Teamnote of KAIMARU

KAIST — 신민철, 이종서, 이채준

2020 ICPC Seoul Regional Contest

1 Data Structures

1.1 Li-Chao Tree

```
// Li Chao Tree : max query
// Time Complexity : O(Q log N), Space Complexity : O(Q log N)
// call init() before use

struct LiChaoTree{
	const ll inf = 1e18;
	struct Line{
		ll a, b;
		ll f(ll x){ return a*x+b; }
		Line(ll a, ll b) : a(a), b(b) {}
		Line() : Line(0, -inf) {}
	};
	struct Node{
		Node() : l(-1), r(-1), v(0, -inf) {}
		int l, r; Line v;
	};
	vector<Node> nd;
	void init(){ nd.emplace_back(); }
	void update(int node, ll s, ll e, Line v){
		Line lo = nd[node].v, hi = v;
		if(lo.f(s) > hi.f(s)) swap(lo, hi);
		if(lo.f(e) <= hi.f(e)){ nd[node].v = hi; return; }
		ll m = s + e >> 1;
		if(lo.f(m) <= hi.f(m)){
			nd[node].v = hi;
			if(nd[node].r == -1) nd[node].r = nd.size(), nd.emplace_back();
			update(nd[node].r, m+1, e, lo);
		}else{
			nd[node].v = lo;
			if(nd[node].l == -1) nd[node].l = nd.size(), nd.emplace_back();
			update(nd[node].l, s, m, hi);
		}
	}
	ll query(int node, ll s, ll e, ll x){
		if(node == -1) return -inf;
		ll t = nd[node].v.f(x);
		ll m = s + e >> 1;
		if(x <= m) return max(t, query(nd[node].l, s, m, x));
		else return max(t, query(nd[node].r, m+1, e, x));
	}
};
```

1.2 Line Container

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+m,
 * and query maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */
#pragma once

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

struct LineContainer : multiset<Line, less<>> {
	// (for doubles, use inf = 1/.0, div(a,b) = a/b)
	static const ll inf = LLONG_MAX;
	ll div(ll a, ll b) { // floored division
		return a / b - ((a ^ b) < 0 && a % b); }
	bool isect(iterator x, iterator y) {
```

```
		if (y == end()) return x->p = inf, 0;
		if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
		else x->p = div(y->m - x->m, x->k - y->k);
		return x->p >= y->p;
	}
	void add(ll k, ll m) {
		auto z = insert({k, m, 0}), y = z++, x = y;
		while (isect(y, z)) z = erase(z);
		if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
		while ((y = x) != begin() && (--x)->p >= y->p)
			isect(x, erase(y));
	}
	ll query(ll x) {
		assert(!empty());
		auto l = *lower_bound(x);
		return l.k * x + l.m;
	}
};
```

1.3 Persistent Segment Tree

```
struct PST {
    struct NODE {
        int l, r, val;
        NODE(): l(0), r(0), val(0) {}
    };
    NODE T[MAXN * 30];

    int sz = 0;

    int make(int y, int l, int r, int n) {
        if(r < y || y < l) return n;
        int cur = ++sz;
        T[cur] = T[n];
        if(l != r) {
            int mid = l + r >> 1;
            T[cur].l = make(y, l, mid, T[n].l);
            T[cur].r = make(y, mid+1, r, T[n].r);
        }
        T[cur].val++;
        return cur;
    }

    int query(int x, int y, int l, int r, int nx, int ny) {
        if(r < x || y < l) return 0;
        if(x <= l && r <= y) return T[ny].val - T[nx].val;
        int mid = l + r >> 1;
        return query(x, y, l, mid, T[nx].l, T[ny].l) +
            query(x, y, mid+1, r, T[nx].r, T[ny].r);
    }
} pst;
```

2 Trees

2.1 Heavy-Light Decomposition

```
// Heavy Light Decomposition
// dfs : undirected graph(inp) -> directed graph(g)
// dfs1 : get subtree size, node depth, parent node
// dfs2 : make heavy chain, euler tour trick

namespace HLD{
	const int SZ = 101010;
	vector<int> inp[SZ], g[SZ];
	int sz[SZ], par[SZ], dep[SZ], top[SZ], in[SZ], out[SZ];
	void addEdge(int s, int e){
		inp[s].push_back(e); inp[e].push_back(s);
	}
	void dfs(int v, int b = -1){
		for(auto i : inp[v]) if(i != b) g[v].push_back(i), dfs(i);
	}
	void dfs1(int v){
		sz[v] = 1;
		for(auto &i : g[v]){
			dep[i] = dep[v] + 1; par[i] = v;
			dfs1(i); sz[v] += sz[i];
			if(sz[i] > sz[g[v][0]]) swap(i, g[v][0]);
		}
	}
	void dfs2(int v){
		in[v] = ++pv;
		for(auto i : g[v]) top[i] = (i == g[v][0]) ? top[v] : i, dfs2(i);
		out[v] = pv;
	}
	void hld(int root){ dfs(root); dfs1(root); dfs2(root); }
	void updatePath(int u, int v, ll x){
		for(; top[u]!=top[v]; u=par[top[u]]){
			if(dep[top[u]] < dep[top[v]]) swap(u, v);
			update(in[top[u]], in[u], x);
		}
```

```
        if(dep[u] > dep[v]) swap(u, v);
        update(in[u], in[v], x);
        // if edge query, then update(in[u]+1, in[v], x)
    }
    ll queryPath(int u, int v){
        ll ret = 0;
        for(; top[u]!=top[v]; u=par[top[u]]){
            if(dep[top[u]] < dep[top[v]]) swap(u, v);
            ret += query(in[top[u]], in[u]);
        }
        if(dep[u] > dep[v]) swap(u, v);
        ret += query(in[u], in[v]);
        // if edge query, then query(in[u]+1, in[v])
        return ret;
    }
}
```

2.2 Centroid Decomposition

```
int szdfs(int cur, int par) {
    sz[cur] = 1;
    for(int nxt : G[cur]) if(nxt != par && !del[nxt]) sz[cur] += szdfs(nxt, cur);
    return sz[cur];
}

int cdfs(int cur, int par, int cap) {
    for(int nxt : G[cur]) if(nxt != par && !del[nxt] && sz[nxt] > cap)
        return cdfs(nxt, cur, cap);
    return cur;
}

void decompose(int root, int par) {
    int cap = szdfs(root, par);
    int cen = cdfs(root, par, cap / 2);
    del[cen] = 1;
    p[cen] = par;
    for(int i : G[cen]) if(!del[i]) decompose(i, cen);
}
```

3 Graphs

3.1 2-SAT

```
#define rep(i, from, to) for (int i = from; i < (to); ++i)
#define all(x) x.begin(), x.end()
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;


struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2*n) {}

  int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
  }
  void setValue(int x) { either(x, x); }

  void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
```

```
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = low;
      if (values[x>>1] == -1)
        values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

3.2 SCC

```
const int MAXN = 100;
vector<int> graph[MAXN];
int up[MAXN], visit[MAXN], vtime;
vector<int> stk;
int scc_idx[MAXN], scc_cnt;

void dfs(int nod) {
    up[nod] = visit[nod] = ++vtime;
    stk.push_back(nod);
    for (int next : graph[nod]) {
        if (visit[next] == 0) {
            dfs(next);
            up[nod] = min(up[nod], up[next]);
        }
        else if (scc_idx[next] == 0)
            up[nod] = min(up[nod], visit[next]);
    }
    if (up[nod] == visit[nod]) {
        ++scc_cnt;
        int t;
        do {
            t = stk.back();
            stk.pop_back();
            scc_idx[t] = scc_cnt;
        } while (!stk.empty() && t != nod);
    }
}

// find SCCs in given directed graph
// O(V+E)
// the order of scc_idx constitutes a reverse topological sort
void get_scc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    scc_cnt = 0;
    memset(scc_idx, 0, sizeof(scc_idx));
    for (int i = 0; i < n; ++i)
        if (visit[i] == 0) dfs(i);
}
```

3.3 BCC

```
const int MAXN = 100;
vector<pair<int, int>> graph[MAXN];  // { next vertex id, edge id }
int up[MAXN], visit[MAXN], vtime;
vector<int> stk;

int is_cut[MAXN];             // v is cut vertex if  is_cut[v] > 0
vector<int> bridge;           // list of edge ids
vector<int> bcc_edges[MAXN];  // list of edge ids in a bcc
int bcc_cnt;

void dfs(int nod, int par_edge) {
    up[nod] = visit[nod] = ++vtime;
    int child = 0;
    for (const auto& e : graph[nod]) {
        int next = e.first, eid = e.second;
        if (eid == par_edge) continue;
        if (visit[next] == 0) {
            stk.push_back(eid);
            ++child;
            dfs(next, eid);
            if (up[next] == visit[next]) bridge.push_back(eid);
            if (up[next] >= visit[nod]) {
                ++bcc_cnt;
                do {
                    auto lasteid = stk.back();
                    stk.pop_back();
                    bcc_edges[bcc_cnt].push_back(lasteid);
                    if (lasteid == eid) break;
```

```
                } while (!stk.empty());
                is_cut[nod]++;
            }
            up[nod] = min(up[nod], up[next]);
        }
        else if (visit[next] < visit[nod]) {
            stk.push_back(eid);
            up[nod] = min(up[nod], visit[next]);
        }
    }
    if (par_edge == -1 && is_cut[nod] == 1)
        is_cut[nod] = 0;
}

// find BCCs & cut vertexs & bridges in undirected graph
// O(V+E)
void get_bcc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    memset(is_cut, 0, sizeof(is_cut));
    bridge.clear();
    for (int i = 0; i < n; ++i) bcc_edges[i].clear();
    bcc_cnt = 0;
    for (int i = 0; i < n; ++i) {
        if (visit[i] == 0)
            dfs(i, -1);
    }
}
```

3.4 Flow - Dinic's Algorithm

```
// usage:
// MaxFlowDinic::init(n);
// MaxFlowDinic::add_edge(0, 1, 100, 100); // for bidirectional edge
// MaxFlowDinic::add_edge(1, 2, 100); // directional edge
// result = MaxFlowDinic::solve(0, 2); // source -> sink
// graph[i][edgeIndex].res -> residual
//
// in order to find out the minimum cut, use `l'.
// if l[i] == 0, i is unrechable.
//
// O(V*V*E)
// with unit capacities, O(min(V^(2/3), E^(1/2)) * E)
struct MaxFlowDinic {
    typedef int flow_t;
    struct Edge {
        int next;
        size_t inv; /* inverse edge index */
        flow_t res; /* residual */
    };
    int n;
    vector<vector<Edge>> graph;
    vector<int> q, l, start;

    void init(int _n) {
        n = _n;
        graph.resize(n);
        for (int i = 0; i < n; i++) graph[i].clear();
    }
    void add_edge(int s, int e, flow_t cap, flow_t caprev = 0) {
        Edge forward{ e, graph[e].size(), cap };
        Edge reverse{ s, graph[s].size(), caprev };
        graph[s].push_back(forward);
        graph[e].push_back(reverse);
    }
    bool assign_level(int source, int sink) {
        int t = 0;
        memset(&l[0], 0, sizeof(l[0]) * l.size());
        l[source] = 1;
        q[t++] = source;
        for (int h = 0; h < t && !l[sink]; h++) {
            int cur = q[h];
            for (const auto& e : graph[cur]) {
                if (l[e.next] || e.res == 0) continue;
                l[e.next] = l[cur] + 1;
                q[t++] = e.next;
            }
        }
        return l[sink] != 0;
    }
    flow_t block_flow(int cur, int sink, flow_t current) {
        if (cur == sink) return current;
        for (int& i = start[cur]; i < graph[cur].size(); i++) {
            auto& e = graph[cur][i];
            if (e.res == 0 || l[e.next] != l[cur] + 1) continue;
            if (flow_t res = block_flow(e.next, sink, min(e.res, current))) {
                e.res -= res;
                graph[e.next][e.inv].res += res;
                return res;
            }
```

```
        }
        return 0;
    }
    flow_t solve(int source, int sink) {
        q.resize(n);
        l.resize(n);
        start.resize(n);
        flow_t ans = 0;
        while (assign_level(source, sink)) {
            memset(&start[0], 0, sizeof(start[0]) * n);
            while (flow_t flow = block_flow(source, sink,
                numeric_limits<flow_t>::max()))
                ans += flow;
        }
        return ans;
    }
};
```

3.5 Min Cost-Max Flow

```
// precondition: there is no negative cycle.
// usage:
// MinCostFlow mcf(n);
// for(each edges) mcf.addEdge(from, to, cost, capacity);
// mcf.solve(source, sink); // min cost max flow
// mcf.solve(source, sink, 0); // min cost flow
// mcf.solve(source, sink, goal_flow);
    // min cost flow with total_flow >= goal_flow if possible
struct MinCostFlow {
    typedef int cap_t;
    typedef int cost_t;

    bool iszerocap(cap_t cap) { return cap == 0; }

    struct edge {
        int target;
        cost_t cost;
        cap_t residual_capacity;
        cap_t orig_capacity;
        size_t revid;
    };

    int n;
    vector<vector<edge>> graph;

    MinCostFlow(int n) : graph(n), n(n) {}

    void addEdge(int s, int e, cost_t cost, cap_t cap) {
        if (s == e) return;
        edge forward{ e, cost, cap, cap, graph[e].size() };
        edge backward{ s, -cost, 0, 0, graph[s].size() };
        graph[s].emplace_back(forward);
        graph[e].emplace_back(backward);
    }

    pair<cost_t, cap_t> augmentShortest(int s, int e, cap_t flow_limit) {
        auto infinite_cost = numeric_limits<cost_t>::max();
        auto infinite_flow = numeric_limits<cap_t>::max();
        vector<pair<cost_t, cap_t>> dist(n, make_pair(infinite_cost, 0));
        vector<int> from(n, -1), v(n);

        dist[s] = pair<cost_t, cap_t>(0, infinite_flow);
        queue<int> q;
        v[s] = 1; q.push(s);
        while(!q.empty()) {
            int cur = q.front();
            v[cur] = 0; q.pop();
            for (const auto& e : graph[cur]) {
                if (iszerocap(e.residual_capacity)) continue;
                auto next = e.target;
                auto ncost = dist[cur].first + e.cost;
                auto nflow = min(dist[cur].second, e.residual_capacity);
                if (dist[next].first > ncost) {
                    dist[next] = make_pair(ncost, nflow);
                    from[next] = e.revid;
                    if (v[next]) continue;
                    v[next] = 1; q.push(next);
                }
            }
        }

        auto p = e;
        auto pathcost = dist[p].first;
        auto flow = dist[p].second;
        if (iszerocap(flow)|| (flow_limit <= 0 && pathcost >= 0))
            return pair<cost_t, cap_t>(0, 0);
        if (flow_limit > 0) flow = min(flow, flow_limit);

        while (from[p] != -1) {
            auto nedge = from[p];
```

```
            auto np = graph[p][nedge].target;
            auto fedge = graph[p][nedge].revid;
            graph[p][nedge].residual_capacity += flow;
            graph[np][fedge].residual_capacity -= flow;
            p = np;
        }
        return make_pair(pathcost * flow, flow);
    }

    pair<cost_t,cap_t> solve(int s, int e,
        cap_t flow_minimum = numeric_limits<cap_t>::max()) {
        cost_t total_cost = 0;
        cap_t total_flow = 0;
        for(;;) {
            auto res = augmentShortest(s, e, flow_minimum - total_flow);
            if (res.second <= 0) break;
            total_cost += res.first;
            total_flow += res.second;
        }
        return make_pair(total_cost, total_flow);
    }
};
```

3.6 LR Flow

```
struct MaxFlowEdgeDemands{
    MaxFlowDinic mf;
    using flow_t = MaxFlowDinic::flow_t;

    vector<flow_t> ind, outd;
    flow_t D; int n;

    void init(int _n) {
        n = _n; D = 0; mf.init(n + 2);
        ind.clear(); outd.clear();
        ind.resize(n, 0); outd.resize(n, 0);
    }

    void add_edge(int s, int e, flow_t cap, flow_t demands = 0) {
        mf.add_edge(s, e, cap - demands);
        D += demands; ind[e] += demands; outd[s] += demands;
    }

    // returns { false, 0 } if infeasible
    // { true, maxflow } if feasible
    pair<bool, flow_t> solve(int source, int sink) {
        mf.add_edge(sink, source, numeric_limits<flow_t>::max());

        for (int i = 0; i < n; i++) {
            if (ind[i]) mf.add_edge(n, i, ind[i]);
            if (outd[i]) mf.add_edge(i, n + 1, outd[i]);
        }

        if (mf.solve(n, n + 1) != D) return{ false, 0 };

        for (int i = 0; i < n; i++) {
            if (ind[i]) mf.graph[i].pop_back();
            if (outd[i]) mf.graph[i].pop_back();
        }

        return{ true, mf.solve(source, sink) };
    }
};
```

3.7 General Matching

```
struct Blossom{
const int MAXN = 2001;
int vis[MAXN], par[MAXN], orig[MAXN], match[MAXN];
int aux[MAXN], t, N;
vector<int> conn[MAXN];
queue<int> Q;

void addEdge(int u, int v){
    conn[u].push_back(v);
    conn[v].push_back(u);
}

void init(int n){
    N = n; t = 0;
    for (int i=0; i<=n; i++){
        conn[i].clear();
        match[i] = aux[i] = par[i] = 0;
    }
}

void augment(int u, int v){
    int pv = v, nv;
    do{
        pv = par[v]; nv = match[pv];
```

```
        match[v] = pv; match[pv] = v;
        v = nv;
    } while (u != pv);
}

int lca(int v, int w){
    ++t;
    while (true){
        if (v){
            if (aux[v] == t) return v;
            aux[v] = t;
            v = orig[par[match[v]]];
        }
        swap(v, w);
    }
}

void blossom(int v, int w, int a){
    while(orig[v] != a){
        par[v] = w; w = match[v];
        if (vis[w] == 1) Q.push(w), vis[w] = 0;
        orig[v] = orig[w] = a;
        v = par[w];
    }
}

bool bfs(int u){
    fill(vis+1, vis+1+N, -1);
    iota(orig+1, orig+N+1, 1);
    Q = queue<int>(); Q.push(u); vis[u] = 0;
    while(!Q.empty()) {
        int v = Q.front(); Q.pop();
        for(int x: conn[v]) {
            if(vis[x] == -1) {
                par[x] = v; vis[x] = 1;
                if(!match[x]) return augment(u, x), true;
                Q.push(match[x]); vis[match[x]] = 0;
            }
            else if(vis[x] == 0 && orig[v] != orig[x]) {
                int a = lca(orig[v], orig[x]);
                blossom(x, v, a); blossom(v, x, a);
            }
        }
    }
    return false;
}

int Match() {
    int ans = 0;
    //find random matching
    vector<int> V(N-1); iota(V.begin(), V.end(), 1);
    shuffle(V.begin(), V.end(), mt19937(0x94949));
    for(auto x: V) if(!match[x]){
        for(auto y: conn[x]) if(!match[y]) {
            match[x] = y, match[y] = x;
            ++ans; break;
        }
    }
    for(int i=1; i<=N; ++i) if(!match[i] && bfs(i)) ++ans;
    return ans;
}
};
```

3.8 Bipartite Matching

```
// in: n, m, graph
// out: match, matched
// vertex cover: (reached[0][left_node] == 0) || (reached[1][right_node] == 1)
// O(E*sqrt(V))
struct BipartiteMatching {
    int n, m;
    vector<vector<int>> graph;
    vector<int> matched, match, edgeview, level;
    vector<int> reached[2];
    BipartiteMatching(int n, int m) : n(n), m(m), graph(n),
        matched(m, -1), match(n, -1) {}

    bool assignLevel() {
        bool reachable = false;
        level.assign(n, -1);
        reached[0].assign(n, 0);
        reached[1].assign(m, 0);
        queue<int> q;
        for (int i = 0; i < n; i++) {
            if (match[i] == -1) {
                level[i] = 0;
                reached[0][i] = 1;
                q.push(i);
            }
        }
```

```
        while (!q.empty()) {
            auto cur = q.front(); q.pop();
            for (auto adj : graph[cur]) {
                reached[1][adj] = 1;
                auto next = matched[adj];
                if (next == -1) {
                    reachable = true;
                }
                else if (level[next] == -1) {
                    level[next] = level[cur] + 1;
                    reached[0][next] = 1;
                    q.push(next);
                }
            }
        }
        return reachable;
    }

    int findpath(int nod) {
        for (int &i = edgeview[nod]; i < graph[nod].size(); i++) {
            int adj = graph[nod][i];
            int next = matched[adj];
            if (next >= 0 && level[next] != level[nod] + 1) continue;
            if (next == -1 || findpath(next)) {
                match[nod] = adj;
                matched[adj] = nod;
                return 1;
            }
        }
        return 0;
    }

    int solve() {
        int ans = 0;
        while (assignLevel()) {
            edgeview.assign(n, 0);
            for (int i = 0; i < n; i++)
                if (match[i] == -1)
                    ans += findpath(i);
        }
        return ans;
    }
};
```

3.9 General Min-Cut

```
// implementation of Stoer-Wagner algorithm
// O(V^3)
//usage
// MinCut mc;
// mc.init(n);
// for (each edge) mc.addEdge(a,b,weight);
// mincut = mc.solve();
// mc.cut = {0,1}^n describing which side the vertex belongs to.
struct MinCutMatrix
{
    typedef int cap_t;
    int n;
    vector<vector<cap_t>> graph;

    void init(int _n) {
        n = _n;
        graph = vector<vector<cap_t>>(n, vector<cap_t>(n, 0));
    }
    void addEdge(int a, int b, cap_t w) {
        if (a == b) return;
        graph[a][b] += w;
        graph[b][a] += w;
    }

    pair<cap_t, pair<int, int>> stMinCut(vector<int> &active) {
        vector<cap_t> key(n);
        vector<int> v(n);
        int s = -1, t = -1;
        for (int i = 0; i < active.size(); i++) {
            cap_t maxv = -1;
            int cur = -1;
            for (auto j : active) {
                if (v[j] == 0 && maxv < key[j]) {
                    maxv = key[j];
                    cur = j;
                }
            }
            t = s; s = cur;
            v[cur] = 1;
            for (auto j : active) key[j] += graph[cur][j];
        }
        return make_pair(key[s], make_pair(s, t));
    }
```

```
    vector<int> cut;

    cap_t solve() {
        cap_t res = numeric_limits<cap_t>::max();
        vector<vector<int>> grps;
        vector<int> active;
        cut.resize(n);
        for (int i = 0; i < n; i++) grps.emplace_back(1, i);
        for (int i = 0; i < n; i++) active.push_back(i);
        while (active.size() >= 2) {
            auto stcut = stMinCut(active);
            if (stcut.first < res) {
                res = stcut.first;
                fill(cut.begin(), cut.end(), 0);
                for (auto v : grps[stcut.second.first]) cut[v] = 1;
            }

            int s = stcut.second.first, t = stcut.second.second;
            if (grps[s].size() < grps[t].size()) swap(s, t);

            active.erase(find(active.begin(), active.end(), t));
            grps[s].insert(grps[s].end(), grps[t].begin(), grps[t].end());
            for (int i = 0; i < n; i++) { graph[i][s] += graph[i][t]; graph[i][t] = 0; }
            for (int i = 0; i < n; i++) { graph[s][i] += graph[t][i]; graph[t][i] = 0; }
            graph[s][s] = 0;
        }
        return res;
    }
};
```

4 Math

4.1 Basic Number Theory

```
typedef long long ll;
typedef unsigned long long ull;

// calculate lg2(a)
inline int lg2(ll a) {
    return 63 - __builtin_clzll(a);
}

// calculate the number of 1-bits
inline int bitcount(ll a) {
    return __builtin_popcountll(a);
}

// calculate ceil(a/b)
// |a|, |b| <= (2^63)-1 (does not dover -2^63)
ll ceildiv(ll a, ll b) {
    if (b < 0) return ceildiv(-a, -b);
    if (a < 0) return (-a) / b;
    return ((ull)a + (ull)b - 1ull) / b;
}

// calculate floor(a/b)
// |a|, |b| <= (2^63)-1 (does not cover -2^63)
ll floordiv(ll a, ll b) {
    if (b < 0) return floordiv(-a, -b);
    if (a >= 0) return a / b;
    return -(ll)(((ull)(-a) + b - 1) / b);
}

// calculate a*b % m
// x86-64 only
ll large_mod_mul(ll a, ll b, ll m) {
    return ll((__int128)a*(__int128)b%m);
}

// calculate a*b % m
// |m| < 2^62, x86 available
// O(logb)
ll large_mod_mul(ll a, ll b, ll m) {
    a %= m; b %= m; ll r = 0, v = a;
    while (b) {
        if (b&1) r = (r + v) % m;
        b >>= 1;
        v = (v << 1) % m;
    }
    return r;
}

// calculate n^k % m
ll modpow(ll n, ll k, ll m) {
    ll ret = 1;
    n %= m;
    while (k) {
        if (k & 1) ret = large_mod_mul(ret, n, m);
        n = large_mod_mul(n, n, m);
```

```
        k /= 2;
    }
    return ret;
}

// calculate gcd(a, b)
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

// find a pair (c, d) s.t. ac + bd = gcd(a, b)
pair<ll, ll> extended_gcd(ll a, ll b) {
    if (b == 0) return { 1, 0 };
    auto t = extended_gcd(b, a % b);
    return { t.second, t.first - t.second * (a / b) };
}

// find x in [0,m) s.t. ax === gcd(a, m) (mod m)
ll modinverse(ll a, ll m) {
    return (extended_gcd(a, m).first % m + m) % m;
}

// calculate modular inverse for 1 ~ n
void calc_range_modinv(int n, int mod, int ret[]) {
    ret[1] = 1;
    for (int i = 2; i <= n; ++i)
        ret[i] = (ll)(mod - mod/i) * ret[mod%i] % mod;
}
```

4.2 Primality Test - Miller-Rabin

```
bool test_witness(ull a, ull n, ull s) {
    if (a >= n) a %= n;
    if (a <= 1) return true;
    ull d = n >> s;
    ull x = modpow(a, d, n);
    if (x == 1 || x == n-1) return true;
    while (s-- > 1) {
        x = large_mod_mul(x, x, n);
        if (x == 1) return false;
        if (x == n-1) return true;
    }
    return false;
}

// test whether n is prime
// based on miller-rabin test
// O(logn*logn)
bool is_prime(ull n) {
    if (n == 2) return true;
    if (n < 2 || n % 2 == 0) return false;

    ull d = n >> 1, s = 1;
    for(; (d&1) == 0; s++) d >>= 1;

#define T(a) test_witness(a##ull, n, s)
    if (n < 4759123141ull) return T(2) && T(7) && T(61);
    return T(2) && T(325) && T(9375) && T(28178)
        && T(450775) && T(9780504) && T(1795265022);
#undef T
}
```

4.3 Rational Number

```
struct rational {
    long long p, q;

    void red() {
        if (q < 0) {
            p = -p;
            q = -q;
        }
        ll t = gcd((p >= 0 ? p : -p), q);
        p /= t;
        q /= t;
    }

    rational(): p(0), q(1) {}
    rational(long long p_): p(p_), q(1) {}
    rational(long long p_, long long q_): p(p_), q(q_) { red(); }

    bool operator==(const rational& rhs) const {
        return p == rhs.p && q == rhs.q;
    }
    bool operator!=(const rational& rhs) const {
        return p != rhs.p || q != rhs.q;
    }
    bool operator<(const rational& rhs) const {
        return p * rhs.q < rhs.p * q;
    }
```

```
    rational operator+(const rational& rhs) const {
        ll g = gcd(q, rhs.q);
        return rational(p * (rhs.q / g) + rhs.p * (q / g), (q / g) * rhs.q);
    }
    rational operator-(const rational& rhs) const {
        ll g = gcd(q, rhs.q);
        return rational(p * (rhs.q / g) - rhs.p * (q / g), (q / g) * rhs.q);
    }
    rational operator*(const rational& rhs) const {
        return rational(p * rhs.p, q * rhs.q);
    }
    rational operator/(const rational& rhs) const {
        return rational(p * rhs.q, q * rhs.p);
    }
};
```

4.4 Pollard-Rho

```
ll pollard_rho(ll n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<ll> dis(1, n - 1);
    ll x = dis(gen);
    ll y = x;
    ll c = dis(gen);
    ll g = 1;
    while (g == 1) {
        x = (modmul(x, x, n) + c) % n;
        y = (modmul(y, y, n) + c) % n;
        y = (modmul(y, y, n) + c) % n;
        g = gcd(abs(x - y), n);
    }
    return g;
}

// integer factorization
// O(n^0.25 * logn)
void factorize(ll n, vector<ll>& fl) {
    if (n == 1) {
        return;
    }
    if (n % 2 == 0) {
        fl.push_back(2);
        factorize(n / 2, fl);
    }
    else if (is_prime(n)) {
        fl.push_back(n);
    }
    else {
        ll f = pollard_rho(n);
        factorize(f, fl);
        factorize(n / f, fl);
    }
}
```

4.5 Chinese Remainder Theorem

```
// find x s.t.  x === a[0] (mod n[0])
//                === a[1] (mod n[1])
//                ...
// assumption: gcd(n[i], n[j]) = 1
ll chinese_remainder(ll* a, ll* n, int size) {
    if (size == 1) return *a;
    ll tmp = modinverse(n[0], n[1]);
    ll tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
    ll ora = a[1];
    ll tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2;
    n[1] *= n[0] / tgcd;
    ll ret = chinese_remainder(a + 1, n + 1, size - 1);
    n[1] /= n[0] / tgcd;
    a[1] = ora;
    return ret;
}
```

4.6 Gaussian Elimination

```
int n, inv;
vector<int> basis[505];
lint gyesu = 1;

void insert(vector<int> v){
  for(int i=0; i<n; i++){
    if(basis[i].size()) inv ^= 1; // inversion num increases
    if(v[i] && basis[i].empty()){
      basis[i] = v;
      return;
    }
    if(v[i]){
```

```
      lint minv = ipow(basis[i][i], mod - 2) * v[i] % mod;
      for(auto &j : basis[i]) j = (j * minv) % mod;
      gyesu *= minv;
      gyesu %= mod;
      for(int j=0; j<basis[i].size(); j++){
        v[j] += mod - basis[i][j];
        while(v[j] >= mod) v[j] -= mod;
      }
    }
  }
  puts("0");
  exit(0);
}

// Sample: Calculates Determinant in Z_p Field
int main(){
  scanf("%d",&n);
  for(int i=0; i<n; i++){
    vector<int> v(n);
    for(int j=0; j<n; j++) scanf("%d",&v[j]);
    if(i % 2 == 1) inv ^= 1;
    insert(v);
  }
  if(inv) gyesu = mod - gyesu;
  gyesu = ipow(gyesu, mod - 2);
  for(int i=0; i<n; i++) gyesu = gyesu * basis[i][i] % mod;
  cout << gyesu % mod << endl;
}
```

4.7 Linear Algebra

```
const int mod = 998244353;
using lint = long long;
lint ipow(lint x, lint p){
  lint ret = 1, piv = x;
  while(p){
    if(p & 1) ret = ret * piv % mod;
    piv = piv * piv % mod;
    p >>= 1;
  }
  return ret;
}
vector<int> berlekamp_massey(vector<int> x){
  vector<int> ls, cur;
  int lf, ld;
  for(int i=0; i<x.size(); i++){
    lint t = 0;
    for(int j=0; j<cur.size(); j++){
      t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
    }
    if((t - x[i]) % mod == 0) continue;
    if(cur.empty()){
      cur.resize(i+1);
      lf = i;
      ld = (t - x[i]) % mod;
      continue;
    }
    lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
    vector<int> c(i-lf-1);
    c.push_back(k);
    for(auto &j : ls) c.push_back(-j * k % mod);
    if(c.size() < cur.size()) c.resize(cur.size());
    for(int j=0; j<cur.size(); j++){
      c[j] = (c[j] + cur[j]) % mod;
    }
    if(i-lf+(int)ls.size()>=(int)cur.size()){
      tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
    }
    cur = c;
  }
  for(auto &i : cur) i = (i % mod + mod) % mod;
  return cur;
}
int get_nth(vector<int> rec, vector<int> dp, lint n){
  int m = rec.size();
  vector<int> s(m), t(m);
  s[0] = 1;
  if(m != 1) t[1] = 1;
  else t[0] = rec[0];
  auto mul = [&rec](vector<int> v, vector<int> w){
    int m = v.size();
    vector<int> t(2 * m);
    for(int j=0; j<m; j++){
      for(int k=0; k<m; k++){
        t[j+k] += 1ll * v[j] * w[k] % mod;
        if(t[j+k] >= mod) t[j+k] -= mod;
      }
    }
    for(int j=2*m-1; j>=m; j--){
      for(int k=1; k<=m; k++){
```

```
      t[j-k] += 1ll * t[j] * rec[k-1] % mod;
      if(t[j-k] >= mod) t[j-k] -= mod;
    }
  }
  t.resize(m);
  return t;
};
while(n){
  if(n & 1) s = mul(s, t);
  t = mul(t, t);
  n >>= 1;
}
lint ret = 0;
for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i] % mod;
return ret % mod;
}
int guess_nth_term(vector<int> x, lint n){
  if(n < x.size()) return x[n];
  vector<int> v = berlekamp_massey(x);
  if(v.empty()) return 0;
  return get_nth(v, x, n);
}
struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no duplicate please..
vector<int> get_min_poly(int n, vector<elem> M){
  // smallest poly P such that A^i = sum_{j < i} {A^j \times P_j}
  vector<int> rnd1, rnd2;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for(int i=0; i<n; i++){
    rnd1.push_back(randint(1, mod - 1));
    rnd2.push_back(randint(1, mod - 1));
  }
  vector<int> gobs;
  for(int i=0; i<2*n+2; i++){
    int tmp = 0;
    for(int j=0; j<n; j++){
      tmp += 1ll * rnd2[j] * rnd1[j] % mod;
      if(tmp >= mod) tmp -= mod;
    }
    gobs.push_back(tmp);
    vector<int> nxt(n);
    for(auto &i : M){
      nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
      if(nxt[i.x] >= mod) nxt[i.x] -= mod;
    }
    rnd1 = nxt;
  }
  auto sol = berlekamp_massey(gobs);
  reverse(sol.begin(), sol.end());
  return sol;
}
lint det(int n, vector<elem> M){
  vector<int> rnd;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for(int i=0; i<n; i++) rnd.push_back(randint(1, mod - 1));
  for(auto &i : M){
    i.v = 1ll * i.v * rnd[i.y] % mod;
  }
  auto sol = get_min_poly(n, M)[0];
  if(n % 2 == 0) sol = mod - sol;
  for(auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
  return sol;
}
```

4.8 Score Theorem

$D = (d_1, d_2, \cdots, d_n)$, $d_{i-1} \le d_i$가 그래프의 degree sequence가 될 수 있음은 $D' = (d'_1, \cdots, d'_{n-1})$, $d'_i = \begin{cases} d_i & i < n - d_n \\ d_i - 1 & i \ge n - d_n \end{cases}$ 이 그래프의 degree sequence가 될 수 있음과 동치이다.

4.9 Derangement

D_n을 $[n] \to [n]$ 상의 순열 중 고정점이 없는 것의 갯수라고 하자. 다음의 식이 성립한다.

- $D_n = (n-1)(D_{n-1} + D_{n-2}), \quad D_0 = 1,\ D_1 = 0.$
- $D_n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$

4.10 Catalan Number

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

4.11 Burnside's Lemma

Let X be a finite set and G be a permutation group over X. Define followings:

- $G(X) := \{\sigma(x) : \sigma \in G\}$ for each $x \in X$
- $X_\sigma := \{x \in X : \sigma(x) = x\}$ for each $\sigma \in G$
- $X/G := \{G(x) : x \in X\}$

Following holds.

$$|X/G| = \frac{1}{|G|}\sum_\sigma |X_\sigma|.$$

4.12 Fast Fourier Transform

```
const double MPI = 3.14159265358979323846264338327950288 4;

#define sz(v) ((int)(v).size())
#define all(v) (v).begin(),(v).end()
typedef complex<double> base;

void fft(vector <base> &a, bool invert)
{
    int n = sz(a);
    for (int i=1,j=0;i<n;i++){
        int bit = n >> 1;
        for (;j>=bit;bit>>=1) j -= bit;
        j += bit;
        if (i < j) swap(a[i],a[j]);
    }
    for (int len=2;len<=n;len<<=1){
        double ang = 2*MPI/len*(invert?-1:1);
        base wlen(cos(ang),sin(ang));
        for (int i=0;i<n;i+=len){
            base w(1);
            for (int j=0;j<len/2;j++){
                base u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if (invert){
        for (int i=0;i<n;i++) a[i] /= n;
    }
}

void multiply(const vector<ll> &a,const vector<ll> &b,vector<ll> &res)
{
    vector <base> fa(all(a)), fb(all(b));
    int n = 1;
    while (n < max(sz(a),sz(b))) n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false); fft(fb,false);
    for (int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for (int i=0;i<n;i++) res[i] = ll(round(fa[i].real()));
}
```

4.13 Number Theoretic Transform

```
const int A = 7, B = 26, P = A << B | 1, R = 3;
const int SZ = 20, N = 1 << SZ;

int Pow(int x, int y) {
    int r = 1;
    while (y) {
        if (y & 1) r = (long long)r * x % P;
        x = (long long)x * x % P;
        y >>= 1;
    }
    return r;
}

void FFT(int *a, bool f) {
    int i, j, k, x, y, z;
    j = 0;
    for (i = 1; i < N; i++) {
        for (k = N >> 1; j >= k; k >>= 1) j -= k;
        j += k;
        if (i < j) {
            k = a[i];
            a[i] = a[j];
            a[j] = k;
        }
    }
    for (i = 1; i < N; i <<= 1) {
        x = Pow(f ? Pow(R, P - 2) : R, P / i >> 1);
        for (j = 0; j < N; j += i << 1) {
```

```
            y = 1;
            for (k = 0; k < i; k++) {
                z = (long long)a[i | j | k] * y % P;
                a[i | j | k] = a[j | k] - z;
                if (a[i | j | k] < 0) a[i | j | k] += P;
                a[j | k] += z;
                if (a[j | k] >= P) a[j | k] -= P;
                y = (long long)y * x % P;
            }
        }
    }
    if (f) {
        j = Pow(N, P - 2);
        for (i = 0; i < N; i++) a[i] = (long long)a[i] * j % P;
    }
}

int X[N];

int main() {
    int i, n;
    scanf("%d", &n);
    for (i = 0; i <= n; i++) scanf("%d", &X[i]);
    FFT(X, false);
    for (i = 0; i < N; i++) X[i] = (long long)X[i] * X[i] % P;
    FFT(X, true);
    for (i = 0; i <= n + n; i++) printf("%d ", X[i]);
}
```

4.14 Simplex Method

```
namespace simplex {
  using T = long double;
  const int N = 10, M = 10;
  const T eps = 1e-7;
  int n, m;
  int Left[M], Down[N];
  T a[M][N], b[M], c[N], v, sol[N];

  bool eq(T a, T b) { return fabs(a - b) < eps;  }
  bool ls(T a, T b) { return a < b && !eq(a, b); }

  void init(){
      // initialize
  }

  void pivot(int x,int y) {
    swap(Left[x], Down[y]);
    T k = a[x][y]; a[x][y] = 1;
    vector<int> nz;
    for(int i = 1; i <= n; i++){
      a[x][i] /= k;
      if(!eq(a[x][i], 0)) nz.push_back(i);
    }
    b[x] /= k;

    for(int i = 1; i <= m; i++){
      if(i == x || eq(a[i][y], 0)) continue;
      k = a[i][y]; a[i][y] = 0;
      b[i] -= k*b[x];
      for(int j : nz) a[i][j] -= k*a[x][j];
    }
    if(eq(c[y], 0)) return;
    k = c[y]; c[y] = 0;
    v += k*b[x];
    for(int i : nz) c[i] -= k*a[x][i];
  }

  // 0: found solution, 1: no feasible solution, 2: unbounded
  int solve() {
    for(int i = 1; i <= n; i++) Down[i] = i;
    for(int i = 1; i <= m; i++) Left[i] = n+i;
    while(1) { // Eliminating negative b[i]
      int x = 0, y = 0;
      for(int i = 1; i <= m; i++) if (ls(b[i], 0) && (x == 0 || b[i] < b[x])) x = i;
      if(x == 0) break;
      for(int i = 1; i <= n; i++) if (ls(a[x][i], 0) && (y == 0 || a[x][i] < a[x][y])) y = i;
      if(y == 0) return 1;
      pivot(x, y);
    }
    while(1) {
      int x = 0, y = 0;
      for(int i = 1; i <= n; i++)
        if (ls(0, c[i]) && (!y || c[i] > c[y])) y = i;
      if(y == 0) break;
      for(int i = 1; i <= m; i++)
        if (ls(0, a[i][y]) && (!x || b[i]/a[i][y] < b[x]/a[x][y])) x = i;
      if(x == 0) return 2;
      pivot(x, y);
    }
```

```
    for(int i = 1; i <= m; i++) if(Left[i] <= n) sol[Left[i]] = b[i];
    return 0;
  }
}

using namespace simplex;
/*
n := number of variables
m := number of constraints
a[1~m][1~n] := constraints
b[1~m] := constraints value (b[i] can be negative)
c[1~n] := maximum coefficient
v := results
sol[i] := 등호조건, i번째 변수의 값
ex) Maximize p = 6x + 14y + 13z
    Constraints: 0.5x + 2y + z ≤ 24
                 x + 2y + 4z ≤ 60
    n = 2, m = 3, a = [[0.5, 2, 1], [1, 2, 4]], b = [24, 60], c = [6, 14, 13]
*/
```

4.15 Special Primes

- $n \le 1,000,000$ 약수 최대 240개 (720,720)
- $n \le 1,000,000,000$ 최대 1,344개 (735,134,400)
- up to 10,000: 소수 1,229개 (9,973)
- up to 100,000: 소수 9,592개 (99,991)
- up to 1,000,000: 소수 78,498개 (999,983)
- up to 1,000,000,000: 소수 50,847,534개 (999,999,937)
- 10,007; 10,009; 10,111; 31,567; 70,001; 1,000,003; 1,000,033; 4,000,037
- 99,999,989; 999,999,937; 1,000,000,007; 1,000,000,009; 9,999,999,967
- $998244353 = 119 \times 2^{23} + 1$, primitive 3
- $985661441 = 235 \times 2^{22} + 1$, primitive 3
- $1012924417 = 483 \times 2^{21} + 1$, primitive 5

5 String

5.1 Hashing

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

typedef long long ll;

struct Hashing{
        //1e9-63, 1e9+7, 1e9+9, 1e9+103
        //1e5+3, 1e5+13, 131071, 524287
        vector<ll> hash1, base1;
        ll p, mod;
        Hashing(){
                p = 1e9-63, mod = 179;
        }
        Hashing(ll _p, ll _mod) : p(_p), mod(_mod) {}
        Hashing(string &s, ll _p, ll _mod) : Hashing(_p, _mod) {
                int n = s.size();
                hash1 = vector<ll>(n+1, 0);
                base1 = vector<ll>(n+1, 0);
                base1[0] = 1; base1[1] = p;
                for(int i=n-1; i>=0; i--){
                        hash1[i] = (s[i] + hash1[i+1] * p) % mod;
                }
                for(int i=2; i<=n; i++){
                        base1[i] = base1[i-1] * p % mod;
                }
        }
        int substr(int l, int r){ //[s, e]
                ll ret = hash1[l] - hash1[r+1] * base1[r-l+1];
                ret %= mod; ret += mod; ret %= mod;
                return ret;
        }
}h1, h2;

string s; int m;

bool chk(int q, int x){
        if(x == 0) return 1;
        int a = h1.substr(s.size()-x, s.size()-1);
        int b = h1.substr(q-x, q-1);
        int aa = h2.substr(s.size()-x, s.size()-1);
        int bb = h2.substr(q-x, q-1);
```

```
        return a == b && aa == bb;
}
```

5.2 KMP

```
typedef vector<int> seq_t;

void calculate_pi(vector<int>& pi, const seq_t& str) {
    pi[0] = -1;
    for (int i = 1, j = -1; i < str.size(); i++) {
        while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
        if (str[i] == str[j + 1])
            pi[i] = ++j;
        else
            pi[i] = -1;
    }
}

// returns all positions matched
// O(|text|+|pattern|)
vector<int> kmp(const seq_t& text, const seq_t& pattern) {
    vector<int> pi(pattern.size()), ans;
    if (pattern.size() == 0) return ans;
    calculate_pi(pi, pattern);
    for (int i = 0, j = -1; i < text.size(); i++) {
        while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
        if (text[i] == pattern[j + 1]) {
            j++;
            if (j + 1 == pattern.size()) {
                ans.push_back(i - j);
                j = pi[j];
            }
        }
    }
    return ans;
}
```

5.3 Aho-Corasick

```
const int MAXN = 100005, MAXC = 26;
int trie[MAXN][MAXC], fail[MAXN], term[MAXN], piv;
void init(vector<string> &v){
    memset(trie, 0, sizeof(trie));
    memset(fail, 0, sizeof(fail));
    memset(term, 0, sizeof(term));
    piv = 0;
    for(auto &i : v){
        int p = 0;
        for(auto &j : i){
            if(!trie[p][j]) trie[p][j] = ++piv;
            p = trie[p][j];
        }
        term[p] = 1;
    }
    queue<int> que;
    for(int i=0; i<MAXC; i++){
        if(trie[0][i]) que.push(trie[0][i]);
    }
    while(!que.empty()){
        int x = que.front();
        que.pop();
        for(int i=0; i<MAXC; i++){
            if(trie[x][i]){
                int p = fail[x];
                while(p && !trie[p][i]) p = fail[p];
                p = trie[p][i];
                fail[trie[x][i]] = p;
                if(term[p]) term[trie[x][i]] = 1;
                que.push(trie[x][i]);
            }
        }
    }
}
bool query(string &s){
    int p = 0;
    for(auto &i : s){
        while(p && !trie[p][i]) p = fail[p];
        p = trie[p][i];
        if(term[p]) return 1;
    }
    return 0;
}
```

5.4 Manacher

```
const int MAXN = 1005;
int aux[2 * MAXN - 1];
void solve(int n, int *str, int *ret){
    // *ret : number of nonobvious palindromic character pair
    for(int i=0; i<n; i++){
```

```
        aux[2*i] = str[i];
        if(i != n-1) aux[2*i+1] = -1;
    }
    int p = 0, c = 0;
    for(int i=0; i<2*n-1; i++){
        int cur = 0;
        if(i <= p) cur = min(ret[2 * c - i], p - i);
        while(i - cur - 1 >= 0 && i + cur + 1 < 2*n-1 && aux[i-cur-1] == aux[i+cur+1]){
            cur++;
        }
        ret[i] = cur;
        if(i + ret[i] > p){
            p = i + ret[i];
            c = i;
        }
    }
}
```

5.5 Suffix Array

```
/**
 * Description: Builds suffix array for a string.
 * \texttt{sa[i]} is the starting index of the suffix which
 * is £i£'th in the sorted suffix array.
 * The returned vector is of size £n+1£, and \texttt{sa[0] = n}.
 * The \texttt{lcp} array contains longest common prefixes for
 * neighbouring strings in the suffix array:
 * \texttt{lcp[i] = lcp(sa[i], sa[i-1])}, \texttt{lcp[0] = 0}.
 * The input string must not contain any zero bytes.
 * Time: O(n \log n)
 */
#define rep(i, from, to) for (int i = from; i < (to); ++i)
#define all(x) x.begin(), x.end()
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        rep(i,1,n) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                    s[i + k] == s[j + k]; k++);
    }
};
```

5.6 Z Algorithm

```
/**
 * Description: z[x] computes the length of the longest common prefix of s[i:] and s,
 * except z[0] = 0. (abacaba -> 0010301)
 * Time: O(n)
 */

vi Z(string S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - l]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}
```

6 Geometry

6.1 Basic Geometry

```
const double eps = 1e-9;

inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
```

```cpp
    return (lhs < rhs) ? -1 : 1;
}

inline bool is_between(double check, double a, double b) {
    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}

struct Point {
    double x, y;
    bool operator==(const Point& rhs) const {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    Point operator+(const Point& rhs) const {
        return Point{ x + rhs.x, y + rhs.y };
    }
    Point operator-(const Point& rhs) const {
        return Point{ x - rhs.x, y - rhs.y };
    }
    Point operator*(double t) const {
        return Point{ x * t, y * t };
    }
};

struct Circle {
    Point center;
    double r;
};

struct Line {
    Point pos, dir;
};

inline double inner(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}

inline double outer(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

inline int ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}

inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}

inline double dist(const Point& a, const Point& b) {
    return sqrt(inner(a - b, a - b));
}

inline double dist2(const Point &a, const Point &b) {
    return inner(a - b, a - b);
}

inline double dist(const Line& line, const Point& point, bool segment = false) {
    double c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    double c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1) <= 0) return dist(line.pos + line.dir, point);
    return dist(line.pos + line.dir * (c1 / c2), point);
}

bool get_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    ret = b.pos + b.dir * t2;
    return true;
}

bool get_segment_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t1 = -outer(b.pos - a.pos, b.dir) / mdet;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return false;
    ret = b.pos + b.dir * t2;
    return true;
}

Point inner_center(const Point &a, const Point &b, const Point &c) {
    double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
    double w = wa + wb + wc;
    return Point{ (wa * a.x + wb * b.x + wc * c.x) / w,
            (wa * a.y + wb * b.y + wc * c.y) / w };
```

```
}

Point outer_center(const Point &a, const Point &b, const Point &c) {
    Point d1 = b - a, d2 = c - a;
    double area = outer(d1, d2);
    double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
        + d1.y * d2.y * (d1.y - d2.y);
    double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
        + d1.x * d2.x * (d1.x - d2.y);
    return Point{ a.x + dx / area / 2.0, a.y - dy / area / 2.0 };
}

vector<Point> circle_line(const Circle& circle, const Line& line) {
    vector<Point> result;
    double a = 2 * inner(line.dir, line.dir);
    double b = 2 * (line.dir.x * (line.pos.x - circle.center.x)
        + line.dir.y * (line.pos.y - circle.center.y));
    double c = inner(line.pos - circle.center, line.pos - circle.center)
        - circle.r * circle.r;
    double det = b * b - 2 * a * c;
    int pred = diff(det, 0);
    if (pred == 0)
        result.push_back(line.pos + line.dir * (-b / a));
    else if (pred > 0) {
        det = sqrt(det);
        result.push_back(line.pos + line.dir * ((-b + det) / a));
        result.push_back(line.pos + line.dir * ((-b - det) / a));
    }
    return result;
}

vector<Point> circle_circle(const Circle& a, const Circle& b) {
    vector<Point> result;
    int pred = diff(dist(a.center, b.center), a.r + b.r);
    if (pred > 0) return result;
    if (pred == 0) {
        result.push_back((a.center * b.r + b.center * a.r) * (1 / (a.r + b.r)));
        return result;
    }
    double aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
    double bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
    double tmp = (bb - aa) / 2.0;
    Point cdiff = b.center - a.center;
    if (diff(cdiff.x, 0) == 0) {
        if (diff(cdiff.y, 0) == 0)
            return result; // if (diff(a.r, b.r) == 0): same circle
        return circle_line(a, Line{ Point{ 0, tmp / cdiff.y }, Point{ 1, 0 } });
    }
    return circle_line(a,
        Line{ Point{ tmp / cdiff.x, 0 }, Point{ -cdiff.y, cdiff.x } });
}

Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
    Point ba = b - a, cb = c - b;
    Line p{ (a + b) * 0.5, Point{ ba.y, -ba.x } };
    Line q{ (b + c) * 0.5, Point{ cb.y, -cb.x } };
    Circle circle;
    if (!get_cross(p, q, circle.center))
        circle.r = -1;
    else
        circle.r = dist(circle.center, a);
    return circle;
}

Circle circle_from_2pts_rad(const Point& a, const Point& b, double r) {
    double det = r * r / dist2(a, b) - 0.25;
    Circle circle;
    if (det < 0)
        circle.r = -1;
    else {
        double h = sqrt(det);
        // center is to the left of a->b
        circle.center = (a + b) * 0.5 + Point{ a.y - b.y, b.x - a.x } * h;
        circle.r = r;
    }
    return circle;
}
```

6.2 Convex Hull

```
// find convex hull
// O(n*logn)
vector<Point> convex_hull(vector<Point>& dat) {
    if (dat.size() <= 3) return dat;
    vector<Point> upper, lower;
    sort(dat.begin(), dat.end(), [](const Point& a, const Point& b) {
        return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    });
    for (const auto& p : dat) {
        while (upper.size() >= 2 && ccw(*++upper.rbegin(), *upper.rbegin(), p) >= 0)
```

```
            upper.pop_back();
        while (lower.size() >= 2 && ccw(*++lower.rbegin(), *lower.rbegin(), p) <= 0)
            lower.pop_back();
        upper.emplace_back(p);
        lower.emplace_back(p);
    }
    upper.insert(upper.end(), ++lower.rbegin(), --lower.rend());
    return upper;
}
```

6.3 Rotating Calipers

```
// get all antipodal pairs
// O(n)
void antipodal_pairs(vector<Point>& pt) {
    // calculate convex hull
    sort(pt.begin(), pt.end(), [](const Point& a, const Point& b) {
        return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    });
    vector<Point> up, lo;
    for (const auto& p : pt) {
        while (up.size() >= 2 && ccw(*++up.rbegin(), *up.rbegin(), p) >= 0) up.pop_back();
        while (lo.size() >= 2 && ccw(*++lo.rbegin(), *lo.rbegin(), p) <= 0) lo.pop_back();
        up.emplace_back(p);
        lo.emplace_back(p);
    }
    for (int i = 0, j = (int)lo.size() - 1; i + 1 < up.size() || j > 0; ) {
        get_pair(up[i], lo[j]);  // DO WHAT YOU WANT
        if (i + 1 == up.size()) { --j; }
        else if (j == 0) { ++i; }
        else if ((long long)(up[i + 1].y - up[i].y) * (lo[j].x - lo[j - 1].x)
                > (long long)(up[i + 1].x - up[i].x) * (lo[j].y - lo[j - 1].y)) {
            ++i;
        }
        else {
            --j;
        }
    }
}
```

6.4 Point in Polygon Test

```
typedef double coord_t;

inline coord_t is_left(Point p0, Point p1, Point p2) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

// point in polygon test
// http://geomalgorithms.com/a03-_inclusion.html
bool is_in_polygon(Point p, vector<Point>& poly) {
    int wn = 0;
    for (int i = 0; i < poly.size(); ++i) {
        int ni = (i + 1 == poly.size()) ? 0 : i + 1;
        if (poly[i].y <= p.y) {
            if (poly[ni].y > p.y) {
                if (is_left(poly[i], poly[ni], p) > 0) {
                    ++wn;
                }
            }
        }
        else {
            if (poly[ni].y <= p.y) {
                if (is_left(poly[i], poly[ni], p) < 0) {
                    --wn;
                }
            }
        }
    }
    return wn != 0;
}
```

6.5 Delaunay Triangulation

```
struct triple {
  int i, j, k;
  triple() {}
  triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y) {
  int n = x.size();
  vector<T> z(n);
  vector<triple> ret;
  for (int i = 0; i < n; i++)
    z[i] = x[i] * x[i] + y[i] * y[i];
  for (int i = 0; i < n-2; i++) {
    for (int j = i+1; j < n; j++) {
      for (int k = i+1; k < n; k++) {
        if (j == k) continue;
```

```
        double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j]-z[i]);
        double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k]-z[i]);
        double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j]-y[i]);
        bool flag = zn < 0;
        for (int m = 0; flag && m < n; m++)
          flag = flag && ((x[m]-x[i])*xn +(y[m]-y[i])*yn + (z[m]-z[i])*zn <= 0);
        if (flag) ret.push_back(triple(i, j, k));
      }
    }
  }
  return ret;
}
```

6.6 Sort By Angle

```
inline ll ccw(pi p1, pi p2, pi p3){
    ll tmp = p1.x*p2.y + p2.x*p3.y + p3.x*p1.y;
    tmp -= p1.y*p2.x + p2.y*p3.x + p3.y*p1.x;
    if (tmp > 0) return 1;
    if (tmp < 0) return -1;
    return 0;
}

inline ll hypot(pi p){
    return p.x*p.x + p.y*p.y;
}

inline int cmp(const pi &a, const pi &b){
    if ((a > pi(0, 0)) ^ (b > pi(0, 0))) return a > b;
    if (ccw(a, pi(0, 0), b) != 0) return ccw(a, pi(0, 0), b) > 0;
    return hypot(a) < hypot(b);
}
```

6.7 Half-Plane Intersection

```
const double eps = 1e-8;
typedef pair<long double, long double> pi;
bool z(long double x){ return fabs(x) < eps; }
struct line{
        long double a, b, c;
        bool operator<(const line &l)const{
                bool flag1 = pi(a, b) > pi(0, 0);
                bool flag2 = pi(l.a, l.b) > pi(0, 0);
                if(flag1 != flag2) return flag1 > flag2;
                long double t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b));
                return z(t) ? c * hypot(l.a, l.b) < l.c * hypot(a, b) : t > 0;
        }
        pi slope(){ return pi(a, b); }
};
pi cross(line a, line b){
        long double det = a.a * b.b - b.a * a.b;
        return pi((a.c * b.b - a.b * b.c) / det, (a.a * b.c - a.c * b.a) / det);
}
bool bad(line a, line b, line c){
        if(ccw(pi(0, 0), a.slope(), b.slope()) <= 0) return false;
        pi crs = cross(a, b);
        return crs.first * c.a + crs.second * c.b >= c.c;
}
bool solve(vector<line> v, vector<pi> &solution){ // ax + by <= c;
        sort(v.begin(), v.end());
        deque<line> dq;
        for(auto &i : v){
                if(!dq.empty() && z(ccw(pi(0, 0), dq.back().slope(), i.slope()))) continue;
                while(dq.size() >= 2 && bad(dq[dq.size()-2], dq.back(), i)) dq.pop_back();
                while(dq.size() >= 2 && bad(i, dq[0], dq[1])) dq.pop_front();
                dq.push_back(i);
        }
        while(dq.size() > 2 && bad(dq[dq.size()-2], dq.back(), dq[0])) dq.pop_back();
        while(dq.size() > 2 && bad(dq.back(), dq[0], dq[1])) dq.pop_front();
        vector<pi> tmp;
        for(int i=0; i<dq.size(); i++){
                line cur = dq[i], nxt = dq[(i+1)%dq.size()];
                if(ccw(pi(0, 0), cur.slope(), nxt.slope()) <= eps) return false;
                tmp.push_back(cross(cur, nxt));
        }
        solution = tmp;
        return true;
}
```

7 Miscellaneous

7.1 OSRank

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <functional>
using namespace __gnu_pbds;
using ordered_set = tree<int, null_type, less<int>,
  rb_tree_tag, tree_order_statistics_node_update>;
```

```
// for multi-set like osrank,
// use ordered_set for pair<int, int> with counter global var
int main(){
  ordered_set X;
  for (int i=1; i<10; i+=2) X.insert(i); // 1 3 5 7 9
  cout << *X.find_by_order(2) << endl; // 5
  cout << X.order_of_key(6) << endl; // 3
  cout << X.order_of_key(7) << endl; // 3
  X.erase(3);
}
```

7.2 mt19937 random

```
// dependency :
// how to use mt19937

int rand(mt19937 &rd, int l, int r){
    // mt19937 rd((unsigned)chrono::steady_clock::now().time_since_epoch().count());
    // mt19937 rd(0x917917);
    uniform_int_distribution<int> rnd(l, r);
    return rnd(rd);
}
```

7.3 FastIO

```
static char buf[1 << 19]; // size : any number geq than 1024
static int idx = 0;
static int bytes = 0;
static inline int _read() {
  if (!bytes || idx == bytes) {
    bytes = (int)fread(buf, sizeof(buf[0]), sizeof(buf), stdin);
    idx = 0;
  }
  return buf[idx++];
}
static inline int _readInt() {
  int x = 0, s = 1;
  int c = _read();
  while (c <= 32) c = _read();
  if (c == '-') s = -1, c = _read();
  while (c > 32) x = 10 * x + (c - '0'), c = _read();
  if (s < 0) x = -x;
  return x;
}
```

7.4 Mo's

```
// dependency :
// Mo's Algorithm
// Time Complexity : O((N+Q) sqrt N T(N))

struct Query{
    int s, e, x;
    bool operator < (const Query &t) const {
        return tie(s/400, e) < tie(t.s/400, t.e);
    }
};

while(qry[i].s < l) insert(--l);
while(qry[i].e > r) insert(++r);
while(qry[i].s > l) erase(l++);
while(qry[i].e < r) erase(r--);
res[qry[i].x] = get();
```

7.5 Bit hacks

```
int __builtin_clz(int x);// number of leading zero
int __builtin_ctz(int x);// number of trailing zero
int __builtin_clzll(long long x);// number of leading zero
int __builtin_ctzll(long long x);// number of trailing zero
int __builtin_popcount(int x);// number of 1-bits in x
int __builtin_popcountll(long long 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);

// compute next perm. ex) 00111, 01011, 01101, 01110, 10011, 10101..
long long next_perm(long long v){
        long long t = v | (v-1);
        return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}
```

7.6 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("avx2")` can double performance of vectorized code, but causes crashes on old machines.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

7.7 체계적인 접근을 위한 질문들

"알고리즘 문제 해결 전략" 에서 발췌함

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (brute force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 문제를 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)

7.8 Parallel Binary Search

- 이분탐색의 결정 문제를 원소들을 쭉 훑어보면서 해결할 수 있을 때 묶어서 처리

7.9 DnC Optimization

- $DP[i][j] = Min_{k<j}(DP[i-1][k] + C[k][j])$
- C 배열은 Monge array여야 함
- Monge array: 임의의 $a \le b \le c \le d$에 대해 $C[a][c] + C[b][d] \le C[a][d] + C[b][c]$
- Generic Implementation: Even though implementation varies based on problem, here's a fairly generic template. The function compute computes one row `i` of states `dp_cur`, given the previous row `i1` of states `dp_before`. It has to be called with `compute(0, n-1, 0, n-1)`.

```
int n;
long long C(int i, int j);
vector<long long> dp_before(n), dp_cur(n);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr)
{
    if (l > r)
        return;
    int mid = (l + r) >> 1;
    pair<long long, int> best = {INF, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(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, optr);
}
```

7.10 Knuth Optimization

- Recurrence: $DP[i][j] = Min_{i \le k<j}(DP[i][k] + DP[k+1][j] + C[i][j])$
- Condition: $C[i][j]$ is a Monge array, and satisfies $C[a][d] \ge C[b][c]$ for $a \le b \le c \le d$.
- Complexity: $O(n^3) \to O(n^2)$
- $opt[i][j]$를 $DP[i][j]$에서 최솟값을 주는 k (여러 개 있으면 가장 왼쪽) 라고 할 때, $opt[i][j-1] \le opt[i][j] \le opt[i+1][j]$