MODULO ARITHMETIC FOUNDATIONS

* Modulo is represented by '%' and it gives us the remainder. Ex:

→ 7%3 = 1

→ -7%3 = -1 (using cout in C++)

→ -7%3 = 2 (using print in python)

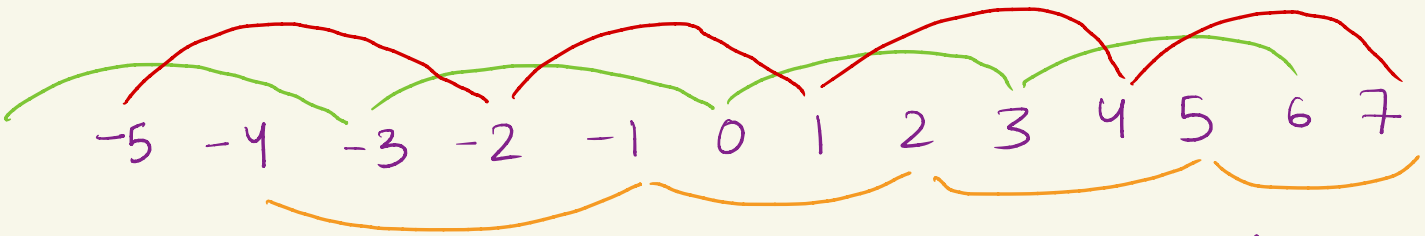
3 ⟌ 7 ⟦2

6

1 → value of 7%3.

Why different values in C++ & python output?

Let's take number line to understand this:



- If we take Modulo 3, all the values at a distance of 3 becomes equivalent, i.e,

  -6 = -3 = 0 = 3 = 6 (as shown in number line)

They become equivalent because they give the same remainder when Modulo 3.

Modulo simply breaks the number line in equivalent parts.

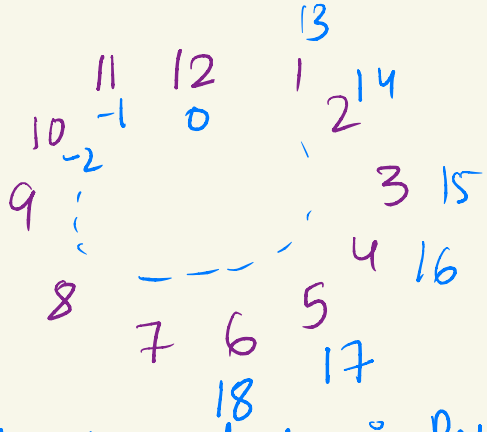
We can say that %3 will gives us the remainder [0,1,2] and anything else will convert to its equivalent form.

→ when you take a remainder with X, it should be in the range [0 ___ (X-1)]

For %3, the remainder space is [0, 1, 2].

In the above case, while we are getting diff. values i.e -1 & 2, we can see that they are equivalent. So both outputs are valid but using above knowledge, we want to try to stick within the remainder range.

This can also be thought of as a clock



[% 12]

* What is the need of Modulo in Problem solving?

C++ has a data limit. i.e,

The largest integer that you can store in INT is 2.1×10^9

even if you take LONG LONG, you can exceed till $\approx 10^{18}$

But, what if we want to compute 2^{100} (this cannot be computed as we don't have any data type that exceeds till this range) Hence, we can use modulo to make our answer fall within a permitted range. In this case & most, we use 10^9+7 for mod. So, $2^{100} \% 10^9+7$ will fall within range and can be computed.

* Why we are using 10^9+7 as our standard Modulo?

1. Prime
2. largest prime less than INT_MAX (which stores the largest value an int data type can store)
3. using standard
4. first prime after 10^9.

helps in keeping our answer within the permitted int range.

Note: 1. Try to Think of expressions in terms of bracketting (using BODMAS), especially while using Modulo. Ex:

$$(((A+B)-(C \times D))+(E^F))\ \%\ M$$

Rule of Modulo:

→ When evaluating expressions with different variables like A, B, C etc. First take the modulo of variable before performing any operation. Once you have something like A % M, B % M then you can go ahead and perform an operation like +, −, * or ÷ (inverse). and Then again take the modulo of the whole expression. Ex. $(A+B)\ \%\ M$

→ $(A\ \%\ M + B\ \%\ M)\ \%\ M$

↳ or any other operation

The above rule follows for bigger expressions too. Just break them down using parantheses & use modulo for first variables & then expressions.

$\Rightarrow E^F \% M \rightarrow (E \% M)^F$

In bigger expression, if the variable is already an int then taking modulo will result in the same number. Hence, in this case we can avoid using modulo for every single variable (given that it is int)

But, if addition, multiplication etc. happens and the resulting value is exceeding the INT range then we shall definitely take mod (also make sure to store multiplication in Long long, because it can overflow)

* Let's evaluate $(((A+B)-(C \times D)) + E^F) \% M$ — (1)

Remember this can overflow

```
6  int binpow(int a,int b,int m){
7      custom written fxn to calculate int pow
8  } because using inbuilt fxn pow() gives
9      floating value.
10 void solve(){
11     int a,b,c,d,e,f,m;
12     cin>>a>>b>>c>>d>>e>>f>>m;
13     int temp1 = (a+b)%m;
14     int temp2 = (c*d)%m;
15     int temp3 = (temp1 - temp2)%m;
16     int temp4 = (temp3 + binpow(e,f,m))%m;
17
18 (2) cout<<(temp4%m+m)%m<<endl;
19 }
20
```

$\rightarrow (A+B) \% M$

$\rightarrow (C*D) \% M$

This val can overflow & hence should be stored in Long Long.

$((A+B) \% M) - ((C+D) \% M))$ → evaluating (1) using binpow.

Here, our answer can be -ve b/c of temp3 so in order to handle that, we can add M.

Let us understand ②.

X can be of the range:

$$-\infty \leq X \leq \infty$$

$$-(M-1) \leq X \% M \leq (M-1)$$

in C++, -ve is also possible if X is -ve.
-2 % 3 = -1
(using cout)

Adding M to above equation

$$1 \leq X \% M + M \leq 2M-1$$

Taking % M

$$\boxed{0 \leq (X \% M + M) \% M \leq M-1}$$

Since M is in the range

1 2 3 --- (M-1) (M) (M+1) --- 2M-1

But this is also in range giving M % M = 0

Taking Mod

1 2 -- M-1

* How to calculate $A^B \% M$.

If B is odd

$\Rightarrow A \cdot A^{B-1} \% M$

If B is even

$\Rightarrow \left(A^{\frac{B}{2}}\right)^2 \% M$

$= \left(A^{\frac{B}{2}} \% M\right)^2 \% M$

binpow (a, b, M) (logic behind algo)

Even

$\left[\text{binpow}\left(a, \frac{b}{2}, M\right)\right]^2 \% M$

↓

This help us calculate in log(b)

ODD

$a \cdot \text{binpow}(a, b-1, M) \% M$

How is it working?

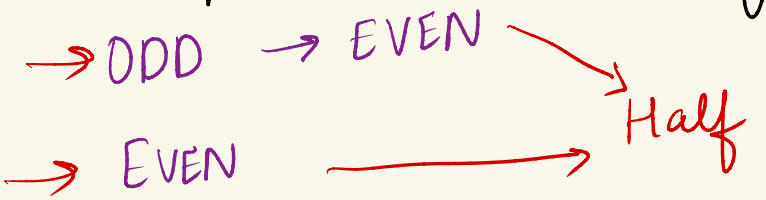
$2^{32} \to 2^{16} \to 2^8 \to 2^4 \to 2^2 \to 2^1 \to 2^0$

What if our power becomes odd while halfing the b:

$2^{28} \to 2^{14} \to 2^7 \to 2^6 \to 2^3 \to 2^2 \to 2^1 \to 2^0$

Whenever it becomes odd, our binpow for odd will make the b even.

Everytime we have ODD 'b', we go to the next even
Then continue doing the halves.
∴ In 2 steps, it will definitely half.



Coding it all :

```
int binpow(int a,int b,int m){
    if(b==0)return 1;
    if(b%2)return (a*binpow(a,b-1,m))%m;
    else{
        int temp = binpow(a,b/2,m);
        return temp*temp%m;
    }
}
```

// O(log b)

* Calculating $\left(\frac{A+B}{C}\right) \% M$ $\quad 1 \le A, B, C, M \le 10^9$

What if the result of above division is fractional like $\frac{1}{2}$ or 0.5. So, the question arise, is $0.5 \% M$, is this expression valid?

No, since Modulo is an integer operand, i.e, it operates on 2 integers. So, we cannot use division to evaluate the above expression.

- Use of Inverses :

$$\left(\frac{P}{Q}\right) \% M$$

$$\Rightarrow \left[(P \% M) * (Q \% M)^{-1}\right] \% M$$

Any number a^{-1} is said to be an inverse of a if $a \cdot a^{-1} = 1 \pmod M$

$$a.a^{-1} \% M = 1$$

For every number, we are saying that inverse is a number that when you multiply it to it and take remainder with M, it will gives us 1. Ex: mod 7.

a	0	1	2	3	4	5	6
a^{-1}	X	1	4	5	2	3	6

using hit & trial, we calculated a^{-1} here.

$a \cdot a^{-1} \% 7 = 1$

$(2 \cdot 4) \% 7 = 1$

$8 \% 7 = 1$

$1 = 1$

$a^{-1} \bmod 6$

a	0	1	2	3	4	5
a^{-1}	✗	1	✗	✗	✗	5

Lots of inverses don't exist.

For a^{-1} to exist :

$$GCD(a, m) == 1$$

else inverse don't exist

↳ This is also why we have standard modulo 10^9+7 which is prime, because its GCD with a number will always be 1 making sure that the inverse exist.

* How to find Inverse : Fermat's Little Theorem

$p \rightarrow$ prime

$$\Rightarrow \boxed{a^{p-1} \% p = 1}$$

a, p are co-prime.

Using this, we can derive

$$a \cdot a^{p-2} \% p = 1$$

, compare this with

$$a \cdot a^{-1} \% p = 1$$

$$\Rightarrow \boxed{a^{-1} \% p = a^{p-2} \% p}$$

Finding inverse of x with mod p.

→ This will give us inverse.

```
int inv(int x,int p){
    return binpow(x,p-2,p);
}
```

→ We can do this in log (p-2) using binpow.

SUMMARY :

① 2 at a time $(a * b) \% M$

Take mod if needed for individual variables.

② Use Long Long `#define int long long`

③ `cout << (ANS % M + M) % M` Handle -ve values

④ Always use binpow function

`bin(a, b, m)`

⑤ $\frac{A}{B} \% M$

→ M will be prime

→ $A \cdot B^{-1} \% M$

→ $A \cdot B^{M-2} \% M$

$$= \left[(A \% M) \cdot \left((B^{M-2}) \% M \right) \right] \% M$$

$$= \left[(A \% M) \left((B \% M)^{M-2} \% M \right) \right] \% M$$