Backlight's Code Template

Backlight @ CSU

2021 年 4 月 30 日

目录

1 ds

1.1 SGT

```
template<typename T>
struct SGTree {
    static constexpr double alpha = 0.75; // alpha \in (0.5, 1)
    int root, tot, buf_size;
    T v[N];
    int s[N], sz[N], sd[N], cnt[N], l[N], r[N], buf[N];


    SGTree()
    {
        root = tot = 0;
    }

    int new_node(T _v)
    {
        ++tot;
        v[tot] = _v;
        s[tot] = sz[tot] = sd[tot] = cnt[tot] = 1;
        l[tot] = r[tot] = 0;
        return tot;
    }

    void push_up(int x)
    {
        if (!x) return;
        int lc = l[x], rc = r[x];
        s[x] = s[lc] + 1 + s[rc];
        sz[x] = sz[lc] + cnt[x] + sz[rc];
        sd[x] = sd[lc] + (cnt[x] != 0) + sd[rc];
    }

    bool balance(int x)
    {
        int lc = l[x], rc = r[x];
        if (alpha * s[x] <= max(s[lc], s[rc])) return false;
        if (alpha * s[x] >= sd[x]) return false;
        return true;
    }

    void flatten(int x)
    {
        if (!x) return;
        flatten(l[x]);
        if (cnt[x]) buf[++buf_size] = x;
        flatten(r[x]);
    }

    void build(int& x, int L, int R)
    {
        if (L > R) {
            x = 0;
            return;
        }
        int mid = (L + R) >> 1;
        x = buf[mid];
        build(l[x], L, mid - 1);
        build(r[x], mid + 1, R);
        push_up(x);
    }

    void rebuild(int& x)
    {
        buf_size = 0;
        flatten(x);
        build(x, 1, buf_size);
    }

    void ins(int& rt, T val)
    {
        if (!rt) {
            rt = new_node(val);
            return;
        }
        if (val == v[rt]) {
            ++cnt[rt];
        } else if (val < v[rt]) {
            ins(l[rt], val);
        } else {
            ins(r[rt], val);
        }
        push_up(rt);
        if (!balance(rt)) rebuild(rt);
    }

    void del(int &rt, T val)
    {
        if (!rt) return;

        if (val == v[rt]) {
            if (cnt[rt]) --cnt[rt];
        } else if (val < v[rt]) {
            del(l[rt], val);
        } else {
            del(r[rt], val);
        }
        push_up(rt);
        if (!balance(rt)) rebuild(rt);
    }

    int getPrevRank(int rt, T val)
    {
        if (!rt) return 0;
        if (v[rt] == val && cnt[rt]) return sz[l[rt]];
        if (v[rt] < val) return sz[l[rt]] + cnt[rt] + getPrevRank(r[rt], val);
        return getPrevRank(l[rt], val);
    }

    int getSuccRank(int rt, T val)
    {
        if (!rt) return 1;
        if (v[rt] == val && cnt[rt]) return sz[l[rt]] + cnt[rt] + 1;
        if (v[rt] < val) return sz[l[rt]] + cnt[rt] + getSuccRank(r[rt], val);
        return getSuccRank(l[rt], val);
    }


    T getKth(int rt, int k)
    {
        if (!rt) return 0;
        if (k <= sz[l[rt]]) return getKth(l[rt], k);
        if (k - sz[l[rt]] <= cnt[rt]) return v[rt];
        return getKth(r[rt], k - sz[l[rt]] - cnt[rt]);
    }

    void ins(T val)
    {
```

```cpp
        ins(root, val);
    }

    void del(T val)
    {
        del(root, val);
    }

    int getRank(T val)
    {
        return getPrevRank(root, val) + 1;
    }

    T getKth(int k)
    {
        return getKth(root, k);
    }

    T getPrev(T val)
    {
        return getKth(getPrevRank(root, val));
    }

    T getSucc(T val)
    {
        return getKth(getSuccRank(root, val));
    }

    void debug(int x)
    {
        if (!x) return;
        debug(l[x]);
        cerr << v[x] << " ";
        debug(r[x]);
    }

    void debug()
    {
        cerr << "SGTree:" << endl;
        debug(root);
        cerr << endl;
    }
};
```

2 graph

2.1 Dijkstra

```cpp
namespace Backlight {

template<typename T>
struct Wraph {
    struct Edge {
        int u, v;
        T w;
        Edge(){}
        Edge(int _u, int _v, T _w): u(_u), v(_v), w(_w) {}
    };

    int V;
    vector<vector<Edge>> G;

    Wraph() : V(0) {}
    Wraph(int _V) : V(_V), G(_V + 1) {}

    inline void addarc(int u, int v, T w) {
        assert(1 <= u && u <= V);
        assert(1 <= v && v <= V);
        G[u].push_back(Edge(u, v, w));
    }

    inline void addedge(int u, int v, T w) {
        addarc(u, v, w);
        addarc(v, u, w);
    }

    /***************************************************/
    vector<T> dijkstra(int S, T T_MAX) {
        typedef pair<T, int> Node;
        priority_queue<Node, vector<Node>, greater<Node>> q;
        vector<T> dis(V + 1);
        for (int i = 1; i <= V; i++) dis[i] = T_MAX;
        dis[S] = 0; q.push(Node(0, S));
        while (!q.empty()){
            Node p = q.top(); q.pop();
            T cost = p.first; int u = p.second;
            if (dis[u] != cost) continue;

            for (Edge e: G[u]){
                int v = e.v;
                T w = e.w;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    q.push(Node(dis[v], v));
                }
            }
        }
        return dis;
    }
};

}
```

3 math

3.1 Lucas

```
namespace Backlight {

// use this when n, m is really large and p is small
namespace Lucas {
    inline ll pow(ll a, ll b, ll p) {
        ll res = 1;
        a %= p;
        while(b) {
            if (b & 1) res = res * a % p;
            a = a * a % p;
            b >>= 1;
        }
        return res;
    }

    inline ll inv1(ll n, ll p) { return pow(n, p - 2, p); }

    inline ll C1(ll n, ll m, ll p) {
        if (m > n) return 0;
    if (m > n - m) m = n - m;
    ll u = 1, d = 1;
    for (ll i = 1; i <= m; ++i) {
        u = u * (n - i + 1) % p;
        d = d * i % p;
    }
    return u * inv1(d, p) % p;
}

// solve n choose m (mod p) while p is a prime
ll lucas(ll n, ll m, ll p) {
    if (m == 0) return 1;
    return C1(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
}


ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll z = x; x = y; y = z - y * (a / b);
    return d;
}

inline ll inv2(ll n, ll p) {
    ll x, y;
    ll d = exgcd(n, p, x, y);
    return d == 1 ? (p + x % p) % p : -1;
}

// n! mod pk without pi^x
ll f(ll n, ll pi, ll pk) {
    if (!n) return 1;
    ll res = 1;
    if (n / pk) {
        for (ll i = 2; i <= pk; ++i)
            if (i % pi) res = res * i % pk;
        res = pow(res, n / pk, pk);
    }
    for (ll i = 2; i <= n % pk; ++i)
        if (i % pi) res = res * i % pk;
    return res * f(n / pi, pi, pk) % pk;
}

ll C2(ll n, ll m, ll p, ll pi, ll pk) {
    if (m > n) return 0;
    ll a = f(n, pi, pk), b = f(m, pi, pk), c = f(n - m, pi, pk);
    ll k = 0;
    for (ll i = n; i; i /= pi) k += i / pi;
    for (ll i = m; i; i /= pi) k -= i / pi;
    for (ll i = n - m; i; i /= pi) k -= i / pi;
    ll ans = a * inv2(b, pk) % pk * inv2(c, pk) % pk * pow(pi, k, pk) % pk;
    ans = ans * (p / pk) % p * inv2(p / pk, pk) % p;
    return ans;
}

// solve n choose m (mod p) while p might not be a prime
ll exlucas(ll n, ll m, ll p) {
    ll x = p;
    ll ans = 0;
    for (ll i = 2; i <= p; ++i) {
        if (x % i == 0) {
            ll pk = 1;
```

```
                while(x % i == 0) pk *= i, x /= i;
                ans = (ans + C2(n, m, p, i, pk)) % p;
            }
        }
        return ans;
    }

} // namespace Lucas

} // namespace Backlight
```

4 other

4.1 BFPRT

```
/**
 * BFPRT: find the kth element of an array in O(n) using Divide and Conquer method.
 * you can use std::nth_element(a, a + k, a + n) instead
**/
namespace BFPRT {
    template<typename T, typename Cmp>
    T kth_index(T* a, int l, int r, int k, Cmp cmp);

    template<typename T, typename Cmp>
    int insert_sort(T* a, int l, int r, Cmp cmp) {
        for (int i = l + 1; i <= r; ++i) {
            int tmp = a[i];
            int j = i - 1;
            while(j >= l && a[j] > tmp) {
                a[j + 1] = a[j];
                --j;
            }
            a[j + 1] = tmp;
        }
        return l + (r - l) / 2;
    }

    template<typename T, typename Cmp>
    int pivot(T* a, int l, int r, Cmp cmp) {
        if (r - l < 5) return insert_sort(a, l, r, cmp);
        int lst = l - 1;
        for (int i = l; i + 4 <= r; i += 5) {
            int p = insert_sort(a, i, i + 4, cmp);
            swap(a[++lst], a[p]);
        }
        return kth_index<T>(a, l, lst, (lst - l + 1) / 2 + 1, cmp);
    }

    template<typename T, typename Cmp>
    int partition(T* a, int l, int r, Cmp cmp) {
        int p = pivot(a, l, r, cmp);
        swap(a[p], a[r]);
        int lst = l - 1;
        for (int i = l; i < r; ++i) {
            if (cmp(a[i], a[r])) swap(a[++lst], a[i]);
        }
        swap(a[++lst], a[r]);
        return lst;
    }

    template<typename T, typename Cmp>
    T kth_index(T* a, int l, int r, int k, Cmp cmp) {
        int p = partition(a, l, r, cmp);
        int d = p - l + 1;
        if (d == k) return p;
        else if (d < k) return kth_index(a, p + 1, r, k - d, cmp);
        else return kth_index(a, l, p - 1, k, cmp);
    }

    template<typename T>
    T kth_index(T* a, int l, int r, int k) {
        return kth_index(a, l, r, k, less<T>());
    }
};
```

5 string

5.1 KMP

```
namespace KMP {
    // pi_i = s[0...i] 最长 border
    void getPi(char* s, int n, int* pi) {
        pi[0] = 0;
        for (int i = 1; i < n; ++i) {
            int j = pi[i - 1];
            while(j > 0 && s[j] != s[i]) j = pi[j - 1];
            if (s[i] == s[j]) ++j;
            pi[i] = j;
        }
    }

    vector<int> getAllMatchPosition(char* s, int n, int* pi, char* t, int m) {
        s[n] = '#'; s[n + 1] = 0; ++n;
        KMP::getPi(s, n, pi);

        vector<int> ans;

        int p = 0;
        for (int i = 0; i < m; ++i) {
            while(p > 0 && t[i] != s[p]) p = pi[p - 1];
            if (t[i] == s[p]) {
                ++p;
                if (p == n - 1) {
                    ans.push_back(i + 2 - n);
                }
            }
        }

        return ans;
    }

    int getPeriod(int n, int* pi) {
        return n - pi[n - 1];
    }
}
```