ACM ICPC Reference

University of São Paulo

November 11, 2017

Contents

1 Começo da prova

Vim

```
set ai si noet ts=4 sw=4 sta sm nu rnu
inoremap {<CR> {<CR>}<Esc>O
inoremap <NL> <ESC>o
nnoremap <NL> o
"carry lines insert
inoremap <C-up> <C-o>:m-2<CR>
inoremap <C-down> <C-o>:m+1<CR>
"carry lines
nnoremap <C-up> :m-2<CR>
nnoremap <C-down> :m+1<CR>
"carry lines visual
vnoremap <C-up> :m-2<CR>gv
vnoremap <C-down> :m'>+1<CR>gv
colors evening
set t_Co=8
syntax on
highlight Normal ctermbg=NONE "No background
highlight nonText ctermbg=NONE
hi Normal ctermbg=none
highlight NonText ctermbg=none
```

---

2 Fluxo

Max Flow

```
namespace f {
        const int maxv = ;
        const int maxe =  * 2;
        typedef int num;
        num inf = /*INT_MAX*/;
        int n = maxv;

        int to[maxe], en, nx[maxe], es[maxe], lv[maxv], qu[maxv], cr[maxv];
        num cp[maxe], fl[maxe];
```

```
	bool bfs(int s, int t) {
		memset(lv, -1, sizeof(int) * n);
		lv[s] = 0;
		int a = 0, b = 0;
		qu[b++] = s; cr[s] = es[s];
		while(a < b) {
			for(int i = es[qu[a]]; i != -1; i = nx[i]) {
				if(cp[i] > fl[i] && lv[to[i]] == -1) {
					lv[to[i]] = lv[qu[a]] + 1;
					qu[b++] = to[i];
					cr[to[i]] = es[to[i]];
					if(to[i] == t) return true;
				}
			}
			a++;
		}
		return false;
	}

	num dfs(int u, int t, num mx) {
		if(u == t) return mx;
		for(int &i = cr[u]; i != -1; i = nx[i])
			if(cp[i] > fl[i] && lv[to[i]] == lv[u] + 1)
				if(num a = dfs(to[i], t, min(mx, cp[i] - fl[i]))) {
					fl[i] += a;
					fl[i ^ 1] -= a;
					return a;
				}
		return 0;
	}


	num max_flow(int s, int t) {
		num fl = 0, a;
		while(bfs(s, t))
			while(a = dfs(s, t, inf))
				fl += a;
		return fl;
	}

	void reset_all(int n2=maxv) { n = n2; en = 0; memset(es, -1, sizeof(int) * n); }
	void reset_flow() { memset(fl, 0, sizeof(num) * en); }

	void add_edge(int a, int b, num c, num rc=0) {
		fl[en] = 0; to[en] = b; cp[en] = c;  nx[en] = es[a]; es[a] = en++;
		fl[en] = 0; to[en] = a; cp[en] = rc; nx[en] = es[b]; es[b] = en++;
	}
}
```

Min Cost

```
namespace f {
const int N = , M =  * 2;
typedef int val;
typedef int num;
int es[N], to[M], nx[M], en, pai[N];
val fl[M], cp[M];
num cs[M], d[N];
const num inf = 1e8, eps = 0;
const val infv = INT_MAX;
int seen[N], tempo;
int qu[N];

num tot;
bool spfa(int s, int t) {
	tempo++;
	int a = 0, b = 0;
	for(int i = 0; i < N; i++) d[i] = inf;
	d[s] = 0;
	qu[b++] = s;
	seen[s] = tempo;
	while(a != b) {
		int u = qu[a++]; if(a == N) a = 0;
		seen[u] = 0;
		for(int e = es[u]; e != -1; e = nx[e])
			if(cp[e] - fl[e] > val(0) && d[u] + cs[e] < d[to[e]] - eps) {
				d[to[e]] = d[u] + cs[e]; pai[to[e]] = e ^ 1;
				if(seen[to[e]] < tempo) { seen[to[e]] = tempo; qu[b++] =
			}
	}
	if(d[t] == inf) return false;
	val mx = infv;
	for(int u = t; u != s; u = to[pai[u]])
		mx = min(mx, cp[pai[u] ^ 1] - fl[pai[u] ^ 1]);
	tot += d[t] * val(mx);
	for(int u = t; u != s; u = to[pai[u]])
		fl[pai[u]] -= mx, fl[pai[u] ^ 1] += mx;
	return mx;
}

void init(int n) {
	en = 0;
} }	memset(es, -1, sizeof(int) * n);
}

val flow;
num mncost(int s, int t) {
	tot = 0; flow = 0;
	while(val a = spfa(s, t)) flow += a;
	return tot;
}

void add_edge(int u, int v, val c, num s) {
	fl[en] = 0; cp[en] = c; to[en] = v; nx[en] = es[u]; cs[en] =
s; es[u] = en++;
	fl[en] = 0; cp[en] = 0; to[en] = u; nx[en] = es[v]; cs[en] = -s; es[v] = en++;
```

```
}
}
```

3 Strings

Z Algorithm

```
// Z Algorithm
//////////////
const int MAXS = ;

int L, R, Z[MAXS];

char seq[MAXS];

inline void ZAlgorithm(char s[]){
        int tam = strlen(s);
        L = R = 0;
        for(int i=1;i<tam;i++){
                if(i > R){
                        L = R = i;
                        while(R < tam && s[R] == s[R-L])
                                R++;
                        Z[i] = R-L;
                        R--;
                }
                else{
                        if(Z[i-L] >= R-i+1){
                                L = i;
                                while(R < tam && s[R] == s[R-L])
                                        R++;
                                Z[i] = R-L;
                                R--;
                        }
                        else
                                Z[i] = Z[i-L];
                }
        }
}

int main (){
        scanf("%s", seq);
        ZAlgorithm(seq);
        return 0;
}
```

4 Grafos

Heavy Light Decomposition

```
int pai[MAXN][LOGN], chainNo[MAXN], chainInd[MAXN], subsize[MAXN], nchain, degs, inicha

vector <int> adj[MAXN], pes[MAXN];
vector <pair<pii,int> > ares;

int tree[8*MAXN];

int h[MAXN], custopai[MAXN];

int res;

void dfs(int v,int ant){
        pai[v][0] = ant;
        h[v] = h[ant]+1;
        for(int a=1;a<LOGN;a++){
                pai[v][a] = pai[pai[v][a-1]][a-1];
        }
        subsize[v] = 1;
        for(int a=0;a<adj[v].size();a++){
                int nxt = adj[v][a];
                if(nxt != ant){
                        custopai[nxt] = pes[v][a];
                        dfs(nxt,v);
                        subsize[v] += subsize[nxt];
                }
        }
}

void HLD(int v){
        chainInd[v] = degs;
        chainNo[v] = nchain;
        int mai=-1, ind=-1, maip = -1;
        for(int a=0;a<adj[v].size();a++){
                int nxt = adj[v][a];
                int cus = pes[v][a];
                if(nxt == pai[v][0]) continue;
                if(mai < subsize[nxt]){
                        mai = subsize[nxt];
                        maip = cus;
                        ind = nxt;
                }
        }
        if(mai != -1){
                if(inichain[nchain] == -1){
                        inichain[nchain] = v;
                }
                s[degs++] = maip;
                HLD(ind);
        }
        else{
```

```
                // eh uma folha
                if(inichain[nchain] == -1){
                        inichain[nchain] = v;
                }
                s[degs++] = 0; //para que todo no tenha uma valor na seg
        }
        for(int a =0;a<adj[v].size();a++){
                int nxt = adj[v][a];
                int cus = pes[v][a];
                if(nxt == pai[v][0] || nxt == ind) continue;
                nchain++;
                HLD(nxt);
        }
}

void build(int idx,int i,int j){
        if(i==j){
                tree[idx] = s[i];
                return;
        }
        int m = (i+j)/2;
        build(idx*2,i,m);
        build(idx*2+1,m+1,j);
        tree[idx] = max(tree[idx*2],tree[idx*2+1]);
}

int LCA(int i,int j){
        if(h[j] > h[i]) swap(i,j);

        if(h[i] > h[j]){
                for(int a = LOGN-1;a>=0;a--){
                        if(h[pai[i][a]] > h[j]){
                                i = pai[i][a];
                        }
                }
                i = pai[i][0];
        }

        if(i==j) return i;

        for(int a=LOGN-1;a>=0;a--){
                if(pai[i][a] != pai[j][a]){
                        i = pai[i][a];
                        j = pai[j][a];
                }
        }
        return pai[i][0];
}

void qry(int idx,int i,int j,int l, int r){
        if(i > r || j < l) return ;
        if(i>=l && j<=r){
                res = max(res,tree[idx]);
                return;
        }
        int m = (i+j)/2;
        qry(idx*2,i,m,l,r);
        qry(idx*2+1,m+1,j,l,r);
}

void qryup(int i,int j){
        while(chainNo[i] != chainNo[j]){
                int j2 = inichain[chainNo[i]];
                int ii = chainInd[i], jj = chainInd[j2];
                if(ii!=jj)
                        qry(1,1,degs-1,jj,ii-1);
                res = max(res, custopai[j2]);
                i = pai[j2][0];
        }
        int ii = chainInd[i], jj = chainInd[j];
        if(ii==jj) return;
        qry(1,1,degs-1,jj,ii-1);
}

void upd(int idx,int i, int j, int l, int val){
        if(i>l || j<l) return ;
        if(i == j){
                tree[idx] = val;
                return ;
        }
        int m = (i+j)/2;
        upd(idx*2,i,m,l,val);
        upd(idx*2+1,m+1,j,l,val);
        tree[idx] = max(tree[idx*2],tree[idx*2+1]);
}

void reset(){
        degs=1;
        nchain = 0;
        memset(inichain,-1,sizeof(inichain));
        memset(tree,0,sizeof(tree));
        ares.clear();
        for(int a=0;a<=n;a++){
                adj[a].clear();
                pes[a].clear();
        }
        h[pri] = -1;
        dfs(pri,pri);
        nchain = 0;
        HLD(pri);
        build(1,1,degs-1);

}
```

5 Geometria

Header

```
/////Header de Geometria/////

// area de calota 2.pi.R.h (h altura)
// volume de calota pi.h/6 * (3r^2 + h^2)

#include <cmath>
#define temp template<typename num>
#define ptn point<num>
temp struct point {
	num x, y;
	ptn() {}
	ptn(num a, num b) : x(a), y(b) {}
	ptn operator + (ptn o) const { return ptn(x + o.x, y + o.y); }
	ptn operator - (ptn o) const { return ptn(x - o.x, y - o.y); }
	num operator * (ptn o) const { return x * o.x + y * o.y; }
	num operator ^ (ptn o) const { return x * o.y - y * o.x; }
	ptn operator * (num i) const { return ptn(x * i, y * i); }
	ptn operator / (num i) const { return ptn(x / i, y / i); }
	point<double> rotate(double deg) {
		double cs = cos(deg), sn = sin(deg);
		return point<double>(x*cs - y*sn, x*sn + y*cs);
	}
	num dist_sqr(ptn o) const { return (*this - o) * (*this - o); }
	bool operator < (ptn o) const { return x < o.x || (x == o.x && y < o.y); }
};
typedef point<int> pti;
typedef point<double> ptd;
temp inline num cross(ptn a, ptn b, ptn c) { return (c - a) ^ (b - a); }
// o ponto c esta no segmento [a, b]?
temp inline bool between_seg(ptn a, ptn b, ptn c) { return cross(a, b, c) == 0 && ((b - c) * (a - c) <= 0); }
// sqr dist de c pro segmento [a, b]
temp double dist_seg_sqr(ptn a, ptn b, ptn c) {
	if((b - a) * (c - b) > 0) return b.dist_sqr(c);
	if((a - b) * (c - a) > 0) return a.dist_sqr(c);
	double d = (b - a) ^ (c - a);
	return d * d / ((b - a) * (b - a));
}
temp int sign(num x) { return (x > 0) - (x < 0); }
// [a, b] intersecta [c, d]?
temp bool inter_seg(ptn a, ptn b, ptn c, ptn d) {
	if(between_seg(a, b, c) || between_seg(a, b, d) || between_seg(c, d, a) || between_seg(c, d, b)) return true;
	if((sign(cross(a, b, c)) * sign(cross(a, b, d)) == -1) && (sign(cross(c, d, a)) * sign(cross(c, d, b)) == -1)) return true;
	return false;
}

temp struct line {
	num a, b, c;
	line() {}
	line(num aa, num bb, num cc) : a(aa), b(bb), c(cc) {}
	line(ptn s, ptn e) : a(e.y - s.y), b(s.x - e.x), c(a*s.x + b*s.y) {}
	line pass(ptn p) { return line(a, b, a*p.x + b*p.y); }
	bool parallel(const line &o) const { return a * o.b - o.a * b == 0; }
	point<double> inter(line o) {
		double d = a * o.b - o.a * b;
		if(d == 0) return point<double>(0, 0); // fudeu
		return point<double>((o.b * c - b * o.c)/d, (a * o.c - o.a * c)/d);
	}
};
typedef line<int> lni;
typedef line<double> lnd;

// convex hull - modifique como necessario
void convex_hull(pti p[], pti st[], int n) {
	sort(p, p + n);
	int sn = 0;
	for(int i = 0; i < n; i++) {
		while(sn >= 2 && cross(st[sn - 2], st[sn - 1], p[i]) > 0)
			sn--;
		st[sn++] = p[i];
	}
	int k = sn;
	for(int i = n - 2; i >= 0; i--) {
		while(sn > k && cross(st[sn - 2], st[sn - 1], p[i]) > 0)
			sn--;
		st[sn++] = p[i];
	}
	sn--;
	// st[0..sn-1] agora tem o convex hull dos pontos p
}
```

6 FFTs

Marcos

```
typedef complex<long double> Complex;
const long double PI = acos(-1.0L);

// Computes the DFT of vector v if type = 1, or the IDFT if type = -1
// If you are calculating the product of polynomials, don't forget to set both
// vectors' degrees to at least the sum of degrees of both polynomials, regardless
// of whether you will use only the first few elements of the resulting array
vector<Complex> FFT(vector<Complex> v, int type) {
    int n = v.size();
    while (n & (n - 1)) { v.push_back(0); n++; }
    int logn = __builtin_ctz(n);
    vector<Complex> v2(n);
    for (int i = 0; i < n; i++) {
        int mask = 0;
        for (int j = 0; j < logn; j++)
            if (i & (1 << j))
                mask |= (1 << (logn - 1 - j));
        v2[mask] = v[i];
    }
    for (int len = 1; 2 * len <= n; len <<= 1) {
        Complex wm(cos(type * PI / len), sin(type * PI / len));
        for (int i = 0; i < n; i += 2 * len) {
            Complex w = 1;
            for (int j = 0; j < len; j++) {
                Complex t = w * v2[i + j + len], u = v2[i + j];
                v2[i + j] = u + t; v2[i + j + len] = u - t;
                w *= wm;
            }
        }
    }
    if (type == -1) for (Complex &c: v2) c /= n;
    return v2;
}
```

Nath

```
#include <bits/stdc++.h>
using namespace std;
using  ll = int64_t;
using ull = uint64_t;

const int P = 7*17*(1<<23) + 1;
const int g = 3;

struct mod{
    int x;
    mod(ll y) : x((y+ll(P))%P) {}
    mod() : x(0) {}
    mod operator+(mod b) { return ll(x) + ll(b.x); }
    mod operator-(mod b) { return ll(x) - ll(b.x); }
    mod operator*(mod b) { return ll(x) * ll(b.x); }
};
mod fexp(mod a, ll e){
    mod t = 1;
    for(;e;a = a*a, e>>=1) if(e&1) t = t*a;
    return t;
}

namespace FFT {
    int n;
    vector<int> p;
    vector<mod> fft, w;
    void dft(vector<mod> &a, bool inv){
        mod t, odd; w[0] = 1;
        for(int i=0;i<n;i++) fft[p[i]] = a[i];
        for(int s=1;s<n;s *= 2){
            t = fexp(g, (P-1)/(2*s));
            if(inv) t = fexp(t, 2*s - 1);
            for(int i=1;i<s;i++) w[i] = t*w[i-1];
            for(int l=0;l<n;l+=2*s)
                for(int i=l;i<l+s;i++){
                    fft[i+s] = fft[i] - (odd = w[i-l]*fft[i+s]);
                    fft[i]   = fft[i] + odd;
                }
        }
        if(inv) {
            t = fexp(n, P-2);
            for(int i=0;i<n;i++) fft[i] = t*fft[i];
        }
        a = fft;
    }
    void conv(vector<mod> a, vector<mod> b, vector<mod> &c){
        n = a.size() + b.size();
        for(;n != (n&-n); n -= (n&-n)); n *= 2;
        a.resize(n); b.resize(n); c.resize(n);
        fft.resize(n); w.resize(n); p.resize(n);
        for(int i=0;i<n;i++) p[i] = ((i&1)*(n>>1))|(p[i>>1]>>1);
        dft(a, 0); dft(b, 0);
        for(int i=0;i<n;i++) c[i] = a[i]*b[i];
        dft(c,1);
    }
}
```

Contents

1 Tudo

1.1 Dinic

```
// Dinico da massa
// Generico: O(V^2 E)
// BipMatch: O(V^(1/2) E)
// UnitCap: O(min{V^(2/3),E^(1/2)} E)
#include <bits/stdc++.h>
using namespace std;

struct dinic {
    int hd[N], nx[M], to[M], ht[M], es;
    ll fl[M], cp[M];
    int n, src, snk;
    int dist[N], seen[N], visi[N], turn;
    int qi, qf, qu[N];

    inline void init () // antes de montar o grafo
    { es = 2; }

    inline void reset () {
        es = 2;
        memset(hd, 0, sizeof hd);
        memset(seen, 0, sizeof seen);
        memset(visi, 0, sizeof visi);
    }

    inline void connect (int i, int j, ll cap) {
 //      printf("%d-%d [%lld]\n", i, j, cap);
        nx[es] = hd[i]; hd[i] = es; to[es] = j; cp[es] = cap; fl[es] = 0;
           es++;
        nx[es] = hd[j]; hd[j] = es; to[es] = i; cp[es] = fl[es] = 0; es++;
    }

    bool bfs () {
        turn++;
        qi = qf = 0;

        qu[qf++] = snk;
        dist[snk] = 0;
        seen[snk] = turn;

        while (qi < qf) {
            int u = qu[qi++];
```

```
        if (visi[u] == turn)
            continue;
        visi[u] = turn;

        for (int ed = hd[u]; ed; ed = nx[ed]) {
            if (cp[ed^1] == fl[ed^1])
                continue;
            int v = to[ed];

            if (seen[v] == turn && dist[v] <= dist[u]+1)
                continue;
            seen[v] = turn;
            dist[v] = dist[u]+1;
            qu[qf++] = v;
        }
    }

    return (seen[src] == turn);
}

ll dfs (int u, ll flw) {
    if (u == snk || flw == 0)
        return flw;

    for (int & ed = ht[u]; ed; ed = nx[ed]) {
        int v = to[ed];
        if (fl[ed] >= cp[ed] || seen[v] != turn || dist[v]+1 != dist[u])
            continue;
        if (ll ret = dfs(v, min(flw, cp[ed] - fl[ed]))) {
            fl[ed] += ret;
            fl[ed^1] -= ret;
            return ret;
        }
    }

    return 0;
}

ll debug () {
    for (int i = 0; i < n; i++){
        printf("%d:", i);
        for (int ed = hd[i]; ed; ed = nx[ed])
            printf(" %d[%lld/%lld]", to[ed], fl[ed], cp[ed]);
        printf("\n");
    }
}
```

```
    ll maxflow () {
        ll res = 0;
        while (bfs()) {
            for (int i = 0; i < n; i++)
                ht[i] = hd[i];
            while (ll val = dfs(src, LLONG_MAX))
                res += val;
        }
        return res;
    }
};
```

1.2 HLD

```
#include <bits/stdc++.h>
using namespace std;
typedef int node;
typedef int edge;
const int N = 1e5;
const int M = 2*N;

int n;
int hd[N];
int to[M], nx[M], es;

int id[N], *leaf, *sz;
int chain[N], d[N], anc[N], tree[2*N];
int ps, ls;
int dfs(node u, node p)
{
    int bst = 0, spc = u;
    sz[id[ps++]=u] = 1;
    d[u] = d[anc[u]=p]+1;
    for(edge e=hd[u];e!=-1;e=nx[e])
        if(sz[to[e]] == 0)
        {
            sz[u] += dfs(to[e], u);
            if(sz[to[e]] > bst)
                bst = sz[to[e]], spc = to[e];
        }
    if(sz[u] == 1) leaf[ls++] = u;
    return sz[chain[u] = chain[spc] = u];
}
void build(node root)
```

```
{
    leaf = tree;
    sz   = tree + n;
    ps = ls = 0;
    dfs(root, root);
    for(int i=0;i<n;i++) chain[id[i]] = chain[chain[id[i]]];
    while(ls)
    {
        node u = leaf[--ls];
        while(u != chain[u])
            u = anc[tree[id[u]=ps++]=u];
        tree[id[u]=ps++] = u;
    }
}
```

1.3 Treap (Implicit)

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5+7;
struct treap {
    int x[N], y[N], sz[N], T[N][2], ts;
    int s[2];
    void init() {
        srand(time(NULL));
        ts = 1;
        for(int i=0;i<N;i++) y[i] = i;
        random_shuffle(y, y+N);
    }
    void split(int t, int k){
        if(!t) return (void)(s[0] = s[1] = 0);
        bool d = sz[T[t][0]] < k;
        split(T[t][d], k - d*(1+sz[T[t][0]])); T[t][d] = s[!d]; s[!d] = t;
        sz[t] = 1 + sz[T[t][0]] + sz[T[t][1]];
    }
    int merge(int tl, int tr){
        int r;
        if(!min(tl,tr)) return max(tl, tr);
        if(y[tl] < y[tr]) T[r=tr][0] = merge(tl, T[tr][0]);
        else              T[r=tl][1] = merge(T[tl][1], tr);
        sz[r] = 1 + sz[T[r][0]] + sz[T[r][1]];
        return r;
    }
    // Tudo abaixo opcional
```

```
    int count_less(int tree, int k){
        int ans = 0, t = tree;
        for(;t && x[t] != k;t=T[t][x[t] < k])
            ans += (x[t] < k)*(1 + sz[T[t][0]]);
        return ans + sz[T[t][0]];
    }
    int kth_element(int tree, int k){
        int t;
        if(sz[tree] < k) return -1;
        split(tree, k);
        for(t=s[0];T[t][1];t=T[t][1]);
        merge(s[0], s[1]);
        return t;
    }
    int insert(int tree, int k){
        int t = tree, less = 0;
        for(;t && x[t] != k;t=T[t][x[t] < k]) less += (x[t] <
           k)*(1+sz[T[t][0]]);
        if(t) return tree;
        split(tree, less); x[ts] = k; sz[ts] = 1;
        return merge(s[0],merge(ts++,s[1]));
    }
    int remove(int t, int k){
        if(!t) return t;
        if(x[t] == k) return merge(T[t][0], T[t][1]);
        T[t][x[t] < k] = remove(T[t][x[t] < k], k);
        sz[t] = 1 + sz[T[t][0]] + sz[T[t][1]];
        return t;
    }
};

int q, tree;
int main(){
    treap T;
    T.init();

    scanf(" %d", &q);
    while(q--){
        char cmd; int tgt;
        scanf(" %c%d", &cmd, &tgt);
        if(cmd == 'I') tree = T.insert(tree, tgt);
        else if(cmd == 'D') tree = T.remove(tree, tgt);
        else if( cmd == 'C') printf("%d\n", T.count_less(tree, tgt));
        else {
            int tgt = T.kth_element(tree, tgt);
            if(tgt == -1) puts("invalid");
```

```
            else printf("%d\n", T.x[tgt]);
        }
    }
}
```

1.4 KMP

```
#include <bits/stdc++.h>
using namespace std;
const int S = 1e6;
// Usa string indexada em 1
int kmp[S+1], m, n;
char P[S+1], T[S+1];

void build()
{
    int k;
    kmp[0] = k = -1;
    for(int i=1;i<m+1;i++)
    {
        while(k >= 0 && P[k+1] != P[i]) k = kmp[k];
        kmp[i] = ++k;
    }
}

void match()
{
    int j = 0;
    for(int i=1;i<n+1;i++)
    {
        while(j >= 0 && P[j+1] != T[i]) j = kmp[j];
        if(++j == m)
        {
            printf("Match comeando em %d\n", i-j+1);
            j = kmp[j];
        }
    }
}
```

1.5 Ordered Set (GCC only)

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
```

```
#define GNU __gnu_pbds

using namespace std;
typedef unsigned long long int ull;
typedef long long int ll;
namespace GNU { typedef tree<int, null_type, less<int>, rb_tree_tag,
   tree_order_statistics_node_update> ordered_set; }
// null_type pode mudar pra virar map
// multiset e multimap no tem suporte aqui, tem que sair usando pair pelo
   jeito
// aquele less ali  o comparador, louco n?
using GNU::ordered_set;

#ifdef ONLINE_JUDGE
#define lld I64d
#define DEBUG(...) {fprintf(stderr, __VA_ARGS__);}
#else
#define DEBUG(...) {}
#endif

ordered_set s;
int q;
int a;

int main () {
    while (scanf("%d %d", &q, &a) != EOF) {
        if (q == 1) {
            s.insert(a);
        } else if (q == 2){
            if (end(s) == s.find_by_order(a))
                printf("no\n");
            else
                printf("%d\n", (int) *s.find_by_order(a)); // se voc
                   esquecer do cast ou do *, o gcc vai falar que sua famlia
                   no presta
        } else {
            printf("%d\n", (int) s.order_of_key(a));
        }
    }

}
```

1.6 Simpson

```
#include <bits/stdc++.h>
```

```
const double eps = 1e-7;

//INTEGRAL NUMERICA PELO METODO DE SIMPSON

// funcao a ser integrada
double f(double x){
    return x*x;
}

//aproximacao da integral no intervalo a,b de Simpson (nao o Homer)
//funciona muito bem para intervalos pequenos
double simpson(double a, double b) {
    return (f(a) + 4*f((a+b)/2) + f(b))*(b-a)/6;
}

//a ideia eh quebrar o intervalo a,b em duas metades e verificar se a
   aproximacao
//ja esta boa o suficiente. se nao estiver, integramos as metades
   recursivamente.
double integrate(double a, double b) {
    double m = (a+b)/2;
    double l = simpson(a,m), r = simpson(m,b), tot = simpson(a,b);
    if(fabs(l+r-tot) < eps) return tot;
    return integrate(a,m) + integrate(m,b);
}
```

1.7 Suffix Array

```
#include<bits/stdc++.h>
using namespace std;
const int S = 5e5+2;
const int L = 30;
char s[S];
int n;
int p[S],
    fst[S], snd[S],
    lcp[S], sa[S];
int buck[S], nx[S], idx[S];

void bucket_sort(int * v)
{
    for(int i=0;i<n;i++)
    {
        idx[i] = sa[i];
        nx[i] = buck[v[sa[i]]];
```

```
        buck[v[sa[i]]] = i;
    }
    int pos = n-1, k = max(CHAR_MAX, n-1);
    while(pos >= 0)
    {
        while(buck[k] == -1) k--;
        int nd = buck[k];
        while(nd != -1)
        {
            sa[pos--] = idx[nd];
            nd = nx[nd];
        }
        buck[k] = -1;
    }
}

bool lexLess(int i, int j)
{ return fst[i] == fst[j] ? snd[i] < snd[j] : fst[i] < fst[j]; }
void build_sa()
{
    n = strlen(s);
    memset(buck, -1, sizeof buck);
    int k;
    p[n] = -1;
    for(int i=0;i<n;i++) p[i] = s[i];
    for(k=1;k<ML;k++)
    {
        for(int i=0;i<n;i++)
        {
            fst[i] = p[sa[i] = i];
            snd[i] = p[min(i+(1<<(k-1)), n)];
        }
        bucket_sort(snd);
        bucket_sort(fst);
        p[sa[0]] = 0;
        for(int i=1;i<n;i++)
            p[sa[i]] = p[sa[i-1]] + (fst[sa[i-1]] != fst[sa[i]] ||
               snd[sa[i-1]] != snd[sa[i]]);
        if( p[sa[n-1]] == n-1 ) break;
    }
    for(int i=0;i<n;i++) sa[p[i]] = i;
    int h = 0;
    for(int i=0;i<n;i++)
        if(p[i] == n-1) h = 0;
        else
        {
```

```
            int j = sa[p[i]+1];
            while(i+h < n && j+h < n && s[i+h] == s[j+h]) h++;
            lcp[p[i]]=h;
            h -= (!!h);
        }
}
```

1.8 Hungarian

```
// Mtodo Hungaro para Emparelhamento Perfeito Bipartido de Peso Mximo
// O(n^3)
// http://www.ime.usp.br/~oshiro/maratona/bipmat/bipmatching.pdf
#include <bits/stdc++.h>

using namespace std;

typedef long long int num;

const int N = 200;
struct hungarian {
    int n;              // tamanho
    int match[N];       // no emparelhado com j (-1 = nenhum)

    num cost[N][N];     // peso da aresta ij
    num pot[2][N];      // potenciais pot[0][i]=y[i] e pot[1][j]=z[j]
    num diff[N];        // delta

    int tree[2][N], empr;
    int visi[N], turn;

    void init() {
        memset(match, -1, sizeof match);

        for (int i = 0; i < n; i++) {
            diff[i] = INT_MAX;
            pot[1][i] = 0;

            pot[0][i] = INT_MIN;
            for (int j = 0; j < n; j++)
                pot[0][i] = max(pot[0][i], cost[i][j]);
        }
    }

    void include (int u) {
        if (tree[0][u] == empr) // muito importante, sem isso vira n^4
```

```
        return;

    tree[0][u] = empr;
    for (int i = 0; i < n; i++)
        diff[i] = min(diff[i], pot[0][u] + pot[1][i] - cost[u][i]);
}

void upd () {
    num del = INT_MAX;
    for (int i = 0; i < n; i++)
        if (tree[1][i] != empr)
            del = min(del, diff[i]);

    for (int i = 0; i < n; i++)
        if (tree[0][i] == empr)
            pot[0][i] -= del;

    for(int i = 0; i < n; i++)
        if (tree[1][i] == empr)
            pot[1][i] += del;
        else
            diff[i] -= del;
}

bool dfs (int u) {
    if (u == -1)
        return 1;
    if (visi[u] == turn)
        return 0;

    visi[u] = turn;
    include(u);

    for (int i = 0; i < n; i++) {
        if (pot[0][u] + pot[1][i] != cost[u][i])
            continue;

        tree[1][i] = empr;
        if (dfs(match[i])) {
            /* DEBUG: imprime caminhos encontrados
            printf("(%d-%d)", u, i);
            if (match[i] == -1)
                printf(" on empr %d\n", empr);
            else
                printf("->");
            */
```

```
                match[i] = u;
                return 1;
            }
        }

        return 0;
    }

    void solve () {
        init();

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                diff[j] = INT_MAX;

            empr++;
            while (++turn && !dfs(i))
                upd();
        }
    }

    num value () {
        /* DEBUG: imprime emparelhamento
        for (int i = 0; i < n; i++)
            printf("%d<->%d\n", match[i], i);
        */

        num res = 0;
        for (int i = 0; i < n; i++)
            res += cost[match[i]][i];
        return res;
    }
};
```