Algorithm Templates Collection for HEOI2018

Kvar_ispw17

April 4, 2018

Life sucks, you're gonna love it.

Contents

1 Data Structure

1.1 Splay

```
#define ls tr[u].ch[0]
#define rs tr[u].ch[1]
const int nil = 0;
int root, tot;

struct node {
    int v, siz, ch[2], fa, rev, sum, tag;
    node() {
        rev = tag = v = sum = 0;
        fa = ch[0] = ch[1] = 0;
        siz = 1;
    }
} tr[N];

void up(int u) {
    tr[u].siz = 1;
    tr[u].sum = tr[u].v;
    if (ls) {
        tr[u].sum += tr[ls].sum;
        tr[u].siz += tr[ls].siz;
    }
    if (rs) {
        tr[u].sum += tr[rs].sum;
        tr[u].siz += tr[rs].siz;
    }
}

void down(int u) {
    if (tr[u].rev) {
        if (ls) tr[ls].rev ^= 1;
        if (rs) tr[rs].rev ^= 1;
        swap(ls, rs);
        tr[u].rev = 0;
    }
    if (tr[u].tag) {
        tr[ls].tag += tr[u].tag;
```

```
        tr[rs].tag += tr[u].tag;
        tr[u].sum += tr[u].tag * tr[u].siz;
        tr[u].v += tr[u].tag;
        tr[u].tag = 0;
    }
}

int son(int u) {
    return u == tr[tr[u].fa].ch[1];
}

int build(int &u, int x[], int l, int r) {
    if (l > r) return 0;
    u = ++tot; if (l == r) { tr[u].v = tr[u].sum = x[l]; return u; }
    int mid = ceil((l + r) / 2.);
    tr[u].v = x[mid];
    ls = build(ls, x, l, mid - 1); if (ls) tr[ls].fa = u;
    rs = build(rs, x, mid + 1, r); if (rs) tr[rs].fa = u;
    up(u); return u;
}

void rotate(int u) {
    int f = tr[u].fa, pf = tr[f].fa; down(f);
    down(u); int d = son(u), pd = pf ? son(f) : 0;
    tr[f].ch[d] = tr[u].ch[d ^ 1];
    if (tr[u].ch[d ^ 1]) tr[tr[u].ch[d ^ 1]].fa = f;
    tr[u].ch[d ^ 1] = f; tr[f].fa = u;
    pf ? tr[pf].ch[pd] = u : root = u;
    tr[u].fa = pf, up(f), up(u);
}

int search(int u, int k) {
    down(u); int siz = 0;
    if (ls) siz = tr[ls].siz;
    if (k < siz + 1) return search(ls, k);
    else if (k > siz + 1) return search(rs, k - (siz + 1));
    else return u;
}

int at(int k) { return search(root, k + 1); }
```

```
void splay(int u, int t) {
    while (tr[u].fa != t) {
        if (tr[tr[u].fa].fa != t)
            rotate(son(u) == son(tr[u].fa) ? tr[u].fa : u);
        rotate(u);
    }
}

void insert(int u, int k, int x[], int l, int r) {
    int t = build(t, x, l, r), p = at(k), q;
    splay(p, nil), q = tr[p].ch[1];
    tr[p].ch[1] = t, tr[t].fa = p;
    splay(p = at(k + tr[t].siz), nil);
    tr[p].ch[1] = q, tr[q].fa = p;
}

void erase(int u, int l, int r) {
    int p = at(l - 1), q = at(r + 1);
    splay(p, nil), splay(q, p);
    tr[q].ch[0] = nil;
}

void reverse(int u, int l, int r) {
    int p = at(l - 1), q = at(r + 1);
    splay(p, nil), splay(q, p);
    tr[tr[q].ch[0]].rev ^= 1;
}

void add(int l, int r, int val) {
    int p = at(l - 1), q = at(r + 1);
    splay(p, nil), splay(q, p);
    int w = tr[q].ch[0];
    tr[w].sum += tr[w].siz * val;
    tr[w].v += val;
    if (tr[w].ch[0]) tr[tr[w].ch[0]].tag += val;
    if (tr[w].ch[1]) tr[tr[w].ch[1]].tag += val;
}

int query(int u, int l, int r) {
    int p = at(l - 1), q = at(r + 1);
```

```
    splay(p, nil), splay(q, p);
    int ret = tr[tr[q].ch[0]].sum;
    return ret;
}
```

1.2 Link-Cut Tree

```
#define null 0x0

class node {
    public:
    int fa, ch[2], rev;
    node() { ch[0] = ch[1] = fa = rev = null; }
} tr[N];

void down(int u) {
    if (tr[u].rev) {
        if (tr[u].ch[0]) tr[tr[u].ch[0]].rev ^= 1;
        if (tr[u].ch[1]) tr[tr[u].ch[1]].rev ^= 1;
        tr[u].rev = 0;
        std::swap(tr[u].ch[0], tr[u].ch[1]);
    }
}

int son(int u) {
    return tr[tr[u].fa].ch[1] == u;
}
int isroot(int u) {
    return tr[tr[u].fa].ch[0] != u && tr[tr[u].fa].ch[1] != u;
}

void rotate(int u) {
    int f = tr[u].fa, pf = tr[f].fa, d = son(u);
    tr[u].fa = pf;
    if (!isroot(f)) tr[pf].ch[son(f)] = u, tr[u].fa = pf;
    tr[f].ch[d] = tr[u].ch[d ^ 1];
    if (tr[f].ch[d]) tr[tr[f].ch[d]].fa = f;
    tr[f].fa = u, tr[u].ch[d ^ 1] = f;
}

```

```
int st[N];
void splay(int u) {
    int top = 0; st[++top] = u;
    for (int v = u; !isroot(v); v = tr[v].fa) st[++top] = tr[v].fa;
    for (int i = top; i; i--) down(st[i]);
    while (!isroot(u)) {
        if (!isroot(tr[u].fa)) rotate(son(u) == son(tr[u].fa) ?
            tr[u].fa : u);
        rotate(u);
    }
}
void access(int u, int t = 0) {
    while (u) { splay(u); tr[u].ch[1] = t; t = u, u = tr[u].fa; }
}

void makeroot(int u) { access(u), splay(u), tr[u].rev ^= 1; }
void link(int u, int v) { makeroot(u), tr[u].fa = v, splay(u); }
void cut(int u, int v) {
    makeroot(u), access(v);
    splay(v), tr[u].fa = tr[v].ch[0] = null;
}
int getroot(int u) {
    access(u), splay(u);
    while (tr[u].ch[0]) u = tr[u].ch[0];
    splay(u); return u;
}
```

1.3 k-D Tree

```
const double fac = .65;
const int N = 5e5 + 5;
const int M = 2e5 + 5;

int D, n, Q, root;
int st[M], cnt, ans, tot;
int lst_ans;

struct node {
    int d[2], ch[2], x[2], y[2], v, w, siz, rd;
    node() {}
    node(int _x, int _y, int v) : v(v), w(v), siz(1) {
        ch[0] = ch[1] = 0;
        x[0] = x[1] = d[0] = _x;
        y[0] = y[1] = d[1] = _y;
        rd = 0;
    }
    void clear() {
        x[0] = x[1] = d[0];
        y[0] = y[1] = d[1];
        ch[0] = ch[1] = 0;
        w = v, siz = 1, rd = 0;
    }
} tr[M];

#define check(p)
    ((p).x[0]<=X1&&X0<=(p).x[1]&&(p).y[0]<=Y1&&Y0<=(p).y[1])
#define cmax(a,b) ((a)<(b)?(a)=(b):1)
#define cmin(a,b) ((a)>(b)?(a)=(b):1)

#define ls tr[p].ch[0]
#define rs tr[p].ch[1]

void mt(int f, int s) {
    tr[f].w += tr[s].w;
    tr[f].siz += tr[s].siz;
    cmin(tr[f].x[0], tr[s].x[0]);
    cmax(tr[f].x[1], tr[s].x[1]);
```

```
    cmin(tr[f].y[0], tr[s].y[0]);
    cmax(tr[f].y[1], tr[s].y[1]);
}

bool cmp(const int &a, const int &b) {
    return tr[a].d[D] < tr[b].d[D];
}

int bt(int l, int r, int d) {
    int mid = (l + r) >> 1;
    D = d;
    nth_element(st + l, st + mid, st + r + 1, cmp);
    int p = st[mid];
    tr[p].clear();
    tr[p].rd = d;
    if (l < mid) ls = bt(l, mid - 1, d ^ 1), mt(p, ls);
    if (r > mid) rs = bt(mid + 1, r, d ^ 1), mt(p, rs);
    return p;
}

void dfs(int p) {
    st[++cnt] = p;
    if(ls) dfs(ls);
    if(rs) dfs(rs);
}

int ins(int p, int nw) {
    if (p == 0) {
        tr[p].rd = rand() & 1;
        return nw;
    }
    mt(p, nw), D = tr[p].rd;
    int &nxt = tr[p].ch[tr[nw].d[D] > tr[p].d[D]];
    if (max(tr[ls].siz, tr[rs].siz) > tr[p].siz * fac) {
        cnt = 0;
        st[++cnt] = nw;
        dfs(p);
        int rot = bt(1, cnt, tr[p].rd);
        if (p == root) root = rot;
        return rot;
```

```
    }
    nxt = ins(nxt, nw);
    return p;
}

void ask(int p, int X0, int Y0, int X1, int Y1) {
    if (X0 <= tr[p].x[0] && tr[p].x[1] <= X1 && \
    Y0 <= tr[p].y[0] && tr[p].y[1] <= Y1) {
        ans += tr[p].w;
        return;
    }
    if (X0 <= tr[p].d[0] && tr[p].d[0] <= X1 && \
    Y0 <= tr[p].d[1] && tr[p].d[1] <= Y1) {
        ans += tr[p].v;
    }
    if (ls && check(tr[ls])) ask(ls, X0, Y0, X1, Y1);
    if (rs && check(tr[rs])) ask(rs, X0, Y0, X1, Y1);
}
```

1.4 2-D Segment Tree

```
int ls[330*N], rs[330*N], rot[4*N], tot; ll tr[330*N], ans;

void __mul(int &p, int l, int r, int L, int R, ll v ) {
    if (p == 0) tr[p = ++tot] = 1;
    if (L <= l && r <= R) { tr[p] = merge(tr[p], v);  return; }
    int mid = (l + r) >> 1;
    if (L <= mid) __mul(ls[p], l, mid, L, R, v);
    if (R > mid) __mul(rs[p], mid + 1, r, L, R, v);
}

void __query(int p, int l, int r, int P) {
    if (p == 0) return;
    ans = merge(ans, tr[p]);
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (P <= mid) __query(ls[p], l, mid, P);
    else __query(rs[p], mid + 1, r, P);
}

void mul(int p, int l, int r, int L1, int R1, int L2, int R2, ll
    v) {
    if (L1 <= l && r <= R1) {
        __mul(rot[p], 0, n + 1, L2, R2, v);
        return;
    }
    int mid = (l + r) >> 1;
    if (L1 <= mid) mul(p << 1, l, mid, L1, R1, L2, R2, v);
    if (R1 > mid) mul(p << 1 | 1, mid + 1, r, L1, R1, L2, R2, v);
}

void query(int p, int l, int r, int P1, int P2) {
    if (rot[p]) __query(rot[p], 0, n + 1, P2);
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (P1 <= mid) query(p << 1, l, mid, P1, P2);
    else query(p << 1 | 1, mid + 1, r, P1, P2);
}
```

1.5 Binary Indexed Tree

```
int maxn; ll c1[N], c2[N];
int lowbit(int x) { return x & -x; }
void init(int n) {
    maxn = n;
    for (int i = 1; i <= n; i++) c1[i] = c2[i] = 0;
}
void add(int x, int v) {
    for (int i = x; i <= maxn; i += lowbit(i))
        c1[i] += v, c2[i] += (ll)x * v;
}
ll sum(int x) {
    ll r1 = 0, r2 = 0;
    for (int i = x; i; i -= lowbit(i))
        r1 += c1[i], r2 += c2[i];
    return r1 * (x + 1) - r2;
}
void add(int l, int r, int v) { add(l, v), add(r + 1, -v); }
ll sum(int l, int r) { return sum(r) - sum(l - 1); }
```

2 String

2.1 Suffix Array

```
int buf1[N], buf2[N], buc[N];

void sort(char str[], int n, int sa[], int rk[], int ht[]) {
    int *x = buf1, *y = buf2, m = 127;
    for (int i = 0; i <= m; i++) buc[i] = 0;
    for (int i = 1; i <= n; i++) buc[x[i] = str[i]]++;
    for (int i = 1; i <= m; i++) buc[i] += buc[i - 1];
    for (int i = n; i; i--) sa[buc[x[i]]--] = i;
    for (int k = 1; k <= n; k <<= 1) {
        int p = 0;
        for (int i = n - k + 1; i <= n; i++) y[++p] = i;
        for (int i = 1; i <= n; i++)
            if (sa[i] > k) y[++p] = sa[i] - k;
        for (int i = 0; i <= m; i++) buc[i] = 0;
        for (int i = 1; i <= n; i++) buc[x[y[i]]]++;
        for (int i = 1; i <= m; i++) buc[i] += buc[i - 1];
        for (int i = n; i; i--) sa[buc[x[y[i]]]--] = y[i];
        swap(x, y), x[sa[1]] = p = 1;
        for (int i = 2; i <= n; i++)
        if (y[sa[i - 1]] == y[sa[i]] && \
              y  [sa[i - 1] + k] == y[sa[i] + k]) x[sa[i]] = p;
        else x[sa[i]] = ++p;
        if ((m = p) >= n) break;
    }
    for (int i = 1; i <= n; i++) rk[sa[i]] = i;
    for (int j = 0, k = 0, i = 1; i <= n; ht[rk[i++]] = k)
    for (k ? k-- : 0, j = sa[rk[i] - 1]; \
          str[i + k] == str[j + k]; k++);
}
```

2.2 Suffix Automaton

```
struct node {
    int nxt[26];
    int fa, len;
} tr[N];

int tot = 1, last = 1, root = 1;

void add(int x) {
    int p = last, np = ++tot;
    tr[np].len = tr[p].len + 1;
    while (p && tr[p].nxt[c] == 0)
        tr[p].nxt[c] = np, p = tr[p].fa;
    if (p == 0) tr[np].fa = root;
    else {
        int q = tr[p].ch[c];
        if (tr[q].len == tr[p].len + 1) tr[np].fa = q;
        else {
            int nq = ++tot;
            tr[nq] = tr[q];
            tr[nq].len = tr[p].len + 1;
            tr[q].fa = tr[np].fa = nq;
            while (p && tr[p].ch[c] == q)
                tr[p].ch[c] = nq, p = tr[p].fa;
        }
    }
}
```

3 Graph Theory

3.1 Divide and Conquer on Graph

3.1.1 Divide on Edges

```
int n, m, Q, color[N], pos[N];
int hd[N], tmp[N], nxt[4*N], to[4*N], w[4*N], tot;

void add(int a, int b, int c) {
    nxt[++tot] = hd[a], to[hd[a] = tot] = b, w[tot] = c;
    nxt[++tot] = hd[b], to[hd[b] = tot] = a, w[tot] = c;
}

void add_tmp(int a, int b, int c) {
    nxt[++tot] = tmp[a], to[tmp[a] = tot] = b, w[tot] = c;
    nxt[++tot] = tmp[b], to[tmp[b] = tot] = a, w[tot] = c;
    assert(tot < 6 * n);
}

void build(vector<int> &ch, int fa, int l, int r) {
    if (l > r) return;
    if (l == r) {
        int e = ch[l];
        add_tmp(fa, to[e], w[e]);
        return;
    }
    int u = ++n;
    color[u] = 1;
    int mid = (l + r) / 2;
    add_tmp(fa, u, 0);
    build(ch, u, l, mid);
    build(ch, u, mid + 1, r);
}

void reconstruct(int u, int fa) {
    vector<int> ch;
    for (int e = hd[u]; e != -1; e = nxt[e]) {
        int v = to[e];
        if (v != fa) {
```

```
            reconstruct(v, u);
            ch.push_back(e);
        }
    }
    if (!ch.empty()) {
        int sz = (int)ch.size() - 1;
        int mid = sz / 2;
        build(ch, u, 0, mid);
        build(ch, u, mid + 1, sz);
    }
}

/* CAUTION : This class $data is used for multiple cases and have multiple means */
class data {
    public:
    int a, b;
    data() {}
    data(int a, int b) : a(a), b(b) {}
    bool operator < (const data &rhs) const { return a == rhs.a ? b < rhs.b : a < rhs.a; }
    bool operator > (const data &rhs) const { return a == rhs.a ? b > rhs.b : a > rhs.a; }
};

int siz[N], del[N];

priority_queue<data> h[2*N];                                        /* $data represented -> { dis_to_root, node_id } */
vector<data> idx[N];                                                        /* $data represented -> { HID, dis_to_root } */
data Heap[N];                                                                  /* $data represented -> { best_ans, idx }    */

data find_center(int u, int fa, int sz) {
    siz[u] = 1;
    data ret = data(inf, -1);
```

```
    /* $data represented -> { bigger_sz, edge_id } */
    for(int v, e = hd[u]; e != -1; e = nxt[e]) {
        if(del[e >> 1] || (v = to[e]) == fa) continue;
        ret = min(ret, find_center(v, u, sz));
        siz[u] += siz[v];
        ret = min(ret, data(max(siz[v], sz - siz[v]), e));
    }
    return ret;
}

int tmp_sz;
void dfs(int u, int fa, int dis, int ID) {
    tmp_sz++;
    if(color[u] == 0) h[ID].emplace(data(dis, u));
    idx[u].emplace_back(data(ID, dis));
    for(int e = hd[u]; e != -1; e = nxt[e]) {
        int v = to[e];
        if(del[e >> 1] || v == fa) continue;
        dfs(v, u, dis + w[e], ID);
    }
}

void divide(int u, int sz) {
    int sz1, sz2;
    if(sz <= 1) return;
    int e = find_center(u, 0, sz).b;
    del[e >> 1] = true;
    tmp_sz = 0, h[e].emplace(data(-inf, -1));
    dfs(to[e], 0, 0, e), sz1 = tmp_sz;
    tmp_sz = 0, h[e^1].emplace(data(-inf, -1));
    dfs(to[e^1], 0, 0, e^1), sz2 = tmp_sz;
    Heap[e >> 1] = data(h[e].top().a + w[e] + h[e^1].top().a, e >> 1);
    divide(to[e], sz1);
    divide(to[e^1], sz2);
}
```

3.2 Heavy-Light Decomposition

```
void dfs1(int u, int _fa, int _dep) {
    fa[u] = _fa;
    dep[u] = _dep;
    s[u] = 1;
    for (int e = hd[u]; e; e = nxt[e]) {
        int v = to[e];
        if (v != _fa) {
            dfs1(v, u, _dep + 1);
            s[u] += s[v];
            if (!u_son[u] || s[v] > s[u_son[u]]) u_son[u] = v;
        }
    }
}

void dfs2(int u, int id) {
    top[u] = id;
    f[u] = ++mark;
    df[mark] = u;
    if (!u_son[u]) return;
    dfs2(u_son[u], id);
    for (int e = hd[u]; e; e = nxt[e]) {
        int v = to[e];
        if (v != u_son[u] && v != fa[u]) dfs2(v, v);
    }
}

void update(int a, int b, int c) {
    for (; top[a] != top[b]; b = fa[top[b]]) {
        if (dep[top[b]] < dep[top[a]]) swap(a, b);
        update(1, mark, f[top[b]], f[b], c, 1);
    }
    if (dep[b] < dep[a]) swap(a, b);
    update(1, mark, f[a], f[b], c, 1);
}
```

4 Mathematics

4.1 Transformation

4.1.1 Fast Fourier transform (FFT)

```
const long double PI = acos(-1.0);
const long double EPS = 1E-8;

class complex {
    public:
    long double re, im;
    complex() {}
    complex(long double re, long double im) : re(re), im(im) {}
    complex operator + (const complex &x) {
        return complex(re + x.re, im + x.im);
    }
    complex operator - (const complex &x) {
        return complex(re - x.re, im - x.im);
    }
    complex operator * (const complex &x) {
        return complex(re * x.re - im * x.im, im * x.re + re *
            x.im);
    }
    complex operator / (const complex &x) {
        return complex((re * x.re + im * x.im) / (x.re * x.re +
            x.im * x.im),
        (im * x.re - re * x.im) / (x.re * x.re + x.im * x.im));
    }
};

int n, rev[N];
complex F[N], w[N];

void FFT(complex * F, int n, int offset) {
    for (int i = 0; i < n; i++)
        if (rev[i] > i) std::swap(F[i], F[rev[i]]);
    for (int i = 2; i <= n; i <<= 1) {
        complex wi(cos(offset * 2 * PI / i), \
                    sin(offset * 2 * PI / i));
```

```
        for (int j = 0; j < n; j += i) {
            complex w(1, 0);
            for (int k = j, h = i >> 1; k < j + h; k++) {
                complex t = w * F[k + h], u = F[k];
                F[k] = u + t;
                F[k + h] = u - t;
                w = w * wi;
            }
        }
    }
    if (offset == -1)
        for (int i = 0; i < n; i++) F[i].re = F[i].re / n;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        double x;
        scanf("%lf", &x);
        F[i].re = x;
    }
    n = 1 << (int)ceil(log2(n));

    for (int i = 0; i < n; i++)
        rev[i] = (rev[i >> 1] >> 1) | \
                  ((i & 1) << ((ll)log2(n) - 1));

    FFT(F, n, 1);
    FFT(F, n, -1);
    for (int i = 0; i < n; i++) print(F[i], '\n');
    return 0;
}
```

4.1.2 Number-Theoretic transform (NTT)

```
ll n, inv_n, F[N], rev[N], q;
const ll MOD = 1004535809; // = 479 * 2 ^ 21 + 1
const ll g = 3;

ll q_pow(ll a, ll b) {
    ll ret = 1;
    while (b) {
        if (b & 1) ret = ret * a % MOD;
        a = a * a % MOD;
        b >>= 1;
    }
    return ret;
}

void NTT(ll F[], ll n, int offset) {
    for (int i = 0; i < n; i++)
        if (rev[i] > i) std::swap(F[i], F[rev[i]]);
    for (int i = 2; i <= n; i <<= 1) {
        ll wi = q_pow(g, offset == 1 ? \
                        (MOD - 1) / i :\
                         MOD - 1 - (MOD - 1) / i);
        for (int j = 0; j < n; j += i) {
            ll w = 1;
            for (int k = j, h = i >> 1; k < j + h; k++) {
                ll t = w * F[k + h], u = F[k];
                F[k] = (u + t) % MOD;
                F[k + h] = ((u - t) % MOD + MOD) % MOD;
                w = w * wi % MOD;
            }
        }
    }
    if (offset == -1)
        for (int i = 0; i < n; i++) F[i] = F[i] * inv_n % MOD;
}

int main() {
    scanf("%lld", &n);
    for (int i = 0; i < n; i++) scanf("%lld", &F[i]);
```

```
    n = 1 << (int)ceil(log2(n));
    inv_n = q_pow(n, MOD - 2);

    for (int i = 0; i < n; i++)
        rev[i] = (rev[i >> 1] >> 1) | \
                   ((i & 1) << ((ll)log2(n) - 1));

    NTT(F, n, 1);
    NTT(F, n, -1);
    for (int i = 0; i < n; i++) printf("\t%lld\n", F[i]);
    return 0;
}
```

4.1.3 Fast WalshfiHadamard transform (FWT)

```
#define mod
int rev; // rev = inverse of 2 in mod

void FWT(int A[], int n) {
    for (int d = 1; d < n; d <<= 1) {
        for (int m = d << 1, i = 0; i < n; i += m) {
            for (int j = 0; j < d; j++) {
                int x = A[i + j], y = A[i + j + d];

                /*
                xor  :  A[i + j] = x + y,
                        A[i + j + d] = (x - y + mod) % mod;
                and  :  A[i + j] = x + y;
                or   :  A[i + j + d] = x + y;
                */

                // example for ^ :
                A[i + j] = (x + y) % mod;
                A[i + j + d] = (x - y + mod) % mod;
            }
        }
    }
}


void UFWT(int A[], int n) {
    for (int d = 1; d < n; d <<= 1) {
        for (int m = d << 1, i = 0; i < n; i += m) {
            for (int j = 0; j < d; j++) {
                int x = A[i + j], y = A[i + j + d];

                /*
                xor  :  A[i + j] = (x + y) / 2,
                        A[i + j + d] = (x - y) / 2;
                and  :  A[i + j] = x - y;
                or   :  A[i + j + d] = y - x;
                */

```

```
                // example for ^ :
                A[i + j] = 1ll * (x + y) * rev % mod;
                A[i + j + d] = (1ll * (x - y) * rev % mod + mod) %
                    mod;
            }
        }
    }
}

void solve(int A[], int B[], int n) {
    FWT(A, n);
    FWT(B, n);
    for (int i = 0; i < n; i++) A[i] = 1ll * A[i] * B[i] % mod;
    UFWT(A, n);
}
```

4.2 Simplex Algorithm

```
double c[N], A[M][N], b[M], ans;
int n, m;

void pivot(int id, int p) {
    A[id][p] = 1 / A[id][p];
    b[id] *= A[id][p];
    for (int i = 1; i <= n; i++) if (i ^ p) A[id][i] *= A[id][p];
    for (int i = 1; i <= m; i++) {
        if ((i ^ id) && A[i][p]) {
            for (int j = 1; j <= n; j++)
            if (j ^ p) A[i][j] -= A[i][p] * A[id][j];
            b[i] -= A[i][p] * b[id];
            A[i][p] *= -A[id][p];
        }
    }
    for (int i = 1; i <= n; i++) if (i ^ p) c[i] -= c[p] * A[id][i];
    ans += c[p] * b[id];
    c[p] *= -A[id][p];
}

double solve() {
    while (true) {
        int p, min_id;
        for (p = 1; p <= n; p++) if (c[p] > 0) break;
        if (p == n + 1) return ans;
        double mn = inf;
        for (int i = 1; i <= m; i++)
        if (A[i][p] > 0 && Min > b[i] / A[i][p]) { mn = b[i];
            min_id = i; }
        if (mn == inf) return mn;
        pivot(min_id, p);
    }
}
```

4.3 Lucas's Theorem

4.3.1 Regular Usage

when *mod* is a prime number.

```
void prepare() {
    inv[1] = 1; fac[0] = facInv[0] = 1;
    for (int i = 1; i <= n; i++) {
        if (i != 1) inv[i] = (P - P / i) * inv[P % i] % P;
        fac[i] = fac[i - 1] * i % P;
        facInv[i] = facInv[i - 1] * inv[i] % P;
    }
}

ll lucas(int n, int m) {
    if (n < m) return 0;
    ll ans = 1;
    for (; m; n /= P, m /= P) ans = ans * C(n % P, m % P) % P;
    return ans;
}
```

4.3.2 Advanced Usage

when *mod* is not a prime number.

```
ll fac(ll n, ll p, ll pR) {
    if (n == 0) return 1;
    ll ret = 1;
    for (ll i = 2; i <= pR; i++) if (i % p) ret = ret * i % pR;
    ret = q_pow(ret, n / pR, pR);
    ll r = n % pR;
    for (int i = 2; i <= r; i++) if (i % p) ret = ret * i % pR;
    return ret * fac(n / p, p, pR) % pR;
}

ll C(ll n, ll m, ll p, ll pR) {
    if (n < m) return 0;
    ll x = fac(n, p, pR), y = fac(m, p, pR), z = fac(n - m, p, pR);
    ll c = 0;
    for (ll i = n; i; i /= p) c += i / p;
    for (ll i = m; i; i /= p) c -= i / p;
    for (ll i = n - m; i; i /= p) c -= i / p;
    ll a = x * Inv(y, pR) % pR * Inv(z, pR) % pR * q_pow(p, c, pR)
        % pR;
    return a * (mod / pR) % mod * Inv(mod / pR, pR) % mod;
}

ll lucas(ll n, ll m) {
    ll x = mod, re = 0;
    for (ll i = 2; i <= mod; i++) if (x % i == 0) {
        ll pR = 1;
        while (x % i == 0) x /= i, pR *= i;
        re = (re + C(n, m, i, pR)) % mod;
    }
    return re;
}
```

4.4 Taylor's Theorem

Let $k \geq 1$ be an integer and let the function $f : R \to R$ be k times differentiable at the point $a \in R$. Then there exists a function $h_k : R \to R$ such that

$$f(x) = f(a)+f'(a)(x-a)+\frac{f''(a)}{2!}(x-a)^2+\cdots+\frac{f^{(k)}(a)}{k!}(x-a)^k+h_k(x)(x-a)^k$$

and $\lim_{x\to a} h_k(x) = 0$. This is called the Peano form of the remainder.

Originally from wikipedia.org

4.5 Lagrange Polynomial

Given a set of $k+1$ data points

$$(x_0, y_0), \ldots, (x_j, y_j), \ldots, (x_k, y_k)$$

where no two x_j are the same, the interpolation polynomial in the Lagrange form is a linear combination

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$$

of Lagrange basis polynomials

$$\ell_j(x) := \prod_{0 \le m \le k, m \ne j} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}$$

where $0 \le j \le k$. Note how, given the initial assumption that no two x_j are the same, $x_j - x_m \ne 0$, so this expression is always well-defined. The reason pairs $x_i = x_j$ with $y_i \ne y_j$ are not allowed is that no interpolation function L such that $y_i = L(x_i)$ would exist; a function can only get one value for each argument x_i. On the other hand, if also $y_i = y_j$, then those two points would actually be one single point.
For all $i \ne j$, $\ell_j(x)$ includes the term $(x - x_i)$ in the numerator, so the whole product will be zero at $x = x_i$:

$$\ell_{j \ne i}(x_i) = \prod_{m \ne j} \frac{x_i - x_m}{x_j - x_m} = \frac{(x_i - x_0)}{(x_j - x_0)} \cdots \frac{(x_i - x_i)}{(x_j - x_i)} \cdots \frac{(x_i - x_k)}{(x_j - x_k)} = 0.$$

On the other hand,

$$\ell_i(x_i) := \prod_{m \ne i} \frac{x_i - x_m}{x_i - x_m} = 1$$

In other words, all basis polynomials are zero at $x = x_i$, except $\ell_i(x)$, for which it holds that $\ell_i(x_i) = 1$, because it lacks the $(x - x_i)$ term.
It follows that $y_i \ell_i(x_i) = y_i$, so at each point x_i, $L(x_i) = y_i + 0 + 0 + \ldots + 0 = y_i$, showing that L interpolates the function exactly.

Originally from wikipedia.org

5 Computational Geometry

```
/* Computational Geometry Base Definition */

const double eps = 1e-10;
const double PI = acos(-1.);

class Point {
    public:
    double x, y;
    Point() {}
    Point(double _x, double _y) {
        x = _x, y = _y;
    }
};

typedef Point Vector;
typedef std::vector<Point> Polygon;

class Circle {
    public:
    Point c;
    double r;
    Circle(Point c, double r) : c(c), r(r) {}
    Point point(double a) {
        return Point(c.x * cos(a) * r, c.y * sin(a) * r);
    }
};

class Line {
    public:
    Point P;
    Vector v;
    double ang;
    Line() {}
    Line(Point P, Vector v) : P(P), v(v) {
        ang = atan2(v.y, v.x);
    }
    bool operator < (const Line &L) const {
```

```
        return ang < L.ang;
    }
};

int fcmp(double x) {
    return fabs(x) < eps ? 0 : x < 0 ? -1 : 1;
}
Vector operator + (Vector a, Vector b) {
    return Vector(a.x + b.x, a.y + b.y);
}
Vector operator - (Vector a, Vector b) {
    return Vector(a.x - b.x, a.y - b.y);
}
Vector operator * (Vector a, int k) {
    return Vector(a.x * k, a.y * k);
}
Vector operator / (Vector a, int k) {
    return Vector(a.x / k, a.y / k);
}
bool operator < (const Point &a, const Point &b) {
    return a.x == b.x ? a.y < a.y : a.x < a.x;
}
bool operator == (const Point &a, const Point &b) {
    return fcmp(a.x - b.x) == 0 && fcmp(a.y - b.y) == 0;
}
double Dot(Vector a, Vector b) {
    return a.x * b.x + a.y * b.y;
}
double Cross(Vector a, Vector b) {
    return a.x * b.y - a.y * b.x;
}
double Area2(Point A, Point B, Point C) {
    return Cross(B - A, C - A);
}
double Length(Vector a) {
    return sqrt(Dot(a, a));
}
double angle(Vector a) {
    return atan2(a.y, a.x);
}
```

```
double Angle(Vector a, Vector b) {
    return acos(Dot(a, b)) / (Length(a) * Length(b));
}
Vector Rotate(Vector a, double rad) {
    return Vector(a.x * cos(rad) - a.y * sin(rad), \
    a.x * sin(rad) + a.y * cos(rad));
}
Vector Normal(Vector a) {
    double L = Length(a);
    return Vector(-a.y / L, a.x / L);
}
double Dist(Point A, Point B) {
    return Length(B - A);
}
bool onLeft(Line L, Point P) {
    return Cross(L.v, P - L.P) > 0;
}
Point GetIntersection(Line a, Line b) {
    Vector u = a.P - b.P;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P + a.v * t;
}

/* Points and Segments Messing UP */

bool isPointOnSegment(Point P, Point A, Point B) {
    return Cross(Vector(P - A), Vector(P - B)) == 0 && \
    (P.x - A.x) * (P.x - B.x) <= 0;
}
bool isSegmentCrossed(Point A, Point B, Point C, Point D) {
    int a = fcmp(Cross(B - A, C - A) * Cross(D - A, B - A)) > 0;
    int b = fcmp(Cross(D - C, B - C) * Cross(A - C, D - C)) > 0;
    return a && b;
}
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
    Vector u = P - Q;
    int t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}
Point GetSegmentIntersection(Point A, Point B, Point C, Point D) {
```

```
    Vector a = B - A;
    double s1 = fabs(Cross(a, C - A));
    double s2 = fabs(Cross(a, D - A));
    return Point((s1 * D.x + s2 * C.x) / (s1 + s2), \
    (s1 * D.y + s2 * C.y) / (s1 + s2));
}
double DistanceToLine(Point P, Point A, Point B) {
    Vector a = B - A, b = P - A;
    return fabs(Cross(a, b)) / Length(a);
}
double DistanceToSegment(Point P, Point A, Point B) {
    if (A == B) return Length(P - A);
    Vector a = B - A, b = P - A, c = P - B;
    if (fcmp(Dot(a, b)) < 0) return Length(b);
    else if (fcmp(Dot(a, c)) > 0) return Length(c);
    else return fabs(Cross(a, b)) / Length(a);
}

/* Polygons and lines messing up */
double Area(Polygon P) {
    double ret = 0;
    Point St = *P.begin();
    int s = P.size();
    for (int i = 1; i < s - 1; i++) {
        Point A = P[i], B = P[i + 1];
        ret += Cross(A - St, B - St);
    }
    return ret;
}

int isPointInPolygon(Point A, Polygon P) {
    int wn = 0;
    int s = P.size();
    for (int i = 0; i < s; i++) {
        if (isPointOnSegment(A, P[i], P[(i + 1) % s])) return -1;
        int k = fcmp(Cross(P[(i + 1) % s] - P[i], A - P[i]));
        int d1 = fcmp(P[i].y - A.y);
        int d2 = fcmp(P[(i + 1) % s].y - A.y);
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
```

```
    }
    return wn ? 1 : 0;
}

int ConvexHull(Point P[], int n, Point ch[]) {
    int m = 0;
    for (int i = 0; i < n; i++) {
        while (m > 1 && Cross(ch[m - 1] - ch[m - 2], \
        P[i] - ch[m - 2]) <= 0) m--;
        ch[m++] = P[i];
    }
    int k = m;
    for (int i = n - 2; i >= 0; i--) {
        while (m > k && Cross(ch[m - 1] - ch[m - 2], \
        P[i] - ch[m - 2]) <= 0) m--;
        ch[m++] = P[i];
    }
    if (n > 1) m--;
    return m;
}

int HalfplainIntersection(Line L[], int n, Point Poly[]) {
    std::sort(L, L + n);
    int hd, tl;
    Point *P = new Point[n];
    Line *q = new Line[n];
    q[hd = tl = 0] = L[0];
    for (int i = 1; i < n; i++) {
        while (hd < tl && !onLeft(L[i], P[tl - 1])) tl--;
        while (hd < tl && !onLeft(L[i], P[hd])) hd++;
        q[++tl] = L[i];
        if (fabs(Cross(q[tl].v, q[tl - 1].v) < eps)) {
            tl--;
            if (onLeft(q[tl], L[i].P)) q[tl] = L[i];
        }
        if (hd < tl) P[tl - 1] = GetIntersection(q[tl - 1], q[tl]);
    }
    while (hd < tl && !onLeft(q[hd], P[tl - 1])) tl--;
    if (tl - hd < 0) return 0;
    P[tl] = GetIntersection(q[tl], q[hd]);
```

```
    int m = 0;
    for (int i = hd; i <= tl; i++) Poly[m++] = P[i];
    return m;
}

double RotatingCalipers(Point P[], int n) {
    int x = 1;
    double ans = 0;
    P[n] = P[0];
    for (int i = 0; i < n; i++) {
        while (Cross(P[i + 1] - P[i], P[x + 1] - P[i]) > \
        Cross(P[i + 1] - P[i], P[x] - P[i])) x = (x + 1) % n;
        ans = std::max(ans, Dist(P[x], P[i]));
        ans = std::max(ans, Dist(P[x + 1], P[i + 1]));
    }
    return ans;
}

using namespace std;

/* Circle and &^%^%#@ messing up */

int getLineCircleIntersection(Line L, Circle C, double &t1, double
    & t2, vector<Point> &sol) {
    double a = L.v.x, b = L.P.x - C.c.x, c = L.v.y, d = L.P.y -
        C.c.y;
    double e = a * a + c * c, f = 2 * (a * b + c * d), g = b * b +
        d * d - C.r * C.r;
    double delta = f * f - 4 * e * g;
    if (fcmp(delta) < 0) return 0;
    if (fcmp(delta) == 0) {
        t1 = t2 = -f / (2 * e);
        sol.push_back(C.point(t1));
        return 1;
    }
    t1 = (-f - sqrt(delta)) / (2 * e);
    sol.push_back(C.point(t1));
    t2 = (-f + sqrt(delta)) / (2 * e);
    sol.push_back(C.point(t2));
    return 2;
```

```
}

int getCircleCircleIntersection(Circle C1, Circle C2,
    vector<Point> &sol) {
    double d = Length(C1.c - C2.c);
    if (fcmp(d) == 0) {
        if (fcmp(C1.r - C2.r) == 0) return -1;
        return 0;
    }
    if (fcmp(C1.r + C2.r - d) < 0) return 0;
    if (fcmp(fabs(C1.r - C2.r) - d) > 0) return 0;
    double a = angle(C2.c - C1.c);
    double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 *
        C1.r * d));
    Point p1 = C1.point(a - da), p2 = C1.point(a + da);
    sol.push_back(p1);
    if (p1 == p2) return 1;
    sol.push_back(p2);
    return 2;
}

int getTangents(Point p, Circle C, Vector *v) {
    Vector u = C.c - p;
    double dist = Length(u);
    if (dist < C.r) return 0;
    else if (fcmp(dist - C.r) == 0) {
        v[0] = Rotate(u, PI / 2);
        return 1;
    } else {
        double ang = asin(C.r / dist);
        v[0] = Rotate(u, -ang);
        v[1] = Rotate(u, +ang);
        return 2;
    }
}

int getTangents(Circle A, Circle B, Point *a, Point *b) {
    int cnt = 0;
    if (A.r < B.r) { swap(A, B); swap(a, b); }
    int d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) *
```

```
        (A.c.y - B.c.y);
    int rdiff = A.r - B.r;
    int rsum = A.r + B.r;
    if (d2 < rdiff * rdiff) return 0;
    double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
    if (d2 == 0 && A.r == B.r) return -1;
    if (d2 == rdiff * rdiff) {
        a[cnt] = A.point(base); b[cnt] = B.point(base); cnt++;
        return 1;
    }
    double ang = acos(A.r - B.r) / sqrt(d2);
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(base + ang); cnt++;
    a[cnt] = A.point(base - ang);
    b[cnt] = B.point(base - ang); cnt++;
    if (d2 == rsum * rsum) {
        a[cnt] = A.point(base);
        b[cnt] = B.point(PI + base); cnt++;
    } else if (d2 > rsum * rsum) {
        double ang = acos(A.r + B.r) / sqrt(d2);
        a[cnt] = A.point(base + ang);
        b[cnt] = B.point(PI + base + ang); cnt++;
        a[cnt] = A.point(base - ang);
        b[cnt] = B.point(PI + base - ang); cnt++;
    }
    return cnt;
}
```

6 Others

6.1 Simulated Annealing

```
/*
* J(y)          Evaluation function value in state y
* S(i)          Indicates the current status
* S(i+1)        Indicates the new status
* r:0.95        Used to control the speed of cooling
* T:1000        The temperature of the system,
*               the system should initially be at a high temperature
* T_min:0.001 The lower limit of the temperature.
*               If the temperature T reaches T_min, stop searching
*/

function SA:

    while (T > T_min):

        dE = J(S(i + 1)) - J(S(i));

        if (dE >= 0)

        // After the expression is moved, A better solution is
            obtained and the mobile is always accepted

            S(i + 1) = S(i);

        // Accepts movement from S(i) to S(i+1)

        else if (exp(dE / T) > random(0, 1))
            S(i + 1) = S(i);

        // Accepts movement from S(i) to S(i+1)

        T = r * T;

        //Cooling annealing, 0<r<1. The bigger r is, the slower the
            cooling is. The smaller r is, the faster the temperature
            is lowered.
```

```
        //If r is too large, the search for the global optimal
            solution may be higher, but the search process is
            longer. If r is too small, the search process will be
            fast, but in the end it may reach a local optimum.

        i++;
```