KTH Royal Institute of Technology

Omogen Heap

Simon Lindholm, Johan Sannemo, Mårten Wiman

2023-11-23

Strings (1)

KMP.h

Usage: 0-base. pmt[i] = s[0..i]'s common longest prefix and suffix. kmp[i] = ith matched begin position.
Time: $\mathcal{O}(n)$

d41d8c, 40 lines

```
vector<int> get_pmt(const string& s) {
    int n = s.size();
    vector<int> pmt(n, 0);
    // except finding itself by searching from s[0]
    int b = 1, m = 0;
    // s[b + m]: letter to compare
    while (b + m < n) {
        if (s[b+m] == s[m]) {
            pmt[b+m] = m + 1;
            m++;
        } else {
            if (m > 0) {
                b += m - pmt[m-1];
                m = pmt[m-1];
            } else {
                b++;
            }
        }
    }
    return pmt;
}
vector<int> KMP(const string& hay, const string& needle) {
    vector<int> pmt = get_pmt(needle);
    vector<int> ret;
    int b = 0, m = 0;
    while (b <= (int)hay.size() - needle.size()) {
        if (m < needle.size() && hay[b+m] == needle[m]) {
            m++;
            if (m == needle.size()) ret.push_back(b);
        } else {
            if (m > 0) {
                b += m - pmt[m-1];
                m = pmt[m-1];
            } else {
                b++;
            }
        }
    }
    return ret;
}
```

Zfunc.h

Usage: Z[i] stores lcp of s[0..] and s[i..]
Time: $\mathcal{O}(n)$

d41d8c, 9 lines

```
vector<int> Z(string &s) {
    vector<int> ret(s.size(), 0); ret[0] = s.size();
    for(int i = 1, l = 0, r = 1; i < s.size(); i++) {
        ret[i] = max(0, min(ret[i-l], r-i));
        while(ret[i]+i < s.size() && s[i+ret[i]] == s[ret[i]])
            ret[i]++;
        if(i+ret[i] > r) r = i+ret[i], l = i;
    }
    return ret;
}
```

Manacher.h

Usage: mana[i] stores radius of maximal palindrome of intervened string. Single char is radius 0. Max element of mana is equal to real longest palindrome length.
Time: $\mathcal{O}(N)$, $N = 10^5$ in 4ms.

d41d8c, 11 lines

```
vector<int> mana(string &s) {
    string t = ".";
    for(auto i : s) { t += i; t += '.'; }
    vector<int> ret(t.size(), 0);
    for(int i = 0, c = 0, r = 0; i < (int)t.size(); i++) {
        if(i < r) ret[i] = min(r-i, ret[2*c-i]);
        while(i-ret[i]-1 >= 0 && i+ret[i]+1 < (int)t.size() &&
            t[i-ret[i]-1] == t[i+ret[i]+1]) ret[i]++;
        if(r < i+ret[i]) r = i+ret[i], c = i;
    }
    return ret;
}
```

SuffixArray.h

Usage: 0-base index. sa[i]: lexicographically (i+1)'th suffix (of d letters). lcp[i]: lcp between sa[i] and sa[i+1]. r[i]: rank of s[i..n-1] when only consider first d letters. nr: temp array for next rank. cnt[i]: number of positions which has i of next rank. rf[r]: lexicographically first position which suffixes (of d letters) has rank r. rdx[i]: lexicographically (i+1)'th suffix when only consider (d+1)'th 2d'th letters.
Time: $\mathcal{O}(n \log n)$, $N = 5 \times 10^5$ in 176ms.

d41d8c, 41 lines

```
void suffix_array(string S, vector<int> &sa, vector<int> &lcp)
    {
    int n = S.size();
    vector<int> r(n), nr(n), rf(n), rdx(n);
    sa.resize(n); lcp.resize(n);

    for (int i = 0; i < n; i++) sa[i] = i;
    sort(sa.begin(), sa.end(), [&](int a, int b) { return S[a]
        < S[b]; });
    for (int i = 1; i < n; i++) r[sa[i]] = r[sa[i - 1]] + (S[sa
        [i - 1]] != S[sa[i]]);

    for (int d = 1; d < n; d <<= 1) {
        for (int i = n - 1; i >= 0; i--) {
            rf[r[sa[i]]] = i;
        }
        int j = 0;
        for (int i = n - d; i < n; i++) rdx[j++] = i;
        for (int i = 0; i < n; i++) {
            if (sa[i] >= d) rdx[j++] = sa[i] - d;
        }
        for (int i = 0; i < n; i++) {
            sa[rf[r[rdx[i]]]++] = rdx[i];
        }
        nr[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            if (r[sa[i]] != r[sa[i - 1]]) {
                nr[sa[i]] = nr[sa[i - 1]] + 1;
            }
            else {
                int prv = (sa[i - 1] + d >= n ? -1 : r[sa[i -
                    1] + d]);
                int cur = (sa[i] + d >= n ? -1 : r[sa[i] + d]);
                nr[sa[i]] = nr[sa[i - 1]] + (prv != cur);
            }
        }
        swap(r, nr);
        if (r[sa[n-1]] == n-1) break;
    }
    for (int i = 0, len = 0; i < n; ++i, len = max(len - 1, 0))
        {
        if (r[i] == n - 1) continue;
        for (int j = sa[r[i] + 1]; S[i + len] == S[j + len]; ++
            len);
        lcp[r[i]] = len;
    }
}
```

DeBruijnSequence.h

Description: Calculate length-L DeBruijn sequence.
Usage: Returns 1-base index. K is the number of alphabet, N is the length of different substring, L is the return length ($0 <= L <= K^N$). vector<int> seq = de_bruijn(K, N, L);
Time: $\mathcal{O}(L)$, $N = L = 10^5, K = 10$ in 12ms.

d41d8c, 23 lines

```
vector<int> de_bruijn(int K, int N, int L) {
    vector<int> ans, tmp;
    function<void(int)> dfs = [&](int T) {
        if((int)ans.size() >= L) return;
        if((int)tmp.size() == N) {
            if(N%T == 0)
                for(int i = 0; i < T && (int)ans.size() < L; i
                    ++)
                    ans.push_back(tmp[i]);
        } else {
            int k = ((int)tmp.size()-T >= 0 ? tmp[(int)tmp.size
                ()-T] : 1);
            tmp.push_back(k);
            dfs(T);
            tmp.pop_back();
            for(int i = k+1; i <= K; i++) {
                tmp.push_back(i);
                dfs((int)tmp.size());
                tmp.pop_back();
            }
        }
    };
    dfs(1);
    return ans;
}
```

SuffixAutomaton.h

Usage: add(c) adds c at the end of string. topo(f) executes f while topological sort when erase edges. f(x, y, c): y->x edge marked as c. Note that c is 0-base.
Time: add is amortized $\mathcal{O}(1)$, topo is $\mathcal{O}(n)$, no data. 10^6 add call and one topo call in 668ms.

d41d8c, 65 lines

```
template <int MAXN>
struct SuffixAutomaton {
    struct Node {
        int nxt[MAXN];
        int len = 0, link = 0;
        Node() { memset(nxt, -1, sizeof nxt); }
    };

    int root = 0;
    vector<Node> V;
```

```
    SuffixAutomaton() {
        V.resize(1);
        V.back().link = -1;
    }

    void add(int c) {
        V.push_back(Node());
        V.back().len = V[root].len+1;
        int tmp = root;
        root = (int)V.size()-1;
        while(tmp != -1 && V[tmp].nxt[c] == -1) {
            V[tmp].nxt[c] = root;
            tmp = V[tmp].link;
        }
        if(tmp != -1) {
            int x = V[tmp].nxt[c];
            if(V[tmp].len+1 < V[x].len) {
                int y = x;
                x = (int)V.size();
                V.push_back(V[y]);
                V.back().len = V[tmp].len+1;
                V[y].link = x;
                while(tmp != -1 && V[tmp].nxt[c] == y) {
                    V[tmp].nxt[c] = x;
                    tmp = V[tmp].link;
                }
            }
            V[root].link = x;
        }
    }

    void topo(function<void(int, int, int)> f) {
        vector<int> indeg(V.size(), 0);
        for(auto &node : V) {
            for(auto j : node.nxt) {
                if(j == -1) continue;
                indeg[j]++;
            }
        }
        queue<int> Q;
        for(int i = 0; i < (int)indeg.size(); i++)
            if(indeg[i] == 0) Q.push(i);
        while(Q.size()) {
            int tmp = Q.front(); Q.pop();
            auto &node = V[tmp];
            for(int j = 0; j < MAXN; j++) {
                if(node.nxt[j] == -1) continue;
                f(node.nxt[j], tmp, j);
                if(--indeg[node.nxt[j]] == 0)
                    Q.push(node.nxt[j]);
            }
        }
    }
};
```

eertree.h

Description: eertree.
Usage: `add is same as suffix automaton. Note that c is 0-base.`
Time: add is amortized $\mathcal{O}(1)$, 10^6 add in 212ms.

d41d8c, 43 lines

```
template <int MAXN>
struct eertree {
    struct Node {
        int len = 0, link = 0, cnt = 0;
        array<int, MAXN> nxt;
        Node() { fill(nxt.begin(), nxt.end(), -1); }
    };

    vector<int> S;
    vector<Node> V;
    int root = 0;

    eertree() {
        V.resize(2);
        V[0].len = -1;
    }

    void add(int c) {
        S.push_back(c);
        for(int tmp = root; ; tmp = V[tmp].link) {
            auto iter = S.rbegin()+V[tmp].len+1;
            if(iter < S.rend() && *iter == c) {
                if(V[tmp].nxt[c] == -1) {
                    root = V.size();
                    V[tmp].nxt[c] = root;
                    V.push_back(Node());
                    V.back().len = V[tmp].len+2;
                    tmp = V[tmp].link;
                    iter = S.rbegin()+V[tmp].len+1;
                    while(iter >= S.rend() || *iter != c) {
                        tmp = V[tmp].link;
                        iter = S.rbegin()+V[tmp].len+1;
                    }
                    tmp = V[tmp].nxt[c];
                    if(V.back().len == 1 || tmp <= 0) V.back().
                        link = 1;
                    else V.back().link = tmp;
                } else root = V[tmp].nxt[c];
                V[root].cnt++;
                break;
            }
        }
    }
};
```

Graph (2)

2.1 Fundamentals

Bridge.h

Description: Undirected connected graph, no self-loop. Find every bridges. Usual graph representation. dfs(here, par): returns fastest vertex which connected by some node in subtree of here, except here-parent edge.
Time: $\mathcal{O}(V+E)$, 180ms for $V=10^5$ and $E=10^6$ graph.

d41d8c, 23 lines

```
const int MAX_N = 1e5 + 1;

vector<int> adj[MAX_N];
vector<pii> bridges;
int in[MAX_N];
int cnt = 0;

int dfs(int here, int parent = -1) {
    in[here] = cnt++;
    int ret = 1e9;
    for (int there: adj[here]) {
        if (there != parent) {
            if (in[there] == -1) {
                int subret = dfs(there, here);
                if (subret > in[here]) bridges.push_back({here,
                    there});
                ret = min(ret, subret);
            } else {
                ret = min(ret, in[there]);
            }
        }
    }
    return ret;
}
```

KthShortestPath.h

Description: Calculate Kth shortest path from s to t.
Usage: `0-base index. Vertex is 0 to n-1. KthShortestPath g(n); g.add_edge(s, e, cost); g.run(s, t, k);`
Time: $\mathcal{O}(E\log V + K\log K)$, $V=E=K=3\times 10^5$ in 312ms, 144MB at yosupo.

d41d8c, 75 lines

```
struct KthShortestPath {
    struct node{
        array<node*, 2> son; pair<ll, ll> val;
        node() : node(make_pair(-1e18, -1e18)) {}
        node(pair<ll, ll> val) : node(nullptr, nullptr, val) {}
        node(node *l, node *r, pair<ll, ll> val) : son({l,r}),
            val(val) {}
    };
    node* copy(node *x){ return x ? new node(x->son[0], x->son
        [1], x->val) : nullptr; }
    node* merge(node *x, node *y){ // precondition: x, y both
        points to new entity
        if(!x || !y) return x ? x : y;
        if(x->val > y->val) swap(x, y);
        int rd = rnd(0, 1);
        if(x->son[rd]) x->son[rd] = copy(x->son[rd]);
        x->son[rd] = merge(x->son[rd], y); return x;
    }

    struct edge{
        ll v, c, i; edge() = default;
        edge(ll v, ll c, ll i) : v(v), c(c), i(i) {}
    };

    vector<vector<edge>> gph, rev;
    int idx;
    vector<int> par, pae; vector<ll> dist; vector<node*> heap;

    KthShortestPath(int n) {
        gph = rev = vector<vector<edge>>(n);
        idx = 0;
    }

    void add_edge(int s, int e, ll x){
        gph[s].emplace_back(e, x, idx);
        rev[e].emplace_back(s, x, idx);
        assert(x >= 0); idx++;
    }

    void dijkstra(int snk){ // replace this to SPFA if edge
        weight is negative
        int n = gph.size();
        par = pae = vector<int>(n, -1);
        dist = vector<ll>(n, 0x3f3f3f3f3f3f3f3f);
        heap = vector<node*>(n, nullptr);
        priority_queue<pair<ll,ll>, vector<pair<ll,ll>>,
            greater<>> pq;
        auto enqueue = [&](int v, ll c, int pa, int pe){
            if(dist[v] > c) dist[v] = c, par[v] = pa, pae[v] =
                pe, pq.emplace(c, v);
        }; enqueue(snk, 0, -1, -1); vector<int> ord;
        while(!pq.empty()){
            auto [c,v] = pq.top(); pq.pop(); if(dist[v] != c)
                continue;
            ord.push_back(v); for(auto e : rev[v]) enqueue(e.v,
                c+e.c, v, e.i);
        }
        for(auto &v : ord){
            if(par[v] != -1) heap[v] = copy(heap[par[v]]);
```

```
        for(auto &e : gph[v]){
            if(e.i == pae[v]) continue;
            ll delay = dist[e.v] + e.c - dist[v];
            if(delay < 1e18) heap[v] = merge(heap[v], new
                node(make_pair(delay, e.v)));
        }
    }
}
vector<ll> run(int s, int e, int k){
    using state = pair<ll, node*>; dijkstra(e); vector<ll>
        ans;
    priority_queue<state, vector<state>, greater<state>> pq
        ;
    if(dist[s] > 1e18) return vector<ll>(k, -1);
    ans.push_back(dist[s]);
    if(heap[s]) pq.emplace(dist[s] + heap[s]->val.first,
        heap[s]);
    while(!pq.empty() && ans.size() < k){
        auto [cst, ptr] = pq.top(); pq.pop(); ans.push_back
            (cst);
        for(int j=0; j<2; j++) if(ptr->son[j])
                                pq.emplace(cst-ptr->val.
                                    first + ptr->son[j
                                    ]->val.first, ptr->
                                    son[j]);
        int v = ptr->val.second;
        if(heap[v]) pq.emplace(cst + heap[v]->val.first,
            heap[v]);
    }
    while(ans.size() < k) ans.push_back(-1);
    return ans;
}
};
```

TreeIsomorphism.h

Description: Calculate hash of given tree.
Usage: 1-base index. t.init(n); t.add_edge(a, b); (size, hash) = t.build(void); // size may contain dummy centroid.
Time: $\mathcal{O}(N\log N)$, $N=30$ and $\sum N\le 10^6$ in 256ms.

d41d8c, 74 lines

```
const int MAX_N = 33;
ull A[MAX_N], B[MAX_N];

struct Tree {
    int n;
    vector<int> adj[MAX_N];
    int sz[MAX_N];
    vector<int> cent; // sz(cent) <= 2
    Tree() {}

    void init(int n) {
        this->n = n;
        for (int i=0; i<n+2; ++i) adj[i].clear();
        fill(sz, sz+n+2, 0);
        cent.clear();
    }

    void add_edge(int s, int e) {
        adj[s].push_back(e);
        adj[e].push_back(s);
    }

    int get_cent(int v, int b = -1) {
        sz[v] = 1;
        for (auto i: adj[v]) {
            if (i != b) {
                int now = get_cent(i, v);
                if (now <= n/2) sz[v] += now;
                else break;
            }
        }
        if (n - sz[v] <= n/2) cent.push_back(v);
        return sz[v];
    }

    int init() {
        get_cent(1);
        if (cent.size() == 1) return cent[0];
        int u = cent[0], v = cent[1], add = ++n;
        adj[u].erase(find(adj[u].begin(), adj[u].end(), v));
        adj[v].erase(find(adj[v].begin(), adj[v].end(), u));
        adj[add].push_back(u); adj[u].push_back(add);
        adj[add].push_back(v); adj[v].push_back(add);
        return add;
    }

    pair<int, ull> build(int v, int p = -1, int d = 1) {
        vector<pair<int, ull>> ch;
        for (auto i: adj[v]) {
            if (i != p) ch.push_back(build(i, v, d+1));
        }
        if (ch.empty()) return { 1, d };

        sort(ch.begin(), ch.end());
        ull ret = d;
        int tmp = 1;
        for (int j=0; j<ch.size(); ++j) {
            ret += A[d] ^ B[j] ^ ch[j].second;
            tmp += ch[j].first;
        }
        return { tmp, ret };
    }

    pair<int, ull> build() {
        return build(init());
    }
};

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());
uniform_int_distribution<ull> urnd;

void solve() {
    for (int i=0; i<MAX_N; ++i) A[i] = urnd(rng), B[i] = urnd(
        rng);
}
```

2.2 Network flow

MinCostMaxFlow.h

Description: Set MAXN. Overflow is not checked.
Usage: MCMF g; g.add_edge(s, e, cap, cost); g.solve(src, sink, total_size);
Time: 216ms on almost K_n graph, for $n=300$.

d41d8c, 91 lines

```
// https://github.com/koosaga/olympiad/blob/master/Library/
    codes/combinatorial_optimization/flow_cost_dijkstra.cpp
const int MAXN = 800 + 5;

struct MCMF {
    struct Edge{ int pos, cap, rev; ll cost; };
    vector<Edge> gph[MAXN];
    void clear(){
        for(int i=0; i<MAXN; i++) gph[i].clear();
    }
    void add_edge(int s, int e, int x, ll c){
        gph[s].push_back({e, x, (int)gph[e].size(), c});
        gph[e].push_back({s, 0, (int)gph[s].size()-1, -c});
    }
    ll dist[MAXN];
    int pa[MAXN], pe[MAXN];
    bool inque[MAXN];
    bool spfa(int src, int sink, int n){
        memset(dist, 0x3f, sizeof(dist[0]) * n);
        memset(inque, 0, sizeof(inque[0]) * n);
        queue<int> que;
        dist[src] = 0;
        inque[src] = 1;
        que.push(src);
        bool ok = 0;
        while(!que.empty()){
            int x = que.front();
            que.pop();
            if(x == sink) ok = 1;
            inque[x] = 0;
            for(int i=0; i<gph[x].size(); i++){
                Edge e = gph[x][i];
                if(e.cap > 0 && dist[e.pos] > dist[x] + e.cost)
                    {
                    dist[e.pos] = dist[x] + e.cost;
                    pa[e.pos] = x;
                    pe[e.pos] = i;
                    if(!inque[e.pos]){
                        inque[e.pos] = 1;
                        que.push(e.pos);
                    }
                }
            }
        }
        return ok;
    }
    ll new_dist[MAXN];
    pair<bool, ll> dijkstra(int src, int sink, int n){
        priority_queue<pii, vector<pii>, greater<pii> > pq;
        memset(new_dist, 0x3f, sizeof(new_dist[0]) * n);
        new_dist[src] = 0;
        pq.emplace(0, src);
        bool isSink = 0;
        while(!pq.empty()) {
            auto tp = pq.top(); pq.pop();
            if(new_dist[tp.second] != tp.first) continue;
            int v = tp.second;
            if(v == sink) isSink = 1;
            for(int i = 0; i < gph[v].size(); i++){
                Edge e = gph[v][i];
                ll new_weight = e.cost + dist[v] - dist[e.pos];
                if(e.cap > 0 && new_dist[e.pos] > new_dist[v] +
                    new_weight){
                    new_dist[e.pos] = new_dist[v] + new_weight;
                    pa[e.pos] = v;
                    pe[e.pos] = i;
                    pq.emplace(new_dist[e.pos], e.pos);
                }
            }
        }
        return make_pair(isSink, new_dist[sink]);
    }

    pair<ll, ll> solve(int src, int sink, int n){
        spfa(src, sink, n);
        pair<bool, ll> path;
        pair<ll,ll> ret = {0,0};
        while((path = dijkstra(src, sink, n)).first){
            for(int i = 0; i < n; i++) dist[i] += min(ll(2e15),
                new_dist[i]);
            ll cap = 1e18;
            for(int pos = sink; pos != src; pos = pa[pos]){
                cap = min(cap, (ll)gph[pa[pos]][pe[pos]].cap);
```

```
            }
            ret.first += cap;
            ret.second += cap * (dist[sink] - dist[src]);
            for(int pos = sink; pos != src; pos = pa[pos]){
                int rev = gph[pa[pos]][pe[pos]].rev;
                gph[pa[pos]][pe[pos]].cap -= cap;
                gph[pos][rev].cap += cap;
            }
        }
        return ret;
    }
};
```

Dinic.h

Description: 0-indexed. cf) $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.
Usage: `Dinic g(n); g.add_edge(u, v, cap_uv, cap_vu); g.max_flow(s, t); g.clear_flow();`

d41d8c, 79 lines

```
struct Dinic {
    struct Edge {
        int a;
        ll flow;
        ll cap;
        int rev;
    };

    int n, s, t;
    vector<vector<Edge>> adj;
    vector<int> level;
    vector<int> cache;
    vector<int> q;
    Dinic(int _n) : n(_n) {
        adj.resize(n);
        level.resize(n);
        cache.resize(n);
        q.resize(n);
    }

    bool bfs() {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        int l = 0, r = 1;
        q[0] = s;
        while (l < r) {
            int here = q[l++];
            for (auto[there, flow, cap, rev]: adj[here]) {
                if (flow < cap && level[there] == -1) {
                    level[there] = level[here] + 1;
                    if (there == t) return true;
                    q[r++] = there;
                }
            }
        }
        return false;
    }

    ll dfs(int here, ll extra_capa) {
        if (here == t) return extra_capa;
        for (int& i=cache[here]; i<adj[here].size(); ++i) {
            auto[there, flow, cap, rev] = adj[here][i];
            if (flow < cap && level[there] == level[here] + 1) {
                ll f = dfs(there, min(extra_capa, cap-flow));
                if (f > 0) {
                    adj[here][i].flow += f;
                    adj[there][rev].flow -= f;
                    return f;
                }
            }
        }
        return 0;
    }

    void clear_flow() {
        for (auto& v: adj) {
            for (auto& e: v) e.flow = 0;
        }
    }

    ll max_flow(int _s, int _t) {
        s = _s, t = _t;
        ll ret = 0;
        while (bfs()) {
            fill(cache.begin(), cache.end(), 0);
            while (true) {
                ll f = dfs(s, 2e18);
                if (f == 0) break;
                ret += f;
            }
        }
        return ret;
    }

    void add_edge(int u, int v, ll uv, ll vu) {
        adj[u].push_back({ v, 0, uv, (int)adj[v].size() });
        adj[v].push_back({ u, 0, vu, (int)adj[u].size()-1 });
    }
};
```

Hungarian.h

Description: Bipartite minimum weight matching. 1-base indexed. A[1..n][1..m] and $n \le m$ needed. pair(cost, matching) will be returned.
Usage: `auto ret = hungarian(A);`
Time: $\mathcal{O}\left(n^2 m\right)$, and 100ms for n = 500.

d41d8c, 41 lines

```
const ll INF = 1e18;

pair<ll, vector<int>> hungarian(const vector<vector<ll>>& A) {
    int n = (int)A.size()-1;
    int m = (int)A[0].size()-1;
    vector<ll> u(n+1), v(m+1), p(m+1), way(m+1);
    for (int i=1; i<=n; ++i) {
        p[0] = i;
        int j0 = 0;
        vector<ll> minv (m+1, INF);
        vector<char> used (m+1, false);
        do {
            used[j0] = true;
            int i0 = p[j0], j1;
            ll delta = INF;
            for (int j=1; j<=m; ++j) {
                if (!used[j]) {
                    ll cur = A[i0][j]-u[i0]-v[j];
                    if (cur < minv[j])
                        minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta)
                        delta = minv[j], j1 = j;
                }
            }
            for (int j=0; j<=m; ++j)
                if (used[j])
                    u[p[j]] += delta, v[j] -= delta;
                else
                    minv[j] -= delta;
            j0 = j1;
        } while (p[j0] != 0);
        do {
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }
    vector<int> match(n+1);
    for (int i=1; i<=m; ++i) match[p[i]] = i;
    return { -v[0], match };
}
```

GlobalMinCut.h

Description: Undirected graph with adj matrix. No edge means $adj[i][j] = 0$. 0-based index, and expect $N \times N$ adj matrix.
Time: $\mathcal{O}\left(V^3\right)$, $\sum V^3 = 5.5 \times 10^8$ in 640ms.

d41d8c, 24 lines

```
const int INF = 1e9;
int getMinCut(vector<vector<int>> &adj) {
    int n = adj.size();
    vector<int> used(n);
    int ret = INF;
    for (int ph=n-1; ph>=0; --ph) {
        vector<int> w = adj[0], added = used;
        int prev, k = 0;
        for (int i=0; i<ph; ++i) {
            prev = k;
            k = -1;
            for (int j = 1; j < n; j++) {
                if (!added[j] && (k == -1 || w[j] > w[k])) k = j;
            }
            if (i+1 == ph) break;
            for (int j = 0; j < n; j++) w[j] += adj[k][j];
            added[k] = 1;
        }
        for (int i=0; i<n; ++i) adj[i][prev] = (adj[prev][i] += adj[k][i]);
        used[k] = 1;
        ret = min(ret, w[k]);
    }
    return ret;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Usage: `0-base index. GomoryHuTree t; auto ret = t.solve(n, edges); 0 is root, ret[i] for` $i > 0$ `contains (cost, par)`
Time: $\mathcal{O}(V)$ Flow Computations, $V = 3000, E = 4500$ and special graph that flow always terminate in $\mathcal{O}(3(V+E))$ time in 4036ms.

d41d8c, 33 lines

```
struct Edge {
    int s, e, x;
};

const int MAX_N = 500 + 1;

bool vis[MAX_N];

struct GomoryHuTree {
    vector<pii> solve(int n, const vector<Edge>& edges) { // i - j cut : i - j minimum edge cost. 0 based.
        vector<pii> ret(n); // if i > 0, stores pair(cost, parent)
        for(int i=1; i<n; i++){
            Dinic g(n);
            for (auto[s, e, x]: edges) g.add_edge(s, e, x, x);
            ret[i].first = g.max_flow(i, ret[i].second);
```

```
            memset(vis, 0, sizeof(vis));
            function<void(int)> dfs = [&](int x) {
                if (vis[x]) return;
                vis[x] = 1;
                for (auto& i: g.adj[x]) {
                    if (i.cap - i.flow > 0) dfs(i.a);
                }
            };

            dfs(i);
            for (int j=i+1; j<n; j++) {
                if (ret[j].second == ret[i].second && vis[j])
                    ret[j].second = i;
            }
        }
        return ret;
    }
};
```

2.3 Matching

2.3.1 Random notes on matching (and bipartite)

In general graph, complement of independent set is vertex cover, and reverse holds too.
In bipartite graph, cardinality of minimum vertex cover is equal to card of maximum matching (konig).
In poset (DAG), card of maximum anti chain is equal to minimum path cover (dilworth).
Poset is DAG which satisfy i-¿j and j-¿k edge means i-¿k (transitivity).

hopcroftKarp.h

Description: It contains several application of bipartite matching.
Usage: Both left and right side of node number starts with 0. HopcraftKarp(n, m); g.add_edge(s, e);
Time: $\mathcal{O}\left(E\sqrt{V}\right)$, min path cover $V = 10^4, E = 10^5$ in 20ms. d41d8c, 92 lines

```
struct HopcroftKarp{
    int n, m;
    vector<vector<int>> g;
    vector<int> dst, le, ri;
    vector<char> visit, track;
    HopcroftKarp(int n, int m) : n(n), m(m), g(n), dst(n), le(n
        , -1), ri(m, -1), visit(n), track(n+m) {}

    void add_edge(int s, int e){ g[s].push_back(e); }
    bool bfs(){
        bool res = false; queue<int> que;
        fill(dst.begin(), dst.end(), 0);
        for(int i=0; i<n; i++)if(le[i] == -1)que.push(i),dst[i
            ]=1;
        while(!que.empty()){
            int v = que.front(); que.pop();
            for(auto i : g[v]){
                if(ri[i] == -1) res = true;
                else if(!dst[ri[i]])dst[ri[i]]=dst[v]+1,que.
                    push(ri[i]);
            }
        }
        return res;
    }
    bool dfs(int v){
        if(visit[v]) return false; visit[v] = 1;
        for(auto i : g[v]){
            if(ri[i] == -1 || !visit[ri[i]] && dst[ri[i]] ==
                dst[v] + 1 && dfs(ri[i])){
                le[v] = i; ri[i] = v; return true;
            }
        }
        return false;
    }
    int maximum_matching(){
        int res = 0; fill(le.begin(), le.end(), -1); fill(ri.
            begin(), ri.end(), -1);
        while(bfs()){
            fill(visit.begin(), visit.end(), 0);
            for(int i=0; i<n; i++) if(le[i] == -1) res += dfs(i
                );
        }
        return res;
    }
    vector<pair<int,int>> maximum_matching_edges(){
        int matching = maximum_matching();
        vector<pair<int,int>> edges; edges.reserve(matching);
        for(int i=0; i<n; i++) if(le[i] != -1) edges.
            emplace_back(i, le[i]);
        return edges;
    }
    void dfs_track(int v){
        if(track[v]) return; track[v] = 1;
        for(auto i : g[v]) track[n+i] = 1, dfs_track(ri[i]);
    }
    tuple<vector<int>, vector<int>, int> minimum_vertex_cover()
        {
        int matching = maximum_matching(); vector<int> lv, rv;
        fill(track.begin(), track.end(), 0);
        for(int i=0; i<n; i++) if(le[i] == -1) dfs_track(i);
        for(int i=0; i<n; i++) if(!track[i]) lv.push_back(i);
        for(int i=0; i<m; i++) if(track[n+i]) rv.push_back(i);
        return {lv, rv, lv.size() + rv.size()}; // s(lv)+s(rv)=
            mat
    }
    tuple<vector<int>, vector<int>, int>
        maximum_independent_set(){
        auto [a,b,matching] = minimum_vertex_cover();
        vector<int> lv, rv; lv.reserve(n-a.size()); rv.reserve(
            m-b.size());
        for(int i=0, j=0; i<n; i++){
            while(j < a.size() && a[j] < i) j++;
            if(j == a.size() || a[j] != i) lv.push_back(i);
        }
        for(int i=0, j=0; i<m; i++){
            while(j < b.size() && b[j] < i) j++;
            if(j == b.size() || b[j] != i) rv.push_back(i);
        } // s(lv)+s(rv)=n+m-mat
        return {lv, rv, lv.size() + rv.size()};
    }
    vector<vector<int>> minimum_path_cover(){ // n == m
        int matching = maximum_matching();
        vector<vector<int>> res; res.reserve(n - matching);
        fill(track.begin(), track.end(), 0);
        auto get_path = [&](int v) -> vector<int> {
            vector<int> path{v}; // ri[v] == -1
            while(le[v] != -1) path.push_back(v=le[v]);
            return path;
        };
        for(int i=0; i<n; i++) if(!track[n+i] && ri[i] == -1)
            res.push_back(get_path(i));
        return res; // sz(res) = n-mat
    }
    vector<int> maximum_anti_chain(){ // n = m
        auto [a,b,matching] = minimum_vertex_cover();
        vector<int> res; res.reserve(n - a.size() - b.size());
        for(int i=0, j=0, k=0; i<n; i++){
            while(j < a.size() && a[j] < i) j++;
            while(k < b.size() && b[k] < i) k++;
            if((j == a.size() || a[j] != i) && (k == b.size()
                || b[k] != i)) res.push_back(i);
        }
        return res; // sz(res) = n-mat
    }
};
```

GeneralMatching.h

Description: Matching for general graphs.
Usage: 1-base index. match[] has real matching (maybe). GeneralMatching g(n); g.add_edge(a, b); int ret = g.run(void);
Time: $\mathcal{O}\left(N^3\right)$, $N = 500$ in 20ms. d41d8c, 93 lines

```
const int MAX_N = 500 + 1;

struct GeneralMatching {
    int n, cnt;
    int match[MAX_N], par[MAX_N], chk[MAX_N], prv[MAX_N], vis[
        MAX_N];
    vector<int> g[MAX_N];
    GeneralMatching(int n): n(n) {
        // init
        cnt = 0;
        for (int i=0; i<=n; ++i) g[i].clear();
        memset(match, 0, sizeof match);
        memset(vis, 0, sizeof vis);
        memset(prv, 0, sizeof prv);
    }

    int find(int x) { return x == par[x] ? x : par[x] = find(
        par[x]); }

    int lca(int u, int v) {
        for (cnt++; vis[u] != cnt; swap(u, v)) {
            if (u) vis[u] = cnt, u = find(prv[match[u]]);
        }
        return u;
    }

    void add_edge(int u, int v) {
        g[u].push_back(v);
        g[v].push_back(u);
    }

    void blossom(int u, int v, int rt, queue<int> &q) {
        for (; find(u) != rt; u = prv[v]) {
            prv[u] = v;
            par[u] = par[v = match[u]] = rt;
            if (chk[v] & 1) q.push(v), chk[v] = 2;
        }
    }

    bool augment(int u) {
        iota(par, par + MAX_N, 0);
        memset(chk, 0, sizeof chk);
        queue<int> q;
        q.push(u);
        chk[u] = 2;

        while (!q.empty()) {
            u = q.front();
            q.pop();
            for (auto v : g[u]) {
                if (chk[v] == 0) {
                    prv[v] = u;
                    chk[v] = 1;
```

```
                q.push(match[v]);
                chk[match[v]] = 2;
                if (!match[v]) {
                    for (; u; v = u) {
                        u = match[prv[v]];
                        match[match[v] = prv[v]] = v;
                    }
                    return true;
                }
            } else if (chk[v] == 2) {
                int l = lca(u, v);
                blossom(u, v, l, q);
                blossom(v, u, l, q);
            }
        }
    }
    return false;
}

int run() {
    int ret = 0;
    vector<int> tmp(n-1); // not necessary, just for
        constant optimization
    iota(tmp.begin(), tmp.end(), 0);
    shuffle(tmp.begin(), tmp.end(), mt19937(0x1557));
    for (auto x: tmp) {
        if (!match[x]) {
            for (auto y: g[x]) {
                if (!match[y]) {
                    match[x] = y;
                    match[y] = x;
                    ret++;
                    break;
                }
            }
        }
    }
    for (int i=1; i<=n; i++) {
        if (!match[i]) ret += augment(i);
    }
    return ret;
}
};
```

GeneralWeightedMatching.h

Description: Given a weighted undirected graph, return maximum matching.

Usage: 1-base index. init(n); add_edge(a, b, w); (tot_weight, n_matches) = _solve(void); Note that get_lca function have a static variable.

Time: $\mathcal{O}(N^3)$, $N = 500$ in 317ms at yosupo.

d41d8c, 228 lines

```
static const int INF = INT_MAX;
static const int N = 500 + 1;

struct Edge {
    int u, v, w;
    Edge() {}
    Edge(int ui, int vi, int wi) : u(ui), v(vi), w(wi) {}
};

int n, n_x;
Edge g[N * 2][N * 2];
int lab[N * 2];
int match[N * 2], slack[N * 2], st[N * 2], pa[N * 2];
int flo_from[N * 2][N + 1], s[N * 2], vis[N * 2];
vector<int> flo[N * 2];
queue<int> q;

int e_delta(const Edge &e) {
    return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
}

void update_slack(int u, int x) {
    if (!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x])
        ) slack[x] = u;
}

void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u) {
        if (g[u][x].w > 0 && st[u] != x && s[st[u]] == 0)
            update_slack(u, x);
    }
}

void q_push(int x) {
    if (x <= n) {
        q.push(x);
    } else {
        for (size_t i = 0; i < flo[x].size(); i++) q_push(flo[x
            ][i]);
    }
}

void set_st(int x, int b) {
    st[x] = b;
    if (x > n) {
        for (size_t i = 0; i < flo[x].size(); ++i) set_st(flo[x
            ][i], b);
    }
}

int get_pr(int b, int xr) {
    int pr = find(flo[b].begin(), flo[b].end(), xr) - flo[b].
        begin();
    if (pr % 2 == 1) {
        reverse(flo[b].begin() + 1, flo[b].end());
        return (int)flo[b].size() - pr;
    } else {
        return pr;
    }
}

void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    Edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i < pr; ++i) set_match(flo[u][i], flo[u][i
        ^ 1]);
    set_match(xr, v);
    rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].end());
}

void augment(int u, int v) {
    for (;;) {
        int xnv = st[match[u]];
        set_match(u, v);
        if (!xnv) return;
        set_match(xnv, st[pa[xnv]]);
        u = st[pa[xnv]], v = xnv;
    }
}

int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
        if (u == 0) continue;
        if (vis[u] == t) return u;
        vis[u] = t;
        u = st[match[u]];
        if (u) u = st[pa[u]];
    }
    return 0;
}

void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, s[b] = 0;
    match[b] = match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]]) {
        flo[b].push_back(x), flo[b].push_back(y = st[match[x]])
            , q_push(y);
    }
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]]) {
        flo[b].push_back(x), flo[b].push_back(y = st[match[x]])
            , q_push(y);
    }
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); ++i) {
        int xs = flo[b][i];
        for (int x = 1; x <= n_x; ++x)
            if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g
                [b][x])) {
                g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            }
        for (int x = 1; x <= n; ++x)
            if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
}

void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i) set_st(flo[b][i
        ], flo[b][i]);
    int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
        int xs = flo[b][i], xns = flo[b][i + 1];
        pa[xs] = g[xns][xs].u;
        s[xs] = 1, s[xns] = 0;
        slack[xs] = 0, set_slack(xns);
        q_push(xns);
    }
    s[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
        int xs = flo[b][i];
        s[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
}

bool on_found_edge(const Edge &e) {
    int u = st[e.u], v = st[e.v];
    if (s[v] == -1) {
        pa[v] = e.u, s[v] = 1;
        int nu = st[match[v]];
        slack[v] = slack[nu] = 0;
        s[nu] = 0, q_push(nu);
    } else if (s[v] == 0) {
```

```
        int lca = get_lca(u, v);
        if (!lca) return augment(u, v), augment(v, u), true;
        else add_blossom(u, lca, v);
    }
    return false;
}

bool matching() {
    memset(s + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && !match[x]) pa[x] = 0, s[x] = 0,
             q_push(x);
    if (q.empty()) return false;
    for (;;) {
        while (q.size()) {
            int u = q.front(); q.pop();
            if (s[st[u]] == 1) continue;
            for (int v = 1; v <= n; ++v)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (e_delta(g[u][v]) == 0) {
                        if (on_found_edge(g[u][v])) return true
                            ;
                    } else update_slack(u, st[v]);
                }
        }
        int d = INF;
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && s[b] == 1) d = min(d, lab[b] / 2)
                ;
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x]) {
                if (s[x] == -1) d = min(d, e_delta(g[slack[x]][
                    x]));
                else if (s[x] == 0) d = min(d, e_delta(g[slack[
                    x]][x]) / 2);
            }
        for (int u = 1; u <= n; ++u) {
            if (s[st[u]] == 0) {
                if (lab[u] <= d) return 0;
                lab[u] -= d;
            } else if (s[st[u]] == 1) lab[u] += d;
        }
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b) {
                if (s[st[b]] == 0) lab[b] += d * 2;
                else if (s[st[b]] == 1) lab[b] -= d * 2;
            }
        q = queue<int>();
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x] && st[slack[x]] != x &&
                 e_delta(g[slack[x]][x]) == 0)
                if (on_found_edge(g[slack[x]][x])) return true;
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && s[b] == 1 && lab[b] == 0)
                 expand_blossom(b);
    }
    return false;
}

pair<long long, int> _solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v) {
            flo_from[u][v] = (u == v ? u : 0);
            w_max = max(w_max, g[u][v].w);
        }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
        if (match[u] && match[u] < u) tot_weight += g[u][match[
            u]].w;
    return make_pair(tot_weight, n_matches);
}

void add_edge(int ui, int vi, int wi) {
    g[ui][vi].w = g[vi][ui].w = wi;
}

void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v) g[u][v] = Edge(u, v, 0);
    }
}
```

2.4 DFS algorithms

2sat.h

Description: Every variable x is encoded to 2i, $!x$ is 2i+1. n of TwoSAT means number of variables.
Usage: `TwoSat g(number of vars);`
`g.addCNF(x, y); // x or y`
`g.atMostOne({ a, b, ... });`
`auto ret = g.solve(void); if impossible empty`
Time: $\mathcal{O}(V+E)$, note that sort in atMostOne function. 10^5 simple cnf clauses 56ms.

d41d8c, 94 lines

```
struct TwoSAT {
    struct SCC {
        int n;
        vector<bool> chk;
        vector<vector<int>> E, F;
        SCC() {}

        void dfs(int x, vector<vector<int>> &E, vector<int> &st
            ) {
            if(chk[x]) return;
            chk[x] = true;
            for(auto i : E[x]) dfs(i, E, st);
            st.push_back(x);
        }

        void init(vector<vector<int>> &E) {
            n = E.size();
            this->E = E;
            F.resize(n);
            chk.resize(n, false);
            for(int i = 0; i < n; i++)
                for(auto j : E[i]) F[j].push_back(i);
        }

        vector<vector<int>> getSCC() {
            vector<int> st;
            fill(chk.begin(), chk.end(), false);
            for(int i = 0; i < n; i++) dfs(i, E, st);
            reverse(st.begin(), st.end());
            fill(chk.begin(), chk.end(), false);
            vector<vector<int>> scc;
            for(int i = 0; i < n; i++) {
                if(chk[st[i]]) continue;
                vector<int> T;
                dfs(st[i], F, T);
                scc.push_back(T);
            }
            return scc;
        }
    };

    int n;
    vector<vector<int>> adj;
    TwoSAT(int n): n(n) {
        adj.resize(2*n);
    }

    int new_node() {
        adj.push_back(vector<int>());
        adj.push_back(vector<int>());
        return n++;
    }

    void add_edge(int a, int b) {
        adj[a].push_back(b);
    }

    void add_cnf(int a, int b) {
        add_edge(a^1, b);
        add_edge(b^1, a);
    }

    // arr elements need to be unique
    // Add n dummy variable, 3n-2 edges
    // yi = x1 | x2 | ... | xi, xi->yi, yi->y(i+1), yi->!x(i+1)
    void at_most_one(vector<int> arr) {
        sort(arr.begin(), arr.end());
        assert(unique(arr.begin(), arr.end()) == arr.end());
        for (int i=0; i<arr.size(); ++i) {
            int now = new_node();
            add_cnf(arr[i]^1, 2*now);
            if (i == 0) continue;
            add_cnf(2*(now-1)+1, 2*now);
            add_cnf(2*(now-1)+1, arr[i]^1);
        }
    }

    vector<int> solve() {
        SCC g;
        g.init(adj);
        auto scc = g.getSCC();

        vector<int> rev(2*n, -1);
        for (int i=0; i<scc.size(); ++i) {
            for (int x: scc[i]) rev[x] = i;
        }
        for (int i=0; i<n; ++i) {
            if (rev[2*i] == rev[2*i+1]) return vector<int>();
        }

        vector<int> ret(n);
        for (int i=0; i<n; ++i) ret[i] = (rev[2*i] > rev[2*i
            +1]);
        return ret;
    }
};
```

2.5 Coloring

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color.

Usage: 1-base index. Vizing g; g.clear(V); g.solve(edges, V); answer saved in G.
Time: $\mathcal{O}(VE)$, $\sum VE = 1.1 \times 10^6$ in 24ms.

d41d8c, 60 lines

```
const int MAX_N = 444 + 1;
struct Vizing { // returns edge coloring in adjacent matrix G. 1 - based
    int C[MAX_N][MAX_N], G[MAX_N][MAX_N];

    void clear(int n){
        for (int i=0; i<=n; i++){
            for (int j=0; j<=n; j++) C[i][j] = G[i][j] = 0;
        }
    }

    void solve(vector<pii> &E, int n){
        int X[MAX_N] = {}, a;

        auto update = [&](int u) {
            for (X[u] = 1; C[u][X[u]]; X[u]++);
        };

        auto color = [&](int u, int v, int c) {
            int p = G[u][v];
            G[u][v] = G[v][u] = c;
            C[u][c] = v;
            C[v][c] = u;
            C[u][p] = C[v][p] = 0;
            if (p) X[u] = X[v] = p;
            else update(u), update(v);
            return p;
        };

        auto flip = [&](int u, int c1, int c2){
            int p = C[u][c1]; swap(C[u][c1], C[u][c2]);
            if (p) G[u][p] = G[p][u] = c2;
            if (!C[u][c1]) X[u] = c1;
            if (!C[u][c2]) X[u] = c2;
            return p;
        };

        for (int i=1; i <= n; i++) X[i] = 1;
        for (int t=0; t<E.size(); ++t) {
            auto[u, v0] = E[t];
            int v = v0, c0 = X[u], c=c0, d;
            vector<pii> L;
            int vst[MAX_N] = {};
            while (!G[u][v0]) {
                L.emplace_back(v, d = X[v]);
                if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c = color(u, L[a].first, c);
                else if (!C[u][d]) for(a=(int)L.size()-1;a>=0;a--) color(u,L[a].first,L[a].second);
                else if (vst[d]) break;
                else vst[d] = 1, v = C[u][d];
            }

            if(!G[u][v0]) {
                for (; v; v = flip(v, c, d), swap(c, d));
                if(C[u][c0]){
                    for(a = (int)L.size()-2; a >= 0 && L[a].second != c; a--);
                    for(; a >= 0; a--) color(u, L[a].first, L[a].second);
                } else t--;
            }
        }
    }
};
```

2.6 Heuristics

2.7 Trees

DirectedMST.h

Description: Directed MST for given root node. If no MST exists, returns -1.
Usage: 0-base index. Vertex is 0 to n-1. typedef ll cost_t.
Time: $\mathcal{O}(E \log V)$, $V = E = 2 \times 10^5$ in 90ms at yosupo.

d41d8c, 87 lines

```
struct Edge{
    int s, e; cost_t x;
    Edge() = default;
    Edge(int s, int e, cost_t x) : s(s), e(e), x(x) {}
    bool operator < (const Edge &t) const { return x < t.x; }
};
struct UnionFind{
    vector<int> P, S;
    vector<pair<int, int>> stk;
    UnionFind(int n) : P(n), S(n, 1) { iota(P.begin(), P.end(), 0); }
    int find(int v) const { return v == P[v] ? v : find(P[v]); }
    int time() const { return stk.size(); }
    void rollback(int t){
        while(stk.size() > t){
            auto [u,v] = stk.back(); stk.pop_back();
            P[u] = u; S[v] -= S[u];
        }
    }
    bool merge(int u, int v){
        u = find(u); v = find(v);
        if(u == v) return false;
        if(S[u] > S[v]) swap(u, v);
        stk.emplace_back(u, v);
        S[v] += S[u]; P[u] = v;
        return true;
    }
};
struct Node{
    Edge key;
    Node *l, *r;
    cost_t lz;
    Node() : Node(Edge()) {}
    Node(const Edge &edge) : key(edge), l(nullptr), r(nullptr), lz(0) {}
    void push(){
        key.x += lz;
        if(l) l->lz += lz;
        if(r) r->lz += lz;
        lz = 0;
    }
    Edge top(){ push(); return key; }
};
Node* merge(Node *a, Node *b){
    if(!a || !b) return a ? a : b;
    a->push(); b->push();
    if(b->key < a->key) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node* &a){ a->push(); a = merge(a->l, a->r); }

// 0-based
pair<cost_t, vector<int>> DirectMST(int n, int rt, vector<Edge> &edges){
    vector<Node*> heap(n);
    UnionFind uf(n);
    for(const auto &i : edges) heap[i.e] = merge(heap[i.e], new Node(i));
    cost_t res = 0;
    vector<int> seen(n, -1), path(n), par(n);
    seen[rt] = rt;
    vector<Edge> Q(n), in(n, {-1,-1, 0}), comp;
    deque<tuple<int, int, vector<Edge>>> cyc;
    for(int s=0; s<n; s++){
        int u = s, qi = 0, w;
        while(seen[u] < 0){
            if(!heap[u]) return {-1, {}};
            Edge e = heap[u]->top();
            heap[u]->lz -= e.x; pop(heap[u]);
            Q[qi] = e; path[qi++] = u; seen[u] = s;
            res += e.x; u = uf.find(e.s);
            if(seen[u] == s){ // found cycle, contract
                Node* nd = 0;
                int end = qi, time = uf.time();
                do nd = merge(nd, heap[w = path[--qi]]); while(uf.merge(u, w));
                u = uf.find(u); heap[u] = nd; seen[u] = -1;
                cyc.emplace_front(u, time, vector<Edge>{&Q[qi], &Q[end]});
            }
        }
        for(int i=0; i<qi; i++) in[uf.find(Q[i].e)] = Q[i];
    }
    for(auto& [u,t,comp] : cyc){
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.e)] = e;
        in[uf.find(inEdge.e)] = inEdge;
    }
    for(int i=0; i<n; i++) par[i] = in[i].s;
    return {res, par};
}
```

ManhattanMST.h

Description: Given 2d points, find MST with taxi distance.
Usage: 0-base index internally. taxiMST(pts); Returns mst's tree edges with (length, a, b); Note that union-find need return value.
Time: $\mathcal{O}(N \log N)$, $N = 2 \times 10^5$ in 363ms at yosupo.

d41d8c, 26 lines

```
struct point { ll x, y; };

vector<tuple<ll, int, int>> taxiMST(vector<point> a){
    int n = a.size();
    vector<int> ind(n);
    iota(ind.begin(), ind.end(), 0);
    vector<tuple<ll, int, int>> edge;
    for(int k=0; k<4; k++){
        sort(ind.begin(), ind.end(), [&](int i,int j){return a[i].x-a[j].x < a[j].y-a[i].y;});
        map<ll, int> mp;
        for(auto i: ind){
            for(auto it=mp.lower_bound(-a[i].y); it!=mp.end(); it=mp.erase(it)){
                int j = it->second; point d = {a[i].x-a[j].x, a[i].y-a[j].y};
                if(d.y > d.x) break;
                edge.push_back({d.x + d.y, i, j});
            }
            mp.insert({-a[i].y, i});
        }
        for(auto &p: a) if(k & 1) p.x = -p.x; else swap(p.x, p.y);
    }
    sort(edge.begin(), edge.end());
    DisjointSet dsu(n);
    vector<tuple<ll, int, int>> res;
```

```
    for(auto [x, i, j]: edge) if(dsu.merge(i, j)) res.push_back
        ({x, i, j});
    return res;
}
```

2.8 Math

2.8.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--`, `mat[b][b]++` (and `mat[b][a]--`, `mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

2.8.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k).$$

Contest (3)

template.cpp — 37 lines

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<int, pii> piii;
typedef pair<ll, ll> pll;
typedef pair<ll, pll> plll;
#define fi first
#define se second
const int INF = 1e9+1;
const int P = 1000000007;
const ll LLINF = (ll)1e18+1;
template <typename T>
ostream& operator<<(ostream& os, const vector<T>& v) { for(auto
     i : v) os << i << " "; os << "\n"; return os; }
template <typename T1, typename T2>
ostream& operator<<(ostream& os, const pair<T1, T2>& p) { os <<
      p.fi << " " << p.se; return os; }
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());
#define rnd(x, y) uniform_int_distribution<int>(x, y)(rng)

ll mod(ll a, ll b) { return ((a%b) + b) % b; }
ll ext_gcd(ll a, ll b, ll &x, ll &y) {
    ll g = a; x = 1, y = 0;
    if(b) g = ext_gcd(b, a % b, y, x), y -= a / b * x;
    return g;
}
ll inv(ll a, ll m) {
    ll x, y; ll g = ext_gcd(a, m, x, y);
    if(g > 1) return -1;
    return mod(x, m);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    return 0;
}
```

troubleshoot.txt — 52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Data structures (4)

LazySegmentTree.h

Description: 0-index, [l, r] interval

Usage: `SegmentTree seg(n); seg.query(l, r); seg.update(l, r, val);`

d41d8c, 64 lines

```
struct SegmentTree {
    int n, h;
    vector<int> arr;
    vector<int> lazy;
    SegmentTree(int _n) : n(_n) {
        h = Log2(n);
        n = 1 << h;
        arr.resize(2*n, 0);
        lazy.resize(2*n, 0);
    }

    void update(int l, int r, int c) {
        l += n, r += n;
        for (int i=h; i>=1; --i) {
            if (l >> i << i != l) push(l >> i);
            if ((r+1) >> i << i != (r+1)) push(r >> i);
        }
        for (int L=l, R=r; L<=R; L/=2, R/=2) {
            if (L & 1) apply(L++, c);
            if (~R & 1) apply(R--, c);
        }
        for (int i=1; i<=h; ++i) {
            if (l >> i << i != l) pull(l >> i);
            if ((r+1) >> i << i != (r+1)) pull(r >> i);
        }
    }

    int query(int l, int r) {
        l += n, r += n;
        for (int i=h; i>=1; --i) {
            if (l >> i << i != l) push(l >> i);
            if ((r+1) >> i << i != (r+1)) push(r >> i);
        }
        int ret = 0;
        for (; l <= r; l/=2, r/=2) {
            if (l & 1) ret = max(ret, arr[l++]);
            if (~r & 1) ret = max(ret, arr[r--]);
        }
        return ret;
    }

    void push(int x) {
        if (lazy[x] != 0) {
            apply(2*x, lazy[x]);
            apply(2*x+1, lazy[x]);
            lazy[x] = 0;
        }
    }

    void apply(int x, int c) {
        arr[x] = max(arr[x], c);
        if (x < n) lazy[x] = c;
    }

    void pull(int x) {
        arr[x] = max(arr[2*x], arr[2*x+1]);
    }

    static int Log2(int x){
        int ret = 0;
        while (x > (1 << ret)) ret++;
        return ret;
    }
};
```

ConvexHullTrick.h

Description: Max query, call init() before use.

d41d8c, 55 lines

```
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(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};
}

// Container where you can add lines of the form kx+m, and
     query maximum values at points x.
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;
  }
};
```

FenwickTree.h

Description: 0-indexed. (1-index for internal bit trick)
Usage: `FenwickTree fen(n); fen.add(x, val); fen.sum(x);`

d41d8c, 12 lines

```
struct FenwickTree {
    vector<int> tree;
    FenwickTree(int size) { tree.resize(size+1, 0); }
    int sum(int pos) {
        int ret = 0;
        for (int i=pos+1; i>0; i &= (i-1)) ret += tree[i];
        return ret;
    }
    void add(int pos, int val) {
        for (int i=pos+1; i<tree.size(); i+=(i & -i)) tree[i]
             += val;
    }
};
```

HLD.h

d41d8c, 54 lines

```
class HLD {
private:
    vector<vector<int>> adj;
    vector<int> in, sz, par, top, depth;
    void traverse1(int u) {
        sz[u] = 1;
        for (int &v: adj[u]) {
            adj[v].erase(find(adj[v].begin(), adj[v].end(), u))
                 ;
            depth[v] = depth[u] + 1;
            traverse1(v);
            par[v] = u;
            sz[u] += sz[v];
            if (sz[v] > sz[adj[u][0]]) swap(v, adj[u][0]);
        }
    }
    void traverse2(int u) {
        static int n = 0;
        in[u] = n++;
        for (int v: adj[u]) {
            top[v] = (v == adj[u][0] ? top[u] : v);
            traverse2(v);
        }
    }
public:
    void link(int u, int v) { // u and v is 1-based
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void init() { // have to call after linking
        top[1] = 1;
        traverse1(1);
        traverse2(1);
    }
    // u is 1-based and returns dfs-order [s, e) 0-based index
    pii subtree(int u) {
        return {in[u], in[u] + sz[u]};
    }
    // u and v is 1-based and returns array of dfs-order [s, e)
         0-based index
    vector<pii> path(int u, int v) {
        vector<pii> res;
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) swap(u, v);
            res.emplace_back(in[top[u]], in[u] + 1);
            u = par[top[u]];
        }
        res.emplace_back(min(in[u], in[v]), max(in[u], in[v]) +
             1);
        return res;
    }
    HLD(int n) { // n is number of vertexes
        adj.resize(n+1); depth.resize(n+1);
        in.resize(n+1); sz.resize(n+1);
        par.resize(n+1); top.resize(n+1);
    }
};
```

PBDS.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. To get a map, change null_type.
Time: $\mathcal{O}(\log N)$

d41d8c, 13 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
template<class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
                  tree_order_statistics_node_update>;
int main() {
    ordered_set<int> 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);
}
```

Rope.h

Description: 1 x y: Move SxSx+1...Sy to front of string. $(0 \leq x \leq y < N)$ 2 x y: Move SxSx+1...Sy to back of string. $(0 \leq x \leq y < N)$ 3 x: Print Sx. $(0 \leq x < N)$ cf. rope.erase(index, count) : erase [index, index+count)

<ext/rope> d41d8c, 23 lines

```
using namespace __gnu_cxx;
int main() {
    string s; cin >> s;
    rope<char> R;
    R.append(s.c_str());
    int q; cin >> q;
    while(q--) {
        int t, x, y; cin >> t;
        switch(t) {
            case 1:
                cin >> x >> y; y++;
                R = R.substr(x, y-x) + R.substr(0, x) + R.
                     substr(y, s.size());
                break;
            case 2:
                cin >> x >> y; y++;
                R = R.substr(0, x) + R.substr(y, s.size()) + R.
                     substr(x, y-x);
                break;
            default:
                cin >> x;
                cout << R[x] << "\n";
        }
    }
}
```

PersistentSegmentTree.h

Description: Point update (addition), range sum query
Usage: `Unknown, but just declare sufficient size. You should achieve root number manually after every query/update.`

d41d8c, 69 lines

```
struct PersistentSegmentTree {
    int size;
    int last_root;
    vector<ll> tree, l, r;

    PersistentSegmentTree(int _size) {
        size = _size;
        init(0, size-1);
        last_root = 0;
    }

    void add_node() {
        tree.push_back(0);
        l.push_back(-1);
        r.push_back(-1);
    }

    int init(int nl, int nr) {
        int n = tree.size();
        add_node();
```

```
        if (nl == nr) {
            tree[n] = 0;
            return n;
        }
        int mid = (nl + nr) / 2;
        l[n] = init(nl, mid);
        r[n] = init(mid+1, nr);
        return n;
    }

    void update(int ori, int pos, int val, int nl, int nr) {
        int n = tree.size();
        add_node();
        if (nl == nr) {
            tree[n] = tree[ori] + val;
            return;
        }

        int mid = (nl + nr) / 2;
        if (pos <= mid) {
            l[n] = tree.size();
            r[n] = r[ori];
            update(l[ori], pos, val, nl, mid);
        } else {
            l[n] = l[ori];
            r[n] = tree.size();
            update(r[ori], pos, val, mid+1, nr);
        }
        tree[n] = tree[l[n]] + tree[r[n]];
    }

    void update(int pos, int val) {
        int new_root = tree.size();
        update(last_root, pos, val, 0, size-1);
        last_root = new_root;
    }

    ll query(int a, int b, int n, int nl, int nr) {
        if (n == -1) return 0;
        if (b < nl || nr < a) return 0;
        if (a <= nl && nr <= b) return tree[n];
        int mid = (nl + nr) / 2;
        return query(a, b, l[n], nl, mid) + query(a, b, r[n],
            mid+1, nr);
    }

    ll query(int x, int root) {
        return query(0, x, root, 0, size-1);
    }
};
```

Geometry (5)

5.1 Analytic Geometry

Area $A = \sqrt{p(p-a)(p-b)(p-c)}$ when $p = (a+b+c)/2$
Circumscribed circle $R = abc/4A$, inscribed circle $r = A/p$
Middle line length $m_a = \sqrt{2b^2 + 2c^2 - a^2}/2$
Bisector line length $s_a = \sqrt{bc[1 - (\frac{a}{b+c})^2]}$

Name	α	β	γ	
R	$a^2\mathcal{A}$	$b^2\mathcal{B}$	$c^2\mathcal{C}$	$\mathcal{A} = b^2 + c^2 - a^2$
r	a	b	c	$\mathcal{B} = a^2 + c^2 - b^2$
G	1	1	1	$\mathcal{C} = a^2 + b^2 - c^2$
H	$\mathcal{BC}$	$\mathcal{CA}$	$\mathcal{AB}$	
Excircle(A)	$-a$	b	c	

$HG : GO = 1 : 2$. H of triangle made by middle point on arc of circumscribed circle is equal to inscribed circle center of original triangle.

5.2 Geometry

ShamosHoey.h

Description: Check whether segments are intersected at least once or not. Strict option available, and it depends on the segment intersection function.
Usage: `0-base index. vector<array<Point, 2>> pts(n); auto ret = ShamosHoey(pts);`
Time: $\mathcal{O}(N \log N)$, $N = 2 \times 10^5$ in 320ms.

d41d8c, 75 lines

```
struct Line{
    static ll CUR_X; ll x1, y1, x2, y2, id;
    Line(Point p1, Point p2, int id) : id(id) {
        if(p2 < p1) swap(p1, p2);
        x1 = p1.x, y1 = p1.y;
        x2 = p2.x, y2 = p2.y;
    } Line() = default;
    int get_k() const { return y1 != y2 ? (x2-x1)/(y1-y2) : -1;
        }
    void convert_k(int k){ // x1,y1,x2,y2 = O(COORD^2), use
        i128 in ccw
        Line res; res.x1=x1+y1*k;res.y1=-x1*k+y1; res.x2=x2+y2*
            k;res.y2=-x2*k+y2;
        x1 = res.x1; y1 = res.y1; x2 = res.x2; y2 = res.y2; if(
            x1 > x2) swap(x1, x2), swap(y1, y2);
    }
    ld get_y(ll offset=0) const { // OVERFLOW
        ld t = ld(CUR_X-x1+offset) / (x2-x1);
        return t * (y2 - y1) + y1;
    }
    bool operator < (const Line &l) const {
        return get_y() < l.get_y();
    }
    // strict
    // bool operator < (const Line &l) const {
    //     auto le = get_y(), ri = l.get_y();
    //     if(abs(le-ri) > 1e-7) return le < ri;
    //     if(CUR_X == x1 || CUR_X == l.x1) return get_y(1) < l
    //       .get_y(1);
    //     else return get_y(-1) < l.get_y(-1);
    // }
}; ll Line::CUR_X = 0;
struct Event{ // f=0 st, f=1 ed
    ll x, y, i, f; Event() = default;
    Event(Line l, ll i, ll f) : i(i), f(f) {
        if(f==0) tie(x,y) = tie(l.x1,l.y1);
        else tie(x,y) = tie(l.x2,l.y2);
    }
    bool operator < (const Event &e) const {
        return tie(x,f,y) < tie(e.x,e.f,e.y);
        // strict
        // return make_tuple(x,-f,y) < make_tuple(e.x,-e.f,e.y)
            ;
    }
};

bool intersect(Line l1, Line l2) {
    Point p1{l1.x1,l1.y1}, p2{l1.x2,l1.y2};
    Point p3{l2.x1,l2.y1}, p4{l2.x2,l2.y2};
    // Intersection logic depends on problem
}

tuple<bool,int,int> ShamosHoey(vector<array<Point,2>> v){
    int n = v.size(); vector<int> use(n+1);
    vector<Line> lines; vector<Event> E; multiset<Line> T;
    for(int i=0; i<n; i++){
        lines.emplace_back(v[i][0], v[i][1], i);
        if(int t=lines[i].get_k(); 0<=t && t<=n) use[t] = 1;
    }
    int k = find(use.begin(), use.end(), 0) - use.begin();
    for(int i=0; i<n; i++){
        lines[i].convert_k(k);
        E.emplace_back(lines[i], i, 0);
        E.emplace_back(lines[i], i, 1);
    }
    sort(E.begin(), E.end());
    for(auto &e : E){
        Line::CUR_X = e.x;
        if(e.f == 0){
            auto it = T.insert(lines[e.i]);
            if(next(it) != T.end() && intersect(lines[e.i], *
                next(it))) return {true, e.i, next(it)->id};
            if(it != T.begin() && intersect(lines[e.i], *prev(
                it))) return {true, e.i, prev(it)->id};
        } else{
            auto it = T.lower_bound(lines[e.i]);
            if(it != T.begin() && next(it) != T.end() &&
              intersect(*prev(it), *next(it))) return {true,
                 prev(it)->id, next(it)->id};
            T.erase(it);
        }
    }
    return {false, -1, -1};
}
```

HalfPlaneIntersection.h

Description: Calculate intersection of left half plane of line (s->t).
Usage: `0-base index. vector<Point> ret = HPI(lines);`
Time: $\mathcal{O}(N \log N)$, no data.

d41d8c, 52 lines

```
struct Line {
    Point s, t;
};

const ld eps = 1e-9;
bool equals(ld a, ld b) { return abs(a-b) < eps; }

bool line_intersect(Point& s1, Point& e1, Point& s2, Point& e2,
     Point& v) {
    ld det = (e2-s2) ^ (e1-s1);
    if (equals(det, 0)) return 0;
    ld s = (ld)((s2.x-s1.x) * (s2.y-e2.y) + (s2.y-s1.y) * (e2.x
        -s2.x)) / det;
    v.x = s1.x + (e1.x-s1.x) * s;
    v.y = s1.y + (e1.y-s1.y) * s;
    return 1;
}

bool bad(Line& a, Line& b, Line& c) {
    Point v;
    if (!line_intersect(a.s, a.t, b.s, b.t, v)) return 0;
    ld crs = (c.t-c.s) ^ (v-c.s);
    return crs < 0 || equals(crs, 0);
}

vector<Point> HPI(vector<Line>& ln) {
    auto lsgn = [&](const Line& a) {
```

```
        if(a.s.y == a.t.y) return a.s.x > a.t.x;
        return a.s.y > a.t.y;
    };
    sort(ln.begin(), ln.end(), [&](const Line& a, const Line& b
        ) {
        if(lsgn(a) != lsgn(b)) return lsgn(a) < lsgn(b);
        return (a.t.x-a.s.x)*(b.t.y-b.s.y)-(a.t.y-a.s.y)*(b.t.x
            -b.s.x)>0;
    });
    deque<Line> dq;
    for(int i=0; i<ln.size(); i++) {
        while(dq.size() >= 2 && bad(dq[dq.size()-2], dq.back(),
            ln[i]))
            dq.pop_back();
        while(dq.size() >= 2 && bad(dq[0], dq[1], ln[i]))
            dq.pop_front();
        if(dq.size() < 2 || !bad(dq.back(), ln[i], dq[0]))
            dq.push_back(ln[i]);
    }
    vector<Point> res;
    if(dq.size() >= 3) {
        for(int i=0; i<dq.size(); i++) {
            int j=(i+1)%dq.size();
            Point v;
            if(!line_intersect(dq[i].s, dq[i].t, dq[j].s, dq[j
                ].t, v)) continue;
            res.push_back(v);
  }
    }
    return res;
}
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ...}, all counter-clockwise.
Usage: `vector<P> tris = triangulate(pts);`
Time: $\mathcal{O}(n \log n)$, $\sum n \log n = 1.3 \times 10^7$ in 2500ms.

d41d8c, 88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

BulldozerTrick.h

Description: ?? Bulldozer trick.
Usage: `vector<Line> V;`
Time: $\mathcal{O}\left(n^2 \log n\right)$, no data but relatively fast.

d41d8c, 27 lines

```
struct Line {
    ll i, j, dx, dy; // dx >= 0
Line(int i, int j, const Point &pi, const Point &pj)
    : i(i), j(j), dx(pj.x-pi.x), dy(pj.y-pi.y) {}
    bool operator < (const Line &l) const {
        return make_tuple(dy*l.dx, i, j) < make_tuple(l.dy*dx,
            l.i, l.j); }
    bool operator == (const Line &l) const {
        return dy * l.dx == l.dy * dx;
    }
};
void Solve(){
    sort(A+1, A+N+1); iota(P+1, P+N+1, 1);
    vector<Line> V; V.reserve(N*(N-1)/2);
    for(int i=1; i<=N; i++)
        for(int j=i+1; j<=N; j++)
            V.emplace_back(i, j, A[i], A[j]);
    sort(V.begin(), V.end());
    for(int i=0, j=0; i<V.size(); i=j){
        while(j < V.size() && V[i] == V[j]) j++;
        for(int k=i; k<j; k++){
            int u = V[k].i, v = V[k].j; // point id, index ->
                Pos[id]
            swap(Pos[u], Pos[v]); swap(A[Pos[u]], A[Pos[v]]);
            if(Pos[u] > Pos[v]) swap(u, v);
            // @TODO
        }
    }
}
```

5.3 Circles

5.4 Polygons

PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)
Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

"Point.h", "sideOf.h" d41d8c, 33 lines

```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
    double ret = 0;
    rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
        P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
        vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
        rep(j,0,sz(poly)) if (i != j) {
            rep(u,0,sz(poly[j])) {
                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly
                    [j])];
                int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
                if (sc != sd) {
                    double sa = C.cross(D, A), sb = C.cross(D,
                        B);
                    if (min(sc, sd) < 0)
                        segs.emplace_back(sa / (sa - sb), sgn(
                            sc - sd));
                } else if (!sc && !sd && j<i && sgn((B-A).dot(D
                    -C))>0){
                    segs.emplace_back(rat(C - A, B - A), 1);
                    segs.emplace_back(rat(D - A, B - A), -1);
                }
            }
        }
        sort(all(segs));
        for (auto& s : segs) s.first = min(max(s.first, 0.0),
            1.0);
        double sum = 0;
        int cnt = segs[0].second;
        rep(j,1,sz(segs)) {
```

```
            if (!cnt) sum += segs[j].first - segs[j - 1].first;
            cnt += segs[j].second;
        }
        ret += A.cross(B) * sum;
    }
    return ret / 2;
}
```

5.5 Misc. Point Set Problems

5.6 3D

Techniques (A)

techniques.txt
159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```