FHQ Treap - 范浩强

```
mt19937 myRand(chrono::steady_clock::now().time_since_epoch().count());

struct Node
{
    int val;
    int rd = myRand();
    Node *lf = nullptr;
    Node *rt = nullptr;
    int cnt = 1;
};

int Cnt(Node *a)
{
    return a ? a->cnt : 0;
}

void Update(Node *a)
{
    a->cnt = Cnt(a->lf) + Cnt(a->rt) + 1;
}

void Split(Node *a, int pivot, Node *&l, Node *&r)
{
    if (a == nullptr)
    {
        l = nullptr;
        r = nullptr;
        return;
    }
    if (a->val < pivot)
    {
        l = a;
        Split(a->rt, pivot, a->rt, r);
        Update(a);
        return;
    }
    r = a;
    Split(a->lf, pivot, l, a->lf);
    Update(a);
}

void SplitRk(Node *a, int pivot, Node *&l, Node *&r)
{
    if (a == nullptr)
    {
        l = nullptr;
        r = nullptr;
        return;
    }
```

```
    if (Cnt(a->lf) < pivot)
    {
        l = a;
        SplitRk(a->rt, pivot - Cnt(a->lf) - 1, a->rt, r);
        Update(a);
        return;
    }
    r = a;
    SplitRk(a->lf, pivot, l, a->lf);
    Update(a);
}

Node *Merge(Node *l, Node *r)
{
    if (l == nullptr)
    {
        return r;
    }
    if (r == nullptr)
    {
        return l;
    }
    if (l->rd < r->rd)
    {
        r->lf = Merge(l, r->lf);
        Update(r);
        return r;
    }
    l->rt = Merge(l->rt, r);
    Update(l);
    return l;
}



void Show(Node *a)
{
    if (a)
    {
        cout << '(';
        Show(a->lf);
        cout << a->val << ',' << a->cnt;
        Show(a->rt);
        cout << ')';
    }
}

void Insert(int x)
{
    Node *l, *r;
    Split(root, x, l, r);
    root = Merge(Merge(l, new Node({x})), r);
}
```

```
void Delete(int x)
{
    Node *l, *mid, *r;
    Split(root, x, l, r);
    SplitRk(r, 1, mid, r);
    delete mid;
    root = Merge(l, r);
}

int Rank(int x)
{
    Node *l, *r;
    Split(root, x, l, r);
    int res = Cnt(l) + 1;
    root = Merge(l, r);
    return res;
}

int Get(int x)
{
    Node *l, *mid, *r;
    SplitRk(root, x - 1, l, r);
    SplitRk(r, 1, mid, r);
    int res = mid->val;
    root = Merge(Merge(l, mid), r);
    return res;
}

int Prev(int x)
{
    Node *l, *mid, *r;
    Split(root, x, l, r);
    SplitRk(l, Cnt(l) - 1, l, mid);
    int res = mid->val;
    root = Merge(Merge(l, mid), r);
    return res;
}

int Next(int x)
{
    Node *l, *mid, *r;
    Split(root, x + 1, l, r);
    SplitRk(r, 1, mid, r);
    int res = mid->val;
    root = Merge(Merge(l, mid), r);
    return res;
}

Node *root;

int n;
int m;
```

Cost Flow - 费用流

```
const int INF = 0x3f3f3f3f;

const int MAXN = 8000;
const int MAXM = 60000;

struct Edge
{
    int u;
    ll cap;
    ll w;
    int nxt;
};

Edge edges[MAXM];
int cc = 2;
int first[MAXN];
int dqh[MAXN]; // 当前弧

void Add(int x, int y, int cap, int w)
{
    edges[cc].u = y;
    edges[cc].cap = cap;
    edges[cc].w = w;
    edges[cc].nxt = first[x];
    first[x] = cc;
    cc++;
}

void AddEdge(int x, int y, int flow, int w)
{
    Add(x, y, flow, w);
    Add(y, x, 0, -w);
}

int n;
int m;
int p[60][60];
int w[60][60];
const int S = 1;
const int T = 2;
int N;

int room[60][60];
int heng[60][60];
int shu[60][60];

ll h[MAXN];
long long cost = 0;
ll flow = 0;
```

```
ll dis[MAXN];

bool SPFA()
{
    memset(h, 0x3f, sizeof(h));
    h[S] = 0;
    queue<pair<ll, int>> que;
    que.push({0, S});
    while (que.size())
    {
        auto [d, x] = que.front();
        que.pop();
        if (d == h[x])
        {
            for (int i = first[x]; i; i = edges[i].nxt)
            {
                int y = edges[i].u;
                if (edges[i].cap && d + edges[i].w < h[y])
                {
                    h[y] = d + edges[i].w;
                    que.push({h[y], y});
                }
            }
        }
    }
    return h[T] < INF;
}

void Flow(int x, ll flow)
{
    edges[x].cap -= flow;
    edges[x ^ 1].cap += flow;
}

bool vis[MAXN];

ll Dfs(int x, ll flow)
{
    if (x == T)
    {
        return flow;
    }
    vis[x] = true;
    ll curFlow = 0;
    for (auto &i = dqh[x]; i; i = edges[i].nxt)
    {
        int y = edges[i].u;
        if (!vis[y] && h[y] == h[x] + edges[i].w && edges[i].cap)
        {
            ll f = Dfs(y, min(flow - curFlow, edges[i].cap));
            if (f)
            {
                Flow(i, f);
                curFlow += f;
```

```
                if (curFlow == flow)
                {
                    vis[x] = false;
                    return flow;
                }
            }
        }
    }
    vis[x] = false;
    return curFlow;
}

void Dinic()
{
    while (SPFA())
    {
        for (int i = 1; i <= N; i++)
        {
            dqh[i] = first[i];
        }
        ll curFlow = Dfs(S, INF);
        flow += curFlow;
        cost += (long long)curFlow * h[T];
    }
}
```

LCT

```
const int MAXN = 100010;

int n;

namespace LCT
{
    int fa[MAXN];
    int ch[MAXN][2];
    int sz[MAXN];
    bool tag[MAXN];
#define ls ch[p][0]
#define rs ch[p][1]
    void PushUp(int p)
    {
        sz[p] = sz[ls] + sz[rs] + 1;
    }
    void Flip(int p)
    {
        tag[p] ^= true;
        swap(ls, rs);
    }
    void PushDown(int p)
    {
```

```
        if (tag[p])
        {
            if (ls)
                Flip(ls);
            if (rs)
                Flip(rs);
            tag[p] = false;
        }
    }
    void PushAll(int p)
    {
        if (tag[p])
        {
            if (ls)
                Flip(ls);
            if (rs)
                Flip(rs);
            PushAll(ls);
            PushAll(rs);
            tag[p] = false;
        }
    }
    void Show()
    {
        for (int j = 1; j <= n; j++)
        {
            cout << "fa " << j << " = " << LCT::fa[j] << ", ";
            cout << "ls " << j << " = " << ch[j][0] << ", ";
            cout << "rs " << j << " = " << ch[j][1] << ", ";
            cout << "tag " << j << " = " << tag[j] << ", ";
            cout << "sz " << j << " = " << sz[j] << endl;
        }
        cout << endl;
    }
#define Get(x) (ch[fa[x]][1] == x)
#define IsRoot(x) (ch[fa[x]][0] != x && ch[fa[x]][1] != x)
    void Rotate(int x)
    {
        int y = fa[x];
        int z = fa[y];
        bool k = Get(x);
        if (!IsRoot(y))
        {
            ch[z][ch[z][1] == y] = x;
        }
        ch[y][k] = ch[x][!k];
        fa[ch[x][!k]] = y;
        ch[x][!k] = y;
        fa[y] = x;
        fa[x] = z;
        PushUp(y);
        PushUp(x);
    }
    void Update(int p)
```

```
    {
        if (!IsRoot(p))
        {
            Update(fa[p]);
        }
        PushDown(p);
    }
    void Splay(int x)
    {
        Update(x);
        for (int f; f = fa[x], !IsRoot(x); Rotate(x))
        {
            if (!IsRoot(f))
            {
                Rotate(Get(f) == Get(x) ? f : x);
            }
        }
    }
    int Access(int x)
    {
        int p;
        for (p = 0; x; p = x, x = fa[x])
        {
            Splay(x);
            ch[x][1] = p;
            PushUp(x);
        }
        return p;
    }
    void MakeRoot(int p)
    {
        p = Access(p);
        Flip(p);
    }
    void Link(int x, int y)
    {
        MakeRoot(x);
        Splay(x);
        fa[x] = y;
    }
    int Dis(int x, int y)
    {
        MakeRoot(x);
        Access(y);
        Splay(y);
        return sz[y] - 1;
    }
}

struct Diameter
{
    int x;
    int y;
    int len;
```

```
};

bool Cmp(const Diameter &a, const Diameter &b)
{
    return a.len < b.len;
}

int p[MAXN];
Diameter diameters[MAXN];

int curRes = 0;

int P(int a)
{
    if (a == p[a])
    {
        return a;
    }
    return p[a] = P(p[a]);
}

pair<int, int> Farthest(int pa, int a)
{
    auto [x, y, d] = diameters[pa];
    int disx = LCT::Dis(x, a);
    int disy = LCT::Dis(y, a);
    if (disx < disy)
    {
        return {y, disy};
    }
    return {x, disx};
}

void Init()
{
    for (int i = 1; i <= n; i++)
    {
        p[i] = i;
        diameters[i] = {i, i, 0};
        LCT::sz[i] = 1;
    }
}

void Link(int a, int b)
{
    int pa = P(a);
    int pb = P(b);
    if (diameters[a].len < diameters[b].len)
    {
        swap(pa, pb);
        swap(a, b);
    }
    auto [fara, disa] = Farthest(pa, a);
    auto [farb, disb] = Farthest(pb, b);
```

```
    Diameter newD = {fara, farb, disa + disb + 1};
    curRes -= diameters[pa].len + diameters[pb].len;
    diameters[pa] = max({diameters[pa], diameters[pb], newD}, Cmp);
    curRes += diameters[pa].len;
    p[pb] = pa;
    LCT::Link(b, a);
}
```

SA 后缀数组

```
#include <iostream>
#include <vector>

using namespace std;

struct SA {
    string s;
    int n;
    vector<int> rk;
    vector<int> oldrk;
    vector<int> sa;
    vector<int> id;
    vector<int> cnt;
    vector<int> height;

    bool Same(int a, int b, int w) {
        return a < n - w && b < n - w && oldrk[a] == oldrk[b] &&
            oldrk[a + w] == oldrk[b + w] ||
            a >= n - w && b >= n - w && oldrk[a] == oldrk[b];
    }

    SA(const string& s, int m) : s(s) {
        n = s.size();
        int p;
        rk.resize(n);
        oldrk.resize(n);
        sa.resize(n);
        id.resize(n);
        cnt.resize(m);
        height.resize(n);
        for (int i = 0; i < n; i++) {
            rk[i] = s[i];
            cnt[rk[i]]++;
        }
        for (int i = 1; i < m; i++) {
            cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; i > -1; i--) {
            sa[--cnt[rk[i]]] = i;
        }
```

```
        for (int w = 1;; w <<= 1, m = p) {
            int cur = 0;
            for (int i = n - w; i < n; i++) {
                id[cur] = i;
                cur++;
            }
            for (int i = 0; i < n; i++) {
                if (sa[i] >= w) {
                    id[cur] = sa[i] - w;
                    cur++;
                }
            }
            cnt.assign(m, 0);
            for (int i = 0; i < n; i++) {
                cnt[rk[i]]++;
            }
            for (int i = 1; i < m; i++) {
                cnt[i] += cnt[i - 1];
            }
            for (int i = n - 1; i > -1; i--) {
                sa[--cnt[rk[id[i]]]] = id[i];
            }
            p = 0;
            swap(rk, oldrk);
            for (int i = 0; i < n; i++) {
                if (i && !Same(sa[i - 1], sa[i], w)) {
                    p++;
                }
                rk[sa[i]] = p;
            }
            p++;
            if (p == n) {
                break;
            }
        }
        for (int i = 0, k = 0; i < n; i++) {
            if (rk[i]) {
                if (k) {
                    k--;
                }
                int last = sa[rk[i] - 1];
                while (i + k < n && last + k < n && s[i + k] == s[last +
k]) {
                    k++;
                }
                height[rk[i]] = k;
            }
        }
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
```

```
    string s;
    cin >> s;
    SA sa(s, 128);
    for (int i = 0; i < sa.n; i++) {
        cout << sa.sa[i] << ' ';
    }
    cout << '\n';
    for (int i = 0; i < sa.n; i++) {
        cout << sa.height[i] << ' ';
    }
    return 0;
}
```

Lyndon 分解

```
#include <iostream>

using namespace std;

int res;
string s;
int n;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> s;
    n = s.length();
    int i = 0, j = 0, k = 1;
    while (i < n) {
        while (k < n && s[k] >= s[j]) {
            if (s[k] == s[j]) {
                j++;
            }
            else {
                j = i;
            }
            k++;
        }
        while (i <= j) {
            i = min(n, i + k - j);
            // cout << i << ',' << j << ',' << k << endl;
            res ^= i;
        }
        // cout << i << ',' << j << ',' << k << endl;
        j = i;
        k = i + 1;
        continue;
    }
    cout << res;
```

```
    return 0;
}
```

SAM 后缀自动机

```
class Solution {
    struct Node {
        array<int, 26> nxt;
        int len = 0;
        int link = -1;
        Node() { nxt.fill(-1); }
    };
    vector<Node> nodes;
    int last;
    void Init() {
        nodes = {{}};
        last = 0;
    }
    void Insert(int c) {
        int cur = nodes.size();
        nodes.emplace_back();
        nodes[cur].len = nodes[last].len + 1;
        int p = last;
        while (p != -1 && nodes[p].nxt[c] == -1) {
            nodes[p].nxt[c] = cur;
            p = nodes[p].link;
        }
        if (p == -1) {
            nodes[cur].link = 0;
        } else {
            int q = nodes[p].nxt[c];
            if (nodes[q].len == nodes[p].len + 1) {
                nodes[cur].link = q;
            } else {
                int clone = nodes.size();
                nodes.push_back(nodes[q]);
                nodes[clone].len = nodes[p].len + 1;
                nodes[cur].link = clone;
                nodes[q].link = clone;
                while (p != -1 && nodes[p].nxt[c] == q) {
                    nodes[p].nxt[c] = clone;
                    p = nodes[p].link;
                }
            }
        }
        last = cur;
    }
    bool Match(const string& s) {
        int cur = 0;
        for (const auto& c : s) {
            cur = nodes[cur].nxt[c - 'a'];
```

```
            if (cur == -1) {
                return false;
            }
        }
        return true;
    }

public:
    int numOfStrings(vector<string>& patterns, const string& word) {
        Init();
        for (const auto& c : word) {
            Insert(c - 'a');
        }
        int res = 0;
        for (const auto& s : patterns) {
            res += Match(s);
        }
        return res;
    }
};
```

PAM 回文自动机

```
#include <iostream>
#include <unordered_map>

using namespace std;

struct Node {
    int len = 0;
    Node* fail = nullptr;
    unordered_map<char, Node*> nxt;
    int cnt = 0;
};

string s;

int k = 0;
Node* ji = new Node({ -1 });
Node* ou = new Node({ 0, ji });
Node* last = ou;
Node* cur = ji;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cin >> s;
    for (int i = 0; i < s.length(); i++) {
        char c = (s[i] - 97 + k) % 26 + 97;
        s[i] = c;
        while (i - last->len <= 0 || s[i - last->len - 1] != c) {
```

```
            last = last->fail;
        }
        auto it = last->nxt.find(c);
        if (it == last->nxt.end()) {
            it = last->nxt.insert({ c, new Node({last->len + 2, nullptr,
{}, 0}) }).first;
        }
        while (i - cur->len <= 0 || s[i - cur->len - 1] != c) {
            cur = cur->fail;
        }
        if (cur == last) {
            cur = cur->fail;
            if (cur) {
                while (i - cur->len <= 0 || s[i - cur->len - 1] != c) {
                    cur = cur->fail;
                }
            }
        }
        cur = cur ? cur->nxt[c] : ou;
        last = it->second;
        last->fail = cur;
        k = last->cnt = cur->cnt + 1;
        cout << k << ' ';
    }
    return 0;
}
```

NTT

```
void Init() {
    while (N <= n + m) {
        N <<= 1;
    }
    rev.resize(N);
    F.resize(N);
    G.resize(N);
    FG.resize(N);
    for (int i = 1; i < N; i++) {
        rev[i] = rev[i >> 1] >> 1 | (i & 1) * (N >> 1);
    }
}

void NTT(vector<int>& f, int d) {
    ll wn;
    ll w;
    int k;
    ll p, q;
    for (int i = 1; i < N; i++) {
        if (i < rev[i]) {
            swap(f[i], f[rev[i]]);
        }
```

```
    }
    for (int len = 2; len <= N; len <<= 1) {
        k = len >> 1;
        wn = Pow(YG, (MOD - 1) / len);
        for (int i = 0; i < N; i += len) {
            w = 1;
            for (int j = i; j < i + k; j++) {
                p = f[j];
                q = f[j + k] * w % MOD;
                f[j] = p + q;
                if (f[j] >= MOD) {
                    f[j] -= MOD;
                }
                f[j + k] = p - q;
                if (f[j + k] < 0) {
                    f[j + k] += MOD;
                }
                w = w * wn % MOD;
            }
        }
    }
    if (d == -1) {
        reverse(f.begin() + 1, f.end());
        ll inv = Inv(N);
        for (int i = 0; i < N; i++) {
            f[i] = f[i] * inv % MOD;
        }
    }
}
```

牛顿迭代

$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

多项式求逆等

```
#include <iostream>
#include <vector>
#include <cstring>
#include <algorithm>
#include <bit>

#define ll long long

using namespace std;

const int MOD = 998244353;
const int YG = 3;

const int N = 1 << 18;
```

```
ll Pow(ll a, int b) {
    ll res = 1;
    while (b) {
        if (b & 1) {
            res = res * a % MOD;
        }
        a = a * a % MOD;
        b >>= 1;
    }
    return res;
}

ll Inv(ll a) {
    return Pow(a, MOD - 2);
}

int n;

int rev[N];
int F[N];
int G[N];
int inv[N];

void InitMain() {
    inv[1] = 1;
    for (int i = 2; i < N; i++) {
        inv[i] = (ll)(MOD - MOD / i) * inv[MOD % i] % MOD;
    }
}

void Init(int lim) {
    for (int i = 1; i < lim; i++) {
        rev[i] = rev[i >> 1] >> 1 | (i & 1) * (lim >> 1);
    }
}

void NTT(int* f, int d, int lim) {
    ll wn;
    ll w;
    int k;
    ll p, q;
    for (int i = 1; i < lim; i++) {
        if (i < rev[i]) {
            swap(f[i], f[rev[i]]);
        }
    }
    for (int len = 2; len <= lim; len <<= 1) {
        k = len >> 1;
        wn = Pow(YG, (MOD - 1) / len);
        for (int i = 0; i < lim; i += len) {
            w = 1;
            for (int j = i; j < i + k; j++) {
                p = f[j];
                q = f[j + k] * w % MOD;
```

```
                f[j] = p + q;
                if (f[j] >= MOD) {
                    f[j] -= MOD;
                }
                f[j + k] = p - q;
                if (f[j + k] < 0) {
                    f[j + k] += MOD;
                }
                w = w * wn % MOD;
            }
        }
    }
    if (d == -1) {
        reverse(f + 1, f + lim);
        ll inv = Inv(lim);
        for (int i = 0; i < lim; i++) {
            f[i] = f[i] * inv % MOD;
        }
    }
}

// fg = 1
// fg - 1 = 0
// f - 1 / g = 0
// g = g - (f - 1 / g) / (1 / g ^ 2) = 2g - fg ^ 2 = g(2 - fg)

int cur1[N];
int cur2[N];

void Inv(int* f, int* g, int lim) {
    g[0] = Inv(f[0]);
    for (int len = 2; len <= lim; len <<= 1) {
        int k = len << 1;
        int s = len >> 1;
        memcpy(cur1, f, len * sizeof(int));
        memset(cur1 + len, 0, len * sizeof(int));
        memcpy(cur2, g, s * sizeof(int));
        memset(cur2 + s, 0, (k - s) * sizeof(int));
        Init(k);
        NTT(cur1, 1, k);
        NTT(cur2, 1, k);
        for (int i = 0; i < k; i++) {
            g[i] = cur2[i] * (2 + MOD - (ll)cur1[i] * cur2[i] % MOD) %
MOD;
        }
        NTT(g, -1, k);
    }
}

// g ^ 2 = f
// g ^ 2 - f = 0
// g = g - (g ^ 2 - f) / 2g = (g ^ 2 + f) / 2g

int cur3[N];
```

```
void Sqrt(int* f, int* g, int lim) {
    g[0] = 1;
    for (int len = 2; len <= lim; len <<= 1) {
        int k = len << 1;
        int s = len >> 1;
        memset(g + s, 0, s * sizeof(int));
        Inv(g, cur3, len);
        memset(cur3 + len, 0, len * sizeof(int));
        memcpy(cur1, f, len * sizeof(int));
        memset(cur1 + len, 0, len * sizeof(int));
        memcpy(cur2, g, s * sizeof(int));
        memset(cur2 + s, 0, (k - s) * sizeof(int));
        Init(k);
        NTT(cur1, 1, k);
        NTT(cur2, 1, k);
        NTT(cur3, 1, k);
        for (int i = 0; i < k; i++) {
            g[i] = ((ll)cur2[i] * cur2[i] + cur1[i]) % MOD * ((MOD + 1) /
2) % MOD * cur3[i] % MOD;
        }
        NTT(g, -1, k);
    }
}

void Derivative(int* f, int* g, int lim) {
    for (int i = 1; i < lim; i++) {
        g[i - 1] = (ll)f[i] * i % MOD;
    }
    g[lim - 1] = 0;
}

void Integrate(int* f, int* g, int lim) {
    g[0] = 0;
    for (int i = 1; i < lim; i++) {
        g[i] = (ll)f[i - 1] * inv[i] % MOD;
    }
}

// g = ln(f)
// dg = df / f
// g = int(df / f)

void Ln(int* f, int* g, int lim) {
    int k = lim << 1;
    Inv(f, cur3, lim);
    memset(cur3 + lim, 0, lim * sizeof(int));
    Derivative(f, cur1, lim);
    memset(cur1 + lim, 0, lim * sizeof(int));
    Init(k);
    NTT(cur3, 1, k);
    NTT(cur1, 1, k);
    for (int i = 0; i < k; i++) {
        cur1[i] = (ll)cur1[i] * cur3[i] % MOD;
```

```
    }
    NTT(cur1, -1, k);
    Integrate(cur1, g, lim);
}

// g = e ^ f
// lng - f = 0
// g = g - (lng - f) / (1 / g) = g(1 - lng + f)

void Exp(int* f, int* g, int lim) {
    g[0] = 1;
    for (int len = 2; len <= lim; len <<= 1) {
        int k = len << 1;
        int s = len >> 1;
        memset(g + s, 0, s * sizeof(int));
        Ln(g, cur3, len);
        memset(cur3 + len, 0, len * sizeof(int));
        memcpy(cur1, f, len * sizeof(int));
        memset(cur1 + len, 0, len * sizeof(int));
        memcpy(cur2, g, s * sizeof(int));
        memset(cur2 + s, 0, (k - s) * sizeof(int));
        Init(k);
        NTT(cur1, 1, k);
        NTT(cur2, 1, k);
        NTT(cur3, 1, k);
        for (int i = 0; i < k; i++) {
            g[i] = (ll)cur2[i] * (MOD + 1 - cur3[i] + cur1[i]) % MOD;
        }
        NTT(g, -1, k);
    }
}

void Solve() {
    cin >> n;
    int lim = bit_ceil((unsigned)n);
    for (int i = 0; i < n; i++) {
        cin >> F[i];
    }
    Exp(F, G, lim);
    for (int i = 0; i < n; i++) {
        cout << G[i] << ' ';
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    InitMain();
    Solve();
    return 0;
}
```