Pharisaeus Added eng version to dubkey and danklang Latest commit ad85b3e on 3 Dec 2015

..

README.md	Added eng version to dubkey and danklang	a year ago
main.dc	Add danklang and dub-key	2 years ago
main1.py	Add danklang and dub-key	2 years ago
main2.py	Update main2.py	2 years ago
main3.py	Add danklang and dub-key	2 years ago

README.md

danklang (re, 100p)

> if you see this task while scroling
>
> you have been visited by the reversing task of the 9447 ctf good flags and points will come to you
>
> but only if you submit '9447{ `dankcode main.dc` }' to this task.
>
> main.dc

###PL ENG

Przepełnione memami zadanie (co widać nawet po wstępie).

Dostajemy długi kod w nieistniejącym języku.

Ciężko go czytać, więc zaczynamy od przepisania go literalnie do pythona: main1.py.

Widać że jest niesamowicie nieoptymalny - więc po prostu uruchomienie kodu main1.py prawdopodobnie nie skończyłoby się za naszego życia (a na pewno nie w trakcie trwania CTFa)

Rozpoczynamy rozpoznawanie funkcji które można zoptymalizować:

Na przykład to nic innego niż fibonacci(memes) % 987654321

```
def brotherman(memes):
    hues = 0
    if memes != 0:
        if memes < 3:
            return 1
        else:
            wew = brotherman(memes - 1)
            hues = wew
            wew = brotherman(memes - 2)
            hues += wew
    return hues % 987654321
```

Jako że maksymalna wartość memes nie jest olbrzymia, możemy po prostu obliczyć wcześniej wszystkie wartości (precomputing):

```
def precompute_fibonacci_mod_987654321():
    table = []
    N = 13379447+1
```

```
    result = [0] * N
    result[1] = 1
    for i in xrange(2, N):
        result[i] = (result[i-2] + result[i-1]) % 987654321
    return result

precomputed_fibonacci = precompute_fibonacci_mod_987654321()

def fibonacci_mod_987654321(number):
    return precomputed_fibonacci[number]
```

Za to tutaj rozpoznajemy funkcję sprawdzającą czy liczba jest pierwsza:

```
def fail(memes, calcium):
    dank = True
    if calcium < memes:
        if memes % calcium == 0:
            dank = False
        else:
            wew = fail(memes, calcium + 1)
            dank = wew
    return dank
```

I przepisujemy ją do takiej postaci:

```
def is_prime(number):
    if number % 2 == 0:
        return False
    else:
        for divisor in range(3, int(sqrt(number)) + 1, 2):
            if number % divisor == 0:
                return False
    return True
```

Dochodzimy do takiego stanu: main2.py

W tym momencie kończą się oczywiste pomysły na optymalizację, a wykonanie dalej jest bardzo powolne. Decydujemy się więc na więcej precomputingu, i obliczać z góry wszystko co się da.

Przepisujemy więc sprawdzanie pierwszych:

```
def precompute_primes():
    limit = 13379447 + 1
    a = [True] * limit
    for i in xrange(2, len(a)):
        isprime = a[i]
        if isprime:
            for n in xrange(i*i, limit, i):
                a[n] = False
    return a

primes = precompute_primes()

def is_prime(number):
    return primes[number]
```

funkcję dootdoot (nie rozpoznaliśmy jej, a wygląda na jakąś znaną funkcję z prostym wzorem matematycznym):

```
def dootdoot(memes, seals):
    if seals <= memes:
        if seals == 0:
            return 1
        else:
            if seals == memes:
                return 1
            else:
                return dootdoot(memes - 1, seals - 1) + dootdoot(memes - 1, seals)
```

Na taką formę:

```
def precompute_dootdoot():
    table = []
    MAXH, MAXW = 6, 13379447+1
    for i in range(MAXH):
        table.append([0] * MAXW)
    for i in range(0, MAXH):
        for j in xrange(0, MAXW):
            if i > j:
                table[i][j] = 0
            elif i == 0:
                table[i][j] = 1
            elif i == j:
                table[i][j] = 1
            else:
                table[i][j] = table[i][j-1] + table[i-1][j-1]
    return table

dootdoot_table = precompute_dootdoot()
def dootdoot(memes, seals):
    return dootdoot_table[seals][memes]
```

Na końcu trzy powiązane funkcje - such, epicfail i bills:

```
def epicfail(memes):
    if memes > 1:
        if dank(memes, 2):
            return 1 + bill(memes - 1)
        else:
            return such(memes - 1)
    return 0

def such(memes):
    wow = dootdoot(memes, 5)
    if wow % 7 == 0:
        wew = bill(memes - 1)
        wow += 1
    else:
        wew = epicfail(memes - 1)
    wow += wew
    return wow

def bill(memes):
    wow = fibonacci_mod_987654321(memes)
    if wow % 3 == 0:
        wew = such(memes - 1)
        wow += 1
    else:
        wew = epicfail(memes - 1)
    wow += wew
    return wow
```

Do takiej postaci:

```
def bill(memes):
    wow = fibonacci_mod_987654321(memes)
    if wow % 3 == 0:
        wew = suchs[memes - 1]
        wow += 1
    else:
        wew = epicfails[memes - 1]
    wow += wew
    return wow

def such(memes):
    wow = dootdoot(memes, 5)
    if wow % 7 == 0:
        wew = bills[memes - 1]
        wow += 1
    else:
        wew = epicfails[memes - 1]
    wow += wew
    return wow

def epicfail(i):
```

```
    if i > 1:
        if is_prime(i):
            return 1 + bill(i - 1)
        else:
            return such(i - 1)
    return 0

epicfails = [0] * (13379447 + 1)
suchs = [0] * (13379447 + 1)
bills = [0] * (13379447 + 1)

def upcompute_epicfails():
    for i in xrange(1, 13379447+1):
        if i % 10000 == 0:
            print i
        epicfails[i] = epicfail(i)
        suchs[i] = such(i)
        bills[i] = bill(i)

upcompute_epicfails()
```

W tym momencie rozwiązanie zadania staje sie trywialne - skoro mamy już wszystkie wartości wyliczone, starczy pobrać wynik z tablicy:

```
def me():
    memes = 13379447
    wew = epicfails[memes]
    print(wew)
```

Przepisywanie tego zajęło dość dużo czasu, ale ostatecznie doszliśmy do takiej formy jak main3.py. Uruchomiony kod wykonywał się dość długo, ale ostatecznie dostaliśmy wynik: `29929595198958502010206163344264641209 87`

Po dodaniu stałych części:

```
9447{29929595198958502010206163344264641209 87}
```

Zdobywamy punkty

ENG version

Task full of memes (which you can see even from the task description).

We get a long code in a made-up language.

It's difficult to read so we rewrite it to python: main1.py.

It is clear that it's not optimal and execution of main1.py would not finish before we die (and for sure not before CTF ends).

We start with trying to recognize some function we could optimize:

For example this function is fibonacci(memes) % 987654321:

```
def brotherman(memes):
    hues = 0
    if memes != 0:
        if memes < 3:
            return 1
        else:
            wew = brotherman(memes - 1)
            hues = wew
            wew = brotherman(memes - 2)
            hues += wew
    return hues % 987654321
```

And since the memes variable is not so big, we can simply calculate all values before (precomputing):

```
def precompute_fibonacci_mod_987654321():
    table = []
    N = 13379447+1
```

```
    result = [0] * N
    result[1] = 1
    for i in xrange(2, N):
        result[i] = (result[i-2] + result[i-1]) % 987654321
    return result

precomputed_fibonacci = precompute_fibonacci_mod_987654321()

def fibonacci_mod_987654321(number):
    return precomputed_fibonacci[number]
```

Here we recognize primarity test:

```
def fail(memes, calcium):
    dank = True
    if calcium < memes:
        if memes % calcium == 0:
            dank = False
        else:
            wew = fail(memes, calcium + 1)
            dank = wew
    return dank
```

And change it for a faster one:

```
def is_prime(number):
    if number % 2 == 0:
        return False
    else:
        for divisor in range(3, int(sqrt(number)) + 1, 2):
            if number % divisor == 0:
                return False
    return True
```

We end up with a new version: main2.py

At this point we run out of obvious ideas to optimize the code, and the execution time is still too long. We decide to do even more precomputing and just calculate all possible values.

We precompute primarity check:

```
def precompute_primes():
    limit = 13379447 + 1
    a = [True] * limit
    for i in xrange(2, len(a)):
        isprime = a[i]
        if isprime:
            for n in xrange(i*i, limit, i):
                a[n] = False
    return a

primes = precompute_primes()

def is_prime(number):
    return primes[number]
```

Function dootdoot (we didn't recognize it at the time, but it is just number of combinations):

```
def dootdoot(memes, seals):
    if seals <= memes:
        if seals == 0:
            return 1
        else:
            if seals == memes:
                return 1
            else:
                return dootdoot(memes - 1, seals - 1) + dootdoot(memes - 1, seals)
```

We change into:

```
def precompute_dootdoot():
    table = []
    MAXH, MAXW = 6, 13379447+1
    for i in range(MAXH):
        table.append([0] * MAXW)
    for i in range(0, MAXH):
        for j in xrange(0, MAXW):
            if i > j:
                table[i][j] = 0
            elif i == 0:
                table[i][j] = 1
            elif i == j:
                table[i][j] = 1
            else:
                table[i][j] = table[i][j-1] + table[i-1][j-1]
    return table

dootdoot_table = precompute_dootdoot()
def dootdoot(memes, seals):
    return dootdoot_table[seals][memes]
```

And finally three connected function - such, epicfail i bills:

```
def epicfail(memes):
    if memes > 1:
        if dank(memes, 2):
            return 1 + bill(memes - 1)
        else:
            return such(memes - 1)
    return 0

def such(memes):
    wow = dootdoot(memes, 5)
    if wow % 7 == 0:
        wew = bill(memes - 1)
        wow += 1
    else:
        wew = epicfail(memes - 1)
    wow += wew
    return wow

def bill(memes):
    wow = fibonacci_mod_987654321(memes)
    if wow % 3 == 0:
        wew = such(memes - 1)
        wow += 1
    else:
        wew = epicfail(memes - 1)
    wow += wew
    return wow
```

Are changed into:

```
def bill(memes):
    wow = fibonacci_mod_987654321(memes)
    if wow % 3 == 0:
        wew = suchs[memes - 1]
        wow += 1
    else:
        wew = epicfails[memes - 1]
    wow += wew
    return wow

def such(memes):
    wow = dootdoot(memes, 5)
    if wow % 7 == 0:
        wew = bills[memes - 1]
        wow += 1
    else:
        wew = epicfails[memes - 1]
    wow += wew
    return wow

def epicfail(i):
```

```
    if i > 1:
        if is_prime(i):
            return 1 + bill(i - 1)
        else:
            return such(i - 1)
    return 0

epicfails = [0] * (13379447 + 1)
suchs = [0] * (13379447 + 1)
bills = [0] * (13379447 + 1)

def upcompute_epicfails():
    for i in xrange(1, 13379447+1):
        if i % 10000 == 0:
            print i
        epicfails[i] = epicfail(i)
        suchs[i] = such(i)
        bills[i] = bill(i)

upcompute_epicfails()
```

And now solving the task is trivial - we have all values calculated so we simply need to read the result from the array:

```
def me():
    memes = 13379447
    wew = epicfails[memes]
    print(wew)
```

It took us a while to rewrite this but we finally got main3.py. The code was still running for quite a while but finally we got the result: `29929595198958502010206163344264641209 87` which resulted in flag:

```
9447{29929595198958502010206163344264641209 87}
```