ICPC Notebook

pedroteosousa

Contents

1 Geometry

1.1 Basic

```
struct point {
    coord x, y;
    point(): x(0), y(0) {}
    point(coord a, coord b): x(a), y(b) {}

    inline point operator+ (point o) { return {x + o.x, y + o.y}; }
    inline point operator- (point o) { return {x - o.x, y - o.y}; }
    inline point operator* (coord o) { return {x * o, y * o}; }
    inline point operator/ (coord o) { return {x / o, y / o}; }

    inline coord operator* (point o) { return x * o.x + y * o.y; }
    inline coord operator^ (point o) { return x * o.y - y * o.x; }

    inline bool operator< (point o) { return make_pair(x, y) < make_pair(o.x, o.y); }

    inline bool operator== (point o) { return abs(x-o.x) < eps && abs(y-o.y) < eps; }

    // cw angle
    inline double ang(point o) {
        point p = *this;
        return atan2(-(p ^ o), p * o);
    }

    inline coord sqr() { return x * x + y * y; }
    inline double len() { return sqrt(sqr()); }

    // rotate cw
    inline point rot90() { return {y, -x}; }
    inline point rotate(double a) { return {cos(a)*x + sin(a)*y, -sin(a)*x + cos(a)*y}; }

    inline int ccw(point o) { coord a = (*this) ^ o; return (eps < a) - (a < -eps); }
    inline int dir(point o) { coord a = (*this) * o; return (eps < a) - (a < -eps); }

    bool in_seg(point a, point b) {
        point p = *this;
        return (p-a).ccw(b-a) == 0 && (p-a).dir(p-b) <= 0;
    }
    double dist_line(point a, point b) {
        point p = *this;
        return (b-a).sqr() <= eps ? (p-a).sqr() : double(abs((a-p) ^ (b-p))) / (b-a).len();
    }
    double dist_seg(point a, point b) {
        point p = *this;
        return (p-a).dir(p-b) <= 0 ? dist_line(a, b) : min((p-a).len(), (p-b).len());
    }
};

struct line {
    point p; coord c;
    line() {}
    line(point s, point e): p((s-e).rot90()), c(p*s) {}
    point inter(line o) {
        if (p.ccw(o.p) == 0) throw 1;
        coord d = (p ^ o.p);
        return point((c * o.p.y - p.y * o.c) / d, (o.c * p.x - o.p.x * c) / d);
    }
};

bool inter_seg(point a, point b, point c, point d) {
    if (a.in_seg(c, d) || b.in_seg(c, d) || c.in_seg(a, b) || d.in_seg(a, b)) return true;
    return ((c-a).ccw(b-a) * (d-a).ccw(b-a) == -1 && (a-c).ccw(d-c) * (b-c).ccw(d-c) == -1);
}
```

1.2 Convex Hull

```
double side(point a, point b, point c) {
    return (a^b) + (b^c) + (c^a);
}

vector<point> convex_hull(vector<point> p) {
    int n = p.size(), k = 0;
    if (n == 1) return p;
    vector<point> hull(2*n);

    sort(p.begin(), p.end());

    for(int i=0; i<n; i++) {
        // use <= when including collinear points
        while(k>=2 && (side(hull[k-2], hull[k-1], p[i]) < 0))
            k--;
        hull[k++] = p[i];
    }

    for(int i=n-2,t=k+1; i>=0; i--) {
        while(k>=t && (side(hull[k-2], hull[k-1], p[i]) < 0))
            k--;
        hull[k++] = p[i];
    }
    hull.resize(k-1);
    return hull;
}
```

2 Graph Algorithms

2.1 Tarjan

```
const int inf = 1791791791;

vector<int> adj[N];

// time complexity: O(V+E)
stack<int> ts;
int tme = 0, ncomp = 0, low[N], seen[N];
int comp[N]; // nodes in the same scc have the same color
int scc_dfs(int n) {
    seen[n] = low[n] = ++tme;
    ts.push(n);
    for (auto a : adj[n]) {
        if (seen[a] == 0)
            scc_dfs(a);
        low[n] = min(low[n], low[a]);
    }
    if (low[n] == seen[n]) {
        int node;
        do {
            node = ts.top(); ts.pop();
            comp[node] = ncomp;
            low[node] = inf;
        } while (n != node && ts.size());
        ncomp++;
    }
    return low[n];
}
```

2.2 Lowest Common Ancestor

```
const int N = 1e6 + 5;
const int L = 20;

vector<int> adj[N];
int prof[N], p[N][L+5];

void dfs(int v, int h = 1) {
    prof[v] = h;
    if (h == 1) p[v][0] = v;
    for (auto u : adj[v])
        if (prof[u] == 0) {
            p[u][0] = v;
            dfs(u, h+1);
        }
}

void init(int n) {
    for (int i = 1; i <= L; i++)
        for (int j = 1; j < n; j++)
            p[j][i] = p[p[j][i-1]][i-1];
}

int lca(int u, int v) {
    if (prof[u] < prof[v]) swap(u, v);
    for (int i = L; i >= 0; i--)
        if (prof[p[u][i]] >= prof[v])
            u = p[u][i];
    for (int i = L; i >= 0; i--)
        if (p[u][i] != p[v][i]) {
            u = p[u][i];
            v = p[v][i];
        }
    while (u != v) {
        u = p[u][0];
        v = p[v][0];
    }
    return u;
}
```

2.3 Centroid

```
vector<int> adj[N], centroid[N];
int sze[N];

int dfs(int v, int p = 0) {
    sze[v] = 1;
    for (int u: adj[v])
        if (u != p && sze[u] != -1) sze[v] += dfs(u, v);
    return sze[v];
}

// returns root of centroid tree
int build(int v) {
    int n = dfs(v, v);
    int w = v;
    do {
        v = w;
        for (int u: adj[v])
            if (sze[u] != -1 && sze[u] < sze[v] && 2 * sze[u] >= n)
                w = u;
    } while (v != w);
    sze[v] = -1;
    for (int u: adj[v])
        if (sze[u] != -1)
            centroid[v].push_back(build(u));
    return v;
}
```

3 Flow

3.1 Dinic's Algorithm

```
struct dinic {
    struct edge {
        int from, to;
        ll c, f;
    };
    vector<edge> edges;
    vector<int> adj[N];

    void addEdge(int i, int j, ll c) {
        edges.push_back({i, j, c, 0}); adj[i].push_back(edges.size() - 1);
        edges.push_back({j, i, 0, 0}); adj[j].push_back(edges.size() - 1);
    }

    int turn, seen[N], dist[N], st[N];
    bool bfs (int s, int t) {
        seen[t] = ++turn;
        dist[t] = 0;
        queue<int> q({t});
        while (q.size()) {
            int u = q.front(); q.pop();
            st[u] = 0;
            for (auto e : adj[u]) {
                int v = edges[e].to;
                if (seen[v] != turn && edges[e^1].c != edges[e^1].f) {
                    seen[v] = turn;
                    dist[v] = dist[u] + 1;
                    q.push(v);
                }
            }
        }
        return seen[s] == turn;
    }

    ll dfs(int s, int t, ll f) {
        if (s == t || f == 0)
            return f;
        for (int &i = st[s]; i < adj[s].size(); i++) {
            int e = adj[s][i], v = edges[e].to;
            if (seen[v] == turn && dist[v] + 1 == dist[s] && edges[e].c > edges[e].f) {
                if (ll nf = dfs(v, t, min(f, edges[e].c - edges[e].f))) {
                    edges[e].f += nf;
                    edges[e^1].f -= nf;
                    return nf;
                }
            }
        }
        return 0ll;
    }

    ll max_flow(int s, int t) {
        ll resp = 0ll;
        while (bfs(s, t))
            while (ll val = dfs(s, t, inf))
                resp += val;
        return resp;
    }
};
```

3.2 Min Cost

```
typedef long long ll;
const ll inf = 1e12;

struct min_cost {
    struct edge {
        int from, to;
        ll cp, fl, cs;
    };
    vector<edge> edges;
    vector<int> adj[N];

    void addEdge(int i, int j, ll cp, ll cs) {
        edges.push_back({i, j, cp, 0, cs}); adj[i].push_back(edges.size() - 1);
        edges.push_back({j, i, 0, 0, -cs}); adj[j].push_back(edges.size() - 1);
    }

    ll seen[N], dist[N], pai[N], cost, flow;
    int turn;
    ll spfa(int s, int t) {
        turn++;
        queue<int> q; q.push(s);
        for (int i = 0; i < N; i++) dist[i] = inf;
        dist[s] = 0;
        seen[s] = turn;
        while (q.size()) {
            int u = q.front(); q.pop();
            seen[u] = 0;
            for (auto e : adj[u]) {
                int v = edges[e].to;
                if (edges[e].cp > edges[e].fl && dist[u] + edges[e].cs < dist[v]) {
                    dist[v] = dist[u] + edges[e].cs;
                    pai[v] = e ^ 1;
                    if (seen[v] < turn) {
                        seen[v] = turn;
                        q.push(v);
                    }
                }
            }
        }
        if (dist[t] == inf) return 0;
        ll nfl = inf;
        for (int u = t; u != s; u = edges[pai[u]].to)
            nfl = min(nfl, edges[pai[u] ^ 1].cp - edges[pai[u] ^ 1].fl);
        cost += dist[t] * nfl;
        for (int u = t; u != s; u = edges[pai[u]].to) {
            edges[pai[u]].fl -= nfl;
            edges[pai[u] ^ 1].fl += nfl;
        }
        return nfl;
    }

    void mncost(int s, int t) {
        cost = flow = 0;
        while (ll fl = spfa(s, t))
            flow += fl;
    }
};
```

4 Data Structures

4.1 Trie

```
struct trie {
    struct node {
        int to[A], freq, end;
    };
    struct node t[N];
    int sz = 0;
    int offset = 'a';

    // init trie
    void init() {
        memset(t, 0, sizeof(struct node));
    }

    // insert string
    void insert(char *s, int p = 0) {
        t[p].freq++;
        if (*s == 0) {
            t[p].end++;
            return;
        }
        if (t[p].to[*s - offset] == 0)
            t[p].to[*s - offset] = ++sz;
        insert(s+1, t[p].to[*s - offset]);
    }

    // check if string is on trie
    int find(char *s, int p = 0) {
        if (*s == 0)
            return t[p].end;
        if (t[p].to[*s - offset] == 0)
            return false;
        return find(s+1, t[p].to[*s - offset]);
    }

    // count the number of strings that have this prefix
    int count(char *s, int p = 0) {
        if (*s == 0)
            return t[p].freq;
        if (t[p].to[*s - offset] == 0)
            return 0;
        return count(s+1, t[p].to[*s - offset]);
    }

    // erase a string
    int erase(char *s, int p = 0) {
        if (*s == 0 && t[p].end) {
            --t[p].end;
            return --t[p].freq;
        }
        if ((*s == 0 && t[p].end == 0) || t[p].to[*s - offset] == 0)
            return -1;
        int count = erase(s+1, t[p].to[*s - offset]);
        if (count == 0)
            t[p].to[*s - offset] = 0;
        if (count == -1)
            return -1;
        return --t[p].freq;
    }
};
```

4.2 Binary Indexed Tree

```
int b[N];

int update(int p, int val, int n) {
    for(;p < n; p += p & -p) b[p] += val;
}

int getsum(int p) {
    int sum = 0;
    for(; p != 0; p -= p & -p) {
        sum += b[p];
    }
    return sum;
}
```

4.3 Lazy Segment Tree

```
typedef long long ll;

const ll N = 1e5 + 5;
const ll inf = 1791791791;

struct seg_tree {
    ll seg[4*N];
    ll lazy[4*N];

    seg_tree() {
        memset(seg, 0, sizeof(seg));
        memset(lazy, 0, sizeof(lazy));
    }

    void do_lazy(ll root, ll left, ll right) {
        seg[root] += lazy[root];
        if (left != right) {
            lazy[2*root+1] += lazy[root];
            lazy[2*root+2] += lazy[root];
        }
        lazy[root] = 0;
    }

    // sum update
    ll update(ll l, ll r, ll val, ll left = 0, ll right = N-1, ll root = 0) {
        do_lazy(root, left, right);
        if (r < left || l > right) return seg[root];
        if (left >= l && right <= r) {
            lazy[root] += val;
            do_lazy(root, left, right);
            return seg[root];
        }
        ll update_left = update(l, r, val, left, (left+right)/2, 2*root+1);
        ll update_right = update(l, r, val, (left+right)/2+1, right, 2*root+2);
        return seg[root] = min(update_left, update_right);
    }

    ll query(ll l, ll r, ll left = 0, ll right = N-1, int root = 0) {
        do_lazy(root, left, right);
        if (r < left || l > right)
```

```
            return inf;
        if (left >= l && right <= r) return seg[root];
        ll query_left = query(l, r, left, (left+right)/2, 2*root+1);
        ll query_right = query(l, r, (left+right)/2+1, right, 2*root+2);
        return min(query_left, query_right);
    }
};
```

4.4 Union Find

```
int p[N], w[N];

void init() {
    for (int i = 0; i < N; i++)
        w[p[i] = i] = 1;
}

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

void join(int a, int b) {
    if ((a = find(a)) == (b = find(b))) return;
    if (w[a] < w[b]) swap(a, b);
    w[a] += w[b];
    p[b] = a;
}
```

5 Mathematics

5.1 Matrix

```
template <int n> struct matrix {
    long long mat[n][n];
    matrix () {
        memset (mat, 0, sizeof (mat));
    }
    matrix (long long temp[n][n]) {
        memcpy (mat, temp, sizeof (mat));
    }
    void identity() {
        memset (mat, 0, sizeof (mat));
        for (int i=0;i<n;i++)
            mat[i][i] = 1;
    }
    matrix<n> mul (const matrix<n> &a, long long m) const {
        matrix<n> temp;
        for (int i=0; i<n; i++)
            for (int j=0; j<n; j++)
                for (int k=0; k<n; k++) {
                    temp.mat[i][j] += (mat[i][k]*a.mat[k][j])%m;
                    temp.mat[i][j] %= m;
                }
        return temp;
    }
    matrix<n> operator% (long long m) {
        matrix<n> temp(mat);
        for (int i=0; i<n; i++)
            for (int j=0; j<n; j++)
                temp.mat[i][j] %= m;
        return temp;
    }
    matrix<n> pow(long long e, long long m) {
        matrix<n> temp;
        if (e == 0) {
            temp.identity();
            return temp%m;
        }
        if (e == 1) {
            memcpy (temp.mat, mat, sizeof (temp.mat));
```

```
                return temp%m;
            }
            temp = pow(e/2, m);
            if (e % 2 == 0)
                return (temp.mul(temp, m))%m;
            else
                return (((temp.mul(temp, m))%m)*pow(1, m))%m;
        }
};
```

5.2 Fast Fourier Transform

```
typedef complex<double> cpx;
const double pi = acos(-1.0);
// DFT if type = 1, IDFT if type = -1
// If you are multiplying, remember to let EACH vector with n >= sum of degrees of both polys
// n is required to be a power of 2
void FFT(vector<cpx> &v, vector<cpx> &ans, int n, int type, int p[]) { // p[n]
    assert(!(n & (n - 1))); int i, sz, o; p[0] = 0;
    for(i = 1; i < n; i++) p[i] = (p[i >> 1] >> 1) | ((i & 1)? (n >> 1) : 0);
    for(i = 0; i < n; i++) ans[i] = v[p[i]];
    for(sz = 1; sz < n; sz <<= 1) {
        const cpx wn(cos(type * pi / sz), sin(type * pi / sz));
        for(o = 0; o < n; o += (sz << 1)) {
            cpx w = 1;
            for(i = 0; i < sz; i++) {
                const cpx u = ans[o + i], t = w * ans[o + sz + i];
                ans[o + i] = u + t;
                ans[o + i + sz] = u - t;
                w *= wn;
            }
        }
    }
    if(type == -1) for(i = 0; i < n; i++) ans[i] /= n;
}
```

5.3 Extended Euclidean Algorithm

```
// x * a + y * b = gcd(a, b)
ll ext(ll a, ll b, ll &x, ll &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    ll x1, y1;
    ll gcd = ext(b%a, a, x1, y1);

    x = y1 - (b/a)*x1;
    y = x1;

    return gcd;
}
```

5.4 Rabin-Miller Primality Test

```
long long llrand(long long mn, long long mx) {
    long long p = rand();
    p <<= 32ll;
    p += rand();
    return p%(mx-mn+1ll)+mn;
}

long long mul_mod(long long a, long long b, long long m) {
    long long x = 0, y = a%m;
    while (b) {
        if (b % 2)
            x = (x+y)%m;
        y = (2*y)%m;
        b >>= 1;
    }
    return x%m;
}

long long exp_mod(long long e, long long n, long long m) {
    if (n == 0)
        return 1ll;
    long long temp = exp_mod(e, n/2, m);
    if (n & 1)
        return mul_mod(mul_mod(temp, temp, m), e, m);
    else
        return mul_mod(temp, temp, m);
}

// complexity: O(t*log2^3(p))
bool isProbablyPrime(long long p, long long t=64) {
    if (p <= 1) return false;
    if (p <= 3) return true;
    srand(time(NULL));
    long long r = 0, d = p-1;
    while (d % 2 == 0) {
        r++;
        d >>= 1;
    }
    while (t--) {
        long long a = llrand(2, p-2);
        a = exp_mod(a, d, p);
        if (a == 1 || a == p-1) continue;
        for (int i=0; i<r-1; i++) {
            a = mul_mod(a, a, p);
            if (a == 1) return false;
            if (a == p-1) break;
        }
        if (a != p-1) return false;
    }
    return true;
}
```

6 Strings

6.1 Z function

```
int z[N];

void Z(string s) {
    int n = s.size();
    int m = -1;
    for (int i = 1; i < n; i++) {
        z[i] = 0;
        if (m != -1 && m + z[m] >= i)
            z[i] = min(m + z[m] - i, z[i-m]);
        while (i + z[i] < n && s[i+z[i]] == s[z[i]])
            z[i]++;
        if (m == -1 || i + z[i] > m + z[m])
            m = i;
    }
}
```

6.2 Knuth–Morris–Pratt Algorithm

```
int kmp[N];

void build(string p) {
    int n = p.size(), k = -1;
    kmp[0] = k;
    for (int i = 1; i < n+1; i++) {
        while (k >= 0 && p[k] != p[i-1]) k = kmp[k];
        kmp[i] = ++k;
    }
}
```

```
vector<int> match(string p, string s) {
    int n = s.size(), m = p.size(), j = 0;
    vector<int> matches;
    for (int i = 1; i < n+1; i++) {
        while (j >= 0 && p[j] != s[i-1]) j = kmp[j];
        if (++j == m) {
            matches.push_back(i-j+1);
            j = kmp[j];
        }
    }
    return matches;
}
```

7 Miscellaneous

7.1 vim settings

```
set ai si noet ts=4 sw=4 sta sm nu rnu
inoremap <NL> <ESC>o
nnoremap <NL> o
inoremap <C-up> <C-o>:m-2<CR>
inoremap <C-down> <C-o>:m+1<CR>
nnoremap <C-up> :m-2<CR>
nnoremap <C-down> :m+1<CR>
vnoremap <C-up> :m-2<CR>gv
vnoremap <C-down> :m'>+1<CR>gv
syntax on
colors evening
highlight Normal ctermbg=none "No background
highlight nonText ctermbg=none
```