



ACM 模板

wenwenla



2017-10-22

目录

FastIO	2
并查集	3
ST 表.....	4
链表	4
树状数组.....	6
LCA	6
SCC	8
Dijkstra	9
Dinic	11
Treap	12
线段树	15
KM 字符串匹配	16
LIS	17
数值积分.....	18

FastIO

```
namespace fastio {
    const static int buf_size = 8388608;//about 8Mb
    static char buf[buf_size];
    char *ps = buf + buf_size, *pe = buf + buf_size;
    int pos;
    bool eof = false;

    inline void read_next() {
        pe = buf + fread(buf, 1, buf_size, stdin);
        ps = buf;
        if(ps == pe) eof = true;
    }

    inline bool blank(char x) {
        return x == ' ' || x == '\n' || x == '\t' || x == '\r';
    }

    inline char nc() {
        if(ps == pe) read_next();
        return *ps++;
    }

    template<typename T>
    inline void read_num(T& res) {
        bool neg = false;
        char now = nc();
        while(blank(now)) now = nc();
        if(now == '-') {
            neg = true;
            now = nc();
        }
        T ret = 0;
        while(!blank(now)) {
            ret = ret * 10 + now - '0';
            now = nc();
        }
        res = (neg ? -ret : ret);
    }

    template<typename T>
    inline void put_num(T x) {
        if(x < 0) {
```

```

        putchar('-');
        x = -x;
    }
    if(x == 0) {
        putchar('0');
    }
    char tmp[32];
    int cnt = 0;
    while(x) {
        tmp[cnt++] = x % 10 + '0';
        x /= 10;
    }
    while(cnt > 0) {
        putchar(tmp[cnt - 1]);
        --cnt;
    }
}
}

```

并查集

```

namespace UFS {
    const int N = 1e5 + 10;
    int fa[N];
    inline void init() { memset(fa, 0xff, sizeof(fa)); }
    inline int id(int u) {
        int rt = u;
        while(fa[rt] > 0) rt = fa[rt];
        while(fa[u] > 0) { int tmp = fa[u]; fa[u] = rt; u = tmp; }
        return rt;
    }
    inline void join(int u, int v) {
        int uf = id(u);
        int vf = id(v);
        if(uf != vf) {
            if(fa[uf] < fa[vf]) { fa[uf] += fa[vf]; fa[vf] = uf; }
            else { fa[vf] += fa[uf]; fa[uf] = vf; }
        }
    }
    inline int size(int u) { return -fa[id(u)]; }
};

```

ST 表

```
struct ST {
    //1 base, query O(1)
    int dp[50005][20];
    int n;
    int arr[50005];

    void make(int _n) {
        assert(n < 50005);
        n = _n;
        for(int i = 1; i <= n; ++i) {
            dp[i][0] = arr[i];
        }
        for(int j = 1; (1 << j) <= n; ++j) {
            for(int i = 1; (i + (1 << (j - 1))) <= n; ++i) {
                dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    int query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return min(dp[l][k], dp[r - (1 << k) + 1][k]);
    }
} st;
```

链表

```
template<typename T>
struct ACM_list {
    static const int MAXN = 100005;
    static const int BEG = 0;
    static const int END = MAXN - 1;
    struct Node {
        T data;
        int next;
        int pre;
    } node[MAXN];
    int free[MAXN], fp;
    int sz;

    void init() {
```

```

    node[BEG] = {T{}, END, -1};
    node[END] = {T{}, -1, BEG};
    fp = MAXN - 3;
    for(int i = 0; i <= fp; ++i) {
        free[i] = i + 1;
    }
    sz = 0;
}

int ins_back(int idx, T val) {
    node[free[fp]].next = node[idx].next;
    node[free[fp]].data = val;
    node[free[fp]].pre = idx;
    node[idx].next = free[fp];
    node[node[free[fp]].next].pre = free[fp];
    ++sz;
    return free[fp--];
}

int ins_pre(int idx, T val) {
    idx = node[idx].pre;
    return ins_back(idx, val);
}

void del(int idx) {
    assert(idx != BEG && idx != END);
    node[node[idx].pre].next = node[idx].next;
    node[node[idx].next].pre = node[idx].pre;
    free[++fp] = idx;
    --sz;
}

Node& operator[](int idx) {
    return node[idx];
}

int begin() const {
    return node[BEG].next;
}

int end() const {
    return END;
}

int size() const {
    return sz;
}

```

```

    }
};
ACM_list<int> li;

```

树状数组

```

struct Bit {
    //1 Base
    int arr[300005];
    int n;

    int lowbit(int x) { return x & (-x); }

    void add(int i, int x) {
        int pos = i;
        while(pos <= n) {
            arr[pos] += x;
            pos += lowbit(pos);
        }
    }

    int sum(int i) {
        int pos = i, ans = 0;
        while(pos) {
            ans += arr[pos];
            pos -= lowbit(pos);
        }
        return ans;
    }

    void init(int _n) {
        n = _n;
        for(int i = 1; i <= n; ++i) {
            arr[i] += sum(i - 1) - sum(i - lowbit(i));
        }
    }
} bit;

```

LCA

```

const int VN = 50000, EN = 100000;

```

```

struct ST {
    //1 base, query O(1)
    int dp[VN << 1][20];
    int n;
    int arr[VN << 1];

    void make(int _n) {
        n = _n;
        for(int i = 1; i <= n; ++i) {
            dp[i][0] = i;
        }
        for(int j = 1; (1 << j) <= n; ++j) {
            for(int i = 1; (i + (1 << (j - 1))) <= n; ++i) {
                int l = dp[i][j - 1], r = dp[i + (1 << (j - 1))][j - 1];
                dp[i][j] = arr[l] < arr[r] ? l : r;
            }
        }
    }

    int query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        int li = dp[l][k], ri = dp[r - (1 << k) + 1][k];
        return arr[li] < arr[ri] ? li : ri;
    }
} st;

struct edge {
    int to, cost, next;
    edge() {}
    edge(int _to, int _cost, int _next) : to(_to), cost(_cost), next(_next) {}
} eg[EN];
int head[VN], dis[VN], id[VN], la[VN << 1], lacnt, tot;
bool vis[VN];

void init() {
    memset(head, 0xff, sizeof(head));
    memset(vis, 0x00, sizeof(vis));
    tot = lacnt = 1;
}

void addedge(int from, int to, int cost) {
    eg[tot] = edge(to, cost, head[from]);
    head[from] = tot++;
}

```



```

void dfs(int v, int height) {
    vis[v] = true;
    id[v] = lacnt;
    st.arr[lacnt] = height;
    la[lacnt++] = v;
    for(int i = head[v]; i != -1; i = eg[i].next) {
        if(!vis[eg[i].to]) {
            dis[eg[i].to] = dis[v] + eg[i].cost;
            dfs(eg[i].to, height + 1);
            st.arr[lacnt] = height;
            la[lacnt++] = v;
        }
    }
}

int lca(int u, int v) {
    int l = min(id[u], id[v]), r = max(id[u], id[v]);
    return la[st.query(l, r)];
}

```

SCC

```

namespace SCC {
    //id 1 to n
    const int VN = 10005, EN = 100005;
    struct edge {
        int to, next;
        edge() {}
        edge(int _to, int _next) : to(_to), next(_next) {}
    } se[EN], re[EN];
    int shead[VN], rhead[VN], vs[VN], cmp[VN], vidx, stot, rtot, n;
    bool vis[VN];

    void init(int _n) {
        n = _n;
        stot = rtot = 0;
        memset(shead, 0xff, sizeof(shead));
        memset(rhead, 0xff, sizeof(rhead));
        vidx = 0;
    }

    void addedge(int from, int to) {

```

```

        se[stot] = edge(to, shead[from]);
        shead[from] = stot++;
        re[rtot] = edge(from, rhead[to]);
        rhead[to] = rtot++;
    }

    void dfs(int v) {
        vis[v] = true;
        for(int i = shead[v]; i != -1; i = se[i].next) {
            if(!vis[se[i].to]) {
                dfs(se[i].to);
            }
        }
        vs[vidx++] = v;
    }

    void rdfs(int v, int k) {
        vis[v] = true;
        cmp[v] = k;
        for(int i = rhead[v]; i != -1; i = re[i].next) {
            if(!vis[re[i].to]) {
                rdfs(re[i].to, k);
            }
        }
    }

    int scc() {
        memset(vis, 0, sizeof(vis));
        for(int i = 1; i <= n; ++i) {
            if(!vis[i]) dfs(i);
        }
        memset(vis, 0, sizeof(vis));
        int k = 0;
        for(int i = vidx - 1; i >= 0; --i) {
            if(!vis[vs[i]]) rdfs(vs[i], k++);
        }
        return k;
    }
}

```

Dijkstra

using int64 = long long;

```

struct edge {
    int to, cost, next;
    edge() {}
    edge(int _to, int _cost, int _next) : to(_to), cost(_cost), next(_next) {}
} e[12500];
int tot;
int head[2505];

void init_graph() {
    tot = 0;
    memset(head, -1, sizeof(head));
}

void addedge(int from, int to, int cost) {
    e[tot] = edge(to, cost, head[from]);
    head[from] = tot++;
}

int64 dis[2505];
bool vis[2505];

int64 dijkstra(int s, int t) {
    using PLI = pair<int64, int>;
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0x00, sizeof(vis));
    priority_queue<PLI, vector<PLI>, greater<PLI>> pq;
    pq.push(PLI(0, s));
    dis[s] = 0;
    while(!pq.empty()) {
        PLI now = pq.top();
        pq.pop();
        if(vis[now.second]) continue;
        vis[now.second] = true;
        for(int i = head[now.second]; i != -1; i = e[i].next) {
            if(!vis[e[i].to] && e[i].cost + now.first < dis[e[i].to]) {
                dis[e[i].to] = e[i].cost + now.first;
                pq.push(PLI(dis[e[i].to], e[i].to));
            }
        }
    }
    return dis[t];
}

```

Dinic

```
namespace Dinic {
    const int V = 1000010, E = 8000010, INF = 1e9;
    int vcnt;
    struct edge {
        int to, next, cap, flow;
        edge() {}
        edge(int _to, int _next, int _cap) : to(_to), next(_next), cap(_cap), flow(0) {}
    } eg[E];
    int head[V], cur[V], dis[V], que[V], qf, qe, ecnt, s, t;
    bool vis[V];

    void init(int _vcnt) {
        vcnt = _vcnt;
        memset(head, 0xff, sizeof(head[0]) * (vcnt + 1));
        ecnt = 0;
    }

    void addedge(int from, int to, int cap) {
        eg[ecnt] = edge(to, head[from], cap);
        head[from] = ecnt++;
        eg[ecnt] = edge(from, head[to], 0);
        head[to] = ecnt++;
    }

    bool bfs() {
        memset(vis, 0, sizeof(vis[0]) * (vcnt + 1));
        qf = 0; qe = 0;
        que[qe++] = s;
        dis[s] = 0; vis[s] = true;
        while(qf < qe) {
            int x = que[qf++];
            for(int i = head[x]; i != -1; i = eg[i].next) {
                const edge& e = eg[i];
                if(!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = true;
                    dis[e.to] = dis[x] + 1;
                    que[qe++] = e.to;
                    if(e.to == t) return true;
                }
            }
        }
        return false;
    }
}
```

```

    }

    int dfs(int x, int a) {
        if(x == t || a == 0) return a;
        int flow = 0, f;
        for(int& i = cur[x]; i != -1; i = eg[i].next) {
            edge& e = eg[i];
            if(dis[x] + 1 == dis[e.to] && (f = dfs(e.to, min(e.cap - e.flow, a)))) {
                e.flow += f;
                eg[i ^ 1].flow -= f;
                flow += f;
                a -= f;
                if(!a) break;
            }
        }
        return flow;
    }
}

int solve(int _s, int _t) {
    s = _s; t = _t;
    int flow = 0;
    while(bfs()) {
        memcpy(cur, head, sizeof(cur[0]) * (vcnt + 1));
        flow += dfs(s, INF);
    }
    return flow;
}
}

```

Treap

```

template<typename T, class _Comp = less<T>>
struct Treap {
    const static int NODECNT = _;
    struct Node {
        int ch[2], p, sz;
        T v;
        void make(int _l, int _r, int _p, const T& _v) {
            ch[0] = _l; ch[1] = _r; p = _p; v = _v; sz = 1;
        }
    } node[NODECNT];
    int m_rt, mp[NODECNT], mp_idx, node_idx;

```

```

void maintain(int x) {
    node[x].sz = 1;
    node[x].sz += node[x].ch[0] == -1 ? 0 : node[node[x].ch[0]].sz;
    node[x].sz += node[x].ch[1] == -1 ? 0 : node[node[x].ch[1]].sz;
}

_Comp cmp;

explicit Treap(const _Comp& c) : cmp(c) { unsigned seed = 19971023; srand(seed); clear(); }

Treap() : cmp(_Comp()) { unsigned seed = 19971023; srand(seed); clear(); }

void clear() { m_rt = -1; mp_idx = -1; node_idx = 0; }

void ins(const T& val) { _ins(m_rt, val); }
void _ins(int& rt, const T& val) {
    if(rt == -1) {
        if(mp_idx == -1) { node[rt = node_idx++] = make(-1, -1, rand(), val); }
        else { node[rt = mp[mp_idx--]] = make(-1, -1, rand(), val); }
    } else {
        int type = cmp(node[rt].v, val);
        _ins(node[rt].ch[type], val);
        maintain(rt);
        if(node[rt].p < node[node[rt].ch[type]].p) rotate(rt, type);
    }
}

void del(const T& val) { _del(m_rt, val); }
void _del(int& rt, const T& val) {
    assert(rt != -1);
    if(node[rt].v == val) {
        if(node[rt].ch[0] == -1) {
            mp[++mp_idx] = rt;
            rt = node[rt].ch[1];
        } else if(node[rt].ch[1] == -1) {
            mp[++mp_idx] = rt;
            rt = node[rt].ch[0];
        } else {
            int next = node[node[rt].ch[0]].p < node[node[rt].ch[1]].p;
            rotate(rt, next);
            _del(node[rt].ch[next ^ 1], val);
            maintain(rt);
        }
    } else {
        _del(node[rt].ch[cmp(node[rt].v, val)], val);
    }
}

```

```

        maintain(rt);
    }
}

int find(const T& val) {
    int rt = m_rt;
    while(rt != -1) {
        if(node[rt].v == val) return rt;
        rt = node[rt].ch[cmp(node[rt].v, val)];
    }
    return -1;
}

void rotate(int& rt, int type) {
    int tmp = node[rt].ch[type];
    node[rt].ch[type] = node[tmp].ch[type ^ 1];
    node[tmp].ch[type ^ 1] = rt;
    maintain(rt); maintain(tmp);
    rt = tmp;
}

int kth(int k) {
    assert(k >= 1 && k <= size());
    int rt = m_rt, res = -1;
    while(rt != -1) {
        int le = node[rt].ch[0] == -1 ? 0 : node[node[rt].ch[0]].sz;
        if(le == k - 1) {
            res = node[rt].v;
            break;
        } else if(le > k - 1) {
            rt = node[rt].ch[0];
        } else {
            k -= le + 1;
            rt = node[rt].ch[1];
        }
    }
    return res;
}

int rank(const T& val) {
    int rt = m_rt, cnt = 0;
    while(rt != -1) {
        int le = node[rt].ch[0] == -1 ? 0 : node[node[rt].ch[0]].sz;
        if(cmp(node[rt].v, val)) {

```

```

        cnt += le + 1;
        rt = node[rt].ch[1];
    } else {
        rt = node[rt].ch[0];
    }
}
return cnt + 1;
}

int size() { return node[m_rt].sz; }
};

```

线段树

```

using ll = long long;
struct seg_tree {
    static const int MAXN = 100005;
    struct Node {
        ll sum, tag;
    } node[MAXN << 2];
    ll arr[MAXN];

    int lson(int x) { return x << 1; }
    int rson(int x) { return (x << 1) + 1; }

    void make(int x, int xl, int xr) {
        if(xl == xr) {
            node[x].sum = arr[xl];
            node[x].tag = 0;
            return;
        }
        int mid = (xl + xr) >> 1;
        make(lson(x), xl, mid);
        make(rson(x), mid + 1, xr);
        node[x].sum = node[lson(x)].sum + node[rson(x)].sum;
        node[x].tag = 0;
    }

    void pushdown(int x, int xl, int xr) {
        node[x].sum += (xr - xl + 1) * node[x].tag;
        if(xl < xr) {
            node[lson(x)].tag += node[x].tag;
            node[rson(x)].tag += node[x].tag;
        }
    }
};

```



```

    }
    node[x].tag = 0;
}

void pushup(int x, int xl, int xr) {
    int mid = (xl + xr) >> 1;
    int lcnt = mid - xl + 1, rcnt = xr - mid;
    node[x].sum = node[lson(x)].sum + lcnt * node[lson(x)].tag + node[rson(x)].sum + rcnt *
node[rson(x)].tag;
}

ll query(int x, int xl, int xr, int ql, int qr) {
    if(xl > qr || xr < ql) return 0;
    if(xl == ql && xr == qr) {
        return node[x].sum + (xr - xl + 1) * node[x].tag;
    }
    ll ans = 0;
    int mid = (xl + xr) >> 1;
    pushdown(x, xl, xr);
    ans += query(lson(x), xl, mid, ql, min(mid, qr));
    ans += query(rson(x), mid + 1, xr, max(ql, mid + 1), qr);
    return ans;
}

void update(int x, int xl, int xr, int cl, int cr, ll dt) {
    if(xr < cl || xl > cr) return;
    if(cl == xl && cr == xr) {
        node[x].tag += dt;
        return;
    }
    pushdown(x, xl, xr);
    int mid = (xl + xr) >> 1;
    update(lson(x), xl, mid, cl, min(mid, cr), dt);
    update(rson(x), mid + 1, xr, max(cl, mid + 1), cr, dt);
    pushup(x, xl, xr);
}
} st;

```

KM 字符串匹配

```

typedef long long ll;
typedef unsigned long long ull;

```

```

int km_match(char* pattern, char* str) {
    int plen = strlen(pattern);
    int slen = strlen(str);
    int dt = max(((plen - 1) >> 1), 1);
    int cnt = 0;
    ull phash = 0, shash = 0;
    ull con = 1;
    for(int i = 0; i < plen; ++i) {
        phash = ((phash << 7) + pattern[i]);
        shash = ((shash << 7) + str[i]);
        con <<= 7;
    }
    con >>= 7;
    for(int i = 0; i < slen - plen + 1; ++i) {
        if(phash == shash) {
            bool flag = true;
            for(int j = 0; j < plen; j += dt) {
                if(pattern[j] != str[i + j]) {
                    flag = false;
                    break;
                }
            }
            if(flag) ++cnt;
        }
        shash = (((shash - str[i] * con) << 7) + str[i + plen]);
    }
    return cnt;
}

```

LIS

```

struct LIS {
    // O(nlgn), strictly increase monotonically
    const static int N = 100005;
    int a[N], b[N];

    void input(int n) {
        for(int i = 1; i <= n; ++i) {
            scanf("%d", a + i);
        }
    }

    int solve(int n) {

```

```

        int len = 0;
        for(int i = 1; i <= n; ++i) {
            int* p = lower_bound(b + 1, b + len + 1, a[i]);
            *p = a[i];
            len = max(len, a[i] - b);
        }
        return len;
    }
} lis;

```

数值积分

```

namespace Int {
    const double eps = 1e-6;
    template<class _Callable>
    double Simpson(double lb, double ub, const _Callable& f) {
        double mid = (lb + ub) * .5;
        return (ub - lb) * (f(lb) + 4 * f(mid) + f(ub)) / 6.;
    }

    template<class _Callable>
    double _asr(double lb, double ub, double pre, double eps, const _Callable& f) {
        double mid = (lb + ub) * .5;
        double L = Simpson(lb, mid, f), R = Simpson(mid, ub, f);
        if(fabs(L + R - pre) < 15 * eps) return (L + R) + (L + R - pre) / 15.;
        return _asr(lb, mid, L, eps / 2, f) + _asr(mid, ub, R, eps / 2, f);
    }

    template<class _Callable>
    double asr(double lb, double ub, const _Callable& f) {
        return _asr(lb, ub, Simpson(lb, ub, f), eps, f);
    }
}

```