)) {
        dq.pop_back();
    }
    else break;
}

if (dq.empty()) {
    dq.push_back({i + 1, n, i});
    // h[i+1]=h[i+2]=...=h[n]=i
}
else {
    auto& [l, r, p] = dq.back();
    int low = l, high = r;
    int pos = r + 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (f[i] + w(i, mid) < f[p] + w(p, mid)) {
            pos = mid, high = mid - 1;
        }
        else {
            low = mid + 1;
        }
    }

    r = pos - 1;
    if (pos <= n) {
        dq.push_back({pos, n, i});
        // h[pos]=h[pos+1]=...=h[n]=i
    }
}
}
}

```

Knapsack on Tree

Problem: Given a tree T with N vertices rooted at vertex 1 ($1 \leq N \leq 5000$). The i -th vertex has a value C_i and a constraint K_i ($|C_i| \leq 10^9$, $1 \leq K_i \leq N$). Choose a subset of vertices such that, in the subtree of every vertex i , there are at most K_i chosen vertices, and the total sum of the chosen vertices' values is maximized.

```

void calc(int V){
    int n = child[V].size();

    for(int v_i: child[V]) {
        calc(v_i);
    }

    for(int i = 0; i <= n; i++) fill(fV[i], fV[i] + N + 1, -INF);
    fV[0][0] = 0;

    for(int i = 1; i <= n; i++){
        int v_i = child[V][i - 1];
        for(int a = 0; a <= sz[V]; a++){
            for(int b = 0; b <= sz[v_i]; b++){
                fV[i][a+b] = max(fV[i][a+b], fV[i-1][a] + dp[v_i][b]);
            }
        }
        sz[V] += sz[v_i];
    }

    for(int k = 0; k <= N; k++){
        if(k > K[V]) dp[V][k] = -INF;
        else {

```

```

            if(k > 0) dp[V][k] = max(fV[n][k], fV[n][k - 1] + C[V]);
            else dp[V][k] = fV[n][k];
        }
    }
    sz[V]++;
}

long long solve() {
    calc(1);
    return *max_element(dp[1], dp[1] + N + 1);
}

```

DP on Broken Profile

count the number of ways you can fill an $n \times m$ grid using 1×2 and 2×1 tiles. ($1 \leq n \leq 10, 1 \leq m \leq 1000$)

```

dp[0][0] = 1;
for (int j = 0; j < m; j++)
    for (int i = 0; i < n; i++) {
        for (int mask = 0; mask < (1 << n); mask++) {
            dp[mask][1] = dp[mask ^ (1 << i)][0]; // Horizontal or no tile
            if (i && !(mask & (1 << i)) && !(mask & (1 << i - 1))) // Vertical tile
                dp[mask][1] += dp[mask ^ (1 << i - 1)][0];
            if (dp[mask][1] >= MOD) dp[mask][1] -= MOD;
        }
        for (int mask = 0; mask < (1 << n); mask++)
            dp[mask][0] = dp[mask][1];
    }
cout << dp[0][0] << '\n';

```

Number Theory

Euler's totient function

```

int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            res -= res / i;
        }
    }
    if (n > 1) res -= res / n;
    return res;
}

```

Euler's totient function from 1 to N

```

void preCompute(int n) {
    iota(phi, phi + N, 0); //phi[i] = i
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}

```

Modular Inverse

```

//if MOD is a prime number then phi(MOD)= MOD - 1
int inv(int x, int MOD){
    return Pow(x, phi(MOD) - 1);
}

```

```
}

```

Extended Euclidean Algorithm

```
//computing gcd(a, b) and finding (x, y) that
//ax + by = gcd(a, b)

//recursive version
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

//iterative version
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}

```

Diophantine

```
bool find_any_solution(int a, int b, int c, int &x0,
    int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) return false;

    x0 *= c / g; y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

//all the solutions have the form:
//x = x0 + k * b/g
//y = y0 - k * a/g

//IN A GIVEN INTERVAL:
void shift(int &x, int &y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx,
    int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g)) return
        0;
    a /= g; b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift(x, y, a, b, (minx - x) / b);

```

```
    if (x < minx) shift(x, y, a, b, sign_b);
    if (x > maxx) return 0;
    int lx1 = x;

    shift(x, y, a, b, (maxx - x) / b);
    if (x > maxx) shift(x, y, a, b, -sign_b);
    int rx1 = x;

    shift(x, y, a, b, -(miny - y) / a);
    if (y < miny) shift(x, y, a, b, -sign_a);
    if (y > maxy) return 0;
    int lx2 = x;

    shift(x, y, a, b, -(maxy - y) / a);
    if (y > maxy) shift(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2) swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx) return 0;
    return (rx - lx) / abs(b) + 1;
}

```

Chinese Remainder Theorem

```
// Combine two congruences:
// x = a1 (mod m1), x = a2 (mod m2)
// Returns (x, lcm) or (-1,-1) if no solution
pair<ll, ll> crt2(ll a1, ll m1, ll a2, ll m2) {
    int x, y;
    ll g = gcd(m1, m2, x, y);

    if ((a2 - a1) % g != 0) {
        return {-1, -1}; // no solution
    }

    ll lcm = m1 / g * m2;

    ll k = (a2 - a1) / g;
    ll mult = (1LL * x * k) % (m2 / g);

    ll ans = (a1 + m1 * mult) % lcm;
    if (ans < 0) ans += lcm;

    return {ans, lcm};
}

//solve a system of congruences:
//x = a1 (mod m1)
//x = a2 (mod m2)
//...
//x = ak (mod mk)
pair<ll, ll> crt(vector<ll> a, vector<ll> m) {
    pair<ll, ll> res = {a[0], m[0]};
    for (int i = 1; i < sz(a); i++) {
        res = crt2(res.first, res.second, a[i], m[i]);
        if (res.first == -1) return {-1, -1};
    }
    return res;
}

//x = sol.first (mod sol.second)

```

Rabin-Miller primality test

```

bool test(ll a, ll n, ll k, ll m){
    ll mod = Pow(a, m, n);
    if (mod == 1 || mod == n - 1) return 1;
    for (int l = 1; l < k; ++l){
        mod = (mod * mod) % n;
        if (mod == n - 1) return 1;
    }
    return 0;
}

//check if n is a prime number
bool RabinMiller(ll n){
    if (n == 2 || n == 3 || n == 5 || n == 7) return 1;
    if (n < 11) return 0;

    ll k = 0, m = n - 1;
    while (!(m & 1)){
        m >>= 1;
        k++;
    }

    const static int repeatTime = 3;
    for (int i = 0; i < repeatTime; ++i){
        ll a = rand() % (n - 3) + 2;
        if (!test(a, n, k, m)) return 0;
    }
    return 1;
}

```

Pollard's Rho prime factorization

- Average Time Complexity: $O(\sqrt[4]{N} \log^2(N))$
- Space Complexity: $O(\log N)$

```

using ll = long long;
using ull = unsigned long long;
using ld = long double;
ll mult(ll x, ll y, ll md) {
    ull q = (ld)x * y / md;
    ll res = ((ull)x * y - q * md);
    if (res >= md) res -= md;
    if (res < 0) res += md;
    return res;
}

ll powMod(ll x, ll p, ll md) {
    if (p == 0) return 1;
    if (p & 1) return mult(x, powMod(x, p - 1, md), md);
    return powMod(mult(x, x, md), p / 2, md);
}

bool checkMillerRabin(ll x, ll md, ll s, int k) {
    x = powMod(x, s, md);
    if (x == 1) return true;
    while (k--) {
        if (x == md - 1) return true;
        x = mult(x, x, md);
        if (x == 1) return false;
    }
    return false;
}

bool isPrime(ll x) {
    if (x == 2 || x == 3 || x == 5 || x == 7) return true;

```

```

    if (x % 2 == 0 || x % 3 == 0 || x % 5 == 0 || x % 7 == 0) return false;
    if (x < 121) return x > 1;
    ll s = x - 1;
    int k = 0;
    while (s % 2 == 0) {
        s >>= 1;
        k++;
    }
    if (x < 1LL << 32) {
        for (ll z : {2, 7, 61}) {
            if (!checkMillerRabin(z, x, s, k)) return false;
        }
    } else {
        for (ll z : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
            if (!checkMillerRabin(z, x, s, k)) return false;
        }
    }
    return true;
}

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
long long get_rand(long long r) {
    return uniform_int_distribution<long long>(0, r - 1)(rng);
}

```

```

void pollard(ll x, vector<ll> &ans) {
    if (isPrime(x)) {
        ans.push_back(x);
        return;
    }
    ll c = 1;
    while (true) {
        c = 1 + get_rand(x - 1);
        auto f = [&](ll y) {
            ll res = mult(y, y, x) + c;
            if (res >= x) res -= x;
            return res;
        };
        ll y = 2;
        int B = 100;
        int len = 1;
        ll g = 1;
        while (g == 1) {
            ll z = y;
            for (int i = 0; i < len; i++) {
                z = f(z);
            }
            ll zs = -1;
            int lft = len;
            while (g == 1 && lft > 0) {
                zs = z;
                ll p = 1;
                for (int i = 0; i < B && i < lft; i++) {
                    p = mult(p, abs(z - y), x);
                    z = f(z);
                }
                g = gcd(p, x);
                lft -= B;
            }
            if (g == 1) {
                y = z;
            }

```

```

        len <= 1;
        continue;
    }
    if (g == x) {
        g = 1;
        z = zs;
        while (g == 1) {
            g = gcd(abs(z - y), x);
            z = f(z);
        }
        if (g == x) break;
        assert(g != 1);
        pollard(g, ans);
        pollard(x / g, ans);
        return;
    }
}

// return list of all prime factors of x (can have
// duplicates)
vector<ll> factorize(ll x) {
    vector<ll> ans;
    for (ll p : {2, 3, 5, 7, 11, 13, 17, 19}) {
        while (x % p == 0) {
            x /= p;
            ans.push_back(p);
        }
    }
    if (x != 1) {
        pollard(x, ans);
    }
    sort(ans.begin(), ans.end());
    return ans;
}

// return pairs of (p, k) where x = product(p^k)
vector<pair<ll, int>> factorize_pk(ll x) {
    auto ps = factorize(x);
    ll last = -1, cnt = 0;
    vector<pair<ll, int>> res;
    for (auto p : ps) {
        if (p == last)
            ++cnt;
        else {
            if (last > 0) res.emplace_back(last, cnt);
            last = p;
            cnt = 1;
        }
    }
    if (cnt > 0) {
        res.emplace_back(last, cnt);
    }
    return res;
}

```

Multiplicative Function (Sieve)

- For all coprimes $n, m \in N$, we have $f(m, n) = f(m)f(n)$
- ex: $I(N) = 1$, $id(n) = n$, $id_k(n) = n^k$, $gcd(n, const)$, $\varphi(n)$, $\mu(n)$, $f_k(n) = \sum_{d|n} d^k$, $\mu(n)$
- Dirichlet Convolution: $h(n) = \sum_{d|n} f(d) \times g(n/d)$ is also a multiplicative function

```

const int MN = 1e6 + 11;

int sieve[MN];
pair<int, int> pk[MN];
int ndiv[MN];

int main() {
    for (int i = 2; i <= 1000; i++)
        if (!sieve[i]) {
            for (int j = i*i; j <= 1000000; j += i)
                sieve[j] = i;
        }

    ndiv[1] = 1;

    for (int i = 2; i <= 1000000; i++) {
        if (!sieve[i]) {
            pk[i] = make_pair(i, 1);
            ndiv[i] = 2;
        }
        else {
            int p = sieve[i];

            if (pk[i/p].first == p) {
                pk[i] = make_pair(p, pk[i/p].second + 1);
                ndiv[i] = pk[i].second + 1;
            }
            else {
                pk[i] = make_pair(-1, 0);
                int u = i, v = 1;
                while (u % p == 0) {
                    u /= p;
                    v = v * p;
                }
                ndiv[i] = ndiv[u] * ndiv[v];
            }
        }
    }
}

```

Multiplicative Function

```

int n;
int res = 1;
for (int i = 2; i*i <= n; i++) {
    if (n % i == 0) {
        int u = 1, k = 0;
        while (n % i == 0) {
            n /= i;
            u = u * i;
            k += 1;
        }
        res = res * f(i, k);
    }
}

if (n > 1) {
    res = res * f(n, 1);
}

```

Discrete Logarithm

- To find min x that $a^x \equiv b \pmod{m}$
- Find $ord_m(a) = \min x$ that $a^x \equiv 1 \pmod{m}$ in $O(\sqrt[4]{m} \log m)$

```
//gcd(a, m) == 1
int discrete_log_BSGS_coprime(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = 1LL * cur * a % m;
    }
    int step = binpow(a, n, m);
    for (int p = 1, f1 = 1; p <= n; p++) {
        f1 = 1LL * f1 * step % m;
        if (vals.count(f1)) {
            return n * p - vals[f1];
        }
    }
    return -1;
}

//gcd != 1
int discrete_log_BSGS(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;

    int k = 1, add = 0, g;
    while ((g = __gcd(a, m)) > 1) {
        if (b == k) return add;
        if (b % g) return -1;
        b /= g, m /= g, ++add;
        a %= m;
        k = (k * 1ll * a / g) % m;
    }
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = 1LL * cur * a % m;
    }
    int step = binpow(a, n, m);
    for (int p = 1, f1 = k; p <= n; p++) {
        f1 = 1LL * f1 * step % m;
        if (vals.count(f1)) {
            int ans = n * p - vals[f1] + add;
            return ans;
        }
    }
    return -1;
}

//ord_m(a):
using ll = long long;
ll powMod(ll x, ll p, ll md);
ll gcd(ll x, ll y);
vector<ll> factorize(ll x); //Pollard's Rho

ll phi(ll n) {
    auto ps = factorize(n);
    ll res = n;
    ll last = -1;
    for (auto p : ps) {
        if (p != last) {
            res = res / p * (p - 1);
            last = p;
        }
    }
    return res;
}

ll ord(ll a, ll m) {
    if (gcd(a, m) != 1) return -1;
```

```
ll res = phi(m);
auto ps = factorize(res);
for (auto p : ps)
    if (powMod(a, res / p, m) == 1) res /= p;
return res;
}
```

More

- $1^2 + \dots + n^2 = n(n+1)(2n+1)/6$
- $1^3 + \dots + n^3 = (n(n+1)/2)^2$
- $1^4 + \dots + n^4 = n(n+1)(2n+1)(3n^2+3n-1)/30$
- $1^5 + \dots + n^5 = n^2(n+1)^2(2n^2+2n-1)/12$

Geometry

Dot Product

```
//remember that u * v = 0 -> u is perpendicular with v
//or (u, v) = pi/2
double dotProduct(Vector u, Vector v){
    return u.x * v.x + u.y * v.y;
}
```

Angle

```
//u * v = |u| * |v| * cos(theta)
//-> theta = acos (u * v / (|u| * |v|))
double Cos(Vector u, Vector v){
    return dotProduct(u, v)/(u.len * v.len);
}

double theta(Vector u, Vector v){
    return acos(Cos(u, v));
}
```

Cross Product

```
//u * v = |u| * |v| * sin(theta)
//u * v = u.x * v.y - u.y * v.x
//|u * v| = area of a parallelogram formed by adjacent vectors u and v
//= double the area of the triangle
double crossProduct(Vector u, Vector v){
    return u.x * v.y - u.y * v.x;
}
```

Distance from a Point to a line

$$d(C, AB) = |\vec{AB} * \vec{AC}|/AB$$

```
// Compute the distance from AB to C
// if isSegment is true, AB is a segment, not a line.
double linePointDist(Point A, Point B, Point C, bool isSegment){
    double res = abs(cross(A, B, C)) / dist(A, B);
    if (isSegment){
        int dot1 = dot(B, A, C);
        if (dot1 < 0) return distance(B, C);
        int dot2 = dot(A, B, C);
        if (dot2 < 0) return distance(A, C);
    }
    return res;
}
```

Template 1

```
struct vec {
    db x, y;
    vec(db _x = 0, db _y = 0) : x(_x), y(_y) {}
    db dot(const vec &other) { // Compute the dot
        product
        return x * other.x + y * other.y;
    }
    db cross(const vec &other) { // Compute the cross
        product
        return x * other.y - y * other.x;
    }
    db length() const {
        return sqrt(x * x + y * y);
    }
};
using point = vec; // or use 'typedef vec point'
vec operator - (const point &B, const point &A) { //
    vecAB = B - A
    return vec(B.x - A.x, B.y - A.y);
}

// if isSegment is true, AB is a segment, not a line.
db linePointDist(const point &A, const point &B,
    const point &C, bool isSegment) {
    db dist = abs((B - A).cross(C - A)) / (A - B).
        length();
    if (isSegment) {
        db dot1 = (A - B).dot(C - B);
        if (dot1 < 0) return (B - C).length();
        db dot2 = (B - A).dot(C - A);
        if (dot2 < 0) return (A - C).length();
    }
    return dist;
}
```

Intersection of 2 lines and bla bla bla (I have no time bro)

Lines will have the form: $ax + by = c$.

$\vec{AB} \times \vec{AC} > 0 \Rightarrow A, B, C$ are counterclockwise.

$\vec{AB} \times \vec{AC} < 0 \Rightarrow A, B, C$ are clockwise.

$\vec{AB} \times \vec{AC} = 0 \Rightarrow A, B, C$ are collinear.

```
const double eps = 1e-9;
int sign(double x) {
    if (x > eps) return 1;
    if (x < -eps) return -1;
    return 0;
}
double cross(Vec AB, Vec AC) {
    return AB.x * AC.y - AC.x * AB.y;
}
double dot(Vec AB, Vec AC) {
    return AB.x * AC.x + AB.y * AC.y;
}
//intersection of 2 segments
bool intersect(Point A, Point B, Point C, Point D) {
    int ABxAC = sign(cross(B - A, C - A));
    int ABxAD = sign(cross(B - A, D - A));
    int CDxCA = sign(cross(D - C, A - C));
    int CDxCB = sign(cross(D - C, B - C));
    if (ABxAC == 0 || ABxAD == 0 || CDxCA == 0 ||
        CDxCB == 0) {
        // C on segment AB if ABxAC = 0 and CA.CB <=
        0
        if (ABxAC == 0 && sign(dot(A - C, B - C)) <=
            0) return true;
    }
```

```
if (ABxAD == 0 && sign(dot(A - D, B - D)) <=
    0) return true;
if (CDxCA == 0 && sign(dot(C - A, D - A)) <=
    0) return true;
if (CDxCB == 0 && sign(dot(C - B, D - B)) <=
    0) return true;
return false;
}
return (ABxAC * ABxAD < 0 && CDxCA * CDxCB < 0);
}
```

Circle passing through 3 points

```
struct Point {
    double x, y;
    Point() { x = y = 0.0; }
    Point(double x, double y) : x(x), y(y) {}

    Point operator + (const Point &a) const { return
        Point(x + a.x, y + a.y); }
    Point operator - (const Point &a) const { return
        Point(x - a.x, y - a.y); }
    Point operator * (double k) const { return Point(
        x * k, y * k); }
    Point operator / (double k) const { return Point(
        x / k, y / k); }
};

struct Line { // Ax + By = C
    double a, b, c;
    Line(double a = 0, double b = 0, double c = 0) :
        a(a), b(b), c(c) {}
    Line(Point A, Point B) {
        a = B.y - A.y;
        b = A.x - B.x;
        c = a * A.x + b * A.y;
    }
};

Line Perpendicular_Bisector(Point A, Point B) {
    Point M = (A + B) / 2;
    Line d = Line(A, B);
    // the equation of a perpendicular line has the
    form: -Bx + Ay = D
    double D = -d.b * M.x + d.a * M.y;
    return Line(-d.b, d.a, D);
}

//Intersection of 2 Perpendicular Bisector is the
center of the circle
```

Symmetry

```
struct Line { // Ax + By = C
    double a, b, c;
    Line(double a = 0, double b = 0, double c = 0) :
        a(a), b(b), c(c) {}
};

Point intersect(Line d1, Line d2) {
    double det = d1.a * d2.b - d2.a * d1.b;
    // det != 0 because d1 is perpendicular to d2
    return Point((d2.b * d1.c - d1.b * d2.c) / det, (
        d1.a * d2.c - d2.a * d1.c) / det);
}

Point Symmetry(Point X, Line d) {
    // the equation of a perpendicular line has the
    form: -Bx + Ay = D
```



```
double D = -d.b * X.x + d.a * X.y;
Line d2 = Line(-d.b, d.a, D);
Point Y = intersect(d, d2);
Point X2 = Point(2 * Y.x - X.x, 2 * Y.y - X.y);
return X2;
}
```

Rotation

To rotate $A(x, y)$ counterclockwise by an angle θ around the origin, we can easily use this formula:

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

```
Point Rotations(Point A, Point C, double rad) {
    Point A2 = A - C;
    Point B2 = Point(A2.x * cos(rad) - A2.y * sin(rad),
        A2.x * sin(rad) + A2.y * cos(rad));
    Point B = B2 + C;
    return B;
}
```

Area of a Polygon

```
double polygonArea(const vector<Point>& poly) {
    int n = poly.size();
    double area = 0.0;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        area += poly[i].x * poly[j].y - poly[j].x * poly[i].y;
    }
    return fabs(area) / 2.0;
}
```

Relative position of a point to a polygon $O(N)$

```
//using Area
//Time Complexity:  $O(N)$  per query
PointPolygonPosition position(Polygon plg, Point p) {
    long long sSumTris = 0;
    for (int i = 0; i < plg.nVertices; i++) {
        int i1 = (i + 1) % plg.nVertices;
        Polygon tri(p, plg.vertices[i], plg.vertices[i1]);
        auto sTri = tri.area2(); //2 * area
        if (!sTri) {
            return BOUNDARY;
        }
        sSumTris += sTri;
    }
    return (sSumTris == plg.area2() ? INSIDE : OUTSIDE);
}
```

Relative position of a point to a polygon $O(\log N)$

```
//using binary search
bool isCW(Point a, Point b, Point c) {
    return (Vector(a, b) ^ Vector(a, c)) < 0;
}

PointPolygonPosition position(Polygon plg, Point p) {
    // Check if P is on A_1A_n
    Vector pa1(p, plg.vertices[0]);
    Vector pan(p, plg.vertices[plg.nVertices - 1]);
    if (pa1 ^ pan == 0) { //cross product
```

```
if (1LL * pa1.x * pan.x <= 0) {
    return BOUNDARY;
}
return OUTSIDE;
}

int l = 1, r = plg.nVertices;
while (r - l > 1) {
    int mid = (l + r) >> 1;
    if (isCW(plg.vertices[0], p, plg.vertices[mid])) {
        l = mid;
    } else {
        r = mid;
    }
}
int k = l;
if (k == plg.nVertices - 1) {
    return OUTSIDE;
}

// Check if P is on the triangle
if (Vector(p, plg.vertices[k]) ^ Vector(p, plg.vertices[k + 1]) == 0) {
    return BOUNDARY;
}
long long ss = 0;
ss += Polygon(p, plg.vertices[0], plg.vertices[k]).area2();
ss += Polygon(p, plg.vertices[k], plg.vertices[k + 1]).area2();
ss += Polygon(p, plg.vertices[k + 1], plg.vertices[0]).area2();
if (ss == Polygon(plg.vertices[0], plg.vertices[k], plg.vertices[k + 1]).area2()) {
    return INSIDE;
}
return OUTSIDE;
}
```

Pick Theorem

$$A = I + B/2 - 1$$

- A = Area of the Polygon
- I = Number of interior lattice points (strictly inside the polygon)
- B = Number of boundary lattice points (on the polygon edges)

Convex Hull (Graham scan)

```
// Cross Product of AB and AC
long long cross(const Point &A, const Point &B, const Point &C) {
    return 1LL * (B.x - A.x) * (C.y - A.y) - 1LL * (C.x - A.x) * (B.y - A.y);
}

// A -> B -> C clockwise (-1), collinear (0), counterclockwise (1)
int ccw(const Point &A, const Point &B, const Point &C) {
    long long S = cross(A, B, C);
    if (S < 0) return -1;
    if (S == 0) return 0;
    return 1;
}
```

```

}

//convex hull listed in counterclockwise order
vector<Point> convexHull(vector<Point> p, int n) {
    for (int i = 1; i < n; ++i) {
        if (p[0].y > p[i].y || (p[0].y == p[i].y && p[0].x > p[i].x)) {
            swap(p[0], p[i]);
        }
    }

    sort(p.begin() + 1, p.end(), [&p](const Point &A, const Point &B) {
        int c = ccw(p[0], A, B);
        if (c > 0) return true;
        if (c < 0) return false;
        return A.x < B.x || (A.x == B.x && A.y < B.y);
    });

    vector<Point> hull;
    hull.push_back(p[0]);

    for (int i = 1; i < n; ++i) {
        while (hull.size() >= 2 && ccw(hull[hull.size() - 2], hull.back(), p[i]) < 0) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }
    return hull;
}

```

Convex Hull (Monotone chain algorithm)

```

bool ccw(const Point &A, const Point &B, const Point &C) {
    return 1LL * (B.x - A.x) * (C.y - A.y) - 1LL * (C.x - A.x) * (B.y - A.y) > 0;
}

vector<Point> convexHull(vector<Point> p, int n) {
    sort(p.begin(), p.end(), [](const Point &A, const Point &B) {
        if (A.x != B.x) return A.x < B.x;
        return A.y < B.y;
    });

    vector<Point> hull;
    hull.push_back(p[0]);

    for (int i = 1; i < n; ++i) {
        while (hull.size() >= 2 && ccw(hull[hull.size() - 2], hull.back(), p[i])) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }

    for (int i = n - 2; i >= 0; --i) {
        while (hull.size() >= 2 && ccw(hull[hull.size() - 2], hull.back(), p[i])) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }

    if (n > 1) hull.pop_back();
}

```

```

    return hull;
}

```

Find Fermat Point

- The Fermat Point is the point P such that the total distance PA+PB+PC is minimal.
- If one angle $\geq 120^\circ$, then the Fermat point is simply the vertex with the largest angle.
- else, the Fermat point P lies inside the triangle and each pair of lines PA, PB, PC forms a 120° angle.

```

//Local Search:
double len = 2000;
int N_ITERATION = 10000;
double RATE = 0.99;

for (int turn = 0; turn < N_ITERATION; turn++) {

    Point best = P;
    double bestDist = best PA + PB + PC;

    for (double angle = 0; angle < 2 * PI; angle += (2 * PI) / 100) {
        Point dir = Point(0, len).rotate(angle); //vector

        Point Q = P + dir; // Q = P'

        if (QA + QB + QC < bestDist) {
            best = Q;
            bestDist = QA + QB + QC;
        }

        P = best;
    }

    //easier way:
    P = centroid(A, B, C)
    step = max(dist(P, A), dist(P, B), dist(P, C))
    while step > eps:
        found = false
        for dir in 4 directions:
            Q = P + step * dir
            if f(Q) < f(P):
                P = Q
                found = true
        if not found:
            step *= 0.5

//for n points:
Point weiszfeld(vector<Point> &a, long double eps = 1e-9) {
    int n = a.size();
    Point p = {0, 0};
    for (auto &pt : a) p.x += pt.x, p.y += pt.y;
    p.x /= n; p.y /= n;
    while (true) {
        long double numx = 0, numy = 0, denom = 0;
        bool nearPoint = false;

        for (auto &pt : a) {
            long double d = dist(p, pt);
            if (d < eps) {
                p = pt;
                nearPoint = true;
            }
        }
    }
}

```

```

        break;
    }
    long double w = 1.0 / d;
    numx += pt.x * w;
    numy += pt.y * w;
    denom += w;
}

if (nearPoint) break;

Point np = {numx / denom, numy / denom};
if (dist(np, p) < eps) return np;
p = np;
}
return p;
}

//check 120
auto ang = [&](int i) {
    Point A = a[i], B = a[(i+1)%3], C = a[(i+2)%3];
    long double b = dist(A, C), c = dist(A, B), a_ =
        dist(B, C);
    long double cosA = (b*b + c*c - a_*a_) / (2*b*c);
    return acosl(cosA);
};
for (int i = 0; i < 3; i++) {
    if (ang(i) >= 2.0L * M_PI / 3.0L) {
        cout << fixed << setprecision(9)
            << a[i].x << " " << a[i].y << "\n";
        return 0;
    }
}
}

```

More

- Number of triangles with no polygon side (no two vertices adjacent): $n(n-4)(n-5)/6$.
- Number of intersection points of diagonals in a convex n -gon: $n(n-1)(n-2)(n-3)/24$
- Number of triangles formed by one diagonal + a third vertex: $\binom{n}{2}(n-4)$
- Partitions of the polygon into triangles using non-crossing diagonals: $C_{n-2} = \frac{1}{n-1} \binom{2n-4}{n-2}$ where C_k is the k -th Catalan number.
- Number of ways to choose a set of non-crossing diagonals C_{n-2}
- Number of ways to choose k non-crossing diagonals in a convex = $N(n, k) = \frac{1}{k+1} \binom{n-3k}{k}$.
- You are given a convex polygon with n sides. You then connect all of the points by drawing lines among the diagonals of the polygon, then cut through the lines all at once. The maximum number of pieces: $R(N) = (n-1)(n-2)(n^2 - 3n + 12)/24$
- With a circle: $+ = n$ (n edges).
- $a/\sin A = b/\sin B = c/\sin C = R$

- Menelaus: a line that cross BC, AC, AB at points D, E, F $\rightarrow AF/FB * BD/DC * CE/EA = 1$
- Ceva: AX, BY, CZ will be concurrent, if: $BX/XC * CY/YA * AZ/ZB = 1$
- Apollonius: Median line AD: $AB^2 + AC^2 = 2(AD^2 + BC^2/4)$
- Euler: $OI^2 = R(R - 2r)$
- Cyclic quadrilateral: $\angle A + \angle C = 180^\circ$, $AB * CD = AD * BC$, $AC * BD = AB * CD + AD * BC$
- Simson line: the feet of the perpendiculars from a point on a triangle's circumcircle to the sides of the triangle are collinear.
- Triangle: $S = \sqrt{p(p-a)(p-b)(p-c)}$
- Cyclic quadrilateral:
- $S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$

Algebra

Matrix Multiplication

Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

```

#define vi vector<int>
struct Matrix{
    vector<vi> a;
    int r, c;
    Matrix(){
        a.clear(); r = c = 0;
    }
    Matrix(vector<vi> a, int r, int c): a(a), r(r), c(c) {}
    Matrix operator * (const Matrix &B) const{
        vector<vi> res(r, vi(B.c, 0));
        vector<vi> b = B.a;

        rep(i, r){
            rep(j, B.c){
                rep(k, c){
                    res[i][j] += a[i][k] * b[k][j] % MOD;
                    res[i][j] %= MOD;
                }
            }
        }

        return Matrix(res, r, B.c);
    }
};

Matrix Pow(Matrix A, int b){
    vector<vi> a(A.r, vi(A.c, 0));
    rep(i, A.r) a[i][i] = 1;

    Matrix res(a, A.r, A.c);

```

```

while (b){
    if (b & 1) res = res * A;
    A = A * A;
    b >>= 1;
}

return res;
}

```

```

rep(i, n) res[i] = (int)(real(fa[i]) + 0.5);

return res;
}
}

```

Number Theory Transform

Fast Fourier Transform

```

namespace FFT{
#define cd complex<long double>
#define vc vector<cd>

const long double PI = acos(-1.0L);
const int N = 1e6 + 5;
int rev[N];

void fft(vc &a, bool inverse = 0){
    int n = sz(a);
    rep(i, n) if (i < rev[i]){
        swap(a[i], a[rev[i]]);
    }

    for (int len = 2; len <= n; len <= 1){
        cd wn = polar(1.0L, PI/len * (inverse ? -2
            : 2));
        for (int i = 0; i < n; i += len){
            cd w = 1;
            rep(j, len/2){
                cd u = a[i + j];
                cd v = a[i + j + len/2] * w;
                a[i + j] = u + v;
                a[i + j + len/2] = u - v;
                w *= wn;
            }
        }
    }

    if (inverse){
        for (cd &x: a){
            x /= n;
        }
    }
}

vi operator * (const vi &a, const vi &b){
    if (a.empty() || b.empty()) return {};

    vc fa(all(a));
    vc fb(all(b));

    int n = 1, L = 0;
    while (n < sz(a) + sz(b) - 1) n <= 1, ++L;
    rep(i, n){
        rev[i] = (rev[i >> 1] | (i & 1) << L) >>
            1;
    }

    fa.resize(n); fb.resize(n);

    fft(fa); fft(fb);

    rep(i, n) fa[i] *= fb[i];
    fft(fa, 1);

    n = sz(a) + sz(b) - 1;
    vi res(n);

```

```

namespace NTT{
const int MOD = 998244353;
const int g = 3; //primitive root
int rev[N];

void ntt(vi &a, bool inverse = 0){
    int n = sz(a);
    rep(i, n) if (i < rev[i]){
        swap(a[i], a[rev[i]]);
    }

    for (int len = 2; len <= n; len <= 1){
        int wn = Pow(g, (MOD - 1)/len);
        if (inverse) wn = Pow(wn, MOD - 2);

        for (int i = 0; i < n; i += len){
            int w = 1;
            rep(j, len/2){
                int u = a[i + j];
                int v = mul(w, a[i + j + len/2]);

                a[i + j] = sum(u, v);
                a[i + j + len/2] = dif(u, v);

                w = mul(w, wn);
            }
        }

        if (inverse){
            int div_n = Pow(n, MOD - 2);
            rep(i, n) a[i] = mul(a[i], div_n);
        }
    }

    vi operator * (const vi &a, const vi &b){
        if (a.empty() || b.empty()) return {};

        vi fa(all(a)), fb(all(b));

        int n = 1, L = 0;
        while (n < sz(a) + sz(b) - 1) n <= 1, ++L;
        rep(i, n){
            rev[i] = (rev[i >> 1] | (i & 1) << L) >>
                1;
        }

        fa.resize(n); fb.resize(n);
        ntt(fa); ntt(fb);

        rep(i, n) fa[i] = mul(fa[i], fb[i]);
        ntt(fa, 1);

        fa.resize(sz(a) + sz(b) - 1);
        return fa;
    }
}

```

Combinatoric

Formula

```
//DP version:
void preCompute(){
    for (int i = 0; i <= n; i++){
        C[i][0] = 1;
        for (int k = 1; k <= i; k++){
            C[i][k] = C[i - 1][k - 1] + C[i - 1][k];
        }
    }
}

// "you know what it is" version:
int C(int n, int k){
    if (n < k || k < 0) return 0;
    return mul(fact[n], mul(ifact[n - k], ifact[k]))
}
```

Catalan

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1} \binom{2n}{n}$$

Applications

- Number of ways to triangulate a convex polygon with $n + 2$ vertices.
- Number of Dyck words of length $2n$ (strings of n X and n Y, every prefix has $\#X \geq \#Y$).
- Number of valid parentheses sequences with n pairs.

For $n = 3$: $((()))$, $((()))$, $((()))$, $(()())$, $(())()$.

- Number of ways to parenthesize $(n + 1)$ factors.
Example ($n = 3$):

$((ab)c)d$, $(a(bc))d$, $(ab)(cd)$, $a((bc)d)$, $a(b(cd))$

- Number of full binary trees with n internal nodes.

Derangement

```
//Principle of Inclusion-Exclusion
int c = 1;
for (int i = 1; i <= n; i++) {
    c = (c * i) + (i % 2 == 1 ? -1 : 1);
    cout << c << ' ';
}

//DP
//dp[n] = (n - 1)(dp[n - 2] + dp[n - 1])
```

Classical Sums

$$\sum_{k=0}^n \binom{n}{k} = 2^n$$

$$\sum_{k=0}^n k \binom{n}{k} = n \cdot 2^{n-1}, \quad \sum_{k=0}^n k^2 \binom{n}{k} = n(n+1) \cdot 2^{n-2}$$

$$\sum_{k=0}^n (-1)^k \binom{n}{k} = 0 \quad (n > 0)$$

$$\sum_{k=0}^n \binom{n}{k}^2 = \binom{2n}{n}$$

$$\sum_{k=0}^r \binom{m}{k} \binom{n}{r-k} = \binom{m+n}{r} \quad (\text{Vandermonde})$$

Useful Identities

$$\sum_{k=r}^n \binom{k}{r} = \binom{n+1}{r+1} \quad (\text{Hockey-stick})$$

$$\sum_{k=0}^r (-1)^k \binom{r}{k} \binom{n+k}{m} = \binom{n}{m-r}$$

$$\binom{n}{k_1, k_2, \dots, k_m} = \frac{n!}{k_1! k_2! \dots k_m!}, \quad \sum k_i = n$$

Stirling Numbers

$$S(n, k) = S(n-1, k-1) + kS(n-1, k)$$

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

Others

Test?

```
ll rand(ll l, ll r){
    return rd() % (r - l + 1) + l;
}

int w[N];

main()
{
    // freopen(name".inp", "r", stdin); freopen(name".
    out", "w", stdout); w
    srand(time(0));
    int c = 0;
    ios_base::sync_with_stdio(false); cin.tie(0);
    cout.tie(0);
    for (int i = 1; i <= 1000; i++){
        ofstream fo("a.inp");
        int n = 1000;
        fo << n << endl;
        REP(i, n) fo << rand(1, 100000000) << endl;
        REP(i, n) fo << rand(1, 100000000) << endl;
        system("ac.exe");
        system("trau.exe");
        if (system("fc a1.out a2.out") == 1) exit(0);
    }
    return 0;
}
```

Big Int

```

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

struct BigInt {
    static const int base = 1000000000; // 1e9
    static const int base_digits = 9;
    vector<int> a; // little-endian (a[0] is least
        significant block)
    int sign;

    // Constructors
    BigInt(): sign(1) {}
    BigInt(long long v) { *this = v; }
    BigInt(const string &s) { read(s); }

    // Assign from integer
    BigInt& operator=(long long v) {
        sign = 1;
        if (v < 0) sign = -1, v = -v;
        a.clear();
        for (; v > 0; v /= base) a.push_back(v % base);
        return *this;
    }

    // Remove leading zeroes
    void trim() {
        while (!a.empty() && a.back() == 0) a.
            pop_back();
        if (a.empty()) sign = 1;
    }

    // Read from string
    void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int)s.size() && (s[pos] == '-' || s[pos] == '+')) {
            if (s[pos] == '-') sign = -sign;
            pos++;
        }
        for (int i = s.size()-1; i >= pos; i -=
            base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1)
                ; j <= i; j++)
                x = x * 10 + (s[j] - '0');
            a.push_back(x);
        }
        trim();
    }

    // Output
    friend ostream& operator<<(ostream &os, const
        BigInt &v) {
        if (v.sign == -1 && !v.isZero()) os << '-';
        if (v.a.empty()) { os << 0; return os; }
        os << v.a.back();
        for (int i = (int)v.a.size()-2; i >= 0; i--)
            os << setw(base_digits) << setfill('0') <<
                v.a[i];
        return os;
    }

    // Compare absolute values

```

```

static int cmpAbs(const BigInt &a, const BigInt &
    b) {
    if (a.a.size() != b.a.size()) return a.a.size
        () < b.a.size() ? -1 : 1;
    for (int i = (int)a.a.size()-1; i >= 0; i--)
        if (a.a[i] != b.a[i]) return a.a[i] < b.a[
            i] ? -1 : 1;
    return 0;
}

// Comparison operators
bool operator<(const BigInt &v) const {
    if (sign != v.sign) return sign < v.sign;
    int cmp = cmpAbs(*this, v);
    return sign == 1 ? cmp < 0 : cmp > 0;
}
bool operator==(const BigInt &v) const { return
    sign == v.sign && a == v.a; }
bool operator!=(const BigInt &v) const { return
    !(*this == v); }
bool operator>(const BigInt &v) const { return v
    < *this; }
bool operator<=(const BigInt &v) const { return
    !(v < *this); }
bool operator>=(const BigInt &v) const { return
    !(*this < v); }

bool isZero() const { return a.empty(); }

// Addition
BigInt operator+(const BigInt &v) const {
    if (sign == v.sign) {
        BigInt res = v;
        int carry = 0;
        for (size_t i = 0; i < max(a.size(), v.a.
            size()) || carry; i++) {
            if (i == res.a.size()) res.a.push_back
                (0);
            long long sum = res.a[i] + carry + (i
                < a.size() ? a[i] : 0);
            carry = sum >= base;
            if (carry) sum -= base;
            res.a[i] = sum;
        }
        return res;
    }
    return *this - (-v);
}

// Negation
BigInt operator-() const {
    BigInt res = *this;
    if (!res.isZero()) res.sign = -sign;
    return res;
}

// Subtraction
BigInt operator-(const BigInt &v) const {
    if (sign == v.sign) {
        if (cmpAbs(*this, v) >= 0) {
            BigInt res = *this;
            int carry = 0;
            for (size_t i = 0; i < v.a.size() ||
                carry; i++) {
                res.a[i] -= carry + (i < v.a.size()
                    ? v.a[i] : 0);
                carry = res.a[i] < 0;
                if (carry) res.a[i] += base;
            }

```

```

    }
    res.trim();
    return res;
}
return -(v - *this);
}
return *this + (-v);
}

// Multiplication
BigInt operator*(const BigInt &v) const {
    BigInt res;
    res.sign = sign * v.sign;
    res.a.assign(a.size()+v.a.size(), 0);
    for (size_t i = 0; i < a.size(); i++) {
        long long carry = 0;
        for (size_t j = 0; j < v.a.size() || carry; j++) {
            long long cur = res.a[i+j] + carry +
                1LL * a[i] * (j < v.a.size() ? v.a[j] : 0);
            res.a[i+j] = int(cur % base);
            carry = cur / base;
        }
    }
    res.trim();
    return res;
}

// Division and modulo
BigInt divmod(const BigInt &v, BigInt &rem) const
{
    int norm = base / (v.a.back() + 1);
    BigInt a = abs() * norm;
    BigInt b = v.abs() * norm;
    BigInt q; q.a.assign(a.a.size(), 0);
    rem = 0;
    rem.a.resize(a.a.size());
    for (int i = (int)a.a.size()-1; i >= 0; i--)
    {
        rem.shiftRight();
        rem.a[0] = a.a[i];
        rem.trim();
        int s1 = rem.a.size() <= b.a.size() ? 0 :
            rem.a[b.a.size()];
        int s2 = rem.a.size() <= b.a.size()-1 ? 0 :
            rem.a[b.a.size()-1];
        long long d = ((long long)base * s1 + s2) / b.a.back();
        BigInt tmp = b * d;
        while (rem < tmp) { d--; tmp = tmp - b; }
        rem = rem - tmp;
        q.a[i] = d;
    }
    q.sign = sign * v.sign;
    rem.sign = sign;
    q.trim();
    rem.trim();
    return q;
}

BigInt operator/(const BigInt &v) const {
    BigInt rem;
    return divmod(v, rem);
}

BigInt operator%(const BigInt &v) const {
    BigInt rem;

```

```

        divmod(v, rem);
        return rem;
    }

    // Helpers
    BigInt abs() const {
        BigInt res = *this;
        res.sign = 1;
        return res;
    }

    void shiftRight() {
        if (a.empty()) a.push_back(0);
        a.insert(a.begin(), 0);
    }
};

```

CODE::BLOCK Set Up

*

Settings -> Compiler... -> Selected compiler:
GNU GCC Compiler

Have g++ follow the C++17 ISO C++ language
standard [-std=c++17]

Enable all common compiler warnings (-Wall)

Enable extra compiler warnings (-Wextra)

-O2

Template

Settings -> Editor -> Abbreviations -> Add

Ctrl + J to use

CODE::BLOCK shortcuts

*

- Ctrl + Space → Autocomplete (symbols, functions, variables).
- Ctrl + Shift + C → Comment selected block.
- Ctrl + Shift + X → Uncomment selected block.
- Ctrl + D → Duplicate current line/selection.
- Ctrl + Shift + ↑ / ↓ → Move current line/selection up or down.
- Ctrl + L → Delete current line.
- Ctrl + Shift + K → Insert new line above current line.
- Ctrl + Shift + J → Insert new line below current line.
- Ctrl + G → Go to line.
- Ctrl + Shift + V → Paste without indentation (useful when pasting code from outside).

Multiplication Function

```
ll mult(ll x, ll y, ll md) {  
    ull q = (ld)x * y / md;  
    ll res = ((ull)x * y - q * md);  
    if (res >= md) res -= md;  
    if (res < 0) res += md;  
    return res;  
}
```

Random Function

```
mt19937_64 rng(chrono::steady_clock::now().  
    time_since_epoch().count());  
long long get_rand(long long r) {  
    return uniform_int_distribution<long long>(0, r -  
        1)(rng);  
}
```