

Contents

1 Basic	1	5 String	4
2 Data Structure	1	5.1 Z-Value	4
2.1 Link Cut Tree	1	5.2 Manacher	4
2.2 LiChaoST	1	5.3 Suffix Array	4
2.3 Leftist Heap	1	5.4 Suffix Automaton	5
3 Graph	2	5.5 Palindrome Tree	5
3.1 Dominator Tree	2	6 Math	6
3.2 MaximumClique	2	6.1 Miller Rabin	6
4 Flow/Matching	2	6.2 Pollard Rho	6
4.1 Dinic	2	6.3 Ext GCD	6
4.2 Min Cost Max Flow	3	6.4 Chinese Remainder Theorem	6
4.3 Kuhn Munkres	3	6.5 Powerful Number Sieve	6
4.4 Hopcroft Karp	4	6.6 Fast Walsh Transform	7
		6.7 Floor Sum	7
		7 Geometry	7
		7.1 Basic	7
		7.2 Minkowski Sum	7

1 Basic

2 Data Structure

2.1 Link Cut Tree

```

struct LCT{
    int ch[maxn]
    ][2], par[maxn], rev[maxn], xr[maxn], val[maxn];
    int get(int x){ return ch[par[x]][1] == x;}
    int isroot(int x){
        return ch[par[x]][0] != x && ch[par[x]][1] != x;}
    void push(int x){
        if(rev[x]){
            if(rs) swap(ch[rs][0], ch[rs][1]), rev[rs] ^= 1;
            if(ls) swap(ch[ls][0], ch[ls][1]), rev[ls] ^= 1;
            rev[x] = 0;
        }
    }
    void pull(int x){
        xr[x] = xr[ls] ^ xr[rs] ^ val[x];
    }
    void rotate(int x){
        int y = par[x], z = par[y], k = get(x);
        if(!isroot(y)) ch[z][ch[z][1] == y] = x;
        ch[y][k] = ch[x][!k], par[ch[x][!k]] = y;
        ch[x][!k] = y, par[y] = x;
        par[x] = z;
        pull(y), pull(x);
    }
    void update(int x){
        if(!isroot(x)) update(par[x]);
        push(x);
    }
    void splay(int x){
        update(x);
        for(int p = par[x]; !isroot(x); rotate(x), p = par[x]){
            if(!isroot(p)) rotate(get(p) == get(x) ? p : x);
        }
    }
    void access(int x){
        for(int p = 0; x != 0; p = x, x = par[x]){
            splay(x);
            ch[x][1] = p;
            pull(x);
        }
    }
    void make_root(int x){
        access(x);
        splay(x);
        swap(ls, rs);
        rev[x] ^= 1;
    }
    void link(int x, int y){
        make_root(x);
        splay(x);
        if(find_root(y) == x) return;
        par[x] = y;
    }
    void cut(int x, int y){
        make_root(x);
        access(y);
        splay(x);
        if(par[y] != x || ch[y][0]) return;
        ch[x][1] = par[y] = 0;
    }
}

```

```

}
int find_root(int x){
    access(x);
    splay(x);
    push(x);
    while(ls) x = ls, push(x);
    splay(x);
    return x;
}
void split(int x, int y){
    make_root(x);
    access(y);
    splay(y);
}
void upd(int x, int y){
    access(x);
    splay(x);
    val[x] = y;
    pull(x);
}
}
} st;

```

2.2 LiChaoST

```

struct line{
    ll m, k;
    line(){}
    line(ll _m, ll _k) : m(_m), k(_k){}
    ll val(ll x){ return m * x + k; }
};

struct node{
    line ans;
    node *l, *r;
    int siz;
    node(){}
    node(line l) : ans(l), l(nullptr), r(nullptr){ }
};

node sgt[maxn];

int root[maxn], cnt = 0;

struct segtree{
    node *rt;
    int n, siz;
    segtree() : n(maxc * 2), siz(0), rt(nullptr){}
    void insert(node* &k, int l, int r, line cur){
        if(!k){
            k = new node(cur);
            siz++;
            return;
        }
        if(l == r){
            if(k->ans.val(l) > cur.val(l)) k->ans = cur;
            return;
        }
        int m = (l + r) / 2;
        if(k->ans.val(m) > cur.val(m)) swap(k->ans, cur);
        if(cur.m > k->ans.m) insert(k->l, l, m, cur);
        else insert(k->r, m + 1, r, cur);
    }
    void insert
        (ll m, ll k) { insert(rt, 0, n, line(m, k)); }
    void insert(line l) { insert(rt, 0, n, l); }
    ll qry(node *k, int l, int r, int pos){
        if(!k) return INF;
        if(l == r) return k->ans.val(pos);
        int m = (l + r) / 2;
        return min(k->ans.val(pos), pos <= m ? qry
            (k->l, l, m, pos) : qry(k->r, m + 1, r, pos));
    }
    ll qry(int pos) { return qry(rt, 0, n, pos); }
};

```

2.3 Leftist Heap

```

struct LeftistTree{
    int cnt, rt[maxn]
    ], lc[maxn * 20], rc[maxn * 20], d[maxn * 20];
    int v[maxn * 20];
    LeftistTree(){}
    int newnode(pll nd){
        cnt++;
        v[cnt] = nd;
        return cnt;
    }
    int merge(int x, int y){

```

```

    if(!x || !y) return x + y;
    if(v[x] > v[y]) swap(x, y);
    int p = ++cnt;
    lc[p] = lc[x], v[p] = v[x];
    rc[p] = merge(rc[x], y);
    if(d[lc[p]] < d[rc[p]]) swap(lc[p], rc[p]);
    d[p] = d[rc[p]] + 1;
    return p;
}
} st;

```

3 Graph

3.1 Dominator Tree

```

int in[maxn], id[maxn], par[maxn], dfn = 0;
int mn[maxn], idom[maxn], sdom[maxn], ans[maxn];
int fa[maxn]; // dsu
int n, m;

struct edge{
    int to, id;
    edge(){}
    edge(int _to, int _id) : to(_to), id(_id){}
};
vector<edge> adj[3][maxn];

void dfs(int pos){
    in[pos] = ++dfn;
    id[dfn] = pos;
    for(auto [x, id] : adj[0][pos]){
        if(in[x]) continue;
        dfs(x);
        par[x] = pos;
    }
}

int find(int x){
    if(fa[x] == x) return x;
    int tmp = fa[x];
    fa[x] = find(fa[x]);
    if(in[sdom[mn[tmp]]] < in[sdom[mn[x]]]){
        mn[x] = mn[tmp];
    }
    return fa[x];
}

void tar(int st){
    dfs(st);
    for(int i = 0; i < n; i++) mn[i] = sdom[i] = fa[i] = i;
    for(int i = dfn; i >= 2; i--){
        int pos = id[i], res = INF; // res : in(x) of sdom
        for(auto [x, id] : adj[1][pos]){
            if(!in[x]) continue;
            find(x);
            if(in[pos] > in[x]) res = min(res, in[x]);
            else res = min(res, in[sdom[mn[x]]]);
        }
        sdom[pos] = id[res];
        fa[pos] = par[pos];
        adj[2][sdom[pos]].eb(pos, 0);
        pos = par[pos];
        for(auto [x, id] : adj[2][pos]){
            find(x);
            if(sdom[mn[x]] == pos){
                idom[x] = pos;
            }
            else{
                idom[x] = mn[x];
            }
        }
        adj[2][pos].clear();
    }
    for(int i = 2; i <= dfn; i++){
        int x = id[i];
        if(idom[x] != sdom[x]) idom[x] = idom[idom[x]];
    }
}

```

3.2 MaximumClique

```

struct MaximumClique{
    typedef bitset<maxn> bst;
    bst adj[maxn], empt;
    int p[maxn], n, ans;
    void init(int _n){

```

```

        n = _n;
        for(int i = 0; i < n; i++) adj[i].reset();
    }
    void BronKerbosch(bst R, bst P, bst X){
        if(P == empt && X == empt){
            ans = max(ans, (int)R.count());
            return;
        }
        bst tmp = P | X;
        if((R | P | X).count() <= ans) return;
        int u;
        for(int i = 0; i < n; i++){
            if(tmp[u = p[i]]) break;
        }
        bst lim = P & ~adj[u];
        for(int i = 0; i < n; i++){
            int v = p[i];
            if(lim[v]){
                R[v] = 1;
                BronKerbosch
                    (R, P & adj[v], X & adj[v]);
                R[v] = 0, P[v] = 0, X[v] = 1;
            }
        }
    }
    void add_edge(int a, int b){
        adj[a][b] = adj[b][a] = 1;
    }
    int solve(){
        bst R, P, X;
        ans = 0, P.flip();
        iota(p, p + n, 0);
        random_shuffle
            (p, p + n), BronKerbosch(R, P, X);
        return ans;
    }
};

```

4 Flow / Matching

4.1 Dinic

```

struct Dinic{
    struct edge{
        ll to, cap;
        edge(){}
        edge(int _to, ll _cap) : to(_to), cap(_cap){}
    };
    vector<edge> e;
    vector<vector<int>> adj;
    vector<int> iter, level;
    int n, s, t;
    void init(int _n, int _s, int _t){
        n = _n, s = _s, t = _t;
        adj = vector<vector<int>>(n);
        iter = vector<int>(n);
        level = vector<int>(n);
        e.clear();
    }
    void add_edge(int from, int to, ll cap){
        adj[from].pb(e.size());, adj[to].pb(e.size() + 1);
        e.pb(edge(to, cap)), e.pb(edge(from, 0));
    }
    void bfs(){
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        queue<int> q;
        q.push(s);
        while(!q.empty()){
            int cur = q.front(); q.pop();
            for(auto id : adj[cur]){
                auto [to, cap] = e[id];
                if(level[to] == -1 && cap){
                    level[to] = level[cur] + 1;
                    q.push(to);
                }
            }
        }
    }
    ll dfs(int pos, ll flow){
        if(pos == t) return flow;
        for(int &i = iter[pos]; i < adj[pos].size(); i++){
            auto [to, cap] = e[adj[pos][i]];
            if(level[to] == level[pos] + 1 && cap){
                ll tmp = dfs(to, min(flow, cap));
                if(tmp){
                    e[adj[pos][i]].cap -= tmp;

```

```

        e[adj[pos][i] ^ 1].cap += tmp;
        return tmp;
    }
}
return 0;
}
ll flow(){
    ll ret = 0;
    while(true){
        bfs();
        if(level[t] == -1) break;
        fill(iter.begin(), iter.end(), 0);
        ll tmp;
        while((tmp = dfs(s, INF)) > 0){
            ret += tmp;
        }
    }
    return ret;
}
vector<pll> cut(){
    vector<pll> ret;
    fill(level.begin(), level.end(), -1);
    level[s] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int cur = q.front(); q.pop();
        for(auto id : adj[cur]){
            auto [to, cap] = e[id];
            if(cap == 0 && level
                [to] == -1) ret.pb({e[id ^ 1].to, to});
            else if(level[to] == -1){
                level[to] = level[cur] + 1;
                q.push(to);
            }
        }
    }
    return ret;
}
} flow;

```

4.2 Min Cost Max Flow

```

struct MCMF{
    using T = ll;
    struct edge{
        int to;
        T cap, cost;
        edge(){}
        edge(int _to, T _cap, T
            _cost) : to(_to), cap(_cap), cost(_cost){}
    };
    vector<edge> e;
    vector<vector<int>> adj;
    vector<int> iter, inq;
    vector<T> dist;
    int n, s, t;
    void init(int _n, int _s, int _t){
        n = _n, s = _s, t = _t;
        adj = vector<vector<int>>(n);
        iter = vector<int>(n);
        dist = vector<T>(n);
        inq = vector<int>(n);
        e.clear();
    }
    void add_edge(int from, int to, T cap, T cost = 0){
        adj[from
            ].pb(e.size()), adj[to].pb(e.size() + 1);
        e.pb(edge(to
            , cap, cost)), e.pb(edge(from, 0, -cost));
    }
    bool spfa(){
        fill(dist.begin(), dist.end(), INF);
        queue<int> q;
        q.push(s);
        dist[s] = 0, inq[s] = 1;
        while(!q.empty()){
            int pos = q.front(); q.pop();
            inq[pos] = 0;
            for(auto id : adj[pos]){
                auto [to, cap, cost] = e[id];
                if(cap && dist[to] > dist[pos] + cost){
                    dist[to] = dist[pos] + cost;
                    if(!inq
                        [to]) q.push(to), inq[to] = 1;
                }
            }
        }
    }
}

```

```

    }
    return dist[t] != INF;
}
T dfs(int pos, T flow){
    if(pos == t) return flow;
    inq[pos] = 1;
    for(int
        &i = iter[pos]; i < adj[pos].size(); i++){
        auto [to, cap, cost] = e[adj[pos][i]];
        if(!inq[to] &&
            dist[to] == dist[pos] + cost && cap){
            T tmp = dfs(to, min(flow, cap));
            if(tmp){
                inq[pos] = 0;
                e[adj[pos][i]].cap -= tmp;
                e[adj[pos][i] ^ 1].cap += tmp;
                return tmp;
            }
        }
    }
    inq[pos] = 0;
    return 0;
}
pair<T, T> mcmf(){
    T flow = 0, cost = 0;
    while(true){
        if(!spfa()) break;
        fill(iter.begin(), iter.end(), 0);
        T tmp;
        while((tmp = dfs(s, INF)) > 0){
            flow += tmp, cost += tmp * dist[t];
        }
    }
    return {flow, cost};
}
} flow;

```

4.3 Kuhn Munkres

```

struct Hungarian{
    using T = ll;
    vector<T> lx, ly, slack;
    vector<int> vx, vy, match;
    vector<vector<T>> w;
    queue<int> q;
    int n;
    void init(int _n){
        n = _n;
        lx.resize(n), ly.resize(n), slack.resize(n);
        vx.resize
            (n), vy.resize(n), match.resize(n, -1);
        w.resize(n, vector<T>(n));
    }
    void inp(int x, int y, int val){
        w[x][y] = val;
        lx[x] = max(lx[x], val);
    }
    int dfs(int x){
        if(vx[x]) return false;
        vx[x] = 1;
        for(int i = 0; i < n; i++){
            if(lx[x] + ly[i] == w[x][i] && !vy[i]){
                vy[i] = true;
                if(match[i] == -1 || dfs(match[i])){
                    match[i] = x;
                    return true;
                }
            }
        }
        return false;
    }
    int pdfs(int x){
        fill(vx.begin(), vx.end(), 0);
        fill(vy.begin(), vy.end(), 0);
        return dfs(x);
    }
    void upd(int x){
        for(int i = 0; i < n; i++){
            if(!slack[i]) continue;
            slack[i] =
                min(slack[i], lx[x] + ly[i] - w[x][i]);
            if(!slack[i] && !vy[i]) q.push(i);
        }
    }
    void relabel(){

```

```

T mn = numeric_limits<T>::max() / 3;
for(int i = 0; i < n; i++){
    if(!vy[i]) mn = min(mn, slack[i]);
}
for(int i = 0; i < n; i++){
    if(vx[i]) lx[i] -= mn;
    if(vy[i]) ly[i] += mn;
    else{
        slack[i] -= mn;
        if(!slack[i]) q.push(i);
    }
}
}
auto solve(){
    for(int i = 0; i < n; i++){
        if(pdfs(i)) continue;
        while(!q.empty()) q.pop();
        fill(slack.begin(), slack.end(), INF);
        for(int j = 0; j < n; j++) if(vx[j]) upd(j);
        int ok = 0;
        while(!ok){
            relabel();
            while(!q.empty()){
                int j = q.front(); q.pop();
                if(match[j] == -1){
                    pdfs(i);
                    ok = 1;
                    break;
                }
            }
            vy[j] = vx
            [match[j]] = 1, upd(match[j]);
        }
    }
    T ans = 0;
    for(int i = 0; i < n; i++){
        ans += w[match[i]][i];
    }
    for(int i = 0; i < n; i++) lx[match[i]] = i;
    return make_pair(ans, lx);
}
} h;

```

4.4 Hopcroft Karp

```

int mx[maxn], my[maxn], dx[maxn], dy[maxn], vis[maxn];
vector<int> adj[maxn];
int l, r, m;

int dfs(int pos){
    for(auto x : adj[pos]){
        if(!vis[x] && dy[x] == dx[pos] + 1){
            vis[x] = 1;
            if(my[x] != -1 && dy[x] == lim) continue;
            if(my[x] == -1 || dfs(my[x])){
                my[x] = pos, mx[pos] = x;
                return true;
            }
        }
    }
    return false;
}

int bfs(){
    fill(dx, dx + l, -1);
    fill(dy, dy + r, -1);
    queue<int> q;
    for(int i = 0; i < l; i++){
        if(mx[i] == -1) dx[i] = 0, q.push(i);
    }
    lim = INF;
    while(!q.empty()){
        int pos = q.front(); q.pop();
        if(dx[pos] > lim) break;
        for(auto x : adj[pos]){
            if(dy[x] == -1){
                dy[x] = dx[pos] + 1;
                if(my[x] == -1) lim = dy[x];
            }
            else dx
                [my[x]] = dy[x] + 1, q.push(my[x]);
        }
    }
    return lim != INF;
}

```

```

void Hopcroft_Karp(){
    int res = 0;
    for(int i = 0; i < l; i++) mx[i] = -1;
    for(int i = 0; i < r; i++) my[i] = -1;
    while(bfs()){
        fill(vis, vis + l + r, 0);
        for(int i = 0; i < l; i++){
            if(mx[i] == -1 && dfs(i)) res++;
        }
    }
}

```

5 String

5.1 Z-Value

```

vector<int> z(string s){
    vector<int> z(s.size());
    int x = 0, y = 0;
    for(int i = 1; i < s.size(); i++){
        z[i] = max(0LL, min(z[i - x], y - i));
        while(i + z[i] < s.size() && s[i + z[i]] == s[z[i]]){
            x = i, y = i + z[i], z[i]++;
        }
    }
    return z;
}

```

5.2 Manacher

```

vector<int> manacher(string s){
    int n = 2 * s.size() + 1;
    string ss(n, '#');
    for(int i = 0; i < n / 2; i++) ss[i * 2 + 1] = s[i];
    swap(s, ss);
    vector<int> f(n);
    int m = 0, len = 0;
    for(int i = 0; i < n; i++){
        f[i] = max(0LL, min(f[m + m - i], m + len - i));
        while(i + f[i] < n && i - f[i] >= 0 && s[i + f[i]] == s[i - f[i]]){
            m = i, len = f[i], f[i]++;
        }
    }
    return f;
}

```

5.3 Suffix Array

```

struct SuffixArray{
    int ch[2][maxn], sa[maxn], cnt[maxn], n;
    string s;
    void init(string _s){
        s = _s, n = s.size();
        Get_SA();
        Get_LCP();
    }
    void Get_SA(){
        int *x = ch[0], *y = ch[1], m = 256;
        for(int i = 0; i < m; i++) cnt[i] = 0;
        for(int i = 0; i < n; i++) cnt[x[i]]++;
        for(int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
        for(int i = 0; i < n; i++) sa[--cnt[x[i]]] = i;
        for(int k = 1; k <= 1){
            for(int i = 0; i < m; i++) cnt[i] = 0;
            for(int i = 0; i < n; i++) cnt[x[i]]++;
            for(int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
            int p = 0;
            for(int i = n - k; i < n; i++) y[p++] = i;
            for(int i = 0; i < n; i++)
                i++ if(sa[i] >= k) y[p++] = sa[i] - k;
            for(int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
            y[sa[0]] = p = 0;
            for(int i = 1; i < n; i++){
                int a = sa[i], b = sa[i - 1];
                if(a + k < n && b + k < n && x[a]
                    == x[b] && x[a + k] == x[b + k]);
                else p++;
                y[a] = p;
            }
            if(p == n - 1) break;
        }
    }
}

```

```

        swap(x, y);
        m = p + 1;
    }
}
int rnk[maxn], lcp[maxn];
void Get_LCP(){
    for(int i = 0; i < n; i++) rnk[sa[i]] = i;
    int val = 0;
    for(int i = 0; i < n; i++){
        if(val) val--;
        if(!rnk[i]){
            lcp[0] = val = 0;
            continue;
        }
        int b = sa[rnk[i] - 1];
        while(b + val < n && i + val
              < n && s[b + val] == s[i + val]) val++;
        lcp[rnk[i]] = val;
    }
}
} sa;

```

5.4 Suffix Automaton

```

struct SuffixAutomaton{
    int len[maxn], link[maxn]; // maxn >= 2 * n - 1
    int cnt[maxn], distinct[maxn];
    bool is_clone[maxn];
    int first_pos[maxn];
    vector<int> inv_link[maxn]; //suffix references
    map<char, int> nxt[maxn];
    int sz = 1, last = 0;
    void init(string s){
        link[0] = -1;
        for(auto x : s) sa_extend(x);
    }
    void sa_extend(char c){
        int cur = sz++;
        len[cur] = len[last] + 1;
        firstpos[cur] = len[cur] - 1;
        int p = last;
        while(p != -1 && !nxt[p].count(c)){
            nxt[p][c] = cur;
            p = link[p];
        }
        if(p == -1) link[cur] = 0;
        else{
            int q = nxt[p][c];
            if(len[q] == len[p] + 1) link[cur] = q;
            else{
                int clone = sz++;
                is_clone[clone] = true;
                firstpos[clone] = q;
                len[clone] = len[p] + 1;
                nxt[clone] = nxt[q];
                link[clone] = link[q];
                while(p != -1 && nxt[p][c] == q) {
                    nxt[p][c] = clone;
                    p = link[p];
                }
                link[cur] = link[q] = clone;
            }
        }
        last = cur;
    }
}
//Applications
ll getDistinct
(int pos){ // number of distinct substrings
           starting at pos, including an empty string
    if(distinct[pos]) return distinct[pos];
    distinct[pos] = 1;
    for(auto [c, next]
        : nxt[pos]) distinct[pos] += getDistinct(next);
    return cnt[pos];
}
ll numDistinct(){
    return getDistinct
        (0) - 1; // excluding an empty string
}
ll numDistinct2(){
    ll tot = 0;
    for(int i
        = 1; i < sz; i++) tot += len[i] - len[link[i]];
    return tot;
}
string distinct_kth(ll k){
    numDistinct();

```

```

string s;
ll cur = 0, tally = 0;
while(tally < k){
    for(auto [c, next] : nxt[cur]){
        if(tally + distinct[next] >= k){
            tally += 1;
            s += c;
            cur = next;
            break;
        }
        tally += distinct[next];
    }
}
return s;
}
//inverse links
void genLink(){
    for(int i = 1; i < sz; i++){
        st[st[v].link].inv.pb(v);
    }
}

void get_all_occur(vector<int>& oc, int v){
    if(!st[v].is_clone) oc.pb(st[v].first_pos);
    for(auto
        u : st[v].inv_link) get_all_occur(oc, u);
}
vector<int> all_occ(string
s){ // get all occurrences of s in automaton
    int cur = 0;
    for(auto x : s){
        if(!st[cur].next.count(x)) return {};
        cur = st[cur].next[x];
    }
    vector<int> oc;
    get_all_occur(oc, cur);
    for(auto &x : oc) x += 1 -
        s.length(); //translate to starting positions
    sort(oc.begin(), oc.end());
    return oc;
}
int lcs(string t){
    int v = 0, l = 0, ans = 0;
    for(auto x : t){
        while(v && !nxt[v].count(x)){
            v = link[v];
            l = len[v];
        }
        if(nxt[v].count(x)){
            v = nxt[v][x];
            l++;
        }
        ans = max(ans, l);
    }
    return ans;
}
};

```

5.5 Palindrome Tree

```

struct EERTREE{
    int sz, tot, last;
    int cnt[maxn], ch[maxn][26],
        len[maxn], fail[maxn], dif[maxn], slink[maxn];
    int g[maxn], dp[maxn];
    char s[maxn];
    int node(int l){
        sz++;
        memset(ch[sz], 0, sizeof(ch[sz]));
        len[sz] = l;
        fail[sz] = cnt[sz] = 0;
        return sz;
    }
    void init(){
        sz = -1;
        last = 0;
        s[tot = 0] = '$';
        node(0);
        node(-1);
        fail[0] = 1;
    }
    int getfail(int x){
        while(s[tot - len[x] - 1] != s[tot]) x = fail[x];
        return x;
    }
    void insert(char c){
        s[++tot] = c;

```

```

int now = getfail(last);
if(!ch[now][c - 'a']){
    int x = node(len[now] + 2);
    fail[x] = ch[getfail(fail[now])][c - 'a'];
    ch[now][c - 'a'] = x;
    dif[x] = len[x] - len[fail[x]];
    if(dif[x] == dif[fail[x]]){
        slink[x] = slink[fail[x]];
    }
    else slink[x] = fail[x];
}
last = ch[now][c - 'a'];
cnt[last]++;
}
int process(string s){
    for(int i = 0; i < s.size(); i++){
        insert(s[i]);
        dp[i] = INF;
        for(int x = last; x > 1; x = slink[x]){
            if(i - len[slink[x]] - dif[x] >= 0) g[x] = dp[i - len[slink[x]] - dif[x]];
            if(dif[x] == dif[fail[x]]) g[x] = min(g[x], g[fail[x]]);
            dp[i] = min(dp[i], g[x] + 1);
        }
    }
    return dp[s.size() - 1];
}
} pam;

```

6 Math

6.1 Miller Rabin

```

using u64 = uint64_t;
using u128 = __uint128_t;

u64 fpow(u64 a, u64 b, u64 n){
    u64 ret = 1;
    while(b > 0){
        if(b & 1) ret = (u128)ret * a % n;
        a = (u128)a * a % n;
        b >>= 1;
    }
    return ret;
}

bool check_composite(u64 n, u64 a, u64 d, int s){
    u64 x = fpow(a, d, n);
    if(x == 1 || x == n - 1) return false;
    for(int r = 1; r < s; r++){
        x = (u128)x * x % n;
        if(x == n - 1) return false;
    }
    return true;
}

bool MillerRabin(u64 n){
    if(n < 2) return false;
    int s = 0;
    u64 d = n - 1;
    while(!(d & 1)){
        d >>= 1;
        s++;
    }
    for(auto a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){ // sufficient for n < 2^64
        if(n == a) return true;
        if(check_composite(n, a, d, s)) return false;
    }
    return true;
}

```

6.2 Pollard Rho

```

ll f(ll t, ll c, ll n){
    return (t * t + c) % n;
}

ll Pollard_Rho(ll x){
    ll t = 0;
    ll c = rand() % (x - 1) + 1;
    ll s = t;
    ll val = 1;
    for(int goal = 1;; goal <= 1, s = t, val = 1){
        for(int step = 1; step <= goal; step++){
            t = f(t, c, x);
            val = val * abs(t - s) % x;
            if(!val) return x;
        }
    }
}

```

```

if(step % 127 == 0){
    ll d = __gcd(val, x);
    if(d > 1) return d;
}
}
ll d = __gcd(val, x);
if(d > 1) return d;
}
}

```

6.3 Ext GCD

```

ll extgcd(ll a, ll b, ll &x, ll &y){
    if(b == 0){
        x = 1, y = 0;
        return a;
    }
    int res = extgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return res;
}

```

6.4 Chinese Remainder Theorem

```

ll CRT(vector<ll> p, vector<ll> a){
    ll n = p.size(), prod = 1, ret = 0;
    for(int i = 0; i < n; i++) prod *= p[i];
    for(int i = 0; i < n; i++){
        ll m = (prod / p[i]);
        ll x, y;
        extgcd(m, p[i], x, y);
        ret = ((ret + a[i] * m * x) % prod + prod) % prod;
    }
    return ret;
}

```

6.5 Powerful Number Sieve

```

void linearsieve(){
    phi[1] = 1;
    for(int i = 2; i < maxn; i++){
        if(!lp[i]) pr.pb(i), lp[i] = i, phi[i] = i - 1;
        for(auto x : pr){
            if(i * x >= maxn) break;
            lp[i * x] = x;
            if(lp[i] == x){
                phi[i * x] = phi[i] * x;
                break;
            }
            phi[i * x] = phi[i] * (x - 1);
        }
    }
    for(int i = 1; i < maxn; i++){
        sum[i] = (sum[i - 1] + i * phi[i]) % N;
    }
}

int s2(int n){
    static const int inv6 = inv(6);
    n %= N;
    return n * (n + 1) % N * (2 * n + 1) % N * inv6 % N;
}

int G(int n){
    static const int inv2 = inv(2);
    if(n < maxn) return sum[n];
    if(mp_G.count(n)) return mp_G[n];
    int ans = s2(n);
    for(int i = 2, j; i <= n; i = j + 1){
        j = n / (n / i);
        (ans -= (i + j) % N * (j - i + 1) % N * inv2 % N * G(n / i) % N - N) %= N;
    }
    return mp_G[n] = ans;
}

void dfs(int d, int hd, int p){ // dfs 出所有 PN
    (ans += hd * G(n / d)) %= N;
    for(int i = p; i < pr.size(); i++){
        if(d > n / pr[i] / pr[i]) break;
        int c = 2;
        for(int x = d * pr[i] * pr[i]; x <= n; x *= pr[i], c++){
            if(!vis[i][c]){
                int f = fpow(pr[i], c);
                f = f * (f - 1) % N;
                int g = pr[i] * (pr[i] - 1) % N;
                int t = pr[i] * pr[i] % N;
            }
        }
    }
}

```

```

    for(int j = 1; j <= c; j++){
        (f -= g * h[i][c - j] % N - N) %= N;
        (g *= t) %= N;
    }
    h[i][c] = f;
    vis[i][c] = true;
}
if(h[i][c]) dfs(x, hd * h[i][c] % N, i + 1);
}
}

linearsieve();
for(int i = 0; i < pr.size(); i++) h[i][0] = 1;
dfs(1, 1, 0);

```

6.6 Fast Walsh Transform

```

void fwt(vector<int> &a, bool inv){
    int n = 1;
    while(n < a.size()) n *= 2;
    a.resize(n);
    for(int len = 1; 2 * len <= n; len <= 1){
        for(int i = 0; i < n; i += 2 * len){
            for(int j = 0; j < len; j++){
                int &u =
                    a[i + j], &v = a[i + j + len]; tie(u, v) =
                    // inv ? pll(u - v, v) : pll(u + v, v); // and
                    // inv ? pll(u, v - u) : pll(u, u + v); // or
                    pll(u + v, u - v); // xor
            }
        }
    }
    if(inv) for(auto &x : a) x /= n; // xor only
}

```

6.7 Floor Sum

```

//f(n, a, b, c) = sum_{0<=i<=n}{(ai + b)/c},
//g(n, a, b, c) = sum_{0<=i<=n}{i(ai + b)/c},
//h(n, a, b, c) = sum_{0<=i<=n}{((ai + b)/c)^2},
const int N = 998244353;
const int i2 = (N + 1) / 2, i6 = 166374059;
struct info{
    ll f, g, h;
    info(){f = g = h = 0;}
};
info calc(ll n, ll a, ll b, ll c){
    ll ac = a / c, bc = b / c,
        m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
    info d;
    if(a == 0){
        d.f = bc * n1 % N;
        d.g = bc * n % N * n1 % N * i2 % N;
        d.h = bc * bc % N * n1 % N;
        return d;
    }
    if(a >= c || b >= c){
        d.f = n * n1 % N * i2 % N * ac % N + bc * n1 % N;
        d.g = ac * n % N * n1 % N * n21
            % N * i6 % N + bc * n % N * n1 % N * i2 % N;
        d.h = ac * ac
            % N * n % N * n1 % N * n21 % N * i6 % N + bc *
            bc % N * n1 % N + ac * bc % N * n % N * n1 % N;

        info e = calc(n, a % c, b % c, c);

        d.h +=
            e.h + 2 * bc * e.f % N + 2 * ac % N * e.g % N;
        d.g += e.g, d.f += e.f;
        d.f %= N, d.g %= N, d.h %= N;
        return d;
    }
    info e = calc(m - 1, c, c - b - 1, a);
    d.f = (n * m % N - e.f + N) % N;
    d.g = m * n % N *
        n1 % N - e.h - e.f; d.g = (d.g * i2 % N + N) % N;
    d.h = n * m % N * (m + 1) % N -
        2 * e.g - 2 * e.f - d.f; d.h = (d.h % N + N) % N;
    return d;
}

```

7 Geometry

7.1 Basic

```

struct pt{

```

```

    double x, y;
    pt(){}
    pt(double _x, double _y) : x(_x), y(_y){}
};

pt operator + (pt a, pt b)
{ return pt(a.x + b.x, a.y + b.y); }
pt operator - (pt a, pt b)
{ return pt(a.x - b.x, a.y - b.y); }
pt operator * (pt a, double p)
{ return pt(a.x * p, a.y * p); }
pt operator / (pt a, double p)
{ return pt(a.x / p, a.y / p); }
bool operator < (const pt &a, const pt &b)
{ return a.x < b.x || (a.x == b.x && a.y < b.y); }
double dot(pt a, pt b)
{ return a.x * b.x + a.y * b.y; }
double cross(pt a, pt b)
{ return a.x * b.y - a.y * b.x; }
double len(pt a)
{ return sqrt(dot(a, a)); }
double angle(pt a, pt b)
{ return acos(dot(a, b) / len(a) / len(b)); }
double area2(pt a, pt b, pt c)
{ return cross(b - a, c - a); }

const double eps = 1e-10;
int dcmp(double x){
    if(fabs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}

bool operator == (const pt &a, const pt &b) {
    return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
}

inline int ori(pt a, pt b, pt c){
    double area = cross(b - a, c - a);
    if(area > -eps && area < eps) return 0;
    return area > 0 ? 1 : -1;
}

inline int btw(pt a, pt b, pt c){ // [a, c, b]
    if(fabs(cross(b - a, c - a)) > eps) return false;
    if(dot(b - a, c - a)
        > -eps && len(c - a) <= len(b - a)) return true;
    return false;
}

bool intersect(pt a, pt b, pt c, pt d){
    if(a == c || a == d || b == c || b == d) return true;
    int a123 = ori(a, b, c), a124 = ori(a,
        b, d), a341 = ori(c, d, a), a342 = ori(c, d, b);
    if(a123 == 0 && a124 == 0){
        if(btw(a, b, c) || btw(a, b, d)
            || btw(c, d, a) || btw(c, d, b)) return true;
        else return false;
    }
    else if(a123
        * a124 <= 0 && a341 * a342 <= 0) return true;
    return false;
}

istream &operator >>(istream &s, pt &a){
    s >> a.x >> a.y;
    return s;
}

```

7.2 Minkowski Sum

```

void reorder(vector<pt> &a){
    int pos = 0;
    for(int j = 1; j < a.size(); j++){
        if(a[j].x < a[pos].x || (a[j].x
            == a[pos].x && a[j].y < a[pos].y)) pos = j;
    }
    rotate(a.begin(), a.begin() + pos, a.end());
}

vector<pt> minkowski(vector<pt> a, vector<pt> b){
    // for(int i = 0;
        i < b.size(); i++) b[i] = {-b[i].x, -b[i].y};
    // 最短距離: 把 Q 鏡像, 找凸包到 (0, 0) 的最短距離
    reorder(a), reorder(b);
    a.pb(a[0]), a.pb(a[1]);
    b.pb(b[0]), b.pb(b[1]);
    vector<pt> res;
    int i = 0, j = 0;

```

```
while(i < a.size() - 2 || j < b.size() - 2){  
    res.pb(a[i] + b[j]);  
    int 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 res;  
}
```