Super Pow

Your task is to calculate a^b mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Example1:

```
a = 2
b = [3]

Result: 8
```

Example2:

```
a = 2
b = [1,0]

Result: 1024
```

Solution 1

One knowledge: a*b % k = (a%k)*(b%k)%k
Since the power here is an array, we'd better handle it digit by digit.
One observation:
a^1234567 % k = (a^1234560 % k) * (a^7 % k) % k = (a^123456 % k)^10 % k * (a^7 % k) % k
Looks complicated? Let me put it other way:
Suppose f(a, b) calculates a^b % k; Then translate above formula to using f :
f(a,1234567) = f(a, 1234560) * f(a, 7) % k = f(f(a, 123456),10) * f(a,7)%k;
Implementation of this idea:

```
class Solution {
    const int base = 1337;
    int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10
    {
        a %= base;
        int result = 1;
        for (int i = 0; i < k; ++i)
            result = (result * a) % base;
        return result;
    }
public:
    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1;
        int last_digit = b.back();
        b.pop_back();
        return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;
    }
};
```

Note: This approach is definitely not the fastest, but it is something one can quickly understand and write out when asked in an interview.
And this approach is not using any built-in "pow" functions, I think this is also what the interviewer would expect you to do.
Hope it helps!

written by fentoyal original link here

Solution 2

1337 only has two divisors 7 and 191 exclusive 1 and itself, so judge if `a` has a divisor of 7 or 191, and note that 7 and 191 are prime numbers, `phi` of them is itself - 1, then we can use the Euler's theorem, see it on wiki https://en.wikipedia.org/wiki/Euler's_theorem, it's just Fermat's little theorem if the mod `n` is prime.

see how 1140 is calculated out:
phi(1337) = phi(7) * phi(191) = 6 * 190 = 1440

optimized solution update at 2016-7-12

Today, seeing @myanonymos 's comments, and I find several days ago I AC it just by fortunate coincidence, it's not the best solution. now I get a better idea.

(1) Firstly, if `a` has both divisor 7 and 191, that's `a % 1337 == 0`, answer is 0.
(2) Then if `a` has neither divisor 7 nor 191, that's a and 1337 are coprime, so$\mathbf{a^b \% 1337 = a^{b \% phi(1337)} \% 1337 = a^{b \% 1140} \% 1337}$.

(3) Finally, `a` could has either divisor 7 or 191, that's similar.
Let it be 7 for example.
Let $\mathbf{a = 7^n x}$
and let $\mathbf{b = 1140p + q}$, where $\mathbf{0 < q <= 1140}$
then:

$$\mathbf{a^b \% 1337}$$
$$\mathbf{= ((7^n x)^b) \% 1337}$$
$$\mathbf{= (7^{nb} x^b) \% 1337}$$
$$\mathbf{= ((7^{nb} \% 1337) * (x^b \% 1337)) \% 1337}$$
$$\mathbf{= ((7^{1140np + nq} \% 1337) * (x^{1140p + q} \% 1337)) \% 1337}$$

now note x and 1337 are coprime, so

$$\mathbf{= ((7^{1140np + nq} \% 1337) * (x^q \% 1337)) \% 1337}$$
$$\mathbf{= (7 * (7^{1140np + nq - 1} \% 191) * (x^q \% 1337)) \% 1337}$$

note 7 and 191 are coprime too, and 1140 is a multiple of 190, where 190 = phi(191). What's more we should assure that q != 0, if b % 1140== 0, then let b = 1140. so

$$\mathbf{= (7 * (7^{nq - 1} \% 191) * (x^q \% 1337)) \% 1337}$$
$$\mathbf{= ((7^{nq} \% 1337) * (x^q \% 1337)) \% 1337}$$
$$\mathbf{= (7^{nq} x^q) \% 1337}$$
$$\mathbf{= ((7^n x)^q) \% 1337}$$
$$\mathbf{= (a^q) \% 1337}$$

now you see condition (2) and (3) can be merged as one solution, if you take care of when `b % 1440 == 0`, and let `b += 1140`. Actually (1) can be merged too, but not efficient.

new code:
C++:

```
int superPow(int a, vector<int>& b) {
    if (a % 1337 == 0) return 0; // this line could also be removed
    int p = 0;
    for (int i : b) p = (p * 10 + i) % 1140;
    if (p == 0) p += 1140;
    return power(a, p, 1337);
}
int power(int x, int n, int mod) {
    int ret = 1;
    for (x %= mod; n; x = x * x % mod, n >>= 1) if (n & 1) ret = ret * x % mod;
    return ret;
}
```

java:

```
public int superPow(int a, int[] b) {
    if (a % 1337 == 0) return 0;
    int p = 0;
    for (int i : b) p = (p * 10 + i) % 1140;
    if (p == 0) p += 1440;
    return power(a, p, 1337);
}
public int power(int a, int n, int mod) {
    a %= mod;
    int ret = 1;
    while (n != 0) {
        if ((n & 1) != 0) ret = ret * a % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return ret;
}
```

Actually, if p == 0 or not, we can always let p += 1140, it doesn't matter.
one line python:

```
def superPow(self, a, b):
    return 0 if a % 1337 == 0 else pow(a, reduce(lambda x, y: (x * 10 + y) % 1140
, b) + 1440, 1337)
```

will this be the best solution?

p.s.
I have testcases that the system missed
574
[1,1,4,0]

764
[1,1,4,0]
in this case if I remove this line of code `if (p == 0) p += 1140;`, it will goes wrong, but also can get AC on OJ.

and I found thar 574 * 574 % 1337 = 574, 764 * 764 % 1337 = 764, how interesting!

written by ShuangquanLi original link here

Solution 3

Solution 1: Using Python's big integers (accepted in 72 ms)

Turn `b` into a Python integer object (they grow arbitrarily large) and just use the `pow` function (which supports a modulo paramenter).

```
def superPow(self, a, b):
    return pow(a, int(''.join(map(str, b))), 1337)
```

Solution 2: Using small ints (accepted in 80 ms)

Originally I went backwards (see solution 5) but then I saw other people go forwards and it's simpler. Sigh. Anyway... my version:

```
def superPow(self, a, b):
    result = 1
    for digit in b:
        result = pow(result, 10, 1337) * pow(a, digit, 1337) % 1337
    return result
```

Explanation: For example for a^{5347}, the above computes a^5, then a^{53}, then a^{534}, and then finally a^{5347}. And a step from one to the next can be done like $a^{5347} = (a^{534})^{10} * a^7$.

Solution 3: Using recursion (accepted in 92 ms)

Obligatory recursive oneliner version of solution 2.

```
def superPow(self, a, b):
    return pow(a, b.pop(), 1337) * pow(self.superPow(a, b), 10, 1337) % 1337 if b
else 1
```

Solution 4: Using `reduce` (accepted in 80 ms)

Obligatory `reduce`-oneliner version of solution 2.

```
def superPow(self, a, b):
    return reduce(lambda result, digit: pow(result, 10, 1337) * pow(a, digit, 133
7) % 1337, b, 1)
```

Solution 5: omg was i stupid (accepted in 72 ms)

My original do-it-yourself before I saw other people's solutions and wrote solutions 2-4.

Using only small ints, also accepted in 72 ms:

```
def superPow(self, a, b):
    result = 1
    apower = a
    for digit in reversed(b):
        result = result * pow(apower, digit, 1337) % 1337
        apower = pow(apower, 10, 1337)
    return result
```

Explanation by example:

a^{5347}
$= a^{5000} * a^{300} * a^{40} * a^{7}$
$= (a^{1000})^{5} * (a^{100})^{3} * (a^{10})^{4} * a^{7}$
$= (((a^{10})^{10})^{10})^{\mathbf{5}} * ((a^{10})^{10})^{\mathbf{3}} * (a^{10})^{\mathbf{4}} * a^{7}$

Computing that from back to front is straightforward (or straightbackward?).

written by StefanPochmann original link here

From Leetcoder.