```
// Given the value of a+b and ab you will have to find the value of
a^n+b^n.
// a and b not necessarily have to be real numbers.

ULLI row,col;
ULLI mod,val;
struct st
{
    ULLI x[17][17];
};

typedef st matrix;

void print_matrix( int r,int c,matrix &a )
{
    for( int i=0; i<r; i++ )
    {
        for(int j=0; j<c; j++ )
            cout<<a.x[i][j]<<" ";
        cout<<endl;
    }
}

matrix set_zero(ULLI r,ULLI c)
{
    matrix rst;
    for( ULLI i=0; i<r; i++ )
    {
        for( ULLI j=0; j<c; j++ )
            rst.x[i][j]=0;
    }
    return rst;
}

matrix set_identity(ULLI r,ULLI c)
{
    matrix identity=set_zero(r,c);
    for( ULLI i=0; i<r; i++ )
    {
        identity.x[i][i]=1;
    }
    return identity;
}
matrix maltiplication( matrix &a,matrix &b)
{
    matrix rst=set_zero(row,col);
    for( ULLI i=0; i<row; i++ )
    {
        for( ULLI j=0; j<col; j++ )
        {
            for( ULLI k=0; k<col; k++ )
            {
                rst.x[i][j]=(rst.x[i][j]+(a.x[i][k]*b.x[k][j]));
                // when mod is needed just use this line
                // rst.x[i][j]=(rst.x[i][j]+(a.x[i][k]*b.x[k][j]))%mod;
            }
        }
    }
    return rst;
}
```

```
matrix big_power( ULLI val ,matrix &a )
{
    matrix rst,tmp=a;
    rst=set_identity(row,col);
    while( val )
    {
        if( val&1 )
        {
            rst=maltiplication(rst,tmp);
        }
        tmp=maltiplication(tmp,tmp);
        val>>=1;
    }
    return rst;
}

matrix set_base(ULLI r,ULLI c ,ULLI ar[])
{
    matrix a;
    memset(a.x,0,sizeof a.x);
    for( int i=0; i<c; i++ ) a.x[0][i]=ar[i]%md;
    for( int i=1,j=0; i<r; i++,j++ )
    {
        a.x[i][j]=1;
    }
    return a;
}

int main()
{
    ULLI a,b,n,m,d,ar[100],p,q;
    int T;
    scanf("%d",&T);
    for( int t=1; t<=T; t++ )
    {
        scanf("%llu%llu%llu",&p,&q,&n);
        matrix fibo,base;

        // change the base matrix depending upon the problem description

        base.x[0][0]=p;            // when modulo is need just p%mod
        base.x[0][1]=-q;
        base.x[1][0]=1;
        base.x[1][1]=0;

        // end of the base matrix

        row=2,col=2;               // base matrix row and column
dimensions
        matrix rst;
        rst=big_power(n-1,base);   // base matrix power calulation

        // building the side matrix
        fibo.x[0][0]=p;
        fibo.x[1][0]=2;

        // end of the side matrix

        // multiplication of base matrix and side matrix
```

```
        // result store in the rst matrix

        rst=maltiplication(rst,fibo);

        if( n == 0 ) printf("Case %d: 2\n",t);
        else
            printf("Case %d: %llu\n",t,rst.x[0][0]);
    }
    return 0;
}
```