题目描述

有一台超级计算机，共有 n 块 CPU，这 n 块 CPU 按照 $1 \sim n$ 从小到大编号。

该计算机需要完成 m 个独立的任务，其会**按顺序执行**每一个任务。每一个任务会给出三个参数：t, w, s。意为在 t 时刻将会下达该任务的计算指令，该次任务的权重为 w，此时**空闲的编号最小的**那块 CPU 将会执行本次计算任务(如果此刻没有空闲的 CPU 则不会执行该项任务的计算)。该块 CPU 进入忙碌状态，并在 $t+s$ 时刻计算结束回归空闲状态。

请你计算出在全部执行完这 m 个任务后，每一块 CPU 上共运行了多少权重的任务。

输入

第一个行两个正整数 n 和 m，表示 CPU 的数量和任务的个数 $(1 \leq n, m \leq 2 \times 10^5)$。

接下来 m 行，每行 3 个整数 t, w, s(保证当 $i < j$ 时 $t_i \leq t_j$，其中 $0 \leq t, w, s \leq 10^9$)。

输出

输出共 n 行，第 i 行表示在第 i 块 CPU 上运行任务的权重和。

审题：数据范围 2e5，o（nlogn）可以过
所以我们可以直接用 nlogn 的优先队列（堆）来进行模拟

思路：用两个优先队列（堆）分别维护 CPU 的最小序号和最小时间，每当出现一个任务时，从维护时间的优先队列(堆)中取出所有工作已经结束的 cpu 存入维护序号的优先队列(堆)，令最小序号的 cpu 进行处理，并将该 cpu 放入维护最小时间的优先队列（堆），以此类推进行模拟。

优先队列：

```
q.size();//返回q里元素个数
q.empty();//返回q是否为空，空则返回1，否则返回0
q.push(k);//在q的末尾插入k
q.pop();//删掉q的第一个元素
q.top();//返回q的第一个元素
```

```
class CPU
{
    public:

        CPU(int num,int now)
        {
            this->num=num;
            this->now=now;
            this->weight=0;
        }

            int num;
            long long now;
            long long weight;

        //  friend bool operator < (const CPU &a,const CPU &b);
};
/*typedef struct cpu
{
    int num;
    long long now;
    long long weight;
}CPU;*/

bool operator <(const CPU &a,const CPU &b){
    return a.num<b.num;
}
bool operator >(const CPU &a,const CPU &b){
    return a.now>b.now;
}
```

```cpp
int main()
{
    priority_queue<CPU,vector<CPU>,less<CPU> > q1;
    priority_queue<CPU,vector<CPU>,greater<CPU> > q2;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
    {
        q1.push(CPU(i,0));
    }
    for(int i=1;i<=m;i++)
    {
        scanf("%lld%lld%lld",&t[i],&w[i],&s[i]);
    }
    for(int i=1;i<=m;i++)
    {
        long long nowtime=t[i];
        while(!q2.empty())
        {
            CPU temp2=q2.top();
            if(temp2.now<=nowtime)
            {
                q2.pop();
                q1.push(temp2);
            }
            else
            {
                break;
            }
        }
        if(!q1.empty())
        {
            CPU temp1=q1.top();
            temp1.now=t[i]+s[i];
            temp1.weight+=w[i];
            q1.pop();
            q2.push(temp1);
        }
    }
    while(!q2.empty())
    {
        CPU temp3=q2.top();
        q1.push(temp3);
        q2.pop();
    }
    for(int i=1;i<=n;i++)
    {
        printf("%lld\n",q1.top().weight);
        q1.pop();
    }
}
```

堆:

```
void swap(struct cpu* a, struct cpu* b) {
    struct cpu c = *b;
    *b = *a;
    *a = c;
}

bool cmp(struct cpu a, struct cpu b) {
    return a.id < b.id;
}
```

```
void down(int i) {
    int smallest = i;
    int left = i << 1;
    int right = i << 1 | 1;
    if ((!cmp(heap[smallest], heap[left])) && (left <= heap_size)) {
        smallest = left;
    }
    if (((!cmp(heap[smallest], heap[right]))) && (right <= heap_size)) {
        smallest = right;
    }
    if (smallest != i) {
        swap(&heap[smallest], &heap[i]);
        down(smallest);
    }
}
```

```
void push(struct cpu tmp) {
    heap[++heap_size] = tmp;
    int i = heap_size;
    while (i > 1 && cmp(heap[i], heap[i >> 1])) {
        swap(&heap[i], &heap[(i >> 1)]);
        i = i >> 1;
    }
}

void pop() {
    heap[1] = heap[heap_size];
    heap_size--;
    down(1);
}
```