Contents

Suffix Tree

```
// Ukkonen's algorithm O(n)
const int A = 27; // Alphabet size
struct SuffixTree {
    struct Node { // [l, r) !!!
        int l, r, link, par;
        int nxt[A];
        Node(): l(-1), r(-1), link(-1), par(-1) { fill(nxt, nxt + A, -1); }
        Node(int _l, int _r, int _link, int _par):
        l(_l), r(_r), link(_link), par(_par) { fill(nxt, nxt + A, -1); }
        int &next(int c) { return nxt[c]; }
        int get_len() const { return r - l; }
    };
    struct State { int v, len; };
    vec< Node > t;
    State cur_state;
    vec< int > s;
    SuffixTree(): cur_state({0, 0}) { t.push_back(Node()); }
    // v -> v + s[l, r) !!!
    State go(State st, int l, int r) {
        while(l < r) {
            if(st.len == t[st.v].get_len()) {
                State nx = State({ t[st.v].next(s[l]), 0 });
                if(nx.v == -1) return nx;
                st = nx;
                continue;
            }
            if(s[ t[st.v].l + st.len ] != s[l]) return State({-1, -1});
            if(r - l < t[st.v].get_len() - st.len)
                return State({st.v, st.len + r - l});
            l += t[st.v].get_len() - st.len;
            st.len = t[st.v].get_len();
        }
        return st;
    }
    int get_vertex(State st) {
        if(t[st.v].get_len() == st.len) return st.v;
        if(st.len == 0) return t[st.v].par;
        Node &v = t[st.v];
        Node &pv = t[v.par];
        Node add(v.l, v.l + st.len, -1, v.par);
        // nxt
        pv.next(s[v.l]) = (int)t.size();
        add.next(s[v.l + st.len]) = st.v;
        // par
```

```
        v.par = (int)t.size();
        // [l, r)
        v.l += st.len;
        t.push_back(add); // !!!
        return (int)t.size() - 1;
    }
    int get_link(int v) {
        if(t[v].link != -1) return t[v].link;
        if(t[v].par == -1) return 0;
        int to = get_link(t[v].par);
        to = get_vertex(
            go(State({to, t[to].get_len()}), t[v].l + (t[v].par == 0), t[v].r)
        );
        return t[v].link = to;
    }
    void add_symbol(int c) {
        assert(0 <= c && c < A);
        s.push_back(c);
        while(1) {
            State hlp = go( cur_state, (int)s.size() - 1, (int)s.size() );
            if(hlp.v != -1) { cur_state = hlp; break; }
            int v = get_vertex(cur_state);
            Node add((int)s.size() - 1, +inf, -1, v);
            t.push_back(add);
            t[v].next(c) = (int)t.size() - 1;
            cur_state.v = get_link(v);
            cur_state.len = t[cur_state.v].get_len();
            if(!v) break;
        }
    }
};
```

Suffix Array

```
const int LOG = 21;
struct SuffixArray {
    string s;
    int n;
    vec< int > p;
    vec< int > c[LOG];
    SuffixArray(): n(0) { }
    SuffixArray(string ss): s(ss) {
        s.push_back(0);
        n = (int)s.size();
        vec< int > pn, cn;
```

```
        vec< int > cnt;
        p.resize(n);
        for(int i = 0;i < LOG;i++) c[i].resize(n);
        pn.resize(n);
        cn.resize(n);
        cnt.assign(300, 0);
        for(int i = 0;i < n;i++) cnt[s[i]]++;
        for(int i = 1;i < (int)cnt.size();i++) cnt[i] += cnt[i - 1];
        for(int i = n - 1;i >= 0;i--) p[--cnt[s[i]]] = i;
        for(int i = 1;i < n;i++) {
            c[0][p[i]] = c[0][p[i - 1]];
            if(s[p[i]] != s[p[i - 1]]) c[0][p[i]]++;
        }
        for(int lg = 0, k = 1;k < n;k <<= 1, lg++) {
            for(int i = 0;i < n;i++) {
                if((pn[i] = p[i] - k) < 0) pn[i] += n;
            }
            cnt.assign(n, 0);
            for(int i = 0;i < n;i++) cnt[c[lg][pn[i]]]++;
            for(int i = 1;i < (int)cnt.size();i++) cnt[i] += cnt[i - 1];
            for(int i = n - 1;i >= 0;i--) p[--cnt[c[lg][pn[i]]]] = pn[i];
            for(int l1, r1, l2, r2, i = 1;i < n;i++) {
                cn[p[i]] = cn[p[i - 1]];
                l1 = p[i - 1];
                l2 = p[i];
                if((r1 = l1 + k) >= n) r1 -= n;
                if((r2 = l2 + k) >= n) r2 -= n;
                if(c[lg][l1] != c[lg][l2] || c[lg][r1] != c[lg][r2]) cn[p[i]]++;
            }
            c[lg + 1] = cn;
        }
        p.erase(p.begin(), p.begin() + 1);
        n--;
    }
    int get_lcp(int i, int j) {
        int res = 0;
        for(int lg = LOG - 1;lg >= 0;lg--) {
            if(i + (1 << lg) > n || j + (1 << lg) > n) continue;
            if(c[lg][i] == c[lg][j]) {
                i += (1 << lg);
                j += (1 << lg);
                res += (1 << lg);
            }
        }
        return res;
    }
```

```
};
```

Suffix automaton

```
struct suf_auto {
    vector<int> base, suf, len;
    vector<vector<int>> g;
    int last, sz;
    suf_auto(): base(26, -1), g(1, base), suf(1, -1), len(1, 0), last(0), sz(1){}
    void add_string(const string &s) {
        for (char c : s) {
            c -= 'a';
            int cur = last;
            last = sz++;
            g.push_back(base);
            suf.emplace_back();
            len.push_back(len[cur] + 1);
            while (cur != -1 && g[cur][c] == -1)
                g[cur][c] = last, cur = suf[cur];
            if (cur == -1) {
                suf[last] = 0;
                continue;
            }
            int nx = g[cur][c];
            if (len[nx] == len[cur] + 1) {
                suf[last] = nx;
                continue;
            }
            int cl = sz++;
            g.push_back(g[nx]);
            suf.push_back(suf[nx]);
            len.push_back(len[cur] + 1);
            suf[last] = suf[nx] = cl;
            while (cur != -1 && g[cur][c] == nx)
                g[cur][c] = cl, cur = suf[cur];
        }
    }
};
```

Kasai

```
vector<int> get_lcp(const string& s, const vector<int>& suf){
    int n = (int)suf.size();
    vector<int> back(n);
```

```
    for(int i = 0; i < n; i++) back[suf[i]] = i;
    vector<int> lcp(n - 1);
    for(int i = 0, k = 0; i < n; i++){
        int x = back[i]; k = max(0, k - 1);
        if(x == n - 1){k = 0; continue;}
        while(s[suf[x] + k] == s[suf[x + 1] + k]) k++;
        lcp[x] = k;
    }
    return lcp;
}
```

Z-function

```
vector<int> get_z(const string& s){
    int n = (int)s.length();
    vector<int> z(n);
    for(int i = 1, l = 0, r = 0; i < n; i++){
        if(i < r) z[i] = min(r - i, z[i - l]);
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
        if(i + z[i] > r) l = i, r = i + z[i];
    }
    return z;
}
```

prefix-function(TODO)

```

```

Manacher(TODO)

```

```

Aho

```
const int A = 300; // alphabet size
struct Aho {
    struct Node {
        int nxt[A], go[A];
        int par, pch, link;
        int good;
        Node(): par(-1), pch(-1), link(-1), good(-1) {
```

```
        fill(nxt, nxt + A, -1); fill(go, go + A, -1); }
};
vec< Node > a;
Aho() { a.push_back(Node()); }
void add_string(const string &s) {
    int v = 0;
    for(char c : s) {
        if(a[v].nxt[c] == -1) {
            a[v].nxt[c] = (int)a.size();
            a.push_back(Node());
            a.back().par = v;
            a.back().pch = c;
        }
        v = a[v].nxt[c];
    }
    a[v].good = 1;
}
int go(int v, int c) {
    if(a[v].go[c] == -1) {
        if(a[v].nxt[c] != -1) {
            a[v].go[c] = a[v].nxt[c];
        }else {
            a[v].go[c] = v ? go(get_link(v), c) : 0;
        }
    }
    return a[v].go[c];
}
int get_link(int v) {
    if(a[v].link == -1) {
        if(!v || !a[v].par) a[v].link = 0;
        else a[v].link = go(get_link(a[v].par), a[v].pch);
    }
    return a[v].link;
}
bool is_good(int v) {
    if(!v) return false;
    if(a[v].good == -1) {
        a[v].good = is_good(get_link(v));
    }
    return a[v].good;
}
bool is_there_substring(const string &s) {
    int v = 0;
    for(char c : s) {
        v = go(v, c);
        if(is_good(v)) {
```

```
                return true;
            }
        }
        return false;
    }
};
```

Hungarian

```
vector<int> Hungarian(const vector< vector<int> >& a){ // ALARM: INT everywhere
    int n = (int)a.size();
    vector<int> row(n), col(n), pair(n, -1), back(n, -1), prev(n, -1);
    auto get = [&](int i, int j){ return a[i][j] + row[i] + col[j];};
    for(int v = 0; v < n; v++){
        vector<int> min_v(n, v), A_plus(n), B_plus(n);
        A_plus[v] = 1; int jb;
        while(true){
            int pos_i = -1, pos_j = -1;
            for(int j = 0; j < n; j++){
                if(!B_plus[j] && (pos_i == -1 ||
                    get(min_v[j], j) < get(pos_i, pos_j))) {
                    pos_i = min_v[j], pos_j = j;
                }
            }
            int weight = get(pos_i, pos_j);
            for(int i = 0; i < n; i++) if(!A_plus[i]) row[i] += weight;
            for(int j = 0; j < n; j++) if(!B_plus[j]) col[j] -= weight;
            B_plus[pos_j] = 1, prev[pos_j] = pos_i;
            int x = back[pos_j];
            if(x == -1) { jb = pos_j; break;}
            A_plus[x] = 1;
            for(int j = 0; j < n; j++)
                if(get(x, j) < get(min_v[j], j))
                    min_v[j] = x;
        }
        while(jb != -1){
            back[jb] = prev[jb];
            swap(pair[prev[jb]], jb);
        }
    }
    return pair;
}
```

Hopkroft-Karp

```
struct HopcroftKarp {
    int n, m;
    vec< vec< int > > g;
    vec< int > pl, pr, dist;
    HopcroftKarp(): n(0), m(0) { }
    HopcroftKarp(int _n, int _m): n(_n), m(_m) { g.resize(n); }
    void add_edge(int u, int v) { g[u].push_back(v); }
    bool bfs() {
        dist.assign(n + 1, inf);
        queue< int > q;
        for(int u = 0;u < n;u++) {
            if(pl[u] < m) continue;
            dist[u] = 0;
            q.push(u);
        }
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            if(dist[u] >= dist[n]) continue;
            for(int v : g[u]) {
                if(dist[ pr[v] ] > dist[u] + 1) {
                    dist[ pr[v] ] = dist[u] + 1;
                    q.push(pr[v]);
                }
            }
        }
        return dist[n] < inf;
    }
    bool dfs(int v) {
        if(v == n) return 1;
        for(int to : g[v]) {
            if(dist[ pr[to] ] != dist[v] + 1) continue;
            if(!dfs(pr[to])) continue;
            pl[v] = to;
            pr[to] = v;
            return 1;
        }
        return 0;
    }
    int find_max_matching() {
        pl.resize(n, m);
        pr.resize(m, n);
        int result = 0;
        while(bfs()) {
            for(int u = 0;u < n;u++) {
```

```
                if(pl[u] < m) continue;
                result += dfs(u);
            }
        }
        return result;
    }
};
```

CHT

```
struct Line {
    ll k, b;
    int type;
    ld x;
    Line(): k(0), b(0), type(0), x(0) { }
    Line(ll _k, ll _b, ld _x = 1e18, int _type = 0):
        k(_k), b(_b), x(_x), type(_type) { }
    bool operator<(const Line& other) const {
        if(type + other.type > 0) { return x < other.x;
        }else { return k < other.k; }
    }
    ld intersect(const Line& other) const {
        return ld(b - other.b) / ld(other.k - k);
    }
    ll get_func(ll x0) const {
        return k * x0 + b;
    }
 };
struct CHT {
    set< Line > qs;
    set< Line > :: iterator fnd, help;
    bool hasr(const set< Line > :: iterator& it) {
        return it != qs.end() && next(it) != qs.end(); }
    bool hasl(const set< Line > :: iterator& it) {
        return it != qs.begin(); }
    bool check(const set< Line > :: iterator& it) {
        if(!hasr(it)) return true;
        if(!hasl(it)) return true;
        return it->intersect(*prev(it)) < it->intersect(*next(it)); }
    void update_intersect(const set< Line > :: iterator& it) {
        if(it == qs.end()) return;
        if(!hasr(it)) return;
        Line tmp = *it;
        tmp.x = tmp.intersect(*next(it));
        qs.insert(qs.erase(it), tmp);
```

```
    }
    void add_line(Line L) {
        if(qs.empty()) { qs.insert(L); return; }
        {   fnd = qs.lower_bound(L);
            if(fnd != qs.end() && fnd->k == L.k) {
                if(fnd->b >= L.b) return;
                else qs.erase(fnd);
            }
        }
        fnd = qs.insert(L).first;
        if(!check(fnd)) { qs.erase(fnd); return; }
        while(hasr(fnd) && !check(help = next(fnd))) { qs.erase(help); }
        while(hasl(fnd) && !check(help = prev(fnd))) { qs.erase(help); }
        if(hasl(fnd)) { update_intersect(prev(fnd)); }
        update_intersect(fnd);
    }
    ll get_max(ld x0) {
        if(qs.empty()) return -inf64;
        fnd = qs.lower_bound(Line(0, 0, x0, 1));
        if(fnd == qs.end()) fnd--;
        ll res = -inf64; int i = 0;
        while(i < 2 && fnd != qs.end()) {
            res = max(res, fnd->get_func(x0));
            fnd++;
            i++;
        }
        while(i-- > 0) fnd--;
        while(i < 2) {
            res = max(res, fnd->get_func(x0));
            if(hasl(fnd)) {
                fnd--; i++;
            }else {
                break;
            }
        }
        return res;
    }
};
```

FFT with prime mod

```
const int mod = 998244353;
const int root = 31;
const int LOG = 23;
const int N = 1e5 + 5;
```

```
vec< int > G[LOG + 1];
vec< int > rev[LOG + 1];
inline void _add(int &x, int y);
inline int _sum(int a, int b);
inline int _sub(int a, int b);
inline int _mul(int a, int b);
inline int _binpow(int x, int p);
inline int _rev(int x);
void precalc() {
    for(int start = root, lvl = LOG;lvl >= 0;lvl--, start = _mul(start, start)) {
        int tot = 1 << lvl;
        G[lvl].resize(tot);
        for(int cur = 1, i = 0;i < tot;i++, cur = _mul(cur, start)) {
            G[lvl][i] = cur;
        }
    }
    for(int lvl = 1;lvl <= LOG;lvl++) {
        int tot = 1 << lvl;
        rev[lvl].resize(tot);
        for(int i = 1;i < tot;i++) {
            rev[lvl][i] = ((i & 1) << (lvl - 1)) | (rev[lvl][i >> 1] >> 1);
        }
    }
}
void fft(vec< int > &a, int sz, bool invert) {
    int n = 1 << sz;
    for(int j, i = 0;i < n;i++) {
        if((j = rev[sz][i]) < i) {
            swap(a[i], a[j]);
        }
    }
    for(int f1, f2, lvl = 0, len = 1;len < n;len <<= 1, lvl++) {
        for(int i = 0;i < n;i += (len << 1)) {
            for(int j = 0;j < len;j++) {
                f1 = a[i + j];
                f2 = _mul(a[i + j + len], G[lvl + 1][j]);
                a[i + j] = _sum(f1, f2);
                a[i + j + len] = _sub(f1, f2);
            }
        }
    }
    if(invert) {
        reverse(a.begin() + 1, a.end());
        int rn = _rev(n);
        for(int i = 0;i < n;i++) {
            a[i] = _mul(a[i], rn);
```

```
        }
    }
}
vec< int > multiply(const vec< int > &a, const vec< int > &b) {
    vec< int > fa(ALL(a));
    vec< int > fb(ALL(b));
    int n = (int)a.size();
    int m = (int)b.size();
    int maxnm = max(n, m), sz = 0;
    while((1 << sz) < maxnm) sz++; sz++;
    fa.resize(1 << sz);
    fb.resize(1 << sz);
    fft(fa, sz, false);
    fft(fb, sz, false);
    int SZ = 1 << sz;
    for(int i = 0;i < SZ;i++) { fa[i] = _mul(fa[i], fb[i]); }
    fft(fa, sz, true);
    while((int)fa.size() > 1 && !fa.back()) fa.pop_back();
    return fa;
}
```

FFT with ld

```
struct Complex {
    ld rl = 0, im = 0;
    Complex() = default;
    Complex(ld x, ld y = 0): rl(x), im(y) { }
    Complex & operator = (const Complex &o) {
        if(this != &o) {
            rl = o.rl;
            im = o.im;
        }
        return *this;
    }
    Complex operator + (const Complex &o) const { return {rl + o.rl, im + o.im}; }
    Complex operator - (const Complex &o) const { return {rl - o.rl, im - o.im}; }
    Complex operator * (const Complex &o) const {
        return {rl * o.rl - im * o.im, rl * o.im + im * o.rl}; }
    Complex & operator *= (const Complex &o) {
        (*this) = (*this) * o;
        return *this;
    }
    Complex operator / (const Complex &o) const {
        ld md = o.rl * o.rl + o.im * o.im;
        return {(rl * o.rl + im * o.im) / md, (im * o.rl - rl * o.im) / md};
```

```
    }
    Complex & operator /= (int k) {
        rl /= k;
        im /= k;
        return *this;
    }
    ld real() const { return rl; }
    ld imag() const { return im; }
};
typedef Complex base;
const int LOG = 20;
const int N = 1 << LOG;
int rev[N];
vec< base > PW[LOG + 1];
void precalc() {
    for(int i = 1;i < N;i++) { rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (LOG - 1)); }
    for(int lvl = 0;lvl <= LOG;lvl++) {
        int sz = 1 << lvl;
        ld alpha = 2 * pi / sz;
        base root(cos(alpha), sin(alpha));
        base cur = 1;
        PW[lvl].resize(sz);
        for(int j = 0;j < sz;j++) {
            PW[lvl][j] = cur;
            cur *= root;
        }
    }
}
void fft(base *a, bool invert = 0) {
    for(int j, i = 0;i < N;i++) {
        if((j = rev[i]) > i) swap(a[i], a[j]);
    }
    base u, v;
    for(int lvl = 0;lvl < LOG;lvl++) {
        int len = 1 << lvl;
        for(int i = 0;i < N;i += (len << 1)) {
            for(int j = 0;j < len;j++) {
                u = a[i + j];
                v = a[i + j + len] *
                    (invert ? PW[lvl + 1][j ? (len << 1) - j : 0] : PW[lvl + 1][j]);
                a[i + j] = u + v;
                a[i + j + len] = u - v;
            }
        }
    }
    if(invert) {
```

```
        for(int i = 0;i < N;i++) {
            a[i] /= N;
        }
    }
}
```

Convex-Hull

```
struct Point {
    ll x, y;
    Point(){}
    Point(ll x, ll y): x(x), y(y) {}
    bool operator <(const Point &a) const {
        return make_pair(x, y) < make_pair(a.x, a.y); }
    Point operator +(const Point &a) const { return {x + a.x, y + a.y}; }
    Point operator -(const Point &a) const { return {x - a.x, y - a.y}; }
    ll cross_product(const Point &a) const { return x * a.y - y * a.x; }
    ll len2() const { return x * x + y * y; }
};
vector<Point> convex_hull(vector<Point> v) {
    Point O = v[0];
    int pos = 0;
    for (int i = 0; i < v.size(); i++) {
        if (v[i] < O)
            tie(pos, O) = {i, v[i]};
    }
    swap(v[0], v[pos]);
    v = {v.begin() + 1, v.end()};
    sort(v.begin(), v.end(), [&O](const Point &a, const Point &b){
        ll prod = (a - O).cross_product(b - O);
        if (prod)
            return prod > 0;
        return (a - O).len2() < (b - O).len2();
    });
    vector<Point> ret{O};
    for (Point &p : v) {
        while (ret.size() > 1) {
            Point fr = p - ret[ret.size() - 2],
                sc = ret[ret.size() - 1] - ret[ret.size() - 2];
            ll prod = sc.cross_product(fr);
            if (prod > 0)
                break;
            ret.pop_back();
        }
        ret.push_back(p);
```

```
38      }
39      return ret;
40  };
```