Тимбук

Mamaev Is the Lion of Fortune

2024 год

1 FFT

```
const int MOD = 998244353;
const int LOG_SZ = 18;
const int SZ = (1 << LOG_SZ);

void init_fft(int log_sz) {
    int root = deg(3, (MOD - 1) >> log_sz);
    assert(deg(root, 1 << (log_sz)) == 1 && deg(root, 1 << (log_sz - 1)) != 1);

    int sz = (1 << LOG_SZ);
    for (int i = 0, r = 1; i < sz; i++, r = mul(r, root))
        roots[i] = r;
    for (int i = 1; i < sz; i++)
        perm[i] = (perm[i >> 1] >> 1) | ((i & 1) << (log_sz - 1));
}

void fft(int arr[], int sz) {
    for (int i = 0; i < sz; i++) {
        if (i < perm[i]) swap(arr[i], arr[perm[i]]);
    }
    for (int i = 1, shift = sz / 2; i < sz; i *= 2, shift /= 2) {
        for (int j = 0; j < sz; j += i * 2) {
            for (int k = 0, pos = 0; k < i; k++, pos += shift) {
                int val = mul(arr[i + j + k], roots[pos]);
                arr[i + j + k] = dec(arr[j + k], val);
                arr[j + k] = add(arr[j + k], val);
            }
        }
    }
}

int fftRes[SZ];
```

```
void mult(int a[], int b[], int sz) {

    fill(arra, arra + sz, 0);
    for (int i = 0; i < sz; i++) arra[i] = a[i];
    fft(arra, sz);

    fill(arrb, arrb + sz, 0);
    for (int i = 0; i < sz; i++) arrb[i] = b[i];
    fft(arrb, sz);

    for (int i = 0; i < sz; i++) arra[i] = mul(arra[i], arrb[i]);
    fft(arra, sz);

    reverse(arra + 1, arra + sz);
    vector<int> ans;
    int revsz = deg(sz, MOD - 2);
    for (int i = 0; i < sz; i++) fftRes[i] = mul(arra[i], revsz);
}
```

2 Суфмас + LCP

```
vector<int> suffix_array(string &t) {
    int m = sz(t);
    vector<int> suff(m);
    iota(all(suff), 0);
    sort(all(suff), [&](int i1, int i2) {
        return t[i1] < t[i2];
    });
    vector<int> color(m);
    for (int i = 1; i < m; ++i) {
        color[suff[i]] = color[suff[i - 1]] + (t[suff[i]] != t[suff[i - 1]]);
    }
    auto add = [&](int x, int y) {
        int ans = x + y;
        while (ans >= m) ans -= m;
        return ans;
    };
    auto sub = [&](int x, int y) {
        int ans = x - y;
        while (ans < 0) ans += m;
        return ans;
    };
    vector<int> cnt(m), head(m), new_suff(m), new_color(m);
```

```
    for (int len = 1; len < m; len *= 2) {
        fill(all(cnt), 0);
        for (int i = 0; i < m; ++i) cnt[color[i]]++;
        fill(all(head), 0);
        for (int i = 1; i < m; ++i) head[i] = head[i - 1] + cnt[i - 1];
        fill(all(new_suff), 0);
        for (int i = 0; i < m; ++i) {
            new_suff[head[color[sub(suff[i], len)]]++] =
                    sub(suff[i], len);
        }
        suff.swap(new_suff);
        fill(all(new_color), 0);
        for (int i = 1; i < m; ++i) {
            new_color[suff[i]] = new_color[suff[i - 1]];
            if (color[suff[i]] != color[suff[i - 1]] ||
                color[add(suff[i], len)] != color[add(suff[i - 1], len)]) {
                new_color[suff[i]]++;
            }
        }
        color.swap(new_color);
    }
    return suff;
}

const int inf = 1e9;

vector<int> build_lcp(string &t, vector<int> &suff) {
    int m = sz(t);
    vector<int> pos(m);
    for (int i = 0; i < m; ++i) {
        pos[suff[i]] = i;
    }
    vector<int> lcp(m);
    int curr = 0;
    for (int i = 0; i < m; ++i) {
        if (curr > 0) --curr;
        if (pos[i] == m - 1) {
            lcp[pos[i]] = curr;
            continue;
        }
        int j = suff[pos[i] + 1];
        while (i + curr < m &&
               j + curr < m &&
               t[i + curr] == t[j + curr]) {
            ++curr;
        }
```

```
        lcp[pos[i]] = curr;
    }
    return lcp;
}
```

3 Потоки

3.1 Диниц

```
const int MAXV = 1e5, MAXE = 1e6, INF = 1e9;

struct Edge{
    int from, to, cap, flow;

    Edge(int from = 0, int to = 0, int cap = 0, int flow = 0):
        from(from), to(to), cap(cap), flow(flow) {}
}edge[MAXE];

int dst[MAXV];

vector<int> g[MAXV];

bool bfs(int start, int finish) {
    fill(dst, dst + MAXV, INF);
    dst[start] = 0;
    queue<int> q;
    q.push(start);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int i : g[v]) {
            if (dst[edge[i].to] == INF && edge[i].flow < edge[i].cap) {
                dst[edge[i].to] = dst[v] + 1;
                q.push(edge[i].to);
            }
        }
    }
    return dst[finish] != INF;
}

int nxt[MAXV];

int dfs(int v, int flow, int finish) {
```

```
    if (v == finish) return flow;
    for (int j = nxt[v]; j < sz(g[v]); j++) {
        int i = g[v][j];
        if (edge[i].flow < edge[i].cap && dst[edge[i].to] == dst[v] + 1) {
            int delta = dfs(edge[i].to, min(flow, edge[i].cap - edge[i].flow), finish);
            if (delta > 0) {
                edge[i].flow += delta;
                edge[i ^ 1].flow -= delta;
                return delta;
            }
        }
    }
    return 0;
}

int dinic(int cntv) {
    int ans = 0;
    while (bfs(start, finish)) {
        fill(nxt, nxt + cntv, 0);
        int flow = dfs(start, INF, finish);
        while (flow) {
            ans += flow;
            flow = dfs(start, INF, finish);
        }
    }
    return ans;
}
```

3.2 Min cost max flow

```
template<typename T>
using normal_queue = priority_queue<T, vector<T>, greater<T> >;

const int MAXV = 1e6, MAXE = 1e6, INF = 1e9;

struct Edge{
    int from, to, cap, flow, w;

    Edge(int from = 0, int to = 0, int cap = 0, int w = 0):
        from(from), to(to), cap(cap), flow(0), w(w) {}
}edge[MAXE];

int cntv, cnte;

int dst[MAXV], pot[MAXV], par[MAXV];
```

```
vector<int> g[MAXV];

bool bfs(int start, int finish) {
    fill(dst, dst + cntv, INF);
    fill(par, par + cntv, -1);
    dst[start] = 0;
    normal_queue<pii> q;
    q.push({0, start});
    while (!q.empty()) {
        auto [d, v] = q.top();
        q.pop();
        if (dst[v] < d) continue;
        for (int i : g[v]) {
            int to = edge[i].to, w = edge[i].w + pot[v] - pot[to];
            if (edge[i].flow < edge[i].cap && dst[to] > dst[v] + w) {
                dst[to] = dst[v] + w;
                par[to] = i;
                q.push({dst[to], to});
            }
        }
    }
    return dst[finish] != INF;
}

void ford(int start) {
    fill(pot, pot + cntv, INF);
    pot[start] = 0;
    for (int iter = 0; iter < cntv - 1; iter++) {
        for (int i = 0; i < cnte; i++) {
            int from = edge[i].from, to = edge[i].to, w = edge[i].w;
            if (pot[to] > pot[from] + w) {
                pot[to] = pot[from] + w;
            }
        }
    }
}

ll getFlow() {
    ll ans = 0;
    ford(start);
    while (bfs(start, finish)) {
        for (int i = 0; i < cntv; i++) pot[i] += dst[i];
        ll delta = INF;
        for (v = n - 1; v != 0; v = edge[par[v]].from) {
            int i = par[v];
```

```
            delta = min(delta, edge[i].cap - edge[i].flow);
        }
        for (v = n - 1; v != 0; v = edge[par[v]].from) {
            int i = par[v];
            edge[i].flow += delta;
            edge[i ^ 1].flow -= delta;
            ans += delta * edge[i].w;
        }
    }
}
```