考虑mod B = 0 · · ·B − 1 的最小能被表示的值是多少——大于他们的可以通过+kB 得到。假设这样的最小值分别为m0,m1, · · · ,mB−1，那么小于等于K 的能被表示的非负整数的数量是：

$$Σ^{B-1}_{i=0}⌊\frac{max(K - mi + B, 0)}{B}⌋$$

考虑如何求mi，由反证法可知0, A, · · · , (B − 1)A 在mod B 意义下必然互不相同，而这些数就枚举了mi，因此，当K < AB 时，上式化为：

$$ Σ_{i=0}^{⌊\frac{K}{A}⌋}⌊\frac{K - iA}{B}⌋$$

这是一个非常经典的直线下整点问题，使用类欧几里得算法可以在O(log max(A,B)) 的时间内求解。因此我们得到了计算小于等于K 的能被表示的非负整数的数量的工具，再二分一下就能解决原问题了。时间复杂度O(T log max(A,B) log min(K,AB))，可以通过A,B ≤ 1018, T ≤ 1000 的数据，也即本题原本的数据范围。

考虑到直线下整点是一个掏板子工作，因此放过了暴力。暴力可以这么写：假设A < B，我们尝试计算[kB, (k + 1)B) 这一段里有多少可以被表达出来的，那其实就是{(iA mod B) + kB|iA < (k + 1)B}的集合大小，枚举iA 计算确定答案在哪一段后，再在段内确定就好了——而段内有多少能被表达的已经被枚举了，打好标记for 一下就行了。时间复杂度O(T max(A,B))。

```
#include<stdio.h>

long long solve(long long n, long long a, long long b, long long m)
{
    if(b==0)
    {
        return n*(a/m);
    }
    if(a>=m)
    {
        return n*(a/m)+solve(n,a%m,b,m);
    }
    if(b>=m)
    {
        return (n-1)*n/2*(b/m)+solve(n,a,b%m,m);
    }
    return solve((a+b*n)/m,(a+b*n)%m,m,b);
}

long long A,B,K;

void work()
{
    scanf("%lld%lld%lld",&A,&B,&K);
    long long l=1,r;
    r=(double)A*B>3e18?2e18+10:((long long)(2e18)+10<A*B-1?(long
long)(2e18)+10:A*B-1);// double(A-1)*(B-1)/2>=K
    while(l<r)
    {
        long long mid=(l+r)/2;
        long long n=mid/A+1,m=B,a=mid-(n-1)*A+B,b=A;
        long long tot=solve(n,a,b,m)-1;
        long long cnt=mid-tot;
```

```
        if(cnt>=K)
        {
            r=mid;
        }
        else
        {
            l=mid+1;
        }
    }
    printf("%lld\n",l);
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        work();
    }
}
```