

# Team Note of WhyWA

ehdnjs14, irhi2077, kilogram

Compiled on October 9, 2025

## Contents

### 1 빠른 입출력

#### 1.1 c++

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    return 0;
}
```

#### 1.2 python

```
import sys
# 트릭: 함수 이름 덮어씌우기
# input = sys.stdin.read()

sys.stdin.read()
sys.stdin.readline()
sys.stdout.write() # 기본 줄바꿈 \n 및 str만 가능

# 예제
print(sys.stdin.read()) # (입력 전체)
sys.stdout.write("Hello, World\n")
```

### 2 그래프/트리

#### 2.1 인접 리스트

```
graph = [[] for _ in range(N)]
def add(u, v): graph[u].append(v); graph[v].append(u)
```

#### 2.2 인접 리스트DFS

Time Complexity:  $O(V + E)$

```
def dfs(visit, current):
    visit[current] = True
    # do something
    for adjacent in graph[current]:
        if visit[adjacent]: continue
        dfs(visit, adjacent)
```

### 2.3 인접 리스트BFS

```
from collections import deque

def bfs(visit, init):
    queue = deque([init]) # mle: list로 바꿔볼 것
    visit[init] = True
    while queue:
        current = queue.popleft()
        # do something
        for adjacent in graph[current]:
            if visit[adjacent]: continue
            queue.append(adjacent)
```

### 2.4 위상 정렬(cpp)

Time Complexity:  $O(V + E)$

```
const int N = 1e5 + 9;
vector<int> g[N];
bool vi[N];
vector<int> ord;
void dfs(int u) {
    vi[u] = true;
    for (auto v : g[u]) {
        if (!vi[v]) {
            dfs(v);
        }
    }
    ord.push_back(u);
}

void find(int n) {
    for (int i = 1; i <= n; i++) {
        if (!vi[i]) {
            dfs(i);
        }
    }
    reverse(ord.begin(), ord.end());
}

// check is feasible
vector<int> pos(n + 1);
for (int i = 0; i < (int) ord.size(); i++) {
    pos[ord[i]] = i;
}
for (int u = 1; u <= n; u++) {
    for (auto v : g[u]) {
```

```
        // We need to consider self loops too!
        if (pos[u] >= pos[v]) {
            //print impossible
            return;
        }
    }
}

// print the order
for (auto u : ord) cout << u << ' ';
```

### 2.5 위상 정렬(py)

Time Complexity:  $O(V + E)$

```
from collections import deque

def kahn():
    # build in-degrees
    indeg = [0 for _ in range(N)]
    for node in range(N):
        for adj in graph[node]:
            indeg[adj] += 1

    queue = deque([])
    order = []

    # all nodes with 0-in-degree
    for node in range(N):
        if indeg[node] == 0: queue.append(node)

    while queue:
        curr = queue.popleft()
        order.append(curr)

        for node in graph[curr]:
            indeg[node] -= 1
            if indeg[node] == 0: queue.append(node)

    # has cycle ?
    if len(order) != N: raise CycleError

    return order
```

### 2.6 최단 거리 - Floyd Warshall

Time Complexity:  $O(V^3)$

```
# 가중 그래프
graph = [[] for _ in range(N)]

def add(src, dst, weight):
    graph[src].append((dst, weight))
```

```

def floyd_marshall():
    result = [[float('inf') for _ in range(N)] for _ in range(N)]

    # same node
    for n in range(N): result[n][n] = 0

    for u in range(N):
        for v, w in graph[u]:
            result[u][v] = w

    # for 문 순서 중요! 바꾸지 말 것!
    for m in range(N):
        for s in range(N):
            for e in range(N):
                result[s][e] = min(result[s][e], result[s][m] +
                    result[m][e])

    return result

```

## 2.7 최단 거리 - Dijkstra(cpp)

Time Complexity:  $O(E \log E)$

```

const int N = 3e5 + 9, mod = 998244353;

// you can delete 'cnt' vector if you don't need that
int n, m;
vector<pair<int, int>> g[N], r[N];
vector<long long> dijkstra(int s, int t, vector<int> &cnt) {
    const long long inf = 1e18;
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long
    long, int>> q;
    vector<long long> d(n + 1, inf);
    vector<bool> vis(n + 1, 0);
    q.push({0, s});
    d[s] = 0;
    cnt.resize(n + 1, 0); // number of shortest paths
    cnt[s] = 1;
    while(!q.empty()) {
        auto x = q.top();
        q.pop();
        int u = x.second;
        if(vis[u]) continue;
        vis[u] = 1;
        for(auto y: g[u]) {
            int v = y.first;
            long long w = y.second;
            if(d[u] + w < d[v]) {
                d[v] = d[u] + w;
                q.push({d[v], v});
                cnt[v] = cnt[u];
            } else if(d[u] + w == d[v]) cnt[v] = (cnt[v] + cnt[u]) % mod;
        }
    }
}

```

```

    }
    return d;
}

int u[N], v[N], w[N];

void solve(int n, int m, int s, int t) {
    for(int i = 1; i <= m; i++) {
        cin >> u[i] >> v[i] >> w[i];
        g[u[i]].push_back({v[i], w[i]});
        r[v[i]].push_back({u[i], w[i]});
    }
    vector<int> cnt1, cnt2;
    auto d1 = dijkstra(s, t, cnt1);
    auto d2 = dijkstra(t, s, cnt2);

    long long ans = d1[t];
    for(int i = 1; i <= m; i++) {
        int x = u[i], y = v[i];
        long long nw = d1[x] + w[i] + d2[y];
        if(nw == ans && 1LL * cnt1[x] * cnt2[y] % mod == cnt1[t]) //YES
        else if(nw - ans + 1 < w[i]) // print nw - ans + 1
        else // No
    }
}

```

## 2.8 최단 거리 - Dijkstra(py)

Time Complexity:  $O(E \log E)$

```

import heapq

graph = [[(v, w)] for _ in range(N)]
# 양의 가중치 only

def dijkstra(init):
    dist = [float('inf') for _ in range(N)]
    prev = [None for _ in range(N)]
    dist[init] = 0

    heap = [(0, init)]

    while heap:
        newdst, node = heapq.heappop(heap)
        if newdst != dist[node]: continue

        for node, weight in graph[node]:
            alt = newdst + weight

            if alt < dist[v]:
                dist[v] = alt
                prev[v] = u
                heapq.heappush(heap, (alt, v))

```

```
    return dist, prev
```

## 2.9 최단 거리 - Bellman Ford

Time Complexity:  $O(VE)$

```
graph = [[w for _ in range(N)] for _ in range(N)]

def bellman_ford(init):
    dist = [float('inf') for _ in range(N)]
    parent = [None for _ in range(N)]

    dist[init] = 0

    for u in range(N):
        for v in graph[u]:
            if dist[u] + graph[u][v] < dist[v]:
                dist[v] = dist[u] + graph[u][v]
                parent[v] = u

    for u in range(N):
        for v in range(N):
            if not graph[u][v]: continue

            if dist[u] + graph[u][v] < dist[v]:
                raise NegativeWeight

    return dist, parent
```

## 2.10 유니온 파인드 + 최소 신장 트리(Kruskal)

Time Complexity: UF: 연산마다  $O(\log N)$ , MST:  $O(E \log E)$

```
# no cycle!

graph = [[(v, w)] for _ in range(N)]

def kruskal():
    cost, mst = 0, []

    for u in range(N):
        for v, w in graph[u]:
            if disjoint.root(u) != disjoint.root(v):
                disjoint.union(u, v)
                mst.append((u, v))
                cost += w

    return cost, mst
```

## 2.11 SCC - Kosaraju

Time Complexity:  $O(V + E)$

```
const int N = 3e5 + 9;

// given a directed graph return the minimum number of edges to be added so that the whole
// graph become an SCC
// you need to restore original graph in g and reverse graph in r.
bool vis[N];
vector<int> g[N], r[N], G[N], vec; //G is the condensed graph
void dfs1(int u) {
    vis[u] = 1;
    for(auto v : g[u]) if (!vis[v]) dfs1(v);
    vec.push_back(u);
}

vector<int> comp;
void dfs2(int u) {
    comp.push_back(u);
    vis[u] = 1;
    for(auto v : r[u]) if (!vis[v]) dfs2(v);
}

int idx[N], in[N], out[N];

void find(int n) {
    for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(i);
    reverse(vec.begin(), vec.end());
    memset(vis, 0, sizeof(vis));
    int scc = 0;
    for(auto u : vec) {
        if(!vis[u]) {
            comp.clear();
            dfs2(u);
            scc++;
            for(auto x : comp) idx[x] = scc;
        }
    }
    for(int u = 1; u <= n; u++) {
        for(auto v : g[u]) {
            if(idx[u] != idx[v]) {
                in[idx[v]]++, out[idx[u]]++;
                G[idx[u]].push_back(idx[v]);
            }
        }
    }
    int needed_in = 0, needed_out = 0;
    for(int i = 1; i <= scc; i++) {
        if(!in[i]) needed_in++;
        if(!out[i]) needed_out++;
    }
    int ans = max(needed_in, needed_out);
    if(scc == 1) ans = 0;
    cout << ans;
}
```

## 2.12 Bridge Tree and Articulation Bridges

```
// you should write TECC t(g) int main funtion after that you add edges in graph g.
struct TECC { // 0 indexed
    int n, k;
    vector<vector<int>> g, t;
    vector<bool> used;
    vector<int> comp, ord, low;
    using edge = pair<int, int>;
    vector<edge> br;
    void dfs(int x, int prv, int &c) {
        used[x] = 1; ord[x] = c++; low[x] = n;
        bool mul = 0;
        for (auto y : g[x]) {
            if (used[y]) {
                if (y != prv || mul) low[x] = min(low[x], ord[y]);
                else mul = 1;
                continue;
            }
            dfs(y, x, c);
            low[x] = min(low[x], low[y]);
        }
    }
    void dfs2(int x, int num) {
        comp[x] = num;
        for (auto y : g[x]) {
            if (comp[y] != -1) continue;
            if (ord[x] < low[y]) {
                br.push_back({x, y});
                k++;
                dfs2(y, k);
            } else dfs2(y, num);
        }
    }
    TECC(const vector<vector<int>> &g): g(g), n(g.size()), used(n), comp(n, -1), ord(n),
    low(n), k(0) {
        int c = 0;
        for (int i = 0; i < n; i++) {
            if (used[i]) continue;
            dfs(i, -1, c);
            dfs2(i, k);
            k++;
        }
    }
    // build bridges tree
    void build_tree() {
        t.resize(k);
        for (auto e : br) {
            int x = comp[e.first], y = comp[e.second];
            t[x].push_back(y);
            t[y].push_back(x);
        }
    }
};
```

```
void solve(int n, int m) {
    vector<vector<int>> g(n);
    for (int i = 0; i < m; i++) {
        int a, b; cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    TECC t(g);
    vector<vector<int>> ans(t.k);
    for (int i = 0; i < n; i++) {
        ans[t.comp[i]].push_back(i);
    }

    cout << ans.size() << '\n';
    for (auto x : ans) {
        cout << x.size();
        for (auto y : x) cout << ' ' << y;
        cout << '\n';
    }
}
```

## 2.13 Articulation Points

```
const int N = 3e5 + 9;

// art = check an Articulation point.
// low = maybe restore parent? and update parent of the current node to min(low[u], low[v])
// dis = check whether a current vertex is visited or not.
int T, low[N], dis[N], art[N];
vector<int> g[N];
void dfs(int u, int pre = 0) {
    low[u] = dis[u] = ++T;
    int child = 0;
    for(auto v : g[u]) {
        if(!dis[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dis[u] && pre != 0) art[u] = 1;
            ++child;
        }
        else if(v != pre) low[u] = min(low[u], dis[v]);
    }
    if(pre == 0 && child > 1) art[u] = 1;
}
```

## 2.14 2 - sat

```
const int N = 3e5 + 9, mod = 998244353;

// you can delete 'cnt' vector if you don't need that
int n, m;
vector<pair<int, int>> g[N], r[N];
vector<long long> dijkstra(int s, int t, vector<int> &cnt) {
    const long long inf = 1e18;
```

```

priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> q;
vector<long long> d(n + 1, inf);
vector<bool> vis(n + 1, 0);
q.push({0, s});
d[s] = 0;
cnt.resize(n + 1, 0); // number of shortest paths
cnt[s] = 1;
while(!q.empty()) {
    auto x = q.top();
    q.pop();
    int u = x.second;
    if(vis[u]) continue;
    vis[u] = 1;
    for(auto y: g[u]) {
        int v = y.first;
        long long w = y.second;
        if(d[u] + w < d[v]) {
            d[v] = d[u] + w;
            q.push({d[v], v});
            cnt[v] = cnt[u];
        } else if(d[u] + w == d[v]) cnt[v] = (cnt[v] + cnt[u]) % mod;
    }
}
return d;
}

int u[N], v[N], w[N];

void solve(int n, int m, int s, int t) {
    for(int i = 1; i <= m; i++) {
        cin >> u[i] >> v[i] >> w[i];
        g[u[i]].push_back({v[i], w[i]});
        r[v[i]].push_back({u[i], w[i]});
    }
    vector<int> cnt1, cnt2;
    auto d1 = dijkstra(s, t, cnt1);
    auto d2 = dijkstra(t, s, cnt2);

    long long ans = d1[t];
    for(int i = 1; i <= m; i++) {
        int x = u[i], y = v[i];
        long long nw = d1[x] + w[i] + d2[y];
        if(nw == ans && 1LL * cnt1[x] * cnt2[y] % mod == cnt1[t]) //YES
        else if(nw - ans + 1 < w[i]) // print nw - ans + 1
        else // No
    }
}

```

## 2.15 최소 공통 조상(LCA) and Binary Lifting

Time Complexity: 전처리  $O(N \log N)$ , 쿼리  $O(\log N)$

```

const int N = 3e5 + 9, LG = 18;

vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
// don't forget to call dfs(1)
void dfs(int u, int p = 0) {
    par[u][0] = p;
    dep[u] = dep[p] + 1;
    sz[u] = 1;
    for (int i = 1; i <= LG; i++) par[u][i] = par[par[u][i - 1]][i - 1];
    for (auto v : g[u]) {
        if (v != p) {
            dfs(v, u);
            sz[u] += sz[v];
        }
    }
}
int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    for (int k = LG; k >= 0; k--) if (dep[par[u][k]] >= dep[v]) u = par[u][k];
    if (u == v) return u;
    for (int k = LG; k >= 0; k--) if (par[u][k] != par[v][k]) u = par[u][k], v = par[v][k];
    return par[u][0];
}
// 'kth' function is a Binary Lifting
int kth(int u, int k) {
    assert(k >= 0);
    for (int i = 0; i <= LG; i++) if (k & (1 << i)) u = par[u][i];
    return u;
}
int dist(int u, int v) {
    int l = lca(u, v);
    return dep[u] + dep[v] - (dep[l] << 1);
}
// kth node from u to v, 0th node is u
int go(int u, int v, int k) {
    int l = lca(u, v);
    int d = dep[u] + dep[v] - (dep[l] << 1);
    assert(k <= d);
    if (dep[l] + k <= dep[u]) return kth(u, k);
    k -= dep[u] - dep[l];
    return kth(v, dep[v] - dep[l] - k);
}



## 2.16 Cycle Detection


// It is possible both undigraph and digraph.
const int N = 5e5 + 9;

vector<pair<int, int>> g[N];
int vis[N], par[N], e_id[N];
vector<int> cycle; // simple cycle, contains edge ids

bool dfs(int u) {

```

```

if (!cycle.empty()) return 1;
vis[u] = 1;
for (auto [v, id] : g[u]) {
    if (v != par[u]) {
        if (vis[v] == 0) {
            par[v] = u;
            e_id[v] = id;
            if (dfs(v)) return 1;
        }
        else if (vis[v] == 1) {
            // cycle here
            cycle.push_back(id);
            for (int x = u; x != v; x = par[x]) {
                cycle.push_back(e_id[x]);
            }
            return 1;
        }
    }
}
vis[u] = 2;
return 0;
}

void solve(int n, int m) {
    for (int i = 1; i < m + 1; i++) {
        int u, v;
        cin >> u >> v;
        u++;
        v++;
        g[u].push_back({v, i});
    }
    for (int u = 1; u < n + 1; u++) {
        if (vis[u] == 0 && dfs(u)) {
            // this graph is acyclic
            for (auto x : cycle) {
                // any cycle
            }
            return;
        }
    }
}
}

```

## 2.17 Euler Path Directed

```

const int N = 4e5 + 9;

/*
all the edges should be in the same connected component
#directed graph: euler path: for all -> indeg = outdeg or nodes having indeg > outdeg =
outdeg > indeg = 1 and for others in = out
#directed graph: euler circuit: for all -> indeg = outdeg
*/

//euler path in a directed graph
//it also finds circuit if it exists

```

```

vector<int> g[N], ans;
int done[N];
void dfs(int u) {
    while (done[u] < g[u].size()) dfs(g[u][done[u]++]);
    ans.push_back(u);
}

int find(int n) {
    int edges = 0;
    vector<int> in(n + 1, 0), out(n + 1, 0);
    for (int u = 1; u <= n; u++) {
        for (auto v : g[u]) in[v]++;
        out[u]++;
        edges++;
    }
    int ok = 1, cnt1 = 0, cnt2 = 0, root = 0;
    for (int i = 1; i <= n; i++) {
        if (in[i] - out[i] == 1) cnt1++;
        if (out[i] - in[i] == 1) cnt2++, root = i;
        if (abs(in[i] - out[i]) > 1) ok = 0;
    }
    if (cnt1 > 1 || cnt2 > 1) ok = 0;
    if (!ok) return 0;
    if (root == 0) {
        for (int i = 1; i <= n; i++) if (out[i]) root = i;
    }
    if (root == 0) return 1; //empty graph
    dfs(root);
    if (ans.size() != edges + 1) return 0; //connectivity
    reverse(ans.begin(), ans.end());
    return 1;
}

void solve(int n, int m) {
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
    }
    int ok = find(n);
    if (!ok) {
        // No
        return;
    }
    // Yes
    // The answer is in ans
}

```

## 2.18 Euler Path Undirected

```

const int N = 5e3 + 9;

/*
all the edges should be in the same connected component
#undirected graph: euler path: all degrees are even or exactly two of them are odd.
#undirected graph: euler circuit: all degrees are even

```

```

*/
//euler path in an undirected graph
//it also finds circuit if it exists
vector<pair<int, int>> g[N];
vector<int> ans;
int done[N];
int vis[N * N]; //number of edges
void dfs(int u) {
    while (done[u] < g[u].size()) {
        auto e = g[u][done[u]++;
        if (vis[e.second]) continue;
        vis[e.second] = 1;
        dfs(e.first);
    }
    ans.push_back(u);
}

int find(int n) {
    int edges = 0;
    ans.clear();
    memset(done, 0, sizeof(done));
    memset(vis, 0, sizeof(vis));
    vector<int> deg(n + 1, 0);
    for (int u = 1; u <= n; u++) {
        for (auto e : g[u]) {
            deg[e.first]++;
            deg[u]++;
            edges++;
        }
    }
    int odd = 0, root = 0;
    for (int i = 1; i <= n; i++) {
        if (deg[i] & 1) odd++, root = i;
    }
    if (odd > 2) return 0;
    if (root == 0) {
        for (int i = 1; i <= n; i++) if (deg[i]) root = i;
    }
    if (root == 0) return 1; //empty graph
    dfs(root);
    if (ans.size() != edges / 2 + 1) return 0;
    reverse(ans.begin(), ans.end());
    return 1;
}

void solve(int n, int m) {
    vector<int> deg(n + 1, 0);
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back({v, i});
        g[v].push_back({u, i});
        deg[u]++, deg[v]++;
    }
}

```

```

int sz = m;
for (int i = 1; i <= n; i++) {
    if (deg[i] & 1) {
        ++sz;
        g[n + 1].push_back({i, sz});
        g[i].push_back({n + 1, sz});
    }
}
int ok = find(n + 1);
assert(ok);
// indegree, outdegree
vector<int> in(n + 2, 0), out(n + 2, 0);
for (int i = 0; i + 1 < ans.size(); i++) {
    if (ans[i] != n + 1 && ans[i + 1] != n + 1) {
        in[ans[i + 1]]++;
        out[ans[i]]++;
    }
}
// The answer is in in, out, ans
}



## 2.19 Tree Diameter



```

const int N = 2e5 + 9;

vector<int> g[N];
int farthest(int s, int n, vector<int> &d) {
    static const int inf = N;
    d.assign(n + 1, inf); d[s] = 0;
    vector<bool> vis(n + 1);
    queue<int> q; q.push(s);
    vis[s] = 1; int last = s;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : g[u]) {
            if (vis[v]) continue;
            d[v] = d[u] + 1;
            q.push(v); vis[v] = 1;
        }
        last = u;
    }
    return last;
}

void solve(int n) {
    for (int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    vector<int> dx, dy;
    int x = farthest(1, n, dx);
    int y = farthest(x, n, dx);
    farthest(y, n, dy);
}

```


```

```

int ans = 0;
for (int i = 1; i < n + 1; i++) ans = max(dy[i], ans);
// print ans.
}

```

### 3 자료구조

#### 3.1 세그먼트 트리

Time Complexity:  $O(\log N)$

```

const int N = 3e5 + 9;
int a[N];
struct ST {
    int t[4 * N];
    static const int inf = 1e9;
    ST() {
        memset(t, 0, sizeof t);
    }
    void build(int n, int b, int e) {
        if (b == e) {
            t[n] = a[b];
            return;
        }
        int mid = (b + e) >> 1, l = n << 1, r = l | 1;
        build(l, b, mid);
        build(r, mid + 1, e);
        t[n] = max(t[l], t[r]);
    }
    void upd(int n, int b, int e, int i, int x) {
        if (b > i || e < i) return;
        if (b == e && b == i) {
            t[n] = x;
            return;
        }
        int mid = (b + e) >> 1, l = n << 1, r = l | 1;
        upd(l, b, mid, i, x);
        upd(r, mid + 1, e, i, x);
        t[n] = max(t[l], t[r]);
    }
    int query(int n, int b, int e, int i, int j) {
        if (b > j || e < i) return -inf;
        if (b >= i && e <= j) return t[n];
        int mid = (b + e) >> 1, l = n << 1, r = l | 1;
        int L = query(l, b, mid, i, j);
        int R = query(r, mid + 1, e, i, j);
        return max(L, R);
    }
}t;

```

#### 3.2 세그먼트 트리 + 레이지 프로퍼게이션

Time Complexity:  $O(\log N)$

```

const int N = 5e5 + 9;
int a[N];
struct ST {
#define lc (n << 1)
#define rc ((n << 1) | 1)
    long long t[4 * N], lazy[4 * N];
    ST() {
        memset(t, 0, sizeof t);
        memset(lazy, 0, sizeof lazy);
    }
    inline void push(int n, int b, int e) {
        if (lazy[n] == 0) return;
        t[n] = t[n] + lazy[n] * (e - b + 1);
        if (b != e) {
            lazy[lc] = lazy[lc] + lazy[n];
            lazy[rc] = lazy[rc] + lazy[n];
        }
        lazy[n] = 0;
    }
    inline long long combine(long long a, long long b) {
        return a + b;
    }
    inline void pull(int n) {
        t[n] = t[lc] + t[rc];
    }
    void build(int n, int b, int e) {
        lazy[n] = 0;
        if (b == e) {
            t[n] = a[b];
            return;
        }
        int mid = (b + e) >> 1;
        build(lc, b, mid);
        build(rc, mid + 1, e);
        pull(n);
    }
    void upd(int n, int b, int e, int i, int j, long long v) {
        push(n, b, e);
        if (j < b || e < i) return;
        if (i <= b && e <= j) {
            lazy[n] = v; //set lazy
            push(n, b, e);
            return;
        }
        int mid = (b + e) >> 1;
        upd(lc, b, mid, i, j, v);
        upd(rc, mid + 1, e, i, j, v);
        pull(n);
    }
    long long query(int n, int b, int e, int i, int j) {
        push(n, b, e);
        if (i > e || b > j) return 0; //return null
        if (i <= b && e <= j) return t[n];
    }
}

```

```

    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc, mid + 1, e, i, j));
}
};

```

### 3.3 DFS Tree

```

const int N = 3e5 + 5;

int bridge = 0;
vector<int> adj[N];
int lv[N];
int dp[N];

void dfs (int v) {
    dp[v] = 0;
    for (int u : adj[v]) {
        if (lv[u] == 0) {
            lv[u] = lv[v] + 1;
            dfs(u);
            dp[v] += dp[u];
        } else if (lv[u] < lv[v]) {
            dp[v]++;
        } else if (lv[u] > lv[v]) {
            dp[v]--;
        }
    }
    dp[v]--;
    if (lv[v] > 1 && dp[v] == 0) {
        bridge++;
    }
}

```

### 3.4 Convex Hull Trick

Usage: call init() before use

```

// 직선 개수 N, 쿼리 횟수 Q일 때 O(N+Q)
// (최댓값 쿼리) 삽입하는 직선의 기울기는 단조 증가해야 함
// (최솟값 쿼리) 삽입하는 직선의 기울기는 단조 감소해야 함
// 쿼리를 하는 x좌표는 단조 증가해야 함
struct Line{
    ll a, b, c; // y = ax + b, c = line index
    Line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
    ll f(ll x){ return a * x + b; }
};
vector<Line> v; int pv;
void init(){ v.clear(); pv = 0; }
int chk(const Line &a, const Line &b, const Line &c) const {
    return (_int128_t)(a.b - b.b) * (b.a - c.a) <= (_int128_t)(c.b - b.b) * (b.a - a.a);
}
void insert(Line l){

```

```

    if(v.size() > pv && v.back().a == l.a){
        // if min query, then if(l.b > v.back().b)
        if(l.b < v.back().b) l = v.back();
        v.pop_back();
    }
    while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l)) v.pop_back();
    v.push_back(l);
}
p query(ll x){ // if min query, then v[pv].f(x) >= v[pv+1].f(x)
    while(pv+1 < v.size() && v[pv].f(x) <= v[pv+1].f(x)) pv++;
    return {v[pv].f(x), v[pv].c};
}
//// line container start (max query) //////
// 직선 개수 N 쿼리 개수 Q일 때 O((N+Q) log N)
// 단조성 제약 조건 없음
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; }
}; // (for doubles, use inf = 1/.0, div(a,b) = a/b)
struct LineContainer : multiset<Line, less<> {
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); } // floor
    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;
    }
};

```

### 3.5 퍼시스턴트 세그먼트 트리

Usage: call init(root[0], s, e) before use

```

struct PSTNode{
    PSTNode *l, *r; int v;
    PSTNode(): l = r = nullptr, v = 0 {}
};
PSTNode *root[101010];
PST(): memset(root, 0, sizeof root); // constructor
void init(PSTNode *node, int s, int e){
    if(s == e) return;
    int m = s + e >> 1;

```

```

node->l = new PSTNode; node->r = new PSTNode;
init(node->l, s, m); init(node->r, m+1, e);
}

void update(PSTNode *prv, PSTNode *now, int s, int e, int x){
    if(s == e){ now->v = prv->v ? prv->v + 1 : 1; return; }
    int m = s + e >> 1;
    if(x <= m){
        now->l = new PSTNode; now->r = prv->r;
        update(prv->l, now->l, s, m, x);
    } else{
        now->r = new PSTNode; now->l = prv->l;
        update(prv->r, now->r, m+1, e, x);
    }
    int t1 = now->l ? now->l->v : 0;
    int t2 = now->r ? now->r->v : 0;
    now->v = t1 + t2;
}

int kth(PSTNode *prv, PSTNode *now, int s, int e, int k){
    if(s == e) return s;
    int m = s + e >> 1, diff = now->l->v - prv->l->v;
    if(k <= diff) return kth(prv->l, now->l, s, m, k);
    else return kth(prv->r, now->r, m+1, e, k-diff);
}

```

### 3.6 펜윅 트리

Time Complexity:  $O(\log N)$

```

template <class T>
struct fenwick { //1-indexed
    int n; vector<T> t;
    fenwick() {}
    fenwick(int _n) {
        n = _n; t.assign(n + 1, 0);
    }
    T query(int i) {
        T ans = 0;
        for (; i >= 1; i -= (i & -i)) ans += t[i];
        return ans;
    }
    void upd(int i, T val) {
        if (i <= 0) return;
        for (; i <= n; i += (i & -i)) t[i] += val;
    }
    void upd(int l, int r, T val) {
        upd(l, val);
        upd(r + 1, -val);
    }
    T query(int l, int r) {
        return query(r) - query(l - 1);
    }
};

```

### 3.7 Disjoint Set Union + MST

```

struct DSU {
    vector<int> par, rnk, sz;
    int c;
    DSU(int n) : par(n + 1), rnk(n + 1, 0), sz(n + 1, 1), c(n) {
        for (int i = 1; i <= n; ++i) par[i] = i;
    }
    int find(int i) {
        return (par[i] == i ? i : (par[i] = find(par[i])));
    }
    bool same(int i, int j) {
        return find(i) == find(j);
    }
    int get_size(int i) {
        return sz[find(i)];
    }
    int count() {
        return c;
    }
    int unite(int i, int j) {
        if ((i = find(i)) == (j = find(j))) return -1;
        else --c;
        if (rnk[i] > rnk[j]) swap(i, j);
        par[i] = j;
        sz[j] += sz[i];
        if (rnk[i] == rnk[j]) rnk[j]++;
        return j;
    }
};

void mst(int n, int m) {
    vector<array<int, 3>> ed;
    for (int i = 1; i < m + 1; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        ed.push_back({w, u, v});
    }
    // if you want to find Maximum Spanning Tree,
    // then you should sort all edge of graph
    // in decreasing order of weights.
    sort(ed.begin(), ed.end());
    ll ans = 0;
    DSU d(n);
    for (auto e : ed) {
        int u = e[1], v = e[2], w = e[0];
        if (d.same(u, v)) continue;
        ans += w;
        d.unite(u, v);
    }
    //print ans
}

```

## 4 수학

### 4.1 (python) 순열과 조합

```
import itertools
a=[0]*N

b=list(itertools.combinations(a,r)) #조합
c=list(itertools.permutations(a,r)) #순열
d=list(itertools.product(a,repeat=r)) #중복순열
e=list(itertools.combinations_with_replacement(a,r)) #중복조합
```

### 4.2 나눗셈, 최대공약수, 최소공배수

```
// floor(p / q)
int floor(int p, int q){
    if(q < 0) p = -p, q = -q;
    return p >= 0 ? p / q : (p - q + 1) / q;
}

// ceil(p / q)
int ceil(int p, int q){
    if(q < 0) p = -p, q = -q;
    return p >= 0 ? (p + q - 1) / q : p / q;
}

// a, b >= 0, O(log max(a,b))
// or use std::gcd, std::lcm in C++17
int gcd(int a, int b){ return b ? gcd(b, a % b) : a; }
int lcm(int a, int b){ return a / gcd(a, b) * b; }
```

### 4.3 빠른 거듭제곱

Usage:  $a^b \pmod{c}$ 를 구하는 함수  
Time Complexity:  $O(\log b)$

```
ll PowMod(ll a, ll b, ll c){
    if(c == 1) return 0;
    ll res = 1;
    for(a%=c; b; b >>= 1, a = a * a % c) if(b & 1) res = res * a % c;
    return res;
}
```

### 4.4 소수 판별, 소인수분해

Time Complexity:  $O(\sqrt{N})$

```
// IsPrime(2) = true, IsPrime(4) = false
// Factorize(72) = { {2, 3}, {3, 2} }
bool IsPrime(ll n){
    if(n < 2) return false;
    for(ll i=2; i*i<=n; i++) if(n % i == 0) return false;
    return true;
}
vector<pair<ll,ll>> Factorize(ll n){
```

```
if(n == 1) return {};
vector<pair<ll,ll>> res;
for(ll i=2; i*i<=n; i++){
    if(n % i != 0) continue;
    int cnt = 0;
    while(n % i == 0) n /= i, cnt++;
    res.emplace_back(i, cnt);
}
if(n != 1) res.emplace_back(n, 1);
return res;
}
```

### 4.5 에라토스테네스의 체, 소인수분해

Time Complexity: Sieve:  $O(N \log \log N)$ , Factorize:  $O(\log N)$

```
int SP[5050505]; // SP[i] = i의 가장 작은 소인수
vector<int> Primes;

// n 이하의 모든 소수를 구함
void Sieve(int n){
    for(int i=2; i<=n; i++){
        if(SP[i]) continue;
        for(int j=i; j<=n; j+=i) if(!SP[j]) SP[j] = i;
    }
}

// Sieve 먼저 호출해야 함
vector<pair<int,int>> Factorize(int n){
    vector<pair<int,int>> res;
    while(n != 1){
        if(res.empty() || res.back().first != SP[n]) res.emplace_back(SP[n], 1);
        else res.back().second++;
        n /= SP[n];
    }
    return res;
}
```

### 4.6 선형 시간 체, 곱셈적 함수 전처리

```
// sigma 계산하는 부분 제외하면  $O(n)$ , sigma 계산은  $O(n \log \log n)$ 
// pw(j, e[i*j]) 모두 전처리하면  $O(n)$ 에 계산할 수 있음
// prime: n 이하 소수 목록
// sp : 최소 소인수, 소수라면 0
// tau : 약수 개수, sigma : 약수 합
// phi : n 이하 자연수 중 n과 서로소인 개수
// mu : 2번 이상 곱해진 소인수가 있으면 0, 그렇지 않으면  $(-1)^{\text{소인수 개수}}$ 
// e[i] : i에 sp[i]가 곱해진 횟수
vector<int> prime;
int sp[sz], e[sz], phi[sz], mu[sz], tau[sz], sigma[sz];
phi[1] = mu[1] = tau[1] = sigma[1] = 1;
for(int i=2; i<=n; i++){
    if(!sp[i]){
        prime.push_back(i);
```

```

    e[i] = 1; phi[i] = i-1; mu[i] = -1; tau[i] = 2; sigma[i] = i+1;
}
for(auto j : prime){
    if(i*j >= sz) break;
    sp[i*j] = j;
    if(i % j == 0){
        e[i*j] = e[i]+1; phi[i*j] = phi[i]*j; mu[i*j] = 0;
        tau[i*j] = tau[i]/e[i*j]*(e[i*j]+1);
        sigma[i*j] = sigma[i]*(j-1)/(pw(j, e[i*j]-1)*(pw(j, e[i*j]+1)-1)/(j-1));//overflow
        break;
    }
    e[i*j] = 1; phi[i*j] = phi[i] * phi[j]; mu[i*j] = mu[i] * mu[j];
    tau[i*j] = tau[i] * tau[j]; sigma[i*j] = sigma[i] * sigma[j];
}

```

## 4.7 Segmented Sieve

```

// Generate all primes up to limit using sieve of eratosthenes
vector<int> sieve(int limit) {
    vector<bool> is_prime(limit + 1, true);
    is_prime[0] = is_prime[1] = false;
    for (int p = 2; p * p <= limit; ++p) {
        if (is_prime[p]) {
            for (int i = p * p; i <= limit; i += p) {
                is_prime[i] = false;
            }
        }
    }
    vector<int> primes;
    for (int p = 2; p <= limit; ++p) {
        if (is_prime[p]) {
            primes.push_back(p);
        }
    }
    return primes;
}

// Generate all primes from l to r using segmented sieve in O((r - 1) log (r) + sqrt(r))
vector<ll> segmented_sieve(ll l, ll r) {
    if (l == 1) {
        l++;
    }
    int limit = sqrtl(r);
    while ((ll) limit * limit <= r) limit++;
    while ((ll) limit * limit > r) limit--;
    auto primes = sieve(limit);
    vector<bool> is_prime(r - 1 + 1, true);
    for (ll p : primes) {
        ll start = max((ll)p * p, (ll)(l + p - 1) / p * p);
        for (ll j = start; j <= r; j += p) {
            is_prime[j - 1] = false;
        }
    }
}

```

```

vector<ll> vec;
for (ll i = 1; i <= r; ++i) {
    if (is_prime[i - 1]) {
        vec.push_back(i);
    }
}
return vec;
}

```

## 4.8 Counting Divisors

```

// you can restore any Divisors
ll number_of_divisors(ll num) {
    ll total = 1;
    for (int i = 2; (ll)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);
            total *= e + 1;
        }
    }
    if (num > 1) {
        total *= 2;
    }
    return total;
}

```

## 4.9 Sum of Divisors

```

ll sum_of_divisors(ll num) {
    ll total = 1;

    for (int i = 2; (ll)i * i <= num; i++) {
        if (num % i == 0) {
            int e = 0;
            do {
                e++;
                num /= i;
            } while (num % i == 0);

            ll sum = 0, pow = 1;
            do {
                sum += pow;
                pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1) {
        total *= (1 + num);
    }
}

```

```
    return total;
}
```

## 4.10 확장 유clidean 알고리즘

Time Complexity:  $O(\log \max(a, b))$

```
// 정수 a, b 주어지면 ax + by = gcd(a, b) = g
// 를 만족하는 정수 {g, x, y} 반환
tuple<ll, ll, ll> ext_gcd(ll a, ll b){
    if(b == 0) return {a, 1, 0};
    auto [g,x,y] = ext_gcd(b, a % b);
    return {g, y, x - a/b * y};
}
```

## 4.11 중국인의 나머지 정리

Time Complexity:  $O(k \log m)$

```
// a = a1 (mod m1), a = a2 (mod m2)를 만족하는 {a, lcm(m1, m2)} 반환
// 만약 a가 존재하면 0 <= a < lcm(m1, m2) 에서 유일하게 존재
// a가 존재하지 않는 경우 {-1, -1} 반환
ll mod(ll a, ll b){ return (a % b) >= 0 ? a : a + b; }
pair<ll, ll> crt(ll a1, ll m1, ll a2, ll m2){
    ll g = gcd(m1, m2), m = m1 / g * m2;
    if((a2 - a1) % g) return {-1, -1};
    ll md = m2/g, s = mod((a2-a1)/g, m2/g);
    ll t = mod(get<1>(ext_gcd(m1/g%md, m2/g)), md);
    return { a1 + s * t % md * m1, m };
}
// a = a_i (mod m_i)를 만족하는 {a, lcm(m_1, ..., m_k)} 반환
// a가 존재하지 않는 경우 {-1, -1} 반환
pair<ll, ll> crt(const vector<ll> &a, const vector<ll> &m){
    ll ra = a[0], rm = m[0];
    for(int i=1; i<m.size(); i++){
        auto [aa,mm] = crt(ra, rm, a[i], m[i]);
        if(mm == -1) return {-1, -1}; else tie(ra,rm) = tie(aa,mm);
    }
    return {ra, rm};
}
```

## 4.12 이항 계수를 소수로 나눈 나머지

Time Complexity: 전처리  $O(P)$ , 쿼리  $O(\log P)$

```
// Lucas C(13);
// C.calc(5, 3) = 5C3 = 10
// C.calc(10, 2) = 10C2 % 13 = 45 % 13 = 6
// 주의: P는 소수
// P가 크고(약 10억) n,r이 작으면(1000만 이하)
// 생성자에서 fac, inv를 1000만까지만 구해도 됨

struct Lucas{ // init : O(P), query : O(log P)
    const size_t P;
    vector<ll> fac, inv;
```

```
    ll Pow(ll a, ll b){
        ll res = 1;
        for(; b; b>>=1, a=a*a%P) if(b & 1) res = res * a % P;
        return res;
    }
    Lucas(size_t P) : P(P), fac(P), inv(P) {
        fac[0] = 1; for(int i=1; i<P; i++) fac[i] = fac[i-1] * i % P;
        inv[P-1] = Pow(fac[P-1], P-2); for(int i=P-2; ~i; i--) inv[i] = inv[i+1] * (i+1) % P;
    }
    ll small(ll n, ll r) const { return r <= n ? fac[n] * inv[r] % P * inv[n-r] % P : 0LL; }
    ll calc(ll n, ll r) const {
        if(n < r || n < 0 || r < 0) return 0;
        if(!n || !r || n == r) return 1; else return small(n%P, r%P) * calc(n/P, r/P) % P;
    }
};
```

## 4.13 빠른 소수 판별, 소인수분해 - Miller Rabin, Pollard Rho

Usage: 처음에 Sieve() 호출해야 함

Time Complexity: IsPrime:  $O(\log^2 N)$ , Factorize:  $\approx O(N^{1/4})$

```
constexpr int SZ = 10'000'000;
bool PrimeCheck[SZ+1]; vector<int> Primes;
void Sieve(){
    memset(PrimeCheck, true, sizeof PrimeCheck);
    PrimeCheck[0] = PrimeCheck[1] = false;
    for(int i=2; i<=SZ; i++){
        if(PrimeCheck[i]) Primes.push_back(i);
        for(auto j : Primes){
            if(i*j > SZ) break;
            PrimeCheck[i*j] = false;
            if(i % j == 0) break;
        }
    }
}

using ll = long long;
using ull = unsigned long long;
ull MulMod(ull a, ull b, ull c){ return (_uint128_t)a * b % c; }
ull PowMod(ull a, ull b, ull c){
    ull res = 1; a %= c;
    for(; b; b>>=1, a=MulMod(a,a,c)) if(b & 1) res = MulMod(res,a,c);
    return res;
}
bool MillerRabin(ull n, ull a){
    if(a % n == 0) return true;
    int cnt = __builtin_ctzll(n - 1);
    ull p = PowMod(a, n >> cnt, n);
    if(p == 1 || p == n - 1) return true;
    while(cnt--) if((p=PowMod(p,p,n)) == n - 1) return true;
    return false;
}
```

```

bool IsPrime(ll n){
    if(n <= SZ) return PrimeCheck[n];
    if(n <= 2) return n == 2;
    if(n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n % 7 == 0 || n % 11 == 0) return false;
    // 32bit integer: {2, 7, 61}
    for(int p : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) if(!MillerRabin(n, p))
        return false;
    return true;
}

ll Rho(ll n){
    while(true){
        ll x = rand() % (n - 2) + 2, y = x, c = rand() % (n - 1) + 1;
        while(true){
            x = (MulMod(x, x, n) + c) % n;
            y = (MulMod(y, y, n) + c) % n;
            y = (MulMod(y, y, n) + c) % n;
            ll d = __gcd(abs(x - y), n);
            if(d == 1) continue;
            if(IsPrime(d)) return d;
            else{ n = d; break; }
        }
    }
}

vector<pair<ll,ll>> Factorize(ll n){
    vector<pair<ll,ll>> v;
    int two = __builtin_ctzll(n);
    if(two > 0) v.emplace_back(2, two), n >>= two;
    if(n == 1) return v;
    while(!IsPrime(n)){
        ll d = Rho(n), cnt = 0;
        while(n % d == 0) cnt++, n /= d;
        v.emplace_back(d, cnt);
        if(n == 1) break;
    }
    if(n != 1) v.emplace_back(n, 1);
    return v;
}

```

#### 4.14 가우스 소거법 - RREF, 랭크, 행렬식, 역행렬

Time Complexity:  $O(N^3)$

```

// T Add(T a, T b), Sub, Mul, Div 구현해야 함
// bool IsZero(T x) 구현해야 함
template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, int, T, vector<vector<T>>> Gauss(vector<vector<T>> a, bool
square=true){
    int n = a.size(), m = a[0].size(), rank = 0;
    vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
    for(int i=0; i<n; i++) if(square) out[i][i] = T(1);
    for(int i=0; i<m; i++){
        if(rank == n) break;
        if(IsZero(a[rank][i])){
            T mx = T(0); int idx = -1; // fucking precision error

```

```

            for(int j=rank+1; j<n; j++) if(mx < abs(a[j][i])) mx = abs(a[j][i]), idx = j;
            if(idx == -1 || IsZero(a[idx][i])){ det = 0; continue; }
            for(int k=0; k<m; k++) {
                a[rank][k] = Add(a[rank][k], a[idx][k]);
                if(square) out[rank][k] = Add(out[rank][k], out[idx][k]);
            }
        }
        det = Mul(det, a[rank][i]);
        T coeff = Div(T(1), a[rank][i]);
        for(int j=0; j<m; j++) a[rank][j] = Mul(a[rank][j], coeff);
        for(int j=0; j<m; j++) if(square) out[rank][j] = Mul(out[rank][j], coeff);
        for(int j=0; j<n; j++){
            if(rank == j) continue;
            T t = a[j][i]; // Warning: [j][k], [rank][k]
            for(int k=0; k<m; k++) a[j][k] = Sub(a[j][k], Mul(a[rank][k], t));
            for(int k=0; k<m; k++) if(square) out[j][k] = Sub(out[j][k], Mul(out[rank][k], t));
        }
        rank++;
    }
    return {a, rank, det, out};
}

```

#### 4.15 다항식 곱셈(FFT)

Time Complexity:  $O(N \log N)$

```

// 104,857,601 = 25 * 2^22 + 1, w = 3 | 998,244,353 = 119 * 2^23 + 1, w = 3
// 2,281,701,377 = 17 * 2^27 + 1, w = 3 | 2,483,027,969 = 37 * 2^26 + 1, w = 3
// 2,113,929,217 = 63 * 2^25 + 1, w = 5 | 1,092,616,193 = 521 * 2^21 + 1, w = 3
using real_t = double; using cpx = complex<real_t>;
void FFT(vector<cpx> &a, bool inv_fft=false){
    int N = a.size(); vector<cpx> root(N/2);
    for(int i=1, j=0; i<N; i++){
        int bit = N / 2;
        while(j >= bit) j -= bit, bit >>= 1;
        if(i < (j += bit)) swap(a[i], a[j]);
    }
    long double ang = 2 * acosl(-1) / N * (inv_fft ? -1 : 1);
    for(int i=0; i<N/2; i++) root[i] = cpx(cosl(ang*i), sinl(ang*i));
    /*
    NTT : ang = pow(w, (mod-1)/n) % mod, inv_fft -> ang^{-1}, root[i] = root[i-1] * ang
    XOR Convolution : set roots[*] = 1, a[j+k] = u+v, a[j+k+i/2] = u-v
    OR Convolution : set roots[*] = 1, a[j+k+i/2] += inv_fft ? -u : u;
    AND Convolution : set roots[*] = 1, a[j+k] += inv_fft ? -v : v;
    */
    for(int i=2; i<=N; i<<=1){
        int step = N / i;
        for(int j=0; j<N; j+=i) for(int k=0; k<i/2; k++){
            cpx u = a[j+k], v = a[j+k+i/2] * root[step * k];
            a[j+k] = u+v; a[j+k+i/2] = u-v;
        }
        if(inv_fft) for(int i=0; i<N; i++) a[i] /= N; // skip for AND/OR convolution.
    }
}

```

```

vector<ll> multiply(const vector<ll> &a, const vector<ll> &b){
    vector<cpx> a(all(_a)), b(all(_b));
    int N = 2; while(N < a.size() + b.size()) N <= 1;
    a.resize(N); b.resize(N); FFT(a); FFT(b);
    for(int i=0; i<N; i++) a[i] *= b[i];
    vector<ll> ret(N); FFT(a, 1); // NTT : just return a
    for(int i=0; i<N; i++) ret[i] = llround(a[i].real());
    while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
    return ret;
}

// 더 높은 정밀도
vector<ll> multiply_mod(const vector<ll> &a, const vector<ll> &b, const ull mod){
    int N = 2; while(N < a.size() + b.size()) N <= 1;
    vector<cpx> v1(N), v2(N), r1(N), r2(N);
    for(int i=0; i<a.size(); i++) v1[i] = cpx(a[i] >> 15, a[i] & 32767);
    for(int i=0; i<b.size(); i++) v2[i] = cpx(b[i] >> 15, b[i] & 32767);
    FFT(v1); FFT(v2);
    for(int i=0; i<N; i++){
        int j = i ? N-i : i;
        cpx ans1 = (v1[i] + conj(v1[j])) * cpx(0.5, 0);
        cpx ans2 = (v1[i] - conj(v1[j])) * cpx(0, -0.5);
        cpx ans3 = (v2[i] + conj(v2[j])) * cpx(0.5, 0);
        cpx ans4 = (v2[i] - conj(v2[j])) * cpx(0, -0.5);
        r1[i] = (ans1 * ans3) + (ans1 * ans4) * cpx(0, 1);
        r2[i] = (ans2 * ans3) + (ans2 * ans4) * cpx(0, 1);
    }
    vector<ll> ret(N); FFT(r1, true); FFT(r2, true);
    for(int i=0; i<N; i++){
        ll av = llround(r1[i].real()) % mod;
        ll bv = (llround(r1[i].imag()) + llround(r2[i].real())) % mod;
        ll cv = llround(r2[i].imag()) % mod;
        ret[i] = (av << 30) + (bv << 15) + cv;
        ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
    }
    while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
    return ret;
}

```

## 5 DP

### 5.1 Digit DP

```

vector<int> num;
int a, b, d, k;
int DP[12][12][2];

int find(int pos, int cnt, int f) {
    if (cnt > k) return 0;

    if (pos == num.size()) {
        if (cnt == k) return 1;
        return 0;
    }
}

```

```

if (DP[pos][cnt][f] != -1) return DP[pos][cnt][f];
int res = 0;

int LMT;

// f ? (greater than b) : (already smaller than b)
if (f == 0) LMT = num[pos];
else LMT = 9;

for (int dgt = 0; dgt <= LMT; dgt++) {
    int nf = f;
    int ncnt = cnt;
    if (f == 0 && dgt < LMT) nf = 1; // The number is getting smaller at this position
    if (dgt == d) ncnt++;
    if (ncnt <= k) res += find(pos + 1, ncnt, nf);
}

return DP[pos][cnt][f] = res;
}

int digitcount(int b) {
    num.clear();
    while(b > 0) {
        num.push_back(b % 10);
        b /= 10;
    }
    reverse(num.begin(), num.end());

    memset(DP, -1, sizeof(DP));
    int res = find(0, 0, 0);
    return res;
}

```

### 5.2 피사노 주기

```

// return pisano period(maybe you can modify m * m to m + m in 'for')
// then you should calculate fibonacci value at most Pth (P is pisano period)
// and just find i%pth fibonacci value. That's collect value.
ll pisano(ll m) {
    ll previous = 0;
    ll cur = 1;
    ll temp;

    for (ll i = 0; i < m * m; i++) {
        temp = (previous + cur) % m;
        previous = cur;
        cur = temp;

        // found pisano period
        if (previous == 0 && cur == 1) {
            return i + 1;
        }
    }
}

```

}

### 5.3 가장 긴 증가하는 부분 수열(LIS)

Time Complexity:  $O(N \log N)$

```
import sys
N=int(sys.stdin.readline())
c=[0]*(N)
LIS=[c[0]]
for i in range(N):
    if c[i]>LIS[-1]:
        LIS.append(c[i])
    else:
        left,right=0,len(b)-1
        while(left<=right):
            mid=(left+right)//2
            if LIS[mid]<c[i]:
                left=mid+1
            else:
                right=mid-1
        LIS[left]=c[i]
print(len(c))
```

### 5.4 배낭(knapsack) 문제

```
import sys
N,K=map(int,sys.stdin.readline().split())
goods=[[0,0]]
dp=[[0]*(K+1) for i in range(N+1)]
for i in range(N):
    goods.append(list(map(int,sys.stdin.readline().split())))
for i in range(1,N+1):
    for j in range(1,K+1):
        we=goods[i][0]
        vl=goods[i][1]
        if j<we:
            dp[i][j]=dp[i-1][j]
        else:
            dp[i][j]=max(dp[i-1][j],dp[i-1][j-we]+vl)

print(dp[N][K])
```

## 6 문자열

### 6.1 문자열 해싱

Time Complexity: build:  $O(N)$ , get:  $O(1)$

```
// 전처리 O(N), 부분 문자열의 해시값을 O(1)에 구함
// Hashing<917, 998244353> H;
// H.build("ABCDABCD");
// assert(H.get(1, 4) == H.get(5, 8));
// 주의: get 함수의 인자는 1-based 닫힌 구간
```

```
// 주의: M은 10억 근처의 소수, P는 M과 서로소
// 1e5+3, 1e5+13, 131'071, 524'287, 1'299'709, 1'301'021
// 1e9-63, 1e9+7, 1e9+9, 1e9+103
template<long long P, long long M> struct Hashing {
    vector<long long> h, p;
    void build(const string &s){
        int n = s.size();
        h = p = vector<long long>(n+1); p[0] = 1;
        for(int i=1; i<=n; i++) h[i] = (h[i-1] * P + s[i-1]) % M;
        for(int i=1; i<=n; i++) p[i] = p[i-1] * P % M;
    }
    long long get(int s, int e) const {
        long long res = (h[e] - h[s-1] * p[e-s+1]) % M;
        return res >= 0 ? res : res + M;
    }
};
```

### 6.2 문자열 매칭 - KMP

Time Complexity: GetFail:  $O(|P|)$ ,  $O(|S| + |P|)$

```
// s에서 p가 등장하는 위치 반환
// KMP("ABABCAB", "AB") = {0, 2, 5}
// KMP("AAAAA", "AA") = {0, 1, 2}
```

```
vector<int> GetFail(const string &p){
    int n = p.size();
    vector<int> fail(n);
    for(int i=1, j=0; i<n; i++){
        while(j && p[i] != p[j]) j = fail[j-1];
        if(p[i] == p[j]) fail[i] = ++j;
    }
    return fail;
}

vector<int> KMP(const string &s, const string &p){
    int n = s.size(), m = p.size();
    vector<int> fail = GetFail(p), ret;
    for(int i=0, j=0; i<s.size(); i++){
        while(j && s[i] != p[j]) j = fail[j-1];
        if(s[i] == p[j]){
            if(j + 1 == m) ret.push_back(i-m+1), j = fail[j];
            else j++;
        }
    }
    return ret;
}
```

### 6.3 가장 긴 팰린드롬 부분 문자열 - Manacher

Time Complexity:  $O(N)$

```
// 각 문자를 중심으로 하는 최장 팰린드롬의 반경을 반환
// Manacher("abaaba") = {0,1,0,3,0,1,6,1,0,3,0,1,0}
```

```
// # a # b # a # a # b # a #
// 0 1 0 3 0 1 6 1 0 3 0 1 0
vector<int> Manacher(const string &inp){
    int n = inp.size() * 2 + 1;
    vector<int> ret(n);
    string s = "#";
    for(auto i : inp) s += i, s += "#";
    for(int i=0, p=-1, r=-1; i<n; i++){
        ret[i] = i <= r ? min(r-i, ret[2*p-i]) : 0;
        while(i-ret[i]-1 >= 0 && i+ret[i]+1 < n && s[i-ret[i]-1] == s[i+ret[i]+1])
            ret[i]++;
        if(i+ret[i] > r) r = i+ret[i], p = i;
    }
    return ret;
}
```

## 6.4 문자열 매칭 - Z

Time Complexity:  $O(N)$

```
// Z[i] = LongestCommonPrefix(S[0:N], S[i:N])
//      = S[0:N]과 S[i:N]이 앞에서부터 몇 글자 겹치는지
vector<int> Z(const string &s){
    int n = s.size();
    vector<int> z(n);
    z[0] = n;
    for(int i=1, l=0, r=0; i<n; i++){
        if(i < r) z[i] = min(r-i-1, z[i-1]);
        while(i+z[i] < n && s[i+z[i]] == s[z[i]]) z[i]++;
        if(i+z[i] > r) r = i+z[i], l = i;
    }
    return z;
}
```

## 6.5 접미사 배열

Time Complexity:  $O(N \log N)$

```
// LCP는 1-based
pair<vector<int>, vector<int>> SuffixArray(const string &s){ // O(N log N)
    int n = s.size(), m = max(n, 256);
    vector<int> sa(n), lcp(n), pos(n), tmp(n), cnt(m);
    auto counting_sort = [&](){
        fill(cnt.begin(), cnt.end(), 0);
        for(int i=0; i<n; i++) cnt[pos[i]]++;
        partial_sum(cnt.begin(), cnt.end(), cnt.begin());
        for(int i=n-1; i>=0; i--) sa[--cnt[pos[tmp[i]]]] = tmp[i];
    };
    for(int i=0; i<n; i++) sa[i] = i, pos[i] = s[i], tmp[i] = i;
    counting_sort();
    for(int k=1; ; k<=<1){
        int p = 0;
        for(int i=n-k; i<n; i++) tmp[p++] = i;
        for(int i=0; i<n; i++) if(sa[i] >= k) tmp[p++] = sa[i] - k;
    }
}
```

```
counting_sort();
tmp[sa[0]] = 0;
for(int i=1; i<n; i++){
    tmp[sa[i]] = tmp[sa[i-1]];
    if(sa[i-1]+k < n && sa[i]+k < n && pos[sa[i-1]] == pos[sa[i]] && pos[sa[i-1]+k] == pos[sa[i]+k]) continue;
    tmp[sa[i]] += 1;
}
swap(pos, tmp); if(pos[sa.back()] + 1 == n) break;
}
for(int i=0, j=0; i<n; i++, j=max(j-1, 0)){
    if(pos[i] == 0) continue;
    while(sa[pos[i]-1]+j < n && sa[pos[i]]+j < n && s[sa[pos[i]-1]+j] == s[sa[pos[i]]+j])
        j++;
    lcp[pos[i]] = j;
}
return {sa, lcp};
}
```

## 7 계산 기하

### 7.1 2차원 계산 기하 템플릿 + CCW

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>

Point operator + (Point p1, Point p2){ return {p1.x + p2.x, p1.y + p2.y}; }
Point operator - (Point p1, Point p2){ return {p1.x - p2.x, p1.y - p2.y}; }
ll operator * (Point p1, Point p2){ return p1.x * p2.x + p1.y * p2.y; } // 내적
ll operator / (Point p1, Point p2){ return p1.x * p2.y - p2.x * p1.y; } // 외적
int Sign(ll v){ return (v > 0) - (v < 0); } // 양수면 +1, 음수면 -1, 0이면 0 반환
ll Dist(Point p1, Point p2){ return (p2 - p1) * (p2 - p1); } // 두 점 거리 제곱
ll SignedArea(Point p1, Point p2, Point p3){ return (p2 - p1) / (p3 - p1); }
int CCW(Point p1, Point p2, Point p3){ return Sign(SignedArea(p1, p2, p3)); }
```

### 7.2 360도 각도 정렬

```
/* y축
   ↑
3 2 1
4 -1 0 → x축
5 6 7
원점 기준 각도 정렬
x축 위에 있는 점이 가장 먼저, 그리고 반시계 방향으로
*/
int QuadrantID(const Point p){
    static int arr[9] = { 5, 4, 3, 6, -1, 2, 7, 0, 1 };
    return arr[Sign(p.x)*3+Sign(p.y)+4];
}
sort(points.begin(), points.end(), [&](auto p1, auto p2){
```

```

if(QuadrantID(p1) != QuadrantID(p2)) return QuadrantID(p1) < QuadrantID(p2);
else return p1 / p2 > 0; // 반시계
});

```

### 7.3 다각형 넓이

```

// 다각형의 넓이의 2배를 반환, 항상 정수, O(N)
11 PolygonArea(const vector<Point> &v){
    ll res = 0;
    for(int i=0; i<v.size(); i++) res += v[i] / v[(i+1)%v.size()];
    return abs(res);
}

```

### 7.4 선분 교차 판정

```

// 선분 교차 - 선분 ab와 선분 cd가 만나면 true
bool Cross(Point s1, Point e1, Point s2, Point e2){
    int ab = CCW(s1, e1, s2) * CCW(s1, e1, e2);
    int cd = CCW(s2, e2, s1) * CCW(s2, e2, e1);
    if(ab == 0 && cd == 0){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        return !(e1 < s2 || e2 < s1);
    }
    return ab <= 0 && cd <= 0;
}
// 교차하지 않으면 0
// 교점이 무한히 많으면 -1
// 교점이 1개면 1 반환하고 res에 교점 저장
int Cross(Point s1, Point e1, Point s2, Point e2, pair<double, double> &res){
    if(!Cross(s1, e1, s2, e2)) return 0;
    ll det = (e1 - s1) / (e2 - s2);
    if(!det){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        if(e1 == s2){ res = s2; return 1; }
        if(e2 == s1){ res = s1; return 1; }
        return -1;
    }
    res.x = s1.x + (e1.x - s1.x) * ((s2 - s1) / (e2 - s2) * 1.0 / det);
    res.y = s1.y + (e1.y - s1.y) * ((s2 - s1) / (e2 - s2) * 1.0 / det);
    return 1;
}

```

### 7.5 다각형 내부 판별

```

// 다각형 내부 또는 경계 위에 p가 있으면 true, O(N)
bool PointInPolygon(const vector<Point> &v, Point p){
    int n = v.size(), cnt = 0;
    Point p2(p.x+1, 1'000'000'000 + 1); // 좌표 범위보다 큰 수
    for(int i=0; i<n; i++){
        int j = i + 1 < n ? i + 1 : 0;
        if(min(v[i],v[j]) <= p && p <= max(v[i],v[j]) && CCW(v[i], v[j], p) == 0) return
true;
    }
}

```

```

        if(SegmentIntersection(v[i], v[j], p, p2)) cnt++;
    }
    return cnt % 2 == 1;
}

```

### 7.6 볼록 껍질 - Graham Scan

```

// 모든 점을 포함하는 가장 작은 볼록 다각형, O(N log N)
vector<Point> ConvexHull(vector<Point> points){
    if(points.size() <= 1) return points;
    swap(points[0], *min_element(points.begin(), points.end()));
    sort(points.begin()+1, points.end(), [&](auto a, auto b){
        int dir = CCW(points[0], a, b);
        if(dir != 0) return dir > 0;
        return Dist(points[0], a) < Dist(points[0], b);
    });
    vector<Point> hull;
    for(auto p : points){
        while(hull.size() >= 2 && CCW(hull[hull.size()-2], hull.back(), p) <= 0)
            hull.pop_back();
        hull.push_back(p);
    }
    return hull;
}

```

### 7.7 가장 먼 두 점 - Rotating Calipers

```

// 가장 먼 두 점을 구하는 함수, O(N)
// 주의: hull은 반시계 방향으로 정렬된 볼록 다각형이어야 함
pair<Point, Point> Calipers(vector<Point> hull){
    int n = hull.size(); ll mx = 0; Point a, b;
    for(int i=0, j=0; i<n; i++){
        while(j + 1 < n && (hull[i+1] - hull[i]) / (hull[j+1] - hull[j]) >= 0){
            ll now = Dist(hull[i], hull[j]);
            if(now > mx) mx = now, a = hull[i], b = hull[j];
            j++;
        }
        ll now = Dist(hull[i], hull[j]);
        if(now > mx) mx = now, a = hull[i], b = hull[j];
    }
    return {a, b};
}

```

### 7.8 볼록 다각형 내부 판별

```

// 다각형 내부 또는 경계 위에 p가 있으면 true, O(log N)
// 주의: v는 반시계 방향으로 정렬된 볼록 다각형이어야 함
bool PointInConvexPolygon(const vector<Point> &v, const Point &pt){
    if(CCW(v[0], v[1], pt) < 0) return false; int l = 1, r = v.size() - 1;
    while(l < r){
        int m = l + r + 1 >> 1;
        if(CCW(v[0], v[m], pt) >= 0) l = m; else r = m - 1;
    }
}

```

```

if(l == v.size() - 1) return CCW(v[0], v.back(), pt) == 0 && v[0] <= pt && pt <=
v.back();
return CCW(v[0], v[l], pt) >= 0 && CCW(v[1], v[l+1], pt) >= 0 && CCW(v[l+1], v[0], pt)
>= 0;
}

```

## 8 기타

### 8.1 누적합

```

import sys
N=int(sys.stdin.readline())
a=[0] + list(map(int,sys.stdin.readline().split()))
for i in range(2,N+1):
    a[i]+=a[i-1]

```

### 8.2 이분 탐색

// 주의: 구간 범위에 음수가 들어가면 / 2 말고 >> 1로 해야 함

// 1 1 1 ... 1 1 0 0 ... 0 꼴일 때 마지막 1의 위치

```

int l = 1, r = n;
while(l < r){
    int m = (l + r + 1) / 2;
    if(Check(m)) l = m;
    else r = m - 1;
}

```

// 0 0 0 ... 0 0 1 1 ... 1 꼴일 때 첫 번째 1의 위치

```

int l = 1, r = n;
while(l < r){
    int m = (l + r) / 2;
    if(Check(m)) r = m;
    else l = m + 1;
}

```

### 8.3 삼분 탐색

```

while(s + 3 <= e){ // get minimum / when multiple answer, find minimum `s`
    T l = (s + s + e) / 3, r = (s + e + e) / 3;
    if(Check(l) > Check(r)) s = l; else e = r;
}
T mn = INF, idx = s;
for(T i=s; i<=e; i++) if(T now = Check(i); now < mn) mn = now, idx = i;

```

### 8.4 C++ 랜덤, GCC 확장, 비트마스킹 트리

```

mt19937 rd((unsigned)chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> rnd_int(l, r); // rnd_int(rd)
uniform_real_distribution<double> rnd_real(0, 1); // rnd_real(rd)
/////
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>

```

```

using namespace __gnu_pbds; //ordered_set : find_by_order(order), order_of_key(key)
using namespace __gnu_cxx; //rope : append(str), substr(s, e), at(idx)
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
/////
int __builtin_clz(int x); // number of leading zero
int __builtin_ctz(int x); // number of trailing zero
int __builtin_popcount(int x); // number of 1-bits in x
lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_clzll(n | 1);
long long next_perm(long long v){
    long long t = v | (v-1);
    return (t + 1) | (((t & ~t) - 1) >> (__builtin_ctz(v) + 1));
}
int frq(int n, int i) { // # of digit i in [1, n]
    int j, r = 0;
    for (j = 1; j <= n; j *= 10) if (n / j / 10 >= !i) r += (n / 10 / j - !i) * j + (n / j %
10 > i ? j : n / j % 10 == i ? n % j + 1 : 0);
    return r;
}

```

### 8.5 구간별 약수 최대 개수, 최대 소수

< 10^k	number	divisors	2 3 5 7 11 13 17 19 23 29 31 37
1	6	4	1 1
2	60	12	2 1 1
3	840	32	3 1 1 1
4	7560	64	3 3 1 1
5	83160	128	3 3 1 1 1
6	720720	240	4 2 1 1 1 1
7	8648640	448	6 3 1 1 1 1
8	73513440	768	5 3 1 1 1 1 1
9	735134400	1344	6 3 2 1 1 1 1
10	6983776800	2304	5 3 2 1 1 1 1 1
11	97772875200	4032	6 3 2 2 1 1 1 1
12	963761198400	6720	6 4 2 1 1 1 1 1 1
13	9316358251200	10752	6 3 2 1 1 1 1 1 1 1
14	97821761637600	17280	5 4 2 2 1 1 1 1 1 1
15	866421317361600	26880	6 4 2 1 1 1 1 1 1 1 1
16	8086598962041600	41472	8 3 2 2 1 1 1 1 1 1 1
17	74801040398884800	64512	6 3 2 2 1 1 1 1 1 1 1 1
18	897612484786617600	103680	8 4 2 2 1 1 1 1 1 1 1 1

< 10^k	prime	# of prime	< 10^k	prime
1	7	4	10	9999999967
2	97	25	11	9999999977
3	997	168	12	99999999989
4	9973	1229	13	999999999971
5	99991	9592	14	9999999999973
6	999983	78498	15	99999999999989

7	99999991	664579	16	99999999999999937
8	99999989	5761455	17	9999999999999997
9	99999937	50847534	18	9999999999999989

## 8.6 카탈란 수, 심슨 적분, 그린디 정리, 피의 정리, 페르마 포인트, 오일러 정리

- 카탈란 수

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900

$C_n = \binom{n}{2} / (n+1)$ ;

- 길이가  $2n$ 인 올바른 팔호 수식의 수

-  $n+1$ 개의 리프를 가진 풀 바이너리 트리의 수

-  $n+2$ 각형을  $n$ 개의 삼각형으로 나누는 방법의 수

- Simpson 공식 (적분) : Simpson 공식,  $S_n(f) = \frac{h}{3}[f(x_0) + f(x_n) + 4\sum f(x_{2i+1}) + 2\sum f(x_{2i})]$

-  $M = \max |f^4(x)|$ 이라고 하면 오차 범위는 최대  $E_n \leq \frac{M(b-a)}{180}h^4$

- 알고리즘 개임

- Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0이 아니면 첫번째, 0이면 두번째 플레이어가 승리.

- Grundy Number : 어떤 상황의 Grundy Number는 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state들로 나뉠 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.

- Subtraction Game : 한 번에  $k$ 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를  $k+1$ 로 나눈 나머지를 XOR 합하여 판단한다.

- Index-k Nim : 한 번에 최대  $k$ 개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을  $k+1$ 로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.

- Misere Nim : 모든 돌 무더기가 1이면 N이 훌수일 때 후공 승, 그렇지 않은 경우 XOR 합 0이면 후공 승

- Pick's Theorem

격자점으로 구성된 simple polygon이 주어짐.  $I$ 는 polygon 내부의 격자점 수,  $B$ 는 polygon 선분 위 격자점 수,  $A$ 는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다.  $A = I + B/2 - 1$

```
// number of (x, y) : (0 <= x < n && 0 < y <= k/d x + b/d)
ll count_solve(ll n, ll k, ll b, ll d) { // argument should be positive
    if (k == 0) {
        return (b / d) * n;
    }
    if (k >= d || b >= d) {
        return ((k / d) * (n - 1) + 2 * (b / d)) * n / 2 + count_solve(n, k % d, b % d, d);
    }
    return count_solve((k * n + b) / d, d, (k * n + b) % d, k);
}
```

- 페르마 포인트 : 삼각형의 세 꼭짓점으로부터 거리의 합이 최소가 되는 점

$2\pi/3$  보다 큰 각이 있으면 그 점이 페르마 포인트, 그렇지 않으면 각 변마다 정삼각형 그린 다음, 정삼각형의 끝점에서 반대쪽 삼각형의 꼭짓점으로 연결한 선분의 교점

$2\pi/3$  보다 큰 각이 없으면 거리의 합은  $\sqrt{(a^2 + b^2 + c^2 + 4\sqrt{3}S)/2}$ ,  $S$ 는 넓이

- 오일러 정리: 서로소인 두 정수  $a, n$ 에 대해  $a^{\phi(n)} \equiv 1 \pmod{n}$

모든 정수에 대해  $a^n \equiv a^{n-\phi(n)} \pmod{n}$

$m \geq \log_2 n$ 이면  $a^m \equiv a^{m\% \phi(n)+\phi(n)} \pmod{n}$

- $g^0 + g^1 + g^2 + \dots + g^{p-2} \equiv -1 \pmod{p}$  iff  $g = 1$ , otherwise 0.

## 8.7 경우의 수 - 포함 배제, 스텔링 수, 벨 수

- 공 구별 X, 상자 구별 O, 전사함수 : 포함배제  $\sum_{i=1}^k (-1)^{k-i} \times kCi \times i^n$
- 공 구별 O, 상자 구별 X, 전사함수 : 제 2종 스텔링 수  $S(n, k) = k \times S(n-1, k) + S(n-1, k-1)$   
포함배제하면  $O(K \log N)$ ,  $S(n, k) = 1/k! \times \sum_{i=1}^k (-1)^{k-i} \times kCi \times i^n$
- 공 구별 O, 상자 구별 X, 제약없음 : 벨 수  $B(n, k) = \sum_{i=0}^k S(n, i)$  몇 개의 상자를 버릴지 다 돌아보기 수식 정리하면  $O(\min(N, K) \log N)$ 에 됨.  $B(n, n) = \sum_{i=0}^{n-1} (n-1)Ci \times B(i, i)$   

$$B(n, k) = \sum_{j=0}^k S(n, j) = \sum_{j=0}^k 1/j! \sum_{i=0}^j (-1)^{j-i} iCi \times i^n = \sum_{j=0}^k \sum_{i=0}^j \frac{(-1)^{j-i}}{i!(j-i)!} i^n$$

$$= \sum_{i=0}^k \sum_{j=i}^k \frac{(-1)^{j-i}}{i!(j-i)!} i^n = \sum_{i=0}^k \sum_{j=0}^{k-i} \frac{(-1)^j}{i!j!} i^n = \sum_{i=0}^k \frac{i^n}{i!} \sum_{j=0}^{k-i} \frac{(-1)^j}{j!}$$
- Derangement:  $D(n) = (n-1)(D(n-1) + D(n-2))$
- Signed Stirling 1:  $S_1(n, k) = (n-1)S_1(n-1, k) + S_1(n-1, k-1)$
- Unsigned Stirling 1:  $C_1(n, k) = (n-1)C_1(n-1, k) + C_1(n-1, k-1)$
- Stirling 2:  $S_2(n, k) = kS_2(n-1, k) + S_2(n-1, k-1)$
- Stirling 2:  $S_2(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$
- Partition:  $p(n, k) = p(n-1, k-1) + p(n-k, k)$
- Partition:  $p(n) = \sum (-1)^k p(n-k(3k-1)/2)$
- Bell:  $B(n) = \sum_{k=1}^n \binom{n-1}{k-1} B(n-k)$
- Catalan:  $C_n = \frac{1}{n+1} \binom{2n}{n}$
- Catalan:  $C_n = \binom{2n}{n} - \binom{2n}{n+1}$
- Catalan:  $C_n = \frac{(2n)!}{n!(n+1)!}$
- Catalan:  $C_n = \sum C_i C_{n-i}$

## 8.8 삼각형의 오심 - 외심, 내심, 무게중심, 수심, 방심

변 길이  $a, b, c; p = (a+b+c)/2$

넓이  $A = \sqrt{p(p-a)(p-b)(p-c)}$

외접원 반지름  $R = abc/4A$ , 내접원 반지름  $r = A/p$

중선 길이  $m_a = 0.5\sqrt{2b^2 + 2c^2 - a^2}$

각 이등분선 길이  $s_a = \sqrt{bc(1 - \frac{a^2}{b+c})}$

사인 법칙  $\frac{\sin A}{a} = 1/2R$ , 코사인 법칙  $a^2 = b^2 + c^2 - 2bc \cos A$ , 탄젠트 법칙  $\frac{a+b}{a-b} = \frac{\tan(A+B)/2}{\tan(A-B)/2}$

중심 좌표  $(\frac{\alpha x_a + \beta x_b + \gamma x_c}{\alpha + \beta + \gamma}, \frac{\alpha y_a + \beta y_b + \gamma y_c}{\alpha + \beta + \gamma})$

이름	$\alpha$	$\beta$	$\gamma$	
외심	$a^2\mathcal{A}$	$b^2\mathcal{B}$	$c^2\mathcal{C}$	$\mathcal{A} = b^2 + c^2 - a^2$
내심	$a$	$b$	$c$	$\mathcal{B} = a^2 + c^2 - b^2$
무게중심	1	1	1	$\mathcal{C} = a^2 + b^2 - c^2$
수심	$\mathcal{BC}$	$\mathcal{CA}$	$\mathcal{AB}$	
방심(A)	$-a$	$b$	$c$	

## 8.9 미적분, 뉴턴 랩슨법

- $(\arcsin x)' = 1/\sqrt{1-x^2}$
- $(\tan x)' = 1 + \tan^2 x$
- $\int \tan ax = -\ln |\cos ax|/a$
- $(\arccos x)' = -1/\sqrt{1-x^2}$
- $(\arctan x)' = 1/(1+x^2)$
- $\int x \sin ax = (\sin ax - ax \cos ax)/a^2$
- Newton:  $x_{n+1} = x_n - f(x_n)/f'(x_n)$
- $\oint_C (Ldx + Mdy) = \int \int_D \left( \frac{\partial M}{\partial x} - \frac{\partial L}{\partial y} \right) dx dy$
- where  $C$  is positively oriented, piecewise smooth, simple, closed;  $D$  is the region inside  $C$ ;  $L$  and  $M$  have continuous partial derivatives in  $D$ .