Eötvös Loránd University

ELTE 1

ICPC World Finals 2024

1 Centroid Decomposition

```cpp
struct node{
    vector<int> to;
    vector<pair<int, int>> p; // csak ha kell
    int sz = 0;
    bool vis = false;
};
vector<node> g;
int get_sz(int x, int p = -1){
    g[x].sz = 1;
    for(int y : g[x].to) if(y != p && !g[y].vis) g[x].sz += get_sz(y, x);
    return g[x].sz;
}
pair<int, int> get_c(int x, int n, int p = -1) { // előtte: get_sz
    for(int y : g[x].to) if(y != p && !g[y].vis && g[y].sz * 2 >= n) return g[y].sz * 2 == n ? make_pair(x, y) : get_c(y, n, x);
    return make_pair(x, x);
}
void dfs_sub(int x, int c, int d = 0, int p = -1){
    g[x].p.emplace_back(c, d);
    for(int y : g[x].to) if(y != p && !g[y].vis) dfs_sub(y, c, d + 1, x);
}
void centroid_decomp(int c){
    int sz = get_sz(c);
    c = get_c(c, sz).first;
    g[c].vis = true;
    dfs_sub(c, c); // centroid szülők távolsága a csúcstól (önmagát is beleértve) | a sz itt már nem jó újra kell számolni
    // calc
    for(int y : g[c].to) if(!g[y].vis) centroid_decomp(y);
}
```

2 Heavy Light Decomposition

```cpp
struct node{
    vector<int> to;
    int l, r, i, p, hld_p, sz, d; // l: st bal, r: st jobb, i: st idx, p: ős, hld_p: light edge előtti ős, sz: részfa mérete, d: gyökértől vett távolság
    // heavy út: [l, r], részfa: [i, i + sz), !!! szegmensfában g[x].i-t kell használni
};
vector<node> g;
int dfs_sz(int x, int d = 0, int p = -1){
    g[x].sz = 1;
    g[x].d = d;
    for(int y : g[x].to) if(y != p) g[x].sz += dfs_sz(y, d + 1, x);
    return g[x].sz;
}
int IDX = 0; // reset
int dfs_hld(int x, int hld_p, int p = -1){ // x = hld_p = root
    g[x].i = g[x].r = IDX++;
    g[x].l = g[hld_p].i;
    g[x].p = p;
    g[x].hld_p = hld_p;
    sort(g[x].to.begin(), g[x].to.end(), [](int i, int j){ return g[i].sz > g[j].sz; });
    bool fst = true;
    for(int y : g[x].to){
        if(y == p) continue;
        if(fst) { g[x].r = dfs_hld(y, hld_p, x); fst = false; }
        else dfs_hld(y, y, x);
    }
    return g[x].r;
}
void build_hld(int root) { dfs_sz(root); IDX = 0; dfs_hld(root, root); }
```

3 2 SAT

```cpp
const int c=1000005;
int n, m, compdb, comp[c], res[c];
vector<int> el[c], el2[c];
bool vis[c], vis2[c];
vector<int> sor;
int par(int a) {
    return (a<=n ? a+n : a-n);
}
void add(int a, int b) {
    // a->b;
    el[a].push_back(b);
    el2[b].push_back(a);
    a=par(a), b=par(b);
    el[b].push_back(a);
    el2[a].push_back(b);
}
void dfs(int a) {
    vis[a]=true;
    for (auto x:el[a]) {
        if (!vis[x]) {
            dfs(x);
```

```
        }
    }
    sor.push_back(a);
}
void dfs2(int a, int s) {
    vis2[a]=true;
    comp[a]=s;
    for (auto x:el2[a]) {
        if (!vis2[x]) {
            dfs2(x, s);
        }
    }
}
bool solve() {
    // 1-tol n-ig vannak a valtozok
    // false ha nincs megoldas egyebkent res[i]=1 ha az i. valtozo igaz egy lehetseges megoldasban
    for (int i=1; i<=2*n; i++) {
        if (!vis[i]) {
            dfs(i);
        }
    }
    reverse(sor.begin(), sor.end());
    for (auto x:sor) {
        if (!vis2[x]) {
            dfs2(x, ++compdb);
        }
    }
    for (int i=1; i<=n; i++) {
        int a=i, b=par(i);
        if (comp[a]==comp[b]) {
            return false;
        }

        if (comp[b]>comp[a]) {
            res[i]=1;
        }
    }
    return true;
}
```

4 Bipartite Max Matching

```
using namespace std;

struct HopcroftKarp {
    std::vector<int> G, L, R;
    int flow;
    HopcroftKarp(int n, int m, const std::vector<std::array<int, 2>> &edges) : G(edges.size()), L(n, -1), R(m, -1), flow(0) {
        std::vector<int> deg(n + 1), a, p, q(n);
        for (auto &[x, y] : edges) { deg[x]++; }
        for (int i = 1; i <= n; i++) { deg[i] += deg[i - 1]; }
        for (auto &[x, y] : edges) { G[--deg[x]] = y; }
        while (true) {
            a.assign(n, -1), p.assign(n, -1);
            int t = 0;
            for (int i = 0; i < n; i++) {
                if (L[i] == -1) {
                    q[t++] = a[i] = p[i] = i;
                }
            }
            bool match = false;
            for (int i = 0; i < t; i++) {
                int x = q[i];
                if (L[a[x]] != -1) {
                    continue;
                }
                for (int j = deg[x]; j < deg[x + 1]; j++) {
                    int y = G[j];
                    if (R[y] == -1) {
                        while (y != -1) {
                            R[y] = x, std::swap(L[x], y), x = p[x];
                        }
                        match = true, flow++;
                        break;
                    }
                    if (p[R[y]] == -1) {
                        q[t++] = y = R[y], p[y] = x, a[y] = a[x];
                    }
                }
            }
            if (!match) {
                break;
            }
        }
    }

    std::vector<std::array<int, 2>> get_edges() {
        std::vector<std::array<int, 2>> res;
        for (int i = 0; i < L.size(); i++) {
            if (L[i] != -1) {
                res.push_back({i, L[i]});
            }
        }
        return res;
    }
```

```
};
```

5 Max Matching

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

struct Matching {
  queue<int> q; int ans, n;
  vector<int> fa, s, v, pre, match;
  Matching(auto &&g) : ans(0), n(g.size()), fa(n + 1), s(n + 1), v(n + 1), pre(n + 1, n), match(n + 1, n) {
    for (int x = 0; x < n; ++x) if (match[x] == n) ans += Bfs(g, x, n);
  }
  int Find(int u) {
    return u == fa[u] ? u : fa[u] = Find(fa[u]); }
  int LCA(int x, int y, int n) {
    static int tk = 0; tk++; x = Find(x); y = Find(y);
    for (;; swap(x, y)) if (x != n) {
      if (v[x] == tk) return x;
      v[x] = tk;
      x = Find(pre[match[x]]);
    }
  }
  void Blossom(int x, int y, int l) {
    for (; Find(x) != l; x = pre[y]) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      for (int z: {x, y}) if (fa[z] == z) fa[z] = l;
    }
  }
  bool Bfs(auto &&g, int r, int n) {
    iota(all(fa), 0); ranges::fill(s, -1);
    q = queue<int>(); q.push(r); s[r] = 0;
    for (; !q.empty(); q.pop()) {
      for (int x = q.front(); int u : g[x])
        if (s[u] == -1) {
          if (pre[u] = x, s[u] = 1, match[u] == n) {
            for (int a = u, b = x, last;
                b != n; a = last, b = pre[a])
              last = match[b], match[b] = a, match[a] = b;
            return true;
          }
          q.push(match[u]); s[match[u]] = 0;
        } else if (!s[u] && Find(u) != Find(x)) {
          int l = LCA(u, x, n);
          Blossom(x, u, l); Blossom(u, x, l);
        }
    }
    return false;
  }
}; // init: vector<vector<int>> gráf (n: gráf mérete), párosítás mérete: ans, párosítása i-nek: nincs ->
↪  match[i] == n | van -> match[i]
```

6 Max Weighted Matching

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

namespace weighted_blossom_tree{
        #define d(x) (lab[x.u]+lab[x.v]-e[x.u][x.v].w*2)
        const int N=403*2; using ll = long long; using T = int; // sum of weight, single weight
        const T inf=numeric_limits<T>::max()>>1;
        struct Q{ int u, v; T w; } e[N][N]; vector<int> p[N];
        int n, m=0, id, h, t, lk[N], sl[N], st[N], f[N], b[N][N], s[N], ed[N], q[N]; T lab[N];
        void upd(int u, int v){ if(!sl[v] || d(e[u][v]) < d(e[sl[v]][v])) sl[v] = u; }
        void ss(int v){
                sl[v]=0; for(auto u=1; u<=n; u ++) if(e[u][v].w > 0 && st[u] != v && !s[st[u]]) upd(u, v);
        }
        void ins(int u){ if(u <= n) q[++ t] = u; else for(auto v : p[u]) ins(v); }
        void mdf(int u, int w){ st[u]=w; if(u > n) for(auto v : p[u]) mdf(v, w); }
        int gr(int u,int v){
                if((v=find(p[u].begin(), p[u].end(), v) - p[u].begin()) & 1){
                        reverse(p[u].begin()+1, p[u].end()); return (int)p[u].size() - v;
                }
                return v;
        }
        void stm(int u, int v){
                lk[u] = e[u][v].v;
                if(u <= n) return; Q w = e[u][v];
                int x = b[u][w.u], y = gr(u,x);
                for(auto i=0; i<y; i ++) stm(p[u][i], p[u][i^1]);
                stm(x, v); rotate(p[u].begin(), p[u].begin()+y, p[u].end());
        }
        void aug(int u, int v){
                int w = st[lk[u]]; stm(u, v); if(!w) return;
                stm(w, st[f[w]]); aug(st[f[w]], w);
        }
        int lca(int u, int v){
                for(++ id; u|v; swap(u, v)){
                        if(!u) continue; if(ed[u] == id) return u;
```

```
            ed[u] = id; if(u = st[lk[u]]) u = st[f[u]]; // not ==
        }
        return 0;
    }
    void add(int u, int a, int v){
        int x = n+1; while(x <= m && st[x]) x ++;
        if(x > m) m ++;
        lab[x] = s[x] = st[x] = 0; lk[x] = lk[a];
        p[x].clear(); p[x].push_back(a);
        for(auto i=u, j=0; i!=a; i=st[f[j]]) p[x].push_back(i), p[x].push_back(j=st[lk[i]]), ins(j);
        reverse(p[x].begin()+1, p[x].end());
        for(auto i=v, j=0; i!=a; i=st[f[j]]) p[x].push_back(i), p[x].push_back(j=st[lk[i]]), ins(j);
        mdf(x, x); for(auto i=1; i<=m; i ++) e[x][i].w = e[i][x].w = 0;
        memset(b[x]+1, 0, n*sizeof b[0][0]);
        for(auto u : p[x]){
            for(v=1; v<=m; v ++) if(!e[x][v].w || d(e[u][v]) < d(e[x][v])) e[x][v] = e[u][v],e[v][x] = e[v][u];
            for(v=1; v<=n; v ++) if(b[u][v]) b[x][v] = u;
        }
        ss(x);
    }
    void ex(int u){  // s[u] == 1
        for(auto x : p[u]) mdf(x, x);
        int a = b[u][e[u][f[u]].u],r = gr(u, a);
        for(auto i=0; i<r; i+=2){
            int x = p[u][i], y = p[u][i+1];
            f[x] = e[y][x].u; s[x] = 1; s[y] = 0; sl[x] = 0; ss(y); ins(y);
        }
        s[a] = 1; f[a] = f[u];
        for(auto i=r+1; i<p[u].size(); i ++) s[p[u][i]] = -1, ss(p[u][i]);
        st[u] = 0;
    }
    bool on(const Q &e){
        int u=st[e.u], v=st[e.v], a;
        if(s[v] == -1) f[v] = e.u, s[v] = 1, a = st[lk[v]], sl[v] = sl[a] = s[a] = 0, ins(a);
        else if(!s[v]){
            a = lca(u, v); if(!a) return aug(u,v), aug(v,u), true; else add(u,a,v);
        }
        return false;
    }
    bool bfs(){
        memset(s+1, -1, m*sizeof s[0]); memset(sl+1, 0, m*sizeof sl[0]);
        h = 1; t = 0; for(auto i=1; i<=m; i ++) if(st[i] == i && !lk[i]) f[i] = s[i] = 0, ins(i);
        if(h > t) return 0;
        while(true){
            while(h <= t){
                int u = q[h ++];
                if(s[st[u]] != 1) for(auto v=1; v<=n; v ++) if(e[u][v].w > 0 && st[u] != st[v])
                    if(d(e[u][v])) upd(u, st[v]); else if(on(e[u][v])) return true;
            }
            T x = inf;
            for(auto i=n+1; i<=m; i ++) if(st[i] == i && s[i] == 1) x = min(x, lab[i]>>1);
            for(auto i=1; i<=m; i ++) if(st[i] == i && sl[i] && s[i] != 1) x = min(x, d(e[sl[i]][i])>>s[i]+1);
            for(auto i=1; i<=n; i ++) if(~s[st[i]]) if((lab[i] += (s[st[i]]*2-1)*x) <= 0) return false;
            for(auto i=n+1 ;i<=m; i ++) if(st[i] == i && ~s[st[i]]) lab[i] += (2-s[st[i]]*4)*x;
            h = 1; t = 0;
            for(auto i=1; i<=m; i ++) if(st[i] == i && sl[i] && st[sl[i]] != i && !d(e[sl[i]][i]) && on(e[sl[i]][i])) return true;
            for(auto i=n+1; i<=m; i ++) if(st[i] == i && s[i] == 1 && !lab[i]) ex(i);
        }
        return 0;
    }
    template<typename TT> pair<ll, vector<array<int, 2>>> run(int N, vector<tuple<int,int,TT>> edges){ // 1-based
        for(auto &[u, v, w]: edges) ++ u, ++ v;
        memset(ed+1, 0, m*sizeof ed[0]); memset(lk+1, 0, m*sizeof lk[0]);
        n = m = N; id = 0; iota(st+1, st+n+1, 1); T wm = 0; ll weight = 0;
        for(auto i=1; i<=n; i ++) for(auto j=1; j<=n; j ++) e[i][j] = {i,j,0};
        for(auto [u,v,w] : edges) wm = max(wm, e[v][u].w=e[u][v].w=max(e[u][v].w,(T)w));
        for(auto i=1; i<=n; i ++) p[i].clear();
        for(auto i=1; i<=n; i ++) for(auto j=1; j<=n; j ++) b[i][j] = i*(i==j);
        fill_n(lab+1, n, wm); while(bfs());
        vector<array<int, 2>> matching;
        for(auto i=1; i<=n; i ++) if(i < lk[i]) weight += e[i][lk[i]].w, matching.push_back({i - 1, lk[i] - 1});
        return {weight, matching};
    }
    #undef d
} // call: weighted_blossom_tree::run(n, edges) | returns: pair{weight, vector{edge}}
```

7 Flow

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
```

```
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;
    Dinic(int n, int s, int t) : n(n), s(s), t(t), adj(n), level(n), ptr(n) {}
    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }
    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1 || level[edges[id].u] != -1) continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }
    long long dfs(int v, long long pushed) {
        if (pushed == 0) return 0;
        if (v == t) return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }
    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs()) break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) f += pushed;
        }
        return f;
    }
};
```

8 Min Cost Max Flow

```
struct Edge { int from, to, capacity, cost; };

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
```

```
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

9 Convex Hull Trick

```
const long long INF = 2e18 + 10; // Elég nagynak kell lennie
struct line{ // a * x + b | a, b: kezdetben a legrosszabb egyenes
    mutable long long a = 0, b = INF;
    mutable long double lef = -2e18; bool point = false; // csak a set cht-hoz
    long long get(long long x) const { return a * x + b; }
    long double intersect(const line& e) const { return (long double)(e.b - b) / (a - e.a); }
    bool bad() const { return b == INF; } // ellenőrzi, hogy az egyenes mindennél rosszabb-e (nincsen)
};

struct li_chao{ // update(line) hozzáad egy egyenest, query(x) x-helyen lévő minimum y értéket adja vissza
    struct node{ // Előre foglalt memóriával gyorsabb
        line e;
        node *l = NULL, *r = NULL;
    } *root;
    long long L, R;
    li_chao(long long L, long long R) : root(new node()), L(L), R(R) {}
    void update(node* &p, long long l, long long r, line e){
        if(e.bad()) return;
        if(!p) p = new node();
        int m = (l + r) / 2;
        bool lef = e.get(L) < p->e.get(L);
        bool mid = e.get(m) < p->e.get(m);
        if(mid) swap(e, p->e);
        if(r - l == 1) return;
        else if(lef != mid) update(p->l, l, m, e);
        else update(p->r, m, r, e);
    }
    void update(line e) { update(root, L, R, e); }
    long long query(node *p, long long l, long long r, long long x) {
        if(!p) return INF;
        int m = (l + r) / 2;
        if(x < m) return min(p->e.get(x), query(p->l, l, m, x));
        return min(p->e.get(x), query(p->r, m, r, x));
    }
    long long query(long long x) { return query(root, L, R, x); }
};

struct CHT{
    struct comp{ bool operator()(const line& e1, const line& e2) const { return !e1.point && !e2.point ? e1.a > e2.a : e1.lef < e2.lef; } };
    set<line, comp> lines;
    static inline bool check(const line& a, const line& b, const line& c) { return a.intersect(c) < a.intersect(b); }
    void update(const line& e){
        auto it = lines.insert(e).first;
        if(it->b < e.b) return;
        it->b = e.b;
        auto prv = it == lines.begin() ? lines.end() : prev(it);
        auto nxt = next(it);
        if(prv != lines.end() && nxt != lines.end() && check(*prv, *it, *nxt)) {
            lines.erase(it);
            return;
        }
```

```
        while(prv != lines.end() && prv != lines.begin()){
            auto prv2 = prev(prv);
            if(check(*prv2, *prv, *it)){
                lines.erase(prv);
                prv = prv2;
            } else {
                break;
            }
        }
        while(nxt != lines.end() && next(nxt) != lines.end()){
            auto nxt2 = next(nxt);
            if(check(*it, *nxt, *nxt2)) {
                lines.erase(nxt);
                nxt = nxt2;
            } else{
                break;
            }
        }
        if(prv != lines.end()) it->lef = prv->intersect(*it);
        if(nxt != lines.end()) nxt->lef = it->intersect(*nxt);
    }
    long long query(long long x){
        line tmp;
        tmp.lef = x;
        tmp.point = true;
        auto it = lines.upper_bound(tmp);
        assert(it != lines.begin());
        return prev(it)->get(x);
    }
};
```

10 Float Geometry

```
const long double EPS = 1e-9;

struct point{
    long double x, y;
    point operator+(const point& p) const { return point{x + p.x, y + p.y}; }
    point operator-(const point& p) const { return point{x - p.x, y - p.y}; }
    point operator*(long double t) const { return point{x * t, y * t}; }
    long double len() const { return hypot(x, y); }
    point normalized() const { return (*this) * (1.0 / len()); }
    bool operator<(const point& p) const { return x < p.x - EPS || (abs(x - p.x) < EPS && y < p.y - EPS); }
};

inline long double dot(const point& a, const point& b) { return a.x * b.x + a.y * b.y; }
inline long double cross(const point& a, const point& b) { return a.x * b.y - a.y * b.x; }
inline long double det(long double a, long double b, long double c, long double d) { return a * c - b * d; }
inline long double sqr(long double x) { return x*x; }
inline int sgn(auto x) { return (x > 0 ) - (x < 0); }
inline int dir(const point& a, const point& b, const point& c) { return sgn(cross(b - a, c - a)); }

struct line{ // a * x + b * y + c = 0, normalizáltnak kell lennie
    long double a, b, c;
    line(long double a_, long double b_, long double c_) : a(a_), b(b_), c(c_) {
        long double len = hypot(a, b);
        if(len > EPS) a /= len, b /= len, c /= len;
    }
    line(const point& p1, const point& p2) {
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = -a * p1.x - b * p1.y;
        long double len = hypot(a, b);
        if(len > EPS) a /= len, b /= len, c /= len;
    }

    long double dist(const point& p) { return a * p.x + b * p.y + c; }
};

inline bool paralell(const line& l1, const line& l2) { return abs(det(l1.a, l1.b, l2.a, l2.b)) < EPS; }

inline bool equivalent(const line& l1, const line& l2) {
    return abs(det(l1.a, l1.b, l2.a, l2.b)) < EPS
        && abs(det(l1.a, l1.c, l2.a, l2.c)) < EPS
        && abs(det(l1.b, l1.c, l2.b, l2.c)) < EPS;
}

inline bool intersect1(long double a1, long double a2, long double b1, long double b2){
    return max(min(a1, a2), min(b1, b2)) <= min(max(a1, a2), max(b1, b2)) + EPS;
}

inline bool betw(double l, double r, double x) {
    return min(l, r) <= x + EPS && x <= max(l, r) + EPS;
}

bool intersect(const line& l1, const line& l2, point& res) {
    long double zn = det(l1.a, l1.b, l2.a, l2.b);
    if (abs(zn) < EPS) return false; // párhuzamos
    res.x = -det(l1.c, l1.b, l2.c, l2.b) / zn;
    res.y = -det(l1.a, l1.c, l2.a, l2.c) / zn;
    return true;
}

bool intersect(point a, point b, point c, point d, point& left, point& right) { // ellenőrzi a metszést, metszés esetén a [left, right] szakasz a metszet
    if (!intersect1(a.x, b.x, c.x, d.x) || !intersect1(a.y, b.y, c.y, d.y))
        return false;
```

```
    line m(a, b);
    line n(c, d);
    long double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS) {
        if (abs(m.dist(c)) > EPS || abs(n.dist(a)) > EPS)
            return false;
        if (b < a)
            swap(a, b);
        if (d < c)
            swap(c, d);
        left = max(a, c);
        right = min(b, d);
        return true;
    } else {
        left.x = right.x = -det(m.c, m.b, n.c, n.b) / zn;
        left.y = right.y = -det(m.a, m.c, n.a, n.c) / zn;
        return betw(a.x, b.x, left.x) && betw(a.y, b.y, left.y) &&
               betw(c.x, d.x, left.x) && betw(c.y, d.y, left.y);
    }
}

struct circle{
    point p;
    long double r;
};

vector<point> intersection(const circle& circ, const line& l){ // kör-egyenes metszéspontok
    long double r = circ.r, a = l.a, b = l.b, c = l.c - l.a * circ.p.x - l.b * circ.p.y;
    double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS) return {};
    if (abs (c*c - r*r*(a*a+b*b)) < EPS) return {point{x0, y0} + circ.p};
    double d = r*r - c*c/(a*a+b*b);
    double mult = sqrt (d / (a*a+b*b));
    double ax, ay, bx, by;
    ax = x0 + b * mult;
    bx = x0 - b * mult;
    ay = y0 - a * mult;
    by = y0 + a * mult;
    return {point{ax, ay} + circ.p, point{bx, by} + circ.p};
}

vector<point> intersection(circle circ1, circle circ2){ // kör-kör metszéspontok
    point origo = circ1.p;
    circ2.p = circ2.p - origo;
    circ1.p = {0, 0};
    line l(-2 * circ2.p.x, -2 * circ2.p.y, sqr(circ2.p.x) + sqr(circ2.p.y) + sqr(circ1.r) - sqr(circ2.r));
    auto tmp = intersection(circ1, l);
    for(auto &p : tmp) p = p + origo;
    return tmp;
}

void tangents (point c, double r1, double r2, vector<line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS)  return;
    d = sqrt (abs (d));
    line l(0, 0, 0);
    l.a = (c.x * r + c.y * d) / z;
    l.b = (c.y * r - c.x * d) / z;
    l.c = r1;
    ans.push_back (l);
}

vector<line> tangents (circle a, circle b) { // 2 kör közös érintői
    vector<line> ans;
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents (b.p-a.p, a.r*i, b.r*j, ans);
    for (size_t i=0; i<ans.size(); ++i)
        ans[i].c -= ans[i].a * a.p.x + ans[i].b * a.p.y;
    return ans;
}
```

11 Halfplane Intersection

```
// Redefine epsilon and infinity as necessary. Be mindful of precision errors.
const long double eps = 1e-9, inf = 1e9;
// Basic point/vector struct.
struct Point {
    long double x, y;
    explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {} // Addition, substraction, multiply
    by constant, dot product, cross product.
    friend Point operator + (const Point& p, const Point& q) { return Point(p.x + q.x, p.y + q.y); }
    friend Point operator - (const Point& p, const Point& q) { return Point(p.x - q.x, p.y - q.y); }
    friend Point operator * (const Point& p, const long double& k) { return Point(p.x * k, p.y * k); }
    friend long double dot(const Point& p, const Point& q) { return p.x * q.x + p.y * q.y; }
    friend long double cross(const Point& p, const Point& q) { return p.x * q.y - p.y * q.x; }
};
// Basic half-plane struct.
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the direction vector of the line.
    Point p, pq;
    long double angle;
```

```
    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) { angle = atan2l(pq.y, pq.x); }
    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point& r) { return cross(pq, r - p) < -eps; }
    // Comparator for sorting.
    bool operator < (const Halfplane& e) const { return angle < e.angle; }
    // Intersection point of the lines of two half-planes. It is assumed they're never parallel.
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
// Actual algorithm
vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };
    for(int i = 0; i<4; i++) { // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.push_back(aux);
    }
    // Sort by angle and start algorithm
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < int(H.size()); i++) {
        // Remove from the back of the deque while last half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[len-2]))) { dq.pop_back(); --len; }
        // Remove from the front of the deque while first half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) { dq.pop_front(); --len; }
        // Special case check: Parallel half-planes
        if (len > 0 && fabsl(cross(H[i].pq, dq[len-1].pq)) < eps) {
            // Opposite parallel half-planes that ended up checked against each other.
            if (dot(H[i].pq, dq[len-1].pq) < 0.0) return vector<Point>();
            // Same direction half-plane: keep only the leftmost half-plane.
            if (H[i].out(dq[len-1].p)) { dq.pop_back(); --len; }
            else continue;
        }
        // Add new half-plane
        dq.push_back(H[i]); ++len;
    }
    // Final cleanup: Check half-planes at the front against the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) { dq.pop_back(); --len; }
    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) { dq.pop_front(); --len; }
    // Report empty intersection if necessary
    if (len < 3) return vector<Point>();
    // Reconstruct the convex polygon from the remaining half-planes.
    vector<Point> ret(len);
    for(int i = 0; i+1 < len; i++) { ret[i] = inter(dq[i], dq[i+1]); }
    ret.back() = inter(dq[len-1], dq[0]);
    return ret;
}
```

12 Integer Geometry

```
struct point{
    long long x, y;
    point operator+(const point& p) const { return {x + p.x, y + p.y}; }
    point operator-(const point& p) const { return {x - p.x, y - p.y}; }
    point operator*(long long t) const { return {x * t, y * t}; }
    bool operator==(const point& p) const { return x == p.x && y == p.y; }
    long long len() const { return x * x + y * y; }
};

inline long long dot(const point& a, const point& b) { return a.x * b.x + a.y * b.y; }
inline long long cross(const point& a, const point& b) { return a.x * b.y - b.x * a.y; }
inline int sgn(long long x) { return (x > 0) - (x < 0); }
inline int dir(const point& a, const point& b, const point& c) { return sgn(cross(b - a, c - a)); }

bool comp_args(const point& a, const point& b){ // vektorok rendezése szög alapján (azon belül hossz alapján)
    bool fa = a.y > 0 || (a.y == 0 && a.x >= 0);
    bool fb = b.y > 0 || (b.y == 0 && b.x >= 0);
    if(fa != fb) return fa;
    long long c = cross(a, b);
    return c != 0 ? c > 0 : a.len() < b.len();
}

inline bool contains(const point& a, const point& b, const point& p){ // szakasz tartalmazza-e
    if(dir(a, b, p) != 0) return false;
    long long d = dot(b - a, p - a);
    return 0 <= d && d <= (b-a).len();
}

inline bool intersect1(long long a1, long long a2, long long b1, long long b2){
    return max(min(a1, a2), min(b1, b2)) <= min(max(a1, a2), max(b1, b2));
}
```

```cpp

inline bool intersect(const point& a1, const point& a2, const point& b1, const point& b2){ // szakaszok metszik-e egymást
    if(dir(b1, a1, b2) == 0 && dir(b1, a2, b2) == 0)
        return intersect1(a1.x, a2.x, b1.x, b2.x) && intersect1(a1.y, a2.y, b1.y, b2.y);
    return dir(a1, a2, b1) != dir(a1, a2, b2) && dir(b1, b2, a1) != dir(b1, b2, a2);
}

vector<point> convex_hull(vector<point> a){ // az a pontok konvex burka, minimális pontszámmal
    if(a.empty()) return {};
    int pos = min_element(a.begin(), a.end(), [](const point& a, const point& b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }) - a.begin();
    swap(a[0], a[pos]);
    sort(a.begin() + 1, a.end(), [o = a[0]](const point& a, const point& b) { int d = dir(o, a, b); return d == 1 || (d == 0 && (a-o).len() < (b-o).len()); });
    vector<point> hull;
    for(const point &p : a){
        while(hull.size() > 1 && dir(hull[hull.size() - 2], hull[hull.size() - 1], p) != 1) hull.pop_back();
        hull.push_back(p);
    }
    int j = (int)hull.size() - 2;
    while(j > 0 && dir(hull[j], hull[j+1], hull[0]) != 1) {
        hull.pop_back();
        j--;
    }

    if(hull.size() == 2 && hull[0] == hull[1]) hull.pop_back();

    return hull;
}

vector<point> minkowski_sum(vector<point> a, vector<point> b){ // a és b konvex burkok minkowski összege (konvex burok, minimális pontszámmal)
    if(a.empty() || b.empty()) return {};
    auto comp = [](const point& a, const point& b) { return a.y < b.y || (a.y == b.y && a.x < b.x); };
    int min_a = min_element(a.begin(), a.end(), comp) - a.begin();
    int min_b = min_element(b.begin(), b.end(), comp) - b.begin();
    rotate(a.begin(), a.begin() + min_a, a.end());
    rotate(b.begin(), b.begin() + min_b, b.end());
    a.push_back(a[0]);
    a.push_back(a[1]);
    b.push_back(b[0]);
    b.push_back(b[1]);
    vector<point> hull;
    int i = 0, j = 0;
    while(i < a.size() - 2 || j < b.size() - 2) {
        hull.push_back(a[i] + b[j]);
        long long c = cross(a[i + 1] - a[i], b[j + 1] - b[j]);
        if(c >= 0 && i < a.size() - 2)
            ++i;
        if(c <= 0 && j < b.size() - 2)
            ++j;
    }
    return hull;
}
```

13 Ottoman Bentley

```cpp
struct seg {
    point p, q;
    int id;

    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};

bool intersect(const seg& a, const seg& b) // same as in intersection
{
    return intersect1(a.p.x, a.q.x, b.p.x, b.q.x) &&
           intersect1(a.p.y, a.q.y, b.p.y, b.q.y) &&
           dir(a.p, a.q, b.p) * dir(a.p, a.q, b.q) <= 0 &&
           dir(b.p, b.q, a.p) * dir(b.p, b.q, a.q) <= 0;
}

bool operator<(const seg& a, const seg& b)
{
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}

struct event {
    double x;
    int tp, id;

    event() {}
    event(double x, int tp, int id) : x(x), tp(tp), id(id) {}

    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS)
            return x < e.x;
        return tp > e.tp;
    }
};
```

```cpp
set<seg> s;
vector<set<seg>::iterator> where;

set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}

set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}

// meghatároz egy metsző szakaszpárt az a-ból (x koordináta szerinti legkisebb metszéspont), ezek indexével tér vissza, ha nincs akkor {-1, -1}
pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());

    s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
            if (nxt != s.end() && intersect(*nxt, a[id]))
                return make_pair(nxt->id, id);
            if (prv != s.end() && intersect(*prv, a[id]))
                return make_pair(prv->id, id);
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
            if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
                return make_pair(prv->id, nxt->id);
            s.erase(where[id]);
        }
    }

    return make_pair(-1, -1);
}
```

14 Power series

```cpp
// POWER SERIES OPERATIONS
constexpr int mod = 998244353; // = 2^k * c + 1 | primitív gyöknek jónak kell lennie | 2013265921, 167772161, 2113929217
constexpr int N = 1 << 20; // 2^l, l <= k | max N amit transzformálni lehet
struct mint {
    int x;
    constexpr inline mint(int x = 0) : x(x) {}
    constexpr inline mint operator+(mint o) const { return x + o.x < mod ? x + o.x : x + o.x - mod; }
    constexpr inline mint operator-(mint o) const { return x - o.x < 0 ? x - o.x + mod : x - o.x; }
    constexpr inline mint operator*(mint o) const { return int(uint64_t(x) * o.x % mod); }
    constexpr inline mint &operator+=(mint o) { return *this = *this + o; }
    constexpr inline mint &operator-=(mint o) { return *this = *this - o; }
    constexpr inline mint &operator*=(mint o) { return *this = *this * o; }
    constexpr inline mint inv() const { return pow(mod - 2); }
    constexpr inline mint pow(auto x) const {
        mint a = *this; mint b = 1; for (; x; x >>= 1) { if (x & 1) { b *= a; } a *= a; } return b;
    }
    constexpr inline mint sqrt() const {
        if (pow(mod >> 1).x != 1) return 0;
        int Q = (mod - 1) >> (__countr_zero(mod-1));
        mint x = pow((Q + 1) >> 1), y = pow(Q);
        for (int k = __countr_zero(mod - 1) - 1; k >= 0; --k) // TODO: fix 21
            if (y.pow(1 << k).x != 1) {
                x *= mint(mod_primitive_root()).pow(mod >> (k + 2));
                y *= mint(mod_primitive_root()).pow(mod >> (k + 1));
            }
        return min(x.x, mod - x.x);
    }
    static constexpr long long mod_primitive_root(){ // kiszámítja a moduló egy primitív gyökét
        long long primes[64] = {}; int size = 0; long long p = 2, m = mod-1;
        while(p*p <= m) { if(m % p == 0) primes[size++] = p; while(m % p == 0) m /= p; ++p; } if(m > 1) primes[size++] = m;
        for(long long i = 2; i < mod; i++) { bool ok = true; for(int j = 0; j < size; j++) ok = ok && mint(i).pow((mod - 1) / primes[j]).x != 1; if(ok) return i; }
        return -1;
    }
};

mint w[N];
mint invi[N + 1];
__attribute__((constructor)) void init() {
    invi[1] = w[N / 2] = 1;
    constexpr mint g = mint(mint::mod_primitive_root()).pow(mod / N);
    for (int i = N / 2 + 1; i < N; ++i) w[i] = w[i - 1] * g;
    for (int i = N / 2 - 1; i > 0; --i) w[i] = w[i << 1];
    for (int i = 2; i <= N; i++) invi[i] = invi[mod % i] * (mint() - mint(mod / i));
}
```

```cpp
void dft(mint f[], int n) { // n kettő hatvány
    for (int k = n / 2; k; k /= 2)
        for (int i = 0; i < n; i += k + k)
            for (int j = 0; j < k; ++j) {
                mint x = f[i + j]; mint y = f[i + j + k]; f[i + j] = x + y; f[i + j + k] = (x - y) * w[k + j];
            }
}
void ift(mint f[], int n) { // n kettő hatvány
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += k + k)
            for (int j = 0; j < k; ++j) {
                mint x = f[i + j]; mint y = f[i + j + k] * w[k + j]; f[i + j] = x + y; f[i + j + k] = x - y;
            }
    mint inv = mod - (mod - 1) / n;
    std::reverse(f + 1, f + n);
    for (int i = 0; i < n; ++i) f[i] *= inv;
}
struct poly : std::vector<mint> { using std::vector<mint>::vector;
    poly &add(const poly &o) { if (size() < o.size()) resize(o.size()); for (int i = 0; i < o.size(); ++i) (*this)[i] += o[i]; return *this; }
    poly &sub(const poly &o) { if (size() < o.size()) resize(o.size()); for (int i = 0; i < o.size(); ++i) (*this)[i] -= o[i]; return *this; }
    poly &mul(const poly &o) { if (size() < o.size()) resize(o.size()); for (int i = 0; i < o.size(); ++i) (*this)[i] *= o[i]; return *this; }
    poly &mul(const mint &o) { for (mint &i: *this) i *= o; return *this; }
    poly &derivative() { for(int i = 0; i < (int)size() - 1; i++) (*this)[i] = (*this)[i + 1] * mint(i + 1); pop_back(); return *this; }
    poly &integral()   { resize(size()+1); for(int i = (int)size() - 1; i > 0; i--) (*this)[i] = (*this)[i-1] * invi[i]; (*this)[0] = mint(); return *this; } // lehet overflow invi mérete N !!!

    poly copy() const { return *this; }
    poly &resize(auto sz) { return vector::resize(sz), *this; }
    poly &dft(int n) { return ::dft(resize(n).data(), n), *this; }
    poly &ift(int n) { return ::ift(resize(n).data(), n), *this; }
    poly &ins(int sz) { return insert(begin(), sz, mint()), *this; }
    poly &del(int sz) { return erase(begin(), begin() + sz), *this; }
    poly &reverse() { return std::reverse(begin(), end()), *this;}
    poly pre(int sz) const { return sz < size() ? poly(begin(), begin() + sz) : copy(); }
    poly &reduce() { while(!empty() && back().x == 0) pop_back(); return *this; }

    poly conv(const poly &o){
        int n = __bit_ceil(size() + o.size() - 1);
        return copy().dft(n).mul(o.copy().dft(n)).ift(n).resize(size() + o.size() - 1);
    }
    poly inv() const {
        if (front().x == 0) return {};
        int m = size();
        poly inv = {front().inv()};
        for (int k = 1; k < m; k *= 2) {
            int n = k * 2; poly a = inv.copy().dft(n), b = pre(n).dft(n);
            inv.sub(a.copy().mul(b).ift(n).del(k).dft(n).mul(a).ift(n).resize(k).ins(k));
        }
        return inv.resize(m);
    }
    poly log() const{ // res[0] = 0
        int n = __bit_ceil(size() * 2 - 1);
        return copy().derivative().dft(n).mul(inv().dft(n)).ift(n).integral().resize(size());
    }
    poly exp() const { // p[0] == 0, különben nem valid az ereémény
        if (front().x != 0) return {};
        int m = size();
        poly e = {1};
        for (int k = 1; k < m; k *= 2) {
            int n = k * 2;
            poly elog = e.resize(n).log(); e.dft(n*2);
            e.add(pre(n).sub(elog).dft(n*2).mul(e)).ift(n*2).resize(n);
        }
        return e.resize(m);
    }
    poly pow(auto k) const { // k: int, long long
        if(k == 0) return poly{1}.resize(size());
        int j = 0;
        while(j < size() && (*this)[j].x == 0) ++j;
        if(j == size()) return poly{0}.resize(size());
        mint c = (*this)[j];
        return copy().del(j).mul(c.inv()).log().mul(mint(k % mod)).exp().mul(c.pow(k % (mod - 1))).ins(j > size() / k ? (long long)size() : j * k).resize(size());
    }
    poly sqrt() const { // ha nem létezik akkor az eredmény: {}
        int j = 0;
        while(j < size() && (*this)[j].x == 0) ++j;
        if(j == size()) return poly{0}.resize(size());
        mint c = (*this)[j].sqrt();
        if(c.x == 0 || j % 2 != 0) return {};
        return copy().del(j).mul((*this)[j].inv()).resize(size() - j / 2).pow(mint(2).inv().x).mul(c).ins(j / 2);
    }
    poly div(const poly& o) {
        poly a = copy().reduce().reverse(), b = o.copy().reduce().reverse();
        int m = a.size() - b.size() + 1;
        if(a.empty() || b.empty() || a.size() < b.size()) return b.empty() ? poly{} : poly{0};
```

```
        return a.conv(b.resize(a.size()).inv()).resize(m).reverse();
    }
    poly rem(const poly& o) {
        return copy().sub(div(o).conv(o));
    }
};

/*
poly: ugyanúgy működik, mint az std::vector
műveletek: +, -, * pontoknékt, derivátl, integrál, (resize, dft, ift, ins, del, reverse)
constans műveletek: conv, inv, log, exp, pow, sqrt, div, rem
*/
```

15 String algorithms I.

```
vector<int> prefix_function(string s) {
    // prefix function ABAAB -> (0, 0, 1, 1, 2)
    int n=s.size();
    vector<int> ans(n, 0);
    for (int i=1; i<n; i++) {
        int ert=ans[i-1];
        while (ert && s[i]!=s[ert]) {
            ert=ans[ert-1];
        }
        if (s[i]==s[ert]) {
            ert++;
        }
        ans[i]=ert;
    }
    return ans;
}

vector<int> z_function(string s) {
    // z function ABAAB -> (0, 0, 1, 2, 0);
    int n=s.size();
    vector<int> ans(n, 0);
    int l=0, r=0;
    int lepes=0;
    for (int i=1; i<n; i++) {
        int len=0;
        if (i<r) {
            len=min(r-i, ans[i-l]);
        }
        while (i+len<n && s[i+len]==s[len]) {
            lepes++;
            len++;
        }
        ans[i]=len;
        if (i+len>r) {
            l=i, r=i+len;
        }
    }
    return ans;
}


vector<int> find_periods(string s) {
    // milyen hosszu prefix ismetlesevel kaphato meg s
    // ABABA -> (2, 4, 5)
    // a teljes periodushoz (n\%i==0) feltetel kell
    int n=s.size();
    vector<int> z=z_function(s);
    z[0]=n;
    vector<int> ans;
    for (int i=1; i<n; i++) {
        if (i+z[i]==n) {
            ans.push_back(i);
        }
    }
    ans.push_back(n);
    return ans;
}

int min_rotation(string s) {
    // mennyivel kell elcsusztatni ABAAB -> 2
    int n=s.size();
    s+=s;
    int i=0, pos=0;
    while (i < n) {
        int k=i, j=i+1;
        pos=i;
        while (j<2*n && s[k]<=s[j]) {
            if (s[k]<s[j]) k=i;
            else k++;
            j++;
        }
        while (i<=k) {
            i+=j-k;
        }
    }
    return pos;
    // return s.substr(pos, n);
}
```

16 String algorithms II.

```
vector<int> manacher(string s) {
    // egy 2*n-1 hosszu vektort ad vissza, mindig az i. majd utana az i. es i+1. kozott indulo leghosszabb
    palindromot
    // ABAABB -> (1, 0, 3, 0, 1, 4, 1, 0, 1, 2, 1)
    // akar a d1 (paratlan) es d2 (paros) vektor is hasznos lehet
    int n=s.size();
    vector<int> d1(n, 0), d2(n, 0);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if (i + k > r) {
            l = i - k;
            r = i + k;
        }
    }
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if (i + k > r) {
            l = i - k - 1;
            r = i + k ;
        }
    }
    vector<int> ans;
    for (int i=0; i<n; i++) {
        if (i) ans.push_back(2*d2[i]);
        ans.push_back(2*d1[i]-1);
    }
    return ans;
}

vector<int> sort_cyclic_shifts(string const& s) {
    // ABAAB -> (2, 0, 3, 1, 4)
    int n = s.size();
    const int alphabet = 256;

    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }

    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}

vector<int> suffix_array_construction(string s) {
    // a suffixeket rendezi
    "\$" mindennel kisebb
    // ABAAB -> (2, 3, 0, 4, 1)
    s += "\$";
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
```

```
vector<int> lcp_construction(string const& s, vector<int> const& p) {
    // csak a masikkal egyutt mukodik (ket suffix arrayben szomszedos suffix lcp-je)
    // vector<int> res=lcp_construction(s, suffix_array_construction(s));
    // ABAAB -> (1, 2, 0, 1)
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}
```

17 Treap

```
// TREAP
mt19937 rnd(42123); // mt19937_64 ha long long kell
struct node { // az upd() és push()-t kell implementálni | upd()-et a konstruktor is hívja
        int val, w, size; // val érték (cserélhető), w súly, size a részfa mérete
    node *l, *r; // bal, jobb gyerek
        node(int c) : val(c), w(rnd()), size(1), l(NULL), r(NULL) { upd(); }
    ~node() { delete l; delete r; }
    inline void upd() {} // update az l, r-ből
    inline void push() {} // push l, r-be
} *treap;
int size(node *p) { return p ? p->size : 0; }
void split(node *p, node *&l, node *&r, int val) { // l < val | val <= r
        if(!p) { l = r = NULL; return; }
        p->push();
    if(size(p->l) < val) split(p->r, p->r, r, val - size(p->l) - 1), l = p;
        else split(p->l, l, p->l, val), r = p;
        p->size = 1 + size(p->l) + size(p->r); p->upd();
}
void merge(node *&p, node *l, node *r) {
        if(!l || !r) { p = l ? l : r; return; }
        if (l->w < r->w) l->push(), merge(l->r, l->r, r), p = l;
        else r->push(), merge(r->l, l, r->l), p = r;
        p->size = 1 + size(p->l) + size(p->r); p->upd();
}
```

18 Link Cut Tree

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

using ll = long long;
typedef struct snode *sn;
struct snode {          //////// VARIABLES
        sn p, c[2];         // parent, children
        bool flip = 0;      // subtree flipped or not
        int size;           // # splay tree csúcs, aktuálisban
        ll val;             // value in node
        snode(int _val) : val(_val) { p = c[0] = c[1] = NULL; upd(); }
        friend int get_size(sn x) { return x ? x->size : 0; }
        void prop() {  // lazy prop
                if (!flip) return;
                swap(c[0], c[1]);
                flip = 0;
                for (int i = 0; i < 2; i++)
                        if (c[i]) c[i]->flip ^= 1;
        }
        void upd() {  // recalc vals
                for (int i = 0; i < 2; i++) if (c[i]) c[i]->prop();
                size = 1 + get_size(c[0]) + get_size(c[1]);
                // virtuális részva adatok használata
        }
        void vupd(){}
        //////// SPLAY TREE OPERATIONS
        int dir() {
                if (!p) return -2;
                for (int i = 0; i < 2; i++) if (p->c[i] == this) return i;
                return -1;  // p is path-parent pointer
        }  // -> not in current splay tree
        bool is_root() { return dir() < 0; }
        friend void set_link(sn x, sn y, int d) { if (y) y->p = x; if (d >= 0) x->c[d] = y; }
        void rot() {  // assume p and p->p propagated
```

```
        assert(!is_root());
        int x = dir(); sn pa = p;
        set_link(pa->p, this, pa->dir()); set_link(pa, c[x ^ 1], x); set_link(this, pa, x ^ 1);
        pa->upd();
    }
    void splay() {
        while (!is_root() && !p->is_root()) {
            p->p->prop(), p->prop(), prop();
            dir() == p->dir() ? p->rot() : rot(); rot();
        }
        if (!is_root()) p->prop(), prop(), rot();
        prop(); upd();
    }
    sn fbo(int b) {  // find by order
        prop(); int z = get_size(c[0]);  // of splay tree
        if (b == z) { splay(); return this; }
        return b < z ? c[0]->fbo(b) : c[1]->fbo(b - z - 1);
    }
    //////// BASE OPERATIONS
    void access() {  // bring this to top of tree, propagate
        for (sn v = this, pre = NULL; v; v = v->p) {
            v->splay();  // now switch virtual children
            if (pre) vupd(); // pre törlése (mostantól rendes gyerek)
            if (v->c[1]) vupd(); // c[1] hozzáadása (mostantól virtuális gyerek)
            v->c[1] = pre; v->upd(); pre = v;
        }
        splay();
        assert(!c[1]);  // right subtree is empty
    }
    void make_root() { // ez lesz a fa gyökere
        access(); flip ^= 1; access();
        assert(!c[0] && !c[1]);
    }
    //////// QUERIES
    friend sn lca(sn x, sn y) {
        if (x == y) return x;
        x->access(), y->access();
        if (!x->p) return NULL;
        x->splay();
        return x->p ?: x;  // y was below x in latter case
    }  // access at y did not affect x -> not connected
    friend bool connected(sn x, sn y) { return lca(x, y); }
    int dist_root() { access(); return get_size(c[0]); } // # nodes above
    sn get_root() { // get root of LCT component
        access(); sn a = this;
        while (a->c[0]) a = a->c[0], a->prop();
        a->access();
        return a;
    }
    sn get_par(int b) {  // get b-th parent on path to root | can also get min, max on path to root, etc
        access(); b = get_size(c[0]) - b;
        assert(b >= 0);
        return fbo(b);
    }
    //////// MODIFICATIONS
    void set(ll v) { access(); val = v; upd(); } // changes value
    friend void link(sn x, sn y, bool force = 0) { // ha force: x - y él minden esetben | ha nem force: akkor y-nak gyökérnek kell lenni
        assert(!connected(x, y));
        if (force) y->make_root();  // make x par of y
        else { y->access(); assert(!y->c[0]); }
        x->access(); set_link(y, x, 0); y->upd();
    }
    friend void cut(sn y) {  // cut y from its parent | ha nincs RTE
        y->access();
        assert(y->c[0]);
        y->c[0]->p = y->c[0] = NULL;
        y->upd();
    }
    friend void cut(sn x, sn y) {  // if x, y adj in tree
        x->make_root(); y->access();
        assert(y->c[0] == x && !x->c[0] && !x->c[1]);
        cut(y);
    }
};
/*
Link-cut tree, műveletek: link, cut, set | lca, connected, dist_root, get_root, get_par
Út querry-hez a get_par-hoz hasonló implementáció kell + fbo implementáció, ha nem a teljes út kell. | Ha a, b út kell: 1. make_root(a), 2. query b-től gyökérig
Részfa adatokhoz a vupd()-et kell módosítani, (az upd()-ben is bele kell írni) | fontos kell legyen a műveletnek inverze
*/
```

19 Math

```
int gcd(int a, int b, int& x, int& y) {
    // lehet hogy long long kell
    // x-et es y-t beallitja ugy, hogy a*x+b*y=gcd(a, b) teljesul
```

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