**Strongly Connected Component\_UVa\_11838\_Come and Go.cpp**

**int** vs, es;

**vector**<**int**> g[MAX\_V];

**int** t, low[MAX\_V], dfn[MAX\_V];

**char** instk[MAX\_V];

**int** stk[MAX\_V], top;

**int** scc\_cnt;

**void** dfs(**int** vi)

{

dfn[vi] = t++;

low[vi] = dfn[vi];

stk[++top] = vi;

instk[vi] = 1;

**int** i, sz = g[vi].size();

**for** (i = 0; i < sz; i++)

{

**int** &vj = g[vi][i];

**if** (dfn[vj] == -1)

{

dfs(vj);

low[vi] = **min**(low[vi], low[vj]);

}

**else** **if** (instk[vj])

low[vi] = **min**(low[vi], dfn[vj]);

}

**if** (low[vi] == dfn[vi])

{

scc\_cnt++;

**while** (stk[top] != vi)

{

instk[stk[top]] = 0;

--top;

}

instk[stk[top]] = 0;

--top;

}

}

**int** **main**()

{

**int** i, j;

**int** v1, v2, dir;

**while** (**scanf**("%d%d", &vs, &es) == 2 && (vs | es))

{

**for** (i = 1; i <= vs; i++) g[i].clear();

t = 0;

**memset**(dfn, -1, **sizeof**(dfn));

**memset**(instk, 0, **sizeof**(instk));

top = -1;

scc\_cnt = 0;

**for** (i = 0; i < es; i++)

{

**scanf**("%d%d%d", &v1, &v2, &dir);

g[v1].push\_back(v2);

**if** (dir == 2) g[v2].push\_back(v1);

}

**for** (i = 1; i <= vs; i++)

**if** (dfn[i] == -1)

dfs(i);

**printf**("%d\n", scc\_cnt == 1);

}

**return** 0;

}

**Articulation Point\_UVa\_10765\_Doves and bombs.cpp**

**pair**<**int**, **int**> ans[MAX\_V];

**vector**<**int**> g[10002];

**int** vs, es, t;

**int** low[MAX\_V], dfn[MAX\_V];

**int** dfs(**int** p, **int** vi)

{

dfn[vi] = ++t;

low[vi] = dfn[vi];

**int** i, ch = 0, cnt = 0, sz = g[vi].size();

**for** (i = 0; i < sz; i++)

{

**int** &vj = g[vi][i];

**if** (dfn[vj] == -1)

{

++ch;

dfs(vi, vj);

**if** (low[vj] >= dfn[vi]) cnt++;

low[vi] = **min**(low[vi], low[vj]);

}

**else** **if** (vj != p)

low[vi] = **min**(low[vi], dfn[vj]);

}

**if** (p != -1) // not root

{

**if** (cnt) ans[vi].second = cnt+1;

}

**else**

{

**if** (ch > 1) ans[vi].second = ch;

}

**return** low[vi];

}

**int** **main**()

{

**int** i, j, m, v1, v2;

**while** (**scanf**("%d%d", &vs, &m) == 2 && (vs | m))

{

t = 0;

**memset**(dfn, -1, **sizeof**(dfn));

**for** (i = 0; i < vs; i++)

{

g[i].clear();

ans[i].first = i;

ans[i].second = 1;

}

**while** (**scanf**("%d%d", &v1, &v2) == 2 && !(v1 == -1 && v2 == -1))

{

g[v1].push\_back(v2);

g[v2].push\_back(v1);

}

**for** (i = 0; i < vs; i++)

**if** (dfn[i] == -1)

dfs(-1, i);

**sort**(ans, ans+vs, cmp);

**for** (i = 0; i < m; i++)

**printf**("%d %d\n", ans[i].first, ans[i].second);

**puts**("");

}

**return** 0;

}

**Tarjan's Offline LCA\_POJ\_1470\_Closest Common Ancestors.cpp**

**int** vs, es;

**vector**<**int**> g[MAX];

**int** f[MAX], vleft[MAX], top, lca[MAX][MAX];

**int** pre[MAX], cnt[MAX];

**void** dfs(**int** idx)

{

**int** i, sz = g[idx].size();

**for** (i = 0; i < sz; i++)

{

**int** &idx2 = g[idx][i];

dfs(idx2);

vleft[++top] = idx2;

unionF(idx, idx2);

}

**for** (i = 0; i <= top; i++)

lca[ vleft[i] ][idx] =

lca[idx][ vleft[i] ] = findF( vleft[i] );

}

**int** **main**()

{

**int** i, j, vi, vj, ps;

**while** (**scanf**(" %d", &vs) == 1)

{

top = -1;

**for** (i = 1; i <= vs; i++)

{

g[i].clear();

pre[i] = i;

f[i] = i;

cnt[i] = 0;

}

**for** (i = 0; i < vs; i++)

{

vi = input(); es = input();

**while** (es--)

{

vj = input();

g[vi].push\_back(vj);

pre[vj] = vi;

}

}

**while** (pre[vi] != vi) vi = pre[vi];

dfs(vi);

ps = input();

**while** (ps--)

{

vi = input(); vj = input();

cnt[ lca[vi][vj] ]++;

}

**for** (i = 1; i <= vs; i++)

**if** (cnt[i] > 0)

**printf**("%d:%d\n", i, cnt[i]);

}

**return** 0;

}

**Dinic's Algorithm.cpp**

#define MAX\_V 102

#define INF 1 << 30

**using** **namespace** std;

**struct** edge{ **int** to, res, rev; };

**vector**<edge> g[MAX\_V];

**int** lvl[MAX\_V];

**queue**<**int**> q;

**int** iter[MAX\_V];

**void** addEdge(**int** from, **int** to, **int** fcap, **int** bcap)

{

g[from].push\_back( edge***{*** to, fcap, g[to].size() ***}*** );

g[to].push\_back( edge***{*** from, bcap, g[from].size()-1 ***}*** );

}

**void** doFlow(edge &e, **int** f)

{

e.res -= f;

g[e.to][e.rev].res += f;

}

**int** bfs(**const** **int** &src, **const** **int** &sink)

{

**int** i, sz, qi;

**memset**(lvl, -1, **sizeof**(lvl));

**while** (!q.empty()) q.pop();

lvl[src] = 0;

q.push(src);

**while** (!q.empty())

{

qi = q.front();

sz = g[qi].size();

**for** (i = 0; i < sz; i++)

{

edge &e = g[qi][i];

**if** (e.res > 0 && lvl[e.to] < 0)

{

lvl[e.to] = lvl[qi] + 1;

q.push(e.to);

}

}

q.pop();

}

**return** lvl[sink];

}

**int** dfs(**int** idx, **int** minF, **const** **int** &dest)

{

**if** (idx == dest) **return** minF;

**int** i, sz = g[idx].size(), ret;

**for** (**int** &i = iter[idx]; i < sz; i++)

{

edge &e = g[idx][i];

**if** (e.res > 0 && lvl[idx] < lvl[e.to])

{

ret = dfs(e.to, **min**(minF, e.res), dest);

**if** (ret > 0)

{

doFlow(e, ret);

**return** ret;

}

}

}

**return** 0;

}

**int** maxFlow(**const** **int** &src, **const** **int** &sink)

{

**int** ret, f = 0;

**while** ( bfs(src, sink) >= 0 )

{

**memset**(iter, 0, **sizeof**(iter));

**while** ( (ret = dfs(src, INF, sink)) > 0)

f += ret;

}

**return** f;

}

**int** **main**()

{

**int** i, cases = 0, vs, src, sink, es, v1, v2, cap;

**while** (**scanf**("%d", &vs) == 1 && vs > 0)

{

**for** (i = 1; i <= vs; i++) g[i].clear();

**scanf**("%d%d%d", &src, &sink, &es);

**while** (es--)

{

**scanf**("%d%d%d", &v1, &v2, &cap);

addEdge(v1, v2, cap, cap);

}

**printf**("Network %d\n", ++cases);

**printf**("The bandwidth is %d.\n\n", maxFlow(src, sink));

}

**return** 0;

}

**Knuth Morris Pratt.cpp**

#include <cstdio>

#include <cstring>

#define MAXL 1000

**using** **namespace** std;

**void** pre(**char** \*pat, **int**\* &f)

{

**int** i, j, l=**strlen**(pat);

f = **new** **int**[l];

f[0] = -1;

**for** (i = 1; i < l; i++)

{

j = i-1;

**while** (pat[f[j]+1] != pat[i])

{

j = f[j];

**if** (j == -1)

**break**;

}

f[i] = (j == -1) ? -1 : f[j]+1;

}

**return** ;

}

**void** KMP(**char** \*s, **char** \*pat)

{

**int** i, j, \*f, ls=**strlen**(s), lp=**strlen**(pat);

pre(pat, f);

**for** (i = 0, j = 0; i < ls; )

{

**if** (s[i] == pat[j])

{

i++;

j++;

**if** (j == lp)

{

**printf**("found at %d\n", i-lp);

j = f[j-1] + 1;

}

}

**else**

{

**if** (j == 0)

i++;

**else**

j = f[j-1] + 1;

}

}

**return** ;

}

**int** **main**()

{

**char** s[MAXL+1], pat[MAXL+1]; // char \*pat="abcaabcacab";

**gets**(pat);

**while**( **gets**(s) )

KMP(s, pat);

**return** 0;

}

**Fenwick Tree.cpp**

/\* Arrays numbered from 1 \*/

#include <cstdio>

#include <cstring>

#define MAX 100002

**using** **namespace** std;

**int** a[MAX], fenwick[MAX], maxIdx;

**int** query(**int** idx)

{

**int** sum = 0;

**while** (idx > 0)

{

sum += fenwick[idx];

idx -= idx & (-idx);

}

**return** sum;

}

**void** update(**int** idx, **int** n)

{

**while** (idx <= maxIdx)

{

fenwick[idx] += n;

idx += idx & (-idx);

}

**return** ;

}

**int** **main**()

{

**int** i;

**memset**(fenwick, 0, **sizeof**(fenwick));

maxIdx = 5;

**for** (i = 1; i <= 5; i++)

update(i, +i);

**for** (i = 1; i <= 5; i++)

**printf**("%d\n", query(i));

**return** 0;

}

**Fenwick Tree\_2D.cpp**

**const** **int** R = 502;

**const** **int** C = 502;

**int** r, c;

**int** a[R][C], s[R][C];

**int** query(**int** x, **int** y)

{

**int** yy, ret = 0;

**while** (x >= 1)

{

yy = y;

**while** (yy >= 1)

{

ret += s[x][yy];

yy -= yy & (-yy);

}

x -= x & (-x);

}

**return** ret;

}

**void** update(**int** x, **int** y, **int** v)

{

**int** yy;

**while** (x <= r)

{

yy = y;

**while** (yy <= c)

{

s[x][yy] += v;

yy += yy & (-yy);

}

x += x & (-x);

}

}

**void** init()

{

**memset**(s, 0, **sizeof**(s));

}

**int** **main**()

{

**int** i, j;

**while** (**cin** >> r >> c)

{

init();

**for** (i = 1; i <= r; i++)

**for** (j = 1; j <= c; j++)

{

**scanf**("%d", &a[i][j]);

update(i, j, a[i][j]);

}

**cout** << query(r, c) << **endl**;

}

**return** 0;

}

**Treap\_POJ\_2761\_Feed the dogs.cpp**

#define MAX\_N 100005

#define MAX\_M 50005

**using** **namespace** std;

**struct** treap;

**int** size(treap\* &);

**struct** treap

{

**int** pri;

**int** key, size;

treap \*chi[2];

treap(): pri(**rand**()), size(1) { chi[0] = chi[1] = **NULL**; }

**void** pull()

{

size = ::size(chi[0]) + ::size(chi[1]) + 1;

}

} \*root = **NULL**;

**int** size(treap\* &cur)

{

**return** cur ? cur -> size : 0;

}

**void** split(treap \*cur, **const** **int** &key, treap\* &l, treap\* &r)

{

**if** (!cur) { l = r = **NULL**; **return** ; }

**if** (cur->key <= key)

{

l = cur;

split(l->chi[1], key, l->chi[1], r);

l->pull();

}

**else**

{

r = cur;

split(r->chi[0], key, l, r->chi[0]);

r->pull();

}

}

treap \* **merge**(treap \*l, treap \*r) // l->key < r->key, for all children

{

**if** (!l) **return** r;

**if** (!r) **return** l;

**if** (l->pri > r->pri)

{

l->chi[1] = **merge**(l->chi[1], r);

l->pull();

**return** l;

}

**else**

{

r->chi[0] = **merge**(l, r->chi[0]);

r->pull();

**return** r;

}

}

**void** insert(**const** **int** &key)

{

**if** (root == **NULL**) { root = **new** treap; root->key = key; **return** ; }

treap \*l, \*r, \*n = **new** treap;

n->key = key;

split(root, key, l, r);

root = **merge**(l, n);

root = **merge**(root, r);

}

**void** build(**int** a[], **int** len)

{

**int** i;

**for** (i = 0; i < len; i++)

insert(a[i]);

}

**void** del(**const** **int** &key)

{

treap \*l, \*r, \*ll, \*lr, \*n;

split(root, key, l, r);

**if** (!l) **return** ;

// if exists key

split(l, key-1, ll, lr);

// drop 1 from lr

n = **merge**(lr->chi[0], lr->chi[1]); // can be null

n = **merge**(ll, n); // can be null

root = **merge**(n, r);

**delete** lr;

}

treap \*getRk(**int** rk, treap\* &cur)

{

**int** myRk = size(cur->chi[0]) + 1;

**if** (myRk == rk) **return** cur;

**if** (myRk < rk)

**return** getRk(rk - myRk, cur->chi[1]);

**else**

**return** getRk(rk, cur->chi[0]);

}

**Treap,NoKey\_POJ\_3580\_SuperMemo.cpp**

#include <cstdio>

#include <cstring>

#include <ctime>

#include <cstdlib>

#include <cmath>

#include <algorithm>

**using** **namespace** std;

**struct** treap;

**int** size(treap \*&);

#define L(X) X->c[0^accu\_rev]

#define R(X) X->c[1^accu\_rev]

**struct** treap

{

**int** val, pri, sz;

**int** mi, add;

**char** rev;

treap \*c[2];

treap(**int** \_val = 0): val(\_val), sz(1), pri(**rand**()), mi(\_val), add(0), rev(0)

{

c[0] = c[1] = **NULL**;

}

**void** push(**char** &\_rev)

{

\_rev ^= rev;

}

**void** pull()

{

sz = size(c[0]) + size(c[1]) + 1;

mi = val;

**if** (c[0]) mi = **min**(mi, c[0]->mi + c[0]->add);

**if** (c[1]) mi = **min**(mi, c[1]->mi + c[1]->add);

}

} \*root = **NULL**;

**int** size(treap \*&t) { **return** t ? t->sz : 0; }

**void** split(treap \*t, **int** rk, treap \*&l, treap \*&r, **char** accu\_rev = 0)

{

**if** (!t) { l = r = **NULL**; **return** ; }

t->push(accu\_rev);

treap \*&tl = L(t), \*&**tr** = R(t);

**if** (size(tl) < rk)

{

l = t;

split(**tr**, rk - size(tl) - 1, **tr**, r, accu\_rev);

**if** (r) {

r->add += t->add;

r->rev ^= t->rev;

}

}

**else**

{

r = t;

split(tl, rk, l, tl, accu\_rev);

**if** (l) {

l->add += t->add;

l->rev ^= t->rev;

}

}

t->pull();

}

treap \* **merge**(treap \*&l, treap \*&r, **char** accu\_rev\_l = 0, **char** accu\_rev\_r = 0)

{

**if** (!l) **return** r;

**if** (!r) **return** l;

l->push(accu\_rev\_l);

r->push(accu\_rev\_r);

**if** (l->pri > r->pri)

{

r->add -= l->add;

r->rev ^= l->rev;

**char** &accu\_rev = accu\_rev\_l;

r->push(accu\_rev\_r);

R(l) = **merge**(R(l), r, accu\_rev\_l, accu\_rev\_r);

l->pull();

**return** l;

}

**else**

{

l->add -= r->add;

l->rev ^= r->rev;

**char** &accu\_rev = accu\_rev\_r;

l->push(accu\_rev\_l);

L(r) = **merge**(l, L(r), accu\_rev\_l, accu\_rev\_r);

r->pull();

**return** r;

}

}

**void** insert(**const** **int** &val, **const** **int** &pos) // pos = [1,]

{

treap \*l, \*r, \*n = **new** treap;

n->val = n->mi = val;

split(root, pos-1, l, r);

root = **merge**(l, n);

root = **merge**(root, r);

}

**void** del(**const** **int** &rk)

{

treap \*l, \*r, \*ll, \*lr;

split(root, rk, l, r);

split(l, rk-1, ll, lr);

root = **merge**(ll, r);

**delete** lr;

}

**int** query(**int** type, **int** i, **int** j, **int** n = 0) // i, j = [1,]

{

**int** ret;

treap \*l, \*r, \*rl, \*rr;

**if** (i > j) **swap**(i, j);

split(root, i-1, l, r);

split(r, j-i+1, rl, rr);

**switch** (type)

{

**case 0:** // min

ret = rl->mi + rl->add;

**break**;

**case 1:** // add

rl->add += n;

ret = 1; //meaningless

**break**;

**case 2:** // reverse

rl->rev ^= 1;

ret = 1;

**break**;

**case 3:** // revolve

n %= (j - i + 1);

treap \*ql, \*qr;

split(rl, j - i + 1 - n, ql, qr);

rl = **merge**(qr, ql);

ret = 1;

**break**;

}

root = **merge**(rl, rr);

root = **merge**(l, root);

**return** ret;

}

**void** in(treap \*t, **char** accu\_rev = 0)

{

**if** (!t) **return** ;

t->push(accu\_rev);

in(L(t), accu\_rev);

**printf**("%d(%d) ", t->val, size(t));

in(R(t), accu\_rev);

**return** ;

}

**int** **main**()

{

**int** i, j, n, m, ai, qi, qj, qn;

**char** s[100], cmd[20];

**srand**(**time**(0));

**scanf**("%d", &n);

**for** (i = 1; i <= n; i++)

{

**scanf**("%d", &ai);

insert(ai, i);

}

**scanf**("%d", &m);

**while** (m--)

{

**scanf**(" ");

**gets**(s);

**sscanf**(s, "%s", cmd);

**if** (**strcmp**(cmd, "ADD") == 0)

{

**sscanf**(s, "%s %d %d %d", cmd, &qi, &qj, &qn);

query(1, qi, qj, qn);

}

**else** **if** (**strcmp**(cmd, "REVERSE") == 0)

{

**sscanf**(s, "%s %d %d", cmd, &qi, &qj);

query(2, qi, qj);

}

**else** **if** (**strcmp**(cmd, "REVOLVE") == 0)

{

**sscanf**(s, "%s %d %d %d", cmd, &qi, &qj, &qn);

query(3, qi, qj, qn);

}

**else** **if** (**strcmp**(cmd, "INSERT") == 0)

{

**sscanf**(s, "%s %d %d", cmd, &qi, &qn);

insert(qn, qi+1);

}

**else** **if** (**strcmp**(cmd, "DELETE") == 0)

{

**sscanf**(s, "%s %d", cmd, &qn);

del(qn);

}

**else** **if** (**strcmp**(cmd, "MIN") == 0)

{

**sscanf**(s, "%s %d %d", cmd, &qi, &qj);

**printf**("%d\n", query(0, qi, qj));

}

}

**return** 0;

}

**Persistent Treap\_UVa\_12538\_Version Controlled IDE.cpp**

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cstdlib>

#include <ctime>

#include <cmath>

#define ROOT root[root\_iter]

**using** **namespace** std;

**const** **int** MAX\_VER = 50002;

**const** **int** MAX\_LEN = 105;

**struct** treap;

**int** size(treap \*);

**struct** treap

{

**char** val;

**int** pri, sz;

treap \*l, \*r;

treap(**int** \_val = 0): val(\_val), pri(**rand**()), sz(1), l(**NULL**), r(**NULL**){}

**void** pull()

{

sz = size(l) + size(r) + 1;

}

};

treap \*root[MAX\_VER];

**int** root\_iter;

**int** cnt\_c;

**char** s[MAX\_LEN];

**int** size(treap \*t) { **return** t ? t->sz : 0; }

treap \* **copy**(treap \*t)

{

treap \*n = **new** treap;

\*n = \*t;

**return** n;

}

**void** in(treap \*t)

{

**if** (!t) **return** ;

in(t->l);

**putchar**(t->val);

**if** (t->val == 'c') cnt\_c++;

in(t->r);

}

**void** split(treap \*t, **const** **int** &rk, treap \* &l, treap \* &r)

{

**if** (!t) { l = r = **NULL**; **return** ; }

treap \*t2 = **copy**(t);

**if** (size(t->l) < rk)

{

l = t2;

split(t2->r, rk - size(t->l) - 1, t2->r, r);

}

**else**

{

r = t2;

split(t2->l, rk, l, t2->l);

}

t2->pull();

}

treap \* **merge**(treap \*l, treap \*r)

{

**if** (!l) **return** r;

**if** (!r) **return** l;

**if** (l->pri > r->pri)

{

treap \*n = **copy**(l);

n->r = **merge**(n->r, r);

n->pull();

**return** n;

}

**else**

{

treap \*n = **copy**(r);

n->l = **merge**(l, n->l);

n->pull();

**return** n;

}

}

**void** \_split(treap \*t, **const** **int** &rk, treap \* &l, treap \* &r)

{

**if** (!t) { l = r = **NULL**; **return** ; }

**if** (size(t->l) < rk)

{

l = t;

split(t->r, rk - size(t->l) - 1, t->r, r);

}

**else**

{

r = t;

split(t->l, rk, l, t->l);

}

t->pull();

}

treap \* \_merge(treap \*l, treap \*r)

{

**if** (!l) **return** r;

**if** (!r) **return** l;

**if** (l->pri > r->pri)

{

l->r = **merge**(l->r, r);

l->pull();

**return** l;

}

**else**

{

r->l = **merge**(l, r->l);

r->pull();

**return** r;

}

}

**void** insert(**char** s[], **int** len, **int** pos) // insert after pos

{

**int** i;

treap \*l, \*r, \*n, \*ins = **NULL**;

split(ROOT, pos, l, r);

++root\_iter;

**for** (i = 0; i < len; i++)

{

n = **new** treap;

n->val = s[i];

ins = \_merge(ins, n);

}

ROOT = **merge**(l, ins);

ROOT = **merge**(ROOT, r);

}

**void** **remove**(**const** **int** &st, **const** **int** &len)

{

treap \*l, \*r, \*ll, \*lr;

split(ROOT, st+len-1, l, r);

split(l, st-1, ll, lr);

++root\_iter;

ROOT = **merge**(ll, r);

}

**void** print(**const** **int** &ver, **const** **int** &st, **const** **int** &len)

{

treap \*l, \*r, \*ll, \*lr;

treap \*&rt = root[ver];

\_split(rt, st+len-1, l, r);

\_split(l, st-1, ll, lr);

in(lr); **puts**("");

rt = \_merge(ll, lr);

rt = \_merge(rt, r);

}

**void** init()

{

**srand**(**time**(0));

**memset**(root, 0, **sizeof**(root));

root\_iter = 0; // can be -1

cnt\_c = 0;

}

**int** **main**()

{

**int** i, qs, cmd, pos, len, ver, l;

init();

**scanf**("%d", &qs);

**while** (qs--)

{

**scanf**("%d", &cmd);

**switch** (cmd)

{

**case 1:**

**scanf**("%d %s", &pos, s);

l = **strlen**(s);

pos -= cnt\_c;

insert(s, l, pos + i);

**break**;

**case 2:**

**scanf**("%d %d", &pos, &len);

pos -= cnt\_c;

len -= cnt\_c;

**remove**(pos, len);

**break**;

**case 3:**

**scanf**("%d %d %d", &ver, &pos, &len);

ver -= cnt\_c;

pos -= cnt\_c;

len -= cnt\_c;

print(ver, pos, len);

**break**;

}

}

**return** 0;

}

**Segment Tree\_Sum.cpp**

#include <cstdio>

#include <algorithm>

#define MAX 100002

**using** **namespace** std;

**struct** node

{

**int** l, r; **long** **long** **int** val, add;

**int** left, right;

} seg[MAX<<1];

**int** a[MAX], t=0;

**long** **long** **int** buildSegmentTree(**int** idx, **int** l, **int** r)

{

seg[idx].l = l;

seg[idx].r = r;

seg[idx].add=0;

**if** (l == r)

{

seg[idx].val = a[l];

**return** a[l];

}

**int** m = (l+r)>>1;

seg[idx].left = ++t;

seg[idx].right = ++t;

seg[idx].val = buildSegmentTree(seg[idx].left, l, m) + buildSegmentTree(seg[idx].right, m+1, r);

**return** seg[idx].val;

}

**long** **long** **int** query(**int** idx, **int** ql, **int** qr)

{

**int** &l = seg[idx].l, &r = seg[idx].r, m = (l+r)>>1;

**if** (ql == l && qr == r)

**return** seg[idx].val + seg[idx].add \* (r-l+1);

**long** **long** **int** sum=0;

**if** (ql <= m)

sum += query(seg[idx].left, ql, **min**(qr, m) );

**if** (qr > m)

sum += query(seg[idx].right, **max**(ql, m+1), qr);

**return** sum + seg[idx].add \* (qr-ql+1);

}

**long** **long** **int** update(**int** idx, **int** ul, **int** ur, **int** n)

{

**int** &l = seg[idx].l, &r = seg[idx].r, m = (l+r)>>1;

**if** (ul == l && ur == r)

{

seg[idx].add += n;

**return** seg[idx].val + seg[idx].add \* (r-l+1);

}

**long** **long** **int** sum=0;

**if** (ul <= m)

sum += update(seg[idx].left, ul, **min**(ur, m), n);

**else**

sum += seg[seg[idx].left].val + seg[seg[idx].left].add \* (seg[seg[idx].left].r-seg[seg[idx].left].l+1);

**if** (ur > m)

sum += update(seg[idx].right, **max**(ul, m+1), ur, n);

**else**

sum += seg[seg[idx].right].val + seg[seg[idx].right].add \* (seg[seg[idx].right].r-seg[seg[idx].right].l+1);

seg[idx].val = sum;

**return** sum + seg[idx].add \* (r-l+1);

}

**int** **main**()

{

**int** i, l, r, add, n;

**for**( i=0; i<=5; i++ )

a[i]=i;

buildSegmentTree(0, 0, 5);

/\*

for( i=0; i<=t; i++ )

{

puts("-");

printf("Index: %d\n", i);

printf("L: %d, R: %d\n", seg[i].l, seg[i].r);

printf("Val: %lld\n", seg[i].val);

printf("Add: %lld\n", seg[i].add);

}\*/

**scanf**("%d", &n); // n updates

**while**( n-- )

{

**scanf**("%d%d%d", &l, &r, &add);

update(0, l, r, add);

}

/\*

for( i=0; i<=t; i++ )

{

puts("-");

printf("Index: %d\n", i);

printf("L: %d, R: %d\n", seg[i].l, seg[i].r);

printf("Val: %lld\n", seg[i].val);

printf("Add: %lld\n", seg[i].add);

}\*/

**while**( **scanf**("%d%d", &l, &r)==2 )

**printf**("%lld\n", query(0, l, r));

**return** 0;

}

**Monotone Chain\_UVa\_681\_Convex Hull Finding.cpp**

**const** **int** V = 1502;

**struct** vert

{

**int** x, y;

**bool** **operator** < (**const** vert &rhs)**const**

{

**if** (x != rhs.x)

**return** x < rhs.x;

**else**

**return** y < rhs.y;

}

vert **operator** - (**const** vert &rhs)**const**

{

vert tmp;

tmp.x = x - rhs.x;

tmp.y = y - rhs.y;

**return** tmp;

}

**void** **operator** = (**const** vert &rhs)

{

x = rhs.x;

y = rhs.y;

}

} a[V], r[V];

**char** used[V][V];

**int** cross(**const** vert &a, **const** vert &b)

{

**return** a.x \* b.y - a.y \* b.x;

}

**void** monotone\_chain(**int** v, vert a[], **int** &vr, vert r[])

{

**int** i, j;

vert \*f = **new** vert[v]; **int** it\_f = 0;

vert \*b = **new** vert[v]; **int** it\_b = 0;

**sort**(a, a+v);

**for** (i = 0; i < v; i++)

**if** (it\_f < 2)

f[it\_f++] = a[i];

**else**

{

**while** (it\_f >= 2 && cross(a[i]-f[it\_f-1], f[it\_f-1]-f[it\_f-2]) <= 0) it\_f--;

f[it\_f++] = a[i];

}

**for** (i = v-1; i >= 0; i--)

**if** (it\_b < 2)

b[it\_b++] = a[i];

**else**

{

**while** (it\_b >= 2 && cross(a[i]-b[it\_b-1], b[it\_b-1]-b[it\_b-2]) <= 0) it\_b--;

b[it\_b++] = a[i];

}

**for** (i = 0, j = 0; i < it\_f-1; i++)

r[j++] = f[i];

**for** (i = 0; i < it\_b-1; i++)

r[j++] = b[i];

vr = j;

**delete** [] f;

**delete** [] b;

}

**int** **main**()

{

**int** i, t, tmp\_v, v, vr, min\_idx;

**memset**(used, 0, **sizeof**(used));

**scanf**("%d", &t);

**printf**("%d\n", t);

**while** (t--)

{

v = 0;

**scanf**("%d", &tmp\_v);

**for** (i = 0; i < tmp\_v; i++)

{

**scanf**("%d%d", &a[v].x, &a[v].y);

**if** (!used[a[v].x][a[v].y])

{

used[a[v].x][a[v].y] = 1;

v++;

}

}

monotone\_chain(v, a, vr, r);

**printf**("%d\n", vr + 1);

**for** (i = min\_idx; i < min\_idx + vr; i++)

**printf**("%d %d\n", r[i % vr].x , r[i % vr].y);

**printf**("%d %d\n", r[min\_idx].x, r[min\_idx].y);

}

**return** 0;

}