

# Team notebook

September 13, 2018

## Contents

<b>1 Algorithms</b>	<b>1</b>
1.1 Longest Increasing Subsequence . . . . .	1
1.2 Merge Sort . . . . .	1
1.3 Mo's Algorithm . . . . .	1
1.4 Quick Sort . . . . .	2
1.5 Ternary Search . . . . .	2
<b>2 Data Structures</b>	<b>3</b>
2.1 Fenwick Tree . . . . .	3
2.2 Segment Tree . . . . .	3
2.3 Sparse Table . . . . .	4
2.4 Trie . . . . .	5
2.5 Union Find . . . . .	5
<b>3 Graphs</b>	<b>5</b>
3.1 Bellman Ford . . . . .	5
3.2 Edmonds Karp Max Flow . . . . .	6
3.3 Euler Path . . . . .	6
3.4 Hamiltonian Path . . . . .	6
3.5 LCA Binary Lifting . . . . .	7
3.6 Max Cardinality Bipartite Matching . . . . .	7
3.7 Minimum Spanning Tree . . . . .	7
3.8 Strongly Connected Components . . . . .	8
<b>4 Mathematics</b>	<b>8</b>
4.1 Combinatoric Formulas . . . . .	8
4.2 Combinatoric . . . . .	9
4.3 Cycle Finding . . . . .	9
4.4 Diophantine Equations . . . . .	9
4.5 Extended Euclidean Algorithm . . . . .	9

4.6 Linear Recurrence Equations . . . . .	9
4.7 Pillai Function . . . . .	9
<b>5 Strings</b>	<b>10</b>
5.1 KMP Algorithm . . . . .	10
5.2 Rolling Hashing . . . . .	10
5.3 Split Line . . . . .	10
5.4 Suffix Array . . . . .	10
5.5 Z Algorithm . . . . .	12
<b>6 Template</b>	<b>12</b>
6.1 Main . . . . .	12

## 1 Algorithms

### 1.1 Longest Increasing Subsequence

---

```
// O(n ** 2)
int LIS(vector<int> &v) {
    int n = v.size();
    vector<int> lis(n, 1);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j++)
            if (v[j] < v[i])
                lis[i] = max(lis[i], lis[j] + 1);
    return *max_element(lis.begin(), lis.end());
}

// O(n * log(n))
int LIS(vector<int> &v) {
    int n = v.size(), lis = 0;
    vector<int> L(n);
```

```

    for (int i = 0; i < n; i++) {
        int pos = lower_bound(L.begin(), L.begin() + lis, v[i]) -
            L.begin();
        L[pos] = v[i];
        lis = max(pos + 1, lis);
    }
    return lis;
}

```

---

## 1.2 Merge Sort

```

// O(n * lg(n))
void merge_sort(vector<int> &v, int l, int r) {
    if (l + 1 == r) return;
    int m = (l + r) / 2;
    merge_sort(v, l, m);
    merge_sort(v, m, r);
    vector<int> tmp(r - l);
    for (int i = l, j = m, k = 0; i < m or j < r; k++) {
        if (j == r || (i < m && v[i] <= v[j])) {
            tmp[k] = v[i];
            i++;
        } else {
            tmp[k] = v[j];
            j++;
        }
    }
    for (int i = 0; i < (int)tmp.size(); i++)
        v[l + i] = tmp[i];
}

```

---

## 1.3 Mo's Algorithm

```

// O(n * sqrt(n))
const int ROOT = 200;

struct Query {
    int id;
    int L, R;
    Query() {}
    Query(int _id, int _L, int _R) : id(_id), L(_L), R(_R) {}
}

```

---

```

};

bool cmp(Query a, Query b) {
    if (a.L / ROOT == b.L / ROOT)
        return a.R < b.R;
    return a.L / ROOT < b.L / ROOT;
}

void mo(vi &a, vector<Query> &qs, vi &ans) {
    sort(qs.begin(), qs.end(), cmp);
    int currL = 0, currR = 0;
    int currSum = 0;
    fori(i, 0, SZ(qs)) {
        int L = qs[i].L, R = qs[i].R;
        while (currL < L) {
            currSum -= a[currL];
            currL++;
        }
        while (currL > L) {
            currSum += a[currL - 1];
            currL--;
        }
        while (currR <= R) {
            currSum += a[currR];
            currR++;
        }
        while (currR > R + 1) {
            currSum -= a[currR - 1];
            currR--;
        }
        ans[qs[i].id] = currSum;
    }
}

```

---

## 1.4 Quick Sort

```

// O(n ** 2) | Theta(n * lg(n))
void quick_sort(vector<int> &v, int l, int r) {
    int key = v[l + rand() % (r - l + 1)];
    int i = l, j = r;
    do {
        while (v[i] < key) i++;
        while (v[j] > key) j--;
    }
}

```

---

```

        if (i <= j) {
            swap(v[i], v[j]);
            i++;
            j--;
        }
    } while (i <= j);
    if (l < j) quickSort(l, j);
    if (i < r) quickSort(i, r);
}

```

---

## 1.5 Ternary Search

---

```

// O(log n) Integers
int ternary_search(vector<int> &v) {
    int n = v.size();
    int lo = 0, hi = n - 1;
    while (hi - lo > 1) {
        int mid = (lo + hi) >> 1;
        if (v[mid] < v[mid + 1])
            lo = mid;
        else
            hi = mid;
    }
    return lo + 1;
}

// O(log n) Floatings
double ternary_search(args) {
    double lo = 0.0, hi = oo;
    for (int i = 0; i < num_iterations; i++) {
        double mid1 = (lo * 2 + hi) / 3; // 1 + (r - 1) / 3
        double mid2 = (lo + 2 * hi) / 3; // r - (r - 1) / 3
        if (f(mid1, args) < f(mid2, args))
            lo = mid1;
        else
            hi = mid2;
    }
    return lo;
}

```

---

## 2 Data Structures

### 2.1 Fenwick Tree

---

```

struct FenwickTree {
    int n;
    vi data;
    FenwickTree(int _n) : n(_n), data(vi(n)) {}
    void update(int at, int by) {
        while (at < n) data[at] += by, at |= at + 1;
    }
    int query(int at) {
        int res = 0;
        while (at >= 0) res += data[at], at = (at & (at + 1)) - 1;
        return res;
    }
    int rsq(int a, int b) { return query(b) - query(a - 1); }
};

```

---

### 2.2 Segment Tree

---

```

struct SegmentTree {

    int n;
    vi A;
    vi tree;
    SegmentTree() {}

    SegmentTree(vi &_A) {
        A = _A;
        int n = SZ(_A);
        tree.assign(4 * n, 0);
        build(1, 0, n - 1);
    }

    build(int p, int i, int j) {
        if (i == j) {
            tree[p] = A[i];
        } else {
            int mid = (i + j) / 2;
            build(2 * p, i, mid);
            build(2 * p + 1, mid + 1, j);
        }
    }
}

```

```

        tree[p] = min(tree[2 * p], tree[2 * p + 1]);
    }
}

int query(int p, int i, int j, int L, int R) {
    if (j < L || i > R) return oo;
    if (L <= i && j <= R) return tree[p];
    int mid = (i + j) / 2;
    int q1 = query(2 * p, i, mid, L, R);
    int q2 = query(2 * p + 1, mid + 1, j, L, R);
    return min(q1, q2);
}

int query(int i, int j) { return query(1, 0, n - 1, i, j); }

void update(int p, int i, int j, int at, int by) {
    if (j < at || i > at) return;
    if (i == j) {
        A[i] = by;
        tree[p] = by;
        return;
    }
    int mid = (i + j) / 2;
    update(2 * p, i, mid, at, by);
    update(2 * p + 1, mid + 1, j, at, by);
    tree[p] = min(tree[2 * p], tree[2 * p + 1]);
}

void update(int at, int by) { return update(1, 0, n - 1, at, by); }

};

```

// Segment Tree Lazy Propagation

```

struct SegmentTree {

    int n;
    vi A;
    vi tree;
    vi lazy;

    SegmentTree(vi &_A) {
        A = _A;
        n = _A.size();
        tree.assign(4 * n, 0);
        lazy.assign(4 * n, 0);
    }

```

```

        build(1, 0, n - 1);
    }

    void build(int p, int i, int j) {
        if (i == j) {
            tree[p] = A[i];
        } else {
            int mid = (i + j) / 2;
            build(2 * p, i, mid);
            build(2 * p + 1, mid + 1, j);
            tree[p] = tree[2 * p] + tree[2 * p + 1];
        }
    }

    int query(int p, int i, int j, int L, int R) {
        if (j < L || R < i) return 0;
        if (lazy[p] != 0) {
            tree[p] += (j - i + 1) * lazy[p];
            if (i < j) {
                lazy[2 * p] += lazy[p];
                lazy[2 * p + 1] += lazy[p];
            }
            lazy[p] = 0;
        }
        if (L <= i && j <= R) return tree[p];
        int mid = (i + j) / 2;
        int q1 = query(2 * p, i, mid, L, R);
        int q2 = query(2 * p + 1, mid + 1, j, L, R);
        return q1 + q2;
    }

    int query(int i, int j) { return query(1, 0, n - 1, i, j); }

    void update(int p, int i, int j, int L, int R, int new_value) {
        if (lazy[p] != 0) {
            tree[p] += (j - i + 1) * lazy[p];
            if (i < j) {
                lazy[2 * p] += lazy[p];
                lazy[2 * p + 1] += lazy[p];
            }
            lazy[p] = 0;
        }
        if (j < L || R < i) return;
        if (L <= i && j <= R) {
            tree[p] += (j - i + 1) * new_value;

```

```

        if (i < j) {
            lazy[2 * p] += new_value;
            lazy[2 * p + 1] += new_value;
        }
        return;
    }
    int mid = (i + j) / 2;
    update(2 * p, i, mid, L, R, new_value);
    update(2 * p + 1, mid + 1, j, L, R, new_value);
    tree[p] = tree[2 * p] + tree[2 * p + 1];
}

void update(int i, int j, int new_value) { update(1, 0, n - 1, i,
j, new_value); }

};

```

## 2.3 Sparse Table

```

// RMQ
const int MN = 100000;
const int ML = 18;

int T[MN][ML];
int ln[MN];

void build(vi &v) {
    // log2 0(1)
    ln[2] = ln[3] = 1;
    for(i, 4, MN) ln[i] = 2 * ln[i / 2];
    // Sparse table
    int n = v.size();
    for (int i = 0; i < n; i++)
        T[i][0] = v[i];
    for (int j = 1, p = 2; p <= n; j++, p <= 1)
        for (int i = 0; i + p - 1 < n; i++)
            T[i][j] = min(T[i][j - 1], T[i + (p >> 1)][j - 1]);
}

int query(int l, int r) {
    int k = ln[r - l + 1];
    return min(T[l][k], T[r + 1 - (1 << k)][k]);
}

```

## 2.4 Trie

```

// Solve the problem of find v such that v <= lim and v ^ x is maximum
struct Node {
    int best;
    Node* children[2];
    Node() {
        best = oo;
        children[0] = children[1] = NULL;
    }
};

Node* root;

void insert(int x, int k) {
    Node* u = root;
    u->best = min(u->best, x);
    for (int i = 16; i >= 0; i--) {
        int c = (x >> i) & 1;
        if (u->children[c] == NULL)
            u->children[c] = new Node();
        u = u->children[c];
        u->best = min(u->best, x);
    }
}

int query(int x, int lim) {
    Node* u = root;
    int ans = 0;
    for (int i = 16; i >= 0; i--) {
        int c = (x >> i) & 1;
        Node* a = u->children[1 - c];
        Node* b = u->children[c];
        if (a != NULL && a->best <= lim) {
            u = a;
            ans |= (1 - c) << i;
        } else {
            u = b;
            ans |= c << i;
        }
    }
    return ans;
}

void init() {

```

```

    root = new Node();
}

```

---

## 2.5 Union Find

```

struct UnionFind {
    int num;
    vi p, size;
    UnionFind(int n) : p(n, -1), size(n, 1), num(n) {}
    int findSet(int i) { return p[i] < 0 ? i : (p[i] = findSet(p[i])); }
    void unionSet(int i, int j) { // p[findSet(i)] = findSet(j);
        int a = findSet(j), b = findSet(i);
        if (a != b) { p[a] = b; size[b] += size[a]; num--; }
    }
    int numSets() { return num; }
    int sizeSet(int i) { return size[findSet(i)]; }
};

```

---

## 3 Graphs

### 3.1 Bellman Ford

```

// O(V * E)
int V;
vector<vii> g;
vector<ll> dist;
vector<bool> neg;

void dfs(int u) {
    neg[u] = true;
    for (int i = 0; i < g[u].size(); i++) {
        ii v = g[u][i];
        if (!neg[v.first])
            dfs(v.first);
    }
}

void bellmanFord(int s) {
    dist.assign(V, INF); dist[s] = 0;
}

```

---

```

for (int i = 0; i < V - 1; i++) {
    for (int u = 0; u < V; u++) {
        if (dist[u] == INF) continue;
        for (int j = 0; j < g[u].size(); j++) {
            ii v = g[u][j];
            if (dist[v.first] > dist[u] + v.second)
                dist[v.first] = max(-INF, dist[u] +
                    v.second);
        }
    }
}

neg.assign(V, false);
for (int u = 0; u < V; u++) {
    for (int i = 0; i < g[u].size(); i++) {
        ii v = g[u][i];
        if (!neg[v.first] && dist[u] < INF && dist[v.first]
            > dist[u] + v.second)
            dfs(v.first);
    }
}
}

```

---

### 3.2 Edmonds Karp Max Flow

```

// O(n * m^2) = O(n^3 * m)
int n, res[MN][MN], p[MN], f, s, t;

void augment(int v, int minEdge) {
    if (v == s) { f = minEdge; return; }
    else if (p[v] != -1) {
        augment(p[v], min(minEdge, res[p[v]][v]));
        res[p[v]][v] -= f; res[v][p[v]] += f;
    }
}

int maxFlow() {
    int mf = 0;
    while (true) {
        f = 0;
        vi dist(MN, oo); dist[s] = 0;
        queue<int> q; q.push(s);
    }
}

```

---

```

memset(p, -1, sizeof p);
while (!q.empty()) {
    int u = q.front(); q.pop();
    if (u == t) break;
    fori(v, 0, n) {
        if (res[u][v] > 0 && dist[v] == oo) {
            dist[v] = dist[u] + 1;
            q.push(v);
            p[v] = u;
        }
    }
    augment(t, oo);
    if (f == 0) break;
    mf += f;
}
return mf;
}

```

```

// Vertex capacities
int in(int x) { return 2 * x; }
int out(int x) { return 2 * x + 1; }

```

### 3.3 Euler Path

```

int g[MN][MN];
stack<int> s;

void dfs(int u) {
    fori(v, 0, n) if (g[u][v]) {
        if (used[u][v]) continue;
        used[u][v] = 1;
        dfs(v);
    }
    s.push(u);
}

```

### 3.4 Hamiltonian Path

```

// 0(n * 2 ** n)
int n;

```

```

int g[MAXN];
int dp[1 << (MAXN-1)];

void hamiltonianDP() {
    dp[0] = 1;
    fori(bitmask, 0, 1 << (n-1)) {
        fori(u, 1, n) {
            int bit = 1 << (u-1);
            if ((bitmask & bit) == 0)
                continue;
            int prev = bitmask ^ bit;
            if ((g[u] & dp[prev]) != 0)
                dp[bitmask] |= 1 << u;
        }
    }
}

```

### 3.5 LCA Binary Lifting

```

const int MN = 200000;
const int ML = 18;

vi g[MN];
int h[MN];
int par[MN][ML]; // initialize -1

void dfs(int u, int p) {
    par[u][0] = p;
    if (p != -1) h[u] = h[p] + 1;
    for (int i = 1; i < ML; i++)
        if (par[u][i-1] != -1)
            par[u][i] = par[par[u][i-1]][i-1];
    for (int v : g[u]) if (v != p)
        dfs(v, u);
}

int lca(int u, int v) {
    if (h[u] < h[v])
        swap(u, v);
    for (int i = ML - 1; i >= 0; i--)
        if (par[u][i] != -1 && h[par[u][i]] >= h[v])
            u = par[u][i];
    // h[u] = h[v]
}

```

```

    if (u == v) return u;
    for (int i = ML - 1; i >= 0; i--)
        if (par[u][i] != par[v][i])
            u = par[u][i], v = par[v][i];
    return par[u][0];
}

int dist(int u, int v) {
    return h[u] + h[v] - 2 * h[lca(u, v)];
}

```

---

### 3.6 Max Cardinality Bipartite Matching

```

// O(n * m)
vi g[MN];
int match[MN], vis[MN];

int aug(int u) {
    if (vis[u]) return 0;
    vis[u] = 1;
    for (int v : g[u]) {
        if (match[v] == -1 || aug(match[v])) {
            match[v] = u;
            return 1;
        }
    }
    return 0;
}

int main() {
    /* ... */
    int mcbm = 0;
    memset(match, -1, sizeof match);
    fori(u, 0, n) {
        memset(vis, 0, sizeof vis);
        mcbm += aug(u);
    }
    /* ... */
}

```

---

### 3.7 Minimum Spanning Tree

```

int par[MN];
int to[MN], from[MN], weight[MN];

bool cmp(int i, int j) { return weight[i] < weight[j]; }

int findSet(int i) { return par[i] < 0 ? i : par[i] = findSet(par[i]); }
void unionSet(int i, int j) { par[findSet(i)] = findSet(j); }

int MST() {
    ll cost = 0;
    vi edges(m);
    fori(i, 0, m) edges[i] = i;
    sort(edges.begin(), edges.end(), cmp);
    memset(par, -1, sizeof par);
    fori(i, 0, m) {
        int e = edges[i];
        if (findSet(to[e]) != findSet(from[e])) {
            cost += weight[e];
            unionSet(to[e], from[e]);
        }
    }
    return cost;
}

```

---

### 3.8 Strongly Connected Components

```

int cnt, nComps, compOf[MN];
int low[MN], num[MN], vis[MN];
vi g[MN];
stack<int> st;

void scc(int u) {
    low[u] = num[u] = cnt++;
    st.push(u);
    vis[u] = 1;
    for (int v : g[u]) {
        if (num[v] == -1)
            scc(v);
        if (vis[v])
            low[u] = min(low[u], low[v]);
    }
    if (low[u] == num[u]) {

```



```

        while (true) {
            int v = st.top(); st.pop();
            vis[v] = 0;
            compOf[v] = nComps;
            if (u == v) break;
        }
        nComps++;
    }
}

int main() {
    cnt = 0, nComps = 0;
    memset(num, -1, sizeof num);
    fori(i, 0, n) if (num[i] == -1)
        scc(i);
}

```

## 4 Mathematics

### 4.1 Combinatoric Formulas

$$\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}$$

when  $x = y = 1$ :

$$2^n = \sum_{k=0}^n \binom{n}{k}$$

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k$$

Given an integer  $n$ , how many ways can  $k$  non-negative integers less than or equal to  $n$  add up to  $n$

$$\binom{n+k-1}{k-1}$$

### 4.2 Combinatoric

```

ll C(ll n, ll k) {
    ll ans = 1;
    for (ll i = 1; i <= k; i++, n--) {
        ans *= n;
        ans /= i;
    }
    return ans;
}

```

### 4.3 Cycle Finding

```

int f(int x) { return ; }

ii cycleFinding(int xo) {
    int tortoise = f(xo), hare = f(f(xo));
    while (tortoise != hare) { tortoise = f(tortoise); hare =
        f(f(hare)); }
    int mu = 0; hare = xo;
    while (tortoise != hare) { tortoise = f(tortoise); hare = f(hare);
        mu++; }
    int lambda = 1; hare = f(hare);
    while (tortoise != hare) { hare = f(hare); lambda++; }
    return ii(mu, lambda);
}

```

### 4.4 Diophantine Equations

Let  $a$  and  $b$  be integers with  $d = \gcd(a, b)$ . The equation  $ax + by = c$  has infinitely many integral solutions if  $d \mid c$  is true.

Let  $k = \frac{c}{d}$  and  $(x_1, y_1)$  the solution found with the Extended Euclidean Algorithm, then:

$$x = kx_1 + \lambda\left(\frac{b}{d}\right); y = ky_1 - \lambda\left(\frac{a}{d}\right)$$

### 4.5 Extended Euclidean Algorithm

```

void extendedEuclidean(ll a, ll b, ll &d, ll &x, ll &y) {

```

```

    if (b == 0) {
        d = a;
        x = 1;
        y = 0;
    } else {
        extendedEuclidean(b, a % b, d, x, y);
        ll x1 = x, y1 = y;
        x = y1;
        y = x1 - (a / b) * y1;
    }
}

ll invModular(ll c, ll m) {
    ll d, x, y;
    extendedEuclidean(c, m, d, x, y);
    return (x + m) % m;
}

```

## 4.6 Linear Recurrence Equations

$$f(n) = af(n-1) + bf(n-2) + cf(n-3); n \geq 3$$

$$\begin{vmatrix} f(n) \\ f(n-1) \\ f(n-2) \end{vmatrix} = \begin{vmatrix} a & b & c \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix} * \begin{vmatrix} f(n-1) \\ f(n-2) \\ f(n-3) \end{vmatrix}$$

$$\begin{vmatrix} f(n) \\ f(n-1) \\ f(n-2) \end{vmatrix} = \begin{vmatrix} a & b & c \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix}^{n-2} * \begin{vmatrix} f(2) \\ f(1) \\ f(0) \end{vmatrix}$$

## 4.7 Pillai Function

```
/* Native solution
```

```

int pillai(int x) {
    int ans = 0;
    for (int i = 1; i <= x; i++)
        ans += gcd(i, x);
    return ans;
}

```

```
*/
```

```

int main() {
    sieve();
    for (int i = 1; i < MX; i++) {
        int phi = euler_phi(i);
        for (int j = 1; j * i < MX; j++)
            F[j * i] += j * phi;
    }
    return 0;
}

```

# 5 Strings

## 5.1 KMP Algorithm

```

void kmp(string &T, string &P) {
    int n = T.size(), m = P.size();
    vector<int> b(m + 1); b[0] = -1;
    // Preprocess P
    int i = 0, j = -1;
    while (i < m) {
        while (j >= 0 && P[i] != P[j]) j = b[j];
        i++; j++;
        b[i] = j;
    }
    // Search T
    i = 0, j = 0;
    while (i < n) {
        while (j >= 0 && T[i] != P[j]) j = b[j];
        i++; j++;
        if (j == m) {
            cout << "P is found at index " << i - j << endl;
            j = b[j];
        }
    }
}

```

## 5.2 Rolling Hashing

```
const ll k = 28759543;
```

```

const ll MD = 1e9 + 7;

ll pol[MN], pt[MN];

ll calc(int l, int r) {
    return ((pol[r] - (l > 0 ? pol[l - 1] : 0) + MD) * fastExp(pt[l],
        MD - 2)) % MD;
}

int main() {
    /* ... */
    pt[0] = 1;
    fori(i, 1, MN) pt[i] = (k * pt[i - 1]) % MD;
    string s; cin >> s;
    int n = s.size();
    fori(i, 0, n) pol[i] = (s[i] * pt[i]) % MD;
    fori(i, 1, n) pol[i] = (pol[i] + pol[i - 1]) % MD;
    /* ... */
}

```

### 5.3 Split Line

```

vector<string> splitLine(string line) {
    istringstream iss(line);
    vector<string> tokens;
    copy(istream_iterator<string>(iss), istream_iterator<string>(),
        back_inserter(tokens));
    return tokens;
}

```

### 5.4 Suffix Array

```

struct SuffixArray {
    string T;
    vi SA, RA, LCP;

    void computeLCP() {
        int n = T.size();
        LCP.assign(n, 0);
        vi Phi(n), PLCP(n);
        Phi[SA[0]] = -1;

```

```

        for (int i = 1; i < n; i++)
            Phi[SA[i]] = SA[i - 1];
        for (int i = 0, L = 0; i < n; i++) {
            if (Phi[i] == -1) { PLCP[i] = 0; continue; }
            while (T[i + L] == T[Phi[i] + L]) L++;
            PLCP[i] = L;
            L = max(L - 1, 0);
        }
        for (int i = 0; i < n; i++)
            LCP[i] = PLCP[SA[i]];
    }

    void countingSort(int k) {
        int n = T.size(), maxi = max(300, n);
        vi c(maxi);
        for (int i = 0; i < n; i++)
            c[i + k < n ? RA[i + k] : 0]++;
        for (int i = 0, sum = 0; i < maxi; i++) {
            int t = c[i];
            c[i] = sum;
            sum += t;
        }
        vi tempSA(n);
        for (int i = 0; i < n; i++)
            tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] =
                SA[i];
        for (int i = 0; i < n; i++)
            SA[i] = tempSA[i];
    }

    SuffixArray(string _T) { T = _T;
        int n = T.size();
        for (int i = 0; i < n; i++) RA.push_back(T[i]);
        for (int i = 0; i < n; i++) SA.push_back(i);
        for (int k = 1; k < n; k <= 1) {
            countingSort(k);
            countingSort(0);
            vi tempRA(n);
            int r = 0;
            tempRA[SA[0]] = 0;
            for (int i = 1; i < n; i++)
                tempRA[SA[i]] = (RA[SA[i]] == RA[SA[i - 1]]
                    && RA[SA[i] + k] == RA[SA[i - 1] + k]) ?
                    r : ++r;
            for (int i = 0; i < n; i++)

```

```

        RA[i] = tempRA[i];
        if (RA[SA[n - 1]] == n - 1) break;
    }
    computeLCP();
}

};

// String Matching
vi stringMatching(string T, string P) {
    int n = T.size(), m = P.size(), start, end;
    SuffixArray sa(T);
    int lo = -1, hi = n;
    while (hi - lo > 1) {
        int mid = (hi + lo) / 2;
        if (P > T.substr(sa.SA[mid], m))
            lo = mid;
        else
            hi = mid;
    }
    if (hi == n || P != T.substr(sa.SA[hi], m)) return vi();
    start = hi;
    lo = 0, hi = n;
    while (hi - lo > 1) {
        int mid = (hi + lo) / 2;
        if (P < T.substr(sa.SA[mid], m))
            hi = mid;
        else
            lo = mid;
    }
    end = lo;
    vi ans;
    for (int i = end; i >= start; i--)
        ans.push_back(sa.SA[i]);
    return ans;
}

// Longest Repeated Substring
ii LRS(string s) {
    int n = s.size();
    SuffixArray sa(s);
    int idx = 0, maxLCP = -1;
    for (int i = 1; i < n; i++)
        if (sa.LCP[i] > maxLCP)
            maxLCP = sa.LCP[i], idx = i;
}

```

```

        return ii(maxLCP, sa.SA[idx]);
    }

// Longest Common Substring
ii LCS(string a, string b) {
    a.push_back('$'); b.push_back('#');
    string c = a + b;
    int n = a.size(), m = b.size(), nm = c.size();
    SuffixArray sa(c);
    int idx = 0, maxLCP = -1;
    for (int i = 1; i < nm; i++) {
        int owner1 = (sa.SA[i] < nm - m - 1) ? 1 : 2;
        int owner2 = (sa.SA[i - 1] < nm - m - 1) ? 1 : 2;
        if (owner1 != owner2 && sa.LCP[i] > maxLCP)
            maxLCP = sa.LCP[i], idx = i;
    }
    return ii(maxLCP, sa.SA[idx]);
}

```

## 5.5 Z Algorithm

```

vi compute_z(string &s) {
    int n = s.size();
    vi z(n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (i > r) {
            l = r = i;
            while (r < n && s[r - l] == s[r]) r++;
            z[i] = r - l; r--;
        } else {
            int k = i - l;
            if (z[k] < r - i + 1) {
                z[i] = z[k];
            } else {
                l = i;
                while (r < n && s[r - l] == s[r]) r++;
                z[i] = r - l; r--;
            }
        }
    }
    return z;
}

```

## 6 Template

### 6.1 Main

---

```
#include <bits/stdc++.h>
#define endl '\n'
#define debug(X) cout << #X << " = " << X << endl
#define fori(i,b,e) for (int i = (b); i < (e); ++i)
#define SZ(v) (int)v.size()

using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef pair<int, int> ii;

const int oo = 1e9;

int main() {
    ios_base::sync_with_stdio(false); cin.tie(0);
    /* happy coding :D */
    return 0;
}
```

---