**SheepDev Contest #1 – Editorial**
==========

**A – Tree**

100 คะแนน:
สังเกตกระบวนการ depth first search ดังกล่าว

```
DFS(V):
    push V to stack
    for all children W of V:
        DFS(W)
    pop stack
```

จะเห็นว่า ขณะเวลาที่เข้า DFS(V) และยังไม่ push V จุดยอดบน stack ทั้งหมดจะเป็นบรรพบุรุษของ V โดยเรียงตามความลึก (ไม่รวม V)

ดังนั้นการแก้ปัญหาข้อนี้ทำได้โดยเก็บกองซ้อน S สำหรับเก็บบรรพบุรุษ จากนั้นเปิดกองซ้อน At[color] แยกสำหรับแต่ละสี

เมื่อเข้า DFS(V) ให้ใส่ V เข้าบนกองซ้อน S และใส่ตำแหน่งของ V บน S ลงบนกองซ้อน At[C[V]] และก่อนออกจาก V ให้นำตัวบนสุดของกองซ้อนทั้งสองออก

สำหรับการตอบคำถาม เมื่อเข้า DFS(V) ก่อนที่จะใส่ V ลงกองซ้อนใดๆ ให้ดูกองซ้อน At[C[V]] หากว่างอยู่ คำตอบคือ 0 แต่หากไม่ว่าง ให้ดูตัวบนสุดใน At[C[v]] แล้วตอบ S[ 1 + *Top of At[C[v]]* ]

Time complexity – O(N)
Memory complexity – O(N + max C[])

**SheepDev Contest #1 – Editorial**
==========

**B – A-K Query**

**ปัญหาย่อยที่ 1 (20 คะแนน):**

เนื่องจาก N, Q <= 10, 000 สามารถทำ O(NQ) ได้ โดยสำหรับคำสั่งรูปแบบแรก ให้ทำตรงๆบนอาเรย์ด้วยลูปได้เลย และสำหรับคำสั่งรูปแบบสอง สามารถสร้างลำเนาของอาเรย์ย่อยที่ต้องการ และใช้อัลกอริทึม quick select (std::nth_element ใน standard library C++) บนลำเนานั้น เพื่อตอบตัวที่ k
อย่างไรก็ตามโค้ด O(NQ log N) ที่ใช้การเรียงลำเนาตรงๆสามารถผ่านได้เช่นกัน

**ปัญหาย่อยที่ 2 (30 คะแนน):**

เงื่อนไขของปัญหาย่อยนี้คือ ทุกๆคำสั่งประเภทที่ 1 จะมาก่อนคำสั่งประเภทที่ 2 ดังนั้นสามารถแบ่งปัญหาเป็นสองส่วน และทำแต่ละส่วนในลักษณะ offline ได้

*ส่วนคำสั่งประเภทที่ 1*

ต้องการทราบค่าของอาเรย์สุดท้ายที่ได้หลังการทำคำสั่งประเภทที่ 1 ทุกอัน ซึ่งสามารถทำได้หลายแบบ

- lazy segment tree
- sweep line (ให้แต่ละคำสั่งเป็น <timestamp, value, left, right> และหาค่าของช่องปัจจุบันโดยดูคำสั่งที่ timestamp มากสุดในโครงสร้างข้อมูล เช่น heap/binary search tree)
- -* Chtholly tree ซึ่งจะใช้ในปัญหาย่อยถัดไป

โดยเก็บลำดับของช่วงติดกันที่มีค่าเหมือนกันในอาเรย์ในรูปแบบ <l, r, val> และสามารถเก็บใน binary search tree เช่น std::set ได้ ในการทำแต่ละคำสั่ง (ql, qr, k) สามารถทำโดยพิจารณาทุกช่วงที่อยู่ในช่วงของคำสั่ง แล้วไล่ลบช่วงเหล่านั้นออก จากนั้นใส่ <ql, qr, k> ลงใน set

(ก่อนจะลบ: หากบางช่วง <l, r, val> ตัดกับ [ql, qr] ในลักษณะนี้

```
---------l------ql-------r----------------qr
```

สามารถลบ <l, r, val> ออก และใส่ <l, ql – 1, val>, <ql, r, val> ลงไปแทน)

ถึงการไล่ลบช่วง จะทำตรงๆ แต่โดยเฉลี่ยแล้วจะใช้เวลา O(q log q) เนื่องจากแต่ละคำสั่งจะเพิ่มจำนวนช่วงมาไม่เกิน 2 ดังนั้นจำนวนช่วงที่ถูกลบจะไม่เกิน 2q

*ส่วนคำสั่งประเภทที่ 2*

หลังจากได้อาเรย์ที่เป็นผลลัพธ์จากการทำคำสั่งประเภทที่ 1 หมดแล้ว การตอบตัวที่น้อยสุดเป็นลำดับที่ k เป็นปัญหาคลาสสิกซึ่งทำได้หลายวิธี (merge sort tree/persistent segment tree/wavelet tree/offline + fenwick tree)

(https://www.spoj.com/problems/MKTHNUM/)

**ปัญหาย่อยที่ 3 (50 คะแนน):**
แก้โดยการใช้ Chtholly tree และ 2D segment tree

โดยพิจารณาจุดบนระบบพิกัดสองมิติ

เมื่อช่วง <l, r, val> บน Chtholly tree
ถูกพล็อตลงบนจุด <l, val> และจุดนั้นมีน้ำหนักเป็นความยาวของช่วง = r - l + 1
จากพล็อตจุดในลักษณะดังกล่าวแล้ว การตอบคำสั่งประเภทที่ 2 <ql, qr> ทำได้โดยหาค่า x
น้อยที่สุด ที่ทำให้ สี่เหลี่ยมที่มีจุดล่างซ้ายเป็น <ql,0> และมีจุดบนขวาเป็น <qr,x> มีผลรวมน้ำหนักมากกว่า k
(หากมีช่วง <l, r, val> ที่ตัดกับ <ql, qr> ต้องแก้ไขโดนแบ่งเป็นสองช่วงดังในปัญหาย่อยก่อนหน้าก่อน)

การหาค่า x ดังกล่าวทำได้โดยใช้ 2D segment tree และ binary search บนมิติที่สอง

การทำคำสั่งประเภทที่ 1 ทำบน Chtholly tree เหมือนปัญหาย่อยก่อนหน้าได้เลย แต่ทุกครั้งที่มีการลบหรือเพิ่มช่วงลงใน Chtholly tree จะต้องลบหรือเพิ่มจุดที่ถูกพล็อตใน segment tree ด้วย

Time complexity: O(q lg^2(max k))
Memory Complexity: O(q lg^2(max k))

*สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ Chtholly tree ดูได้ที่ https://codeforces.com/blog/entry/56135

```
#include <stdio.h>
#include <set>
#include <stdlib.h>

#define N 100000
#define Q 100000
#define K 100000
#define N_ (1 << 17)

int n_, n, q, tt[N_ << 1], ii;
struct {int a, l, r;}t[N*18*18];

void add2(int p,int k,int&i,int l,int r){
	if(!i)i=++ii;
	t[i].a+=k;
	if(l==r)return;
	if(p<=(l+r)/2)add2(p,k,t[i].l,l,(l+r)/2);
	else add2(p,k,t[i].r,(l+r)/2+1,r);
}
void add(int p,int p2,int k){
```

```
        for (p+=n_;p;p>>=1)add2(p2,k,tt[p],0,K);
}
int v_[99],o;

void gennodes(int l,int r){
        o=0;
        for(l+=n_-1,r+=n_+1;l^r^1;l>>=1,r>>=1){
                if(l&1^1)v_[o++]=tt[l^1];
                if(r&1)v_[o++]=tt[r^1];
        }
}

int walkwalkwalk(int l,int r,int k){
        if(l==r)return l;
        int suml=0;
        for(int j=0;j<o;++j)suml+=t[t[v_[j]].l].a;
        if(suml>=k){
                for(int j=0;j<o;++j)v_[j]=t[v_[j]].l;
                return walkwalkwalk(l,(l+r)/2,k);
        }
        else {
                for(int j=0;j<o;++j)v_[j]=t[v_[j]].r;
                return walkwalkwalk((l+r)/2+1,r,k-suml);
        }
}

struct node {
        mutable int l,r,v;
        bool operator<(const node&o)const{return l<o.l;}
};
std::set<node>chtholly;
std::set<node>::iterator split(int i){
        auto it=chtholly.lower_bound(node{i, 0, 0});
        if(it!=chtholly.end()&&it->l==i)return it;
        auto iy=std::prev(it);
        node x=*iy;
        add(x.l,x.v,-(x.r-x.l+1));
        add(x.l,x.v,i-x.l);
        add(i,x.v,x.r-i+1);
        chtholly.erase(iy);
        chtholly.insert(node{x.l,i-1,x.v});
        return chtholly.insert(node{i,x.r,x.v}).first;
}

int main() {
        scanf("%d%d", &n, &q);
        for(n_=1;n_<=n;n_<<=1);
        chtholly={{0,n-1,0},{n,n,0}};
        add(0,0,n);
        for (int op, s, t, k, i = 0; i < q; ++i) {
                scanf("%d%d%d%d", &op, &s, &t, &k);
                if (op == 1) {
                        auto itr=split(t+1),itl=split(s);
                        for(auto it=itl;it!=itr;++it)
                                add(it->l,it->v,-(it->r-it->l+1));
                        chtholly.erase(itl,itr);
                        chtholly.insert(node{s,t,k});
                        add(s,k,t-s+1);
                } else {
                        split(t+1); split(s); gennodes(s,t);
                        printf("%d\n",walkwalkwalk(0,K,k));
                }
        }

        return 0;
```

}

**SheepDev Contest #1 – Editorial**
**==========**

## C – Suitable Weights

ปัญหาย่อยที่ 1 (20 คะแนน):

สามารถเขียนจำลองสถานการณ์ตรงๆได้เลย

Time complexity: O((N+M)Q) หรือ O((N+M)Q logQ) หากใช้ DSU

ปัญหาย่อยที่ 2 (30 คะแนน):

นำคำถามมาเรียงตาม y จากน้อยไปมาก และนำเส้นเชื่อมมาเรียงตามน้ำหนักจากน้อยไปมาก จากนั้นหาคำตอบโดยไล่คำถามตามลำดับที่เรียง แล้วใส่เส้นเชื่อมที่มีน้ำหนักน้อยกว่าเท่ากับ y ของคำถามปัจจุบัน

สามารถทำได้ด้วย Disjoint Set Union

Time complexity: O(M log M + Q log Q + Q log N)

ปัญหาย่อยที่ 3 (50 คะแนน):

นำเส้นเชื่อมในกราฟมาเรียงตามน้ำหนักจากน้อยไปมาก สังเกตว่าแต่ละคำถามจะมีเส้นเชื่อมที่ไม่ถูกลบ ติดกันเป็นช่วง [l, r]

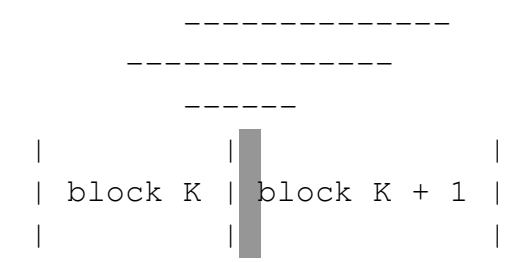
เลือก B เป็นค่าที่เหมาะสม

หากความยาวของช่วงที่ถาม (r – l + 1) ไม่มากกว่า B ให้ใช้ DSU with rollback ในการตอบคำถามได้เลย ใช้เวลา O(B log N) ต่อคำถาม

จากนั้นพิจารณาคำถามที่ช่วงที่ถามยาวกว่า B

แบ่งเส้นเชื่อมที่เรียงแล้วเป็นบล็อค บล็อคละ B เส้น
ให้ belong[i] แทนหมายเลขบล็อคที่เส้นเชื่อมที่ i อยู่
สำหรับคำถาม [l, r] ให้นำคำถามที่มีค่า belong[l] เหมือนกันจับกลุ่มกัน

จากนั้นสำหรับทุกๆ N / B บล็อค ที่บล็อค k จะหาคำตอบสำหรับคำถามที่ belong[l] = k โดยนำคำถามเหล่านี้มาเรียงตาม r จากน้อยไปมาก




เนื่องจาก r – l + 1 > B เรารู้ว่า r จะอยู่บล็อคทางขวาของ K แน่ๆ

หลังเรียงคำถามแล้ว สามารถเก็บ ptr_r โดยเริ่มแรกเท่ากับหมายเลขของคำถามแรกที่อยู่บล็อค K + 1 และไล่คำถาม [l, r], เมื่อ ptr_r <= r ให้ใส่เส้นเชื่อมที่ ptr_r ลงใน DSU เมื่อใส่จน ptr_r = r + 1 แล้วสังเกตว่า DSU ของเรามีเส้นเชื่อมที่ต้องการใน [l, r] แต่ยังขาดเส้นเชื่อมในบล็อค K ไป ดังนั้นสามารถไล่เส้นเชื่อมในบล็อค K ที่เราต้องการ และใส่ลงใน DSU จากนั้นตอบคำถาม [l, r] บันทึกไว้ แล้วลบเส้นเชื่อมในบล็อค K ที่เพิ่มมาออก (ด้วย DSU rollback)

เมื่อประมวลผลคำถามในบล็อค K ครบแล้ว สามารถ reset DSU ได้เลย เพื่อประมวลผลบล็อคถัดไป

Time complexity : O(N^2log N / B + NB logN)
เมื่อให้ B มีค่าประมาณ N ^ 0.5 จะได้ Time complexity = O(N^1.5 logN)

(เมื่อพิจารณาว่า O(Q) ~ O(M) ~ O(N))

*การทำดังกล่าว เป็นรูปแบบหนึ่งของ Mo’s Algorithm

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <vector>
#include <algorithm>

#define N 70000
#define Q 70000
#define M 70000

#define B 200
#define MB (M / B + 5)

int n, m, q, a[M], b[M], c[M], a_[M], b_[M], c_[M], ii[M], ans[M]
                      , bel[M], l[Q], r[Q];

std::vector<int> tag[MB];

struct dsurollback {
	int cmp, ds[N];
	int stamp, roll1[M + 1], roll2[M + 1], roll3[M + 1];

	void reset() {
		stamp = 0;
		cmp = n;
		memset(ds, -1, sizeof ds);
		memset(roll1, -1, sizeof roll1);
	}

	int find(int i) {
		return ds[i] < 0 ? i : find(ds[i]);
	}
	void unite(int i, int j) {
		++stamp;
		i = find(i), j = find(j);
		if (i == j) return;
		if (-ds[i] < -ds[j])
			i ^= j, j ^= i, i ^= j;
		roll1[stamp] = i;
		roll2[stamp] = j;
```

```
                roll3[stamp] = ds[j];
                ds[i] += ds[j];
                ds[j]  = i;
                --cmp;
        }
        void rollback() {
                if (! ~roll1[stamp]) {
                        --stamp;
                        return;
                }
                int i, j, dsj;
                i = roll1[stamp], j = roll2[stamp], dsj = roll3[stamp];
                ds[i] -= dsj;
                ds[j]  = dsj;
                roll1[stamp] = -1;
                --stamp;
                ++cmp;
        }
        int count() {
                return cmp;
        }
} dsu;

int main() {
        scanf("%d%d%d", &n, &m, &q);
        for (int i = 0; i < m; ++i) {
                scanf("%d%d%d", a_ + i, b_ + i, c_ + i);
                ii[i] = i;
        }
        std::sort(ii, ii + m, [](int i, int j) { return c_[i] < c_[j]; });
        for (int i, i_ = 0; i_ < m; ++i_) {
                i = ii[i_];
                a[i_] = a_[i], b[i_] = b_[i], c[i_] = c_[i];
        }

        for (int i = 0; i < m; ++i)
                bel[i] = i / B;

        dsu.reset();
        for (int x, y, i = 0; i < q; ++i) {
                scanf("%d%d", &x, &y);

                int ll, rr;
                ll = -1, rr = m;

                for (int j = 1 << 17; j; j >>= 1)
                        if (ll + j < m && c[ll + j] < x)
                                ll += j;
                for (int j = 1 << 17; j; j >>= 1)
                        if (rr - j >= 0 && c[rr - j] > y)
                                rr -= j;
                ++ll, --rr;

                if (ll > rr) {
                        ans[i] = n;
                        continue;
                }

                if (rr - ll + 1 <= B) {
                        for (int j = ll; j <= rr; ++j)
                                dsu.unite(a[j], b[j]);
                        ans[i] = dsu.count();
                        for (int j = ll; j <= rr; ++j)
                                dsu.rollback();
                } else {
```

```
                l[i] = ll, r[i] = rr;
                tag[bel[ll]].push_back(i);
        }
    }

    for (int i = 0; i * B < m; ++i) {
        dsu.reset();
        std::sort(tag[i].begin(), tag[i].end(), [](int i, int j) {
                return r[i] < r[j];
        });
        int ptr = i * B + B;
        for (auto j : tag[i]) {
                while (ptr <= r[j])
                        dsu.unite(a[ptr], b[ptr]), ++ptr;

                for (int k = i * B + B - 1; k >= l[j]; --k)
                        dsu.unite(a[k], b[k]);

                ans[j] = dsu.count();
                for (int k = i * B + B - 1; k >= l[j]; --k)
                        dsu.rollback();
        }
    }

    for (int i = 0; i < q; ++i)
        printf("%d\n", ans[i]);
    return 0;
}
```