

pf-13. アルゴリズム

(Python 入門)

URL: <https://www.kkaneko.jp/pro/pf/index.html>

金子邦彦

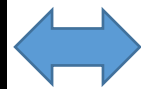


算法 (アルゴリズム) の例



$5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5$
は **152587890625**

```
In [1]: a = 5 * 5
In [2]: b = a * a
In [3]: c = b * b
In [4]: d = c * c
In [5]: print(d)
152587890625
```



同じ
結果

```
In [6]: print(5*5*5*5*5*5*5*5*5*5*5*5*5*5*5)
152587890625
```

掛け算: 15 回

掛け算: 4 回

算法 (アルゴリズム) の工夫で, 掛け算の回数を削減できる場合がある

次の2つのプログラムは、同じ答えが出る。

```
a = 5 * 5
```

```
b = a * a
```

```
c = b * b
```

```
d = c * c
```

```
print(d)
```

```
print(5*5*5*5*5*5*5*5*5*5*5*5*5*5*5*5)
```

Print output (drag lower right corner to resi

```
152587890625  
152587890625
```

Frames

Obj

Global frame

a	25
b	625
c	390625
d	152587890625

次の2つのプログラムは、同じ答えが出る

```
a = 10 * 10
```

```
b = a * a
```

```
c = b * b
```

```
print(c)
```

```
print(10*10*10*10*10*10*10*10)
```

Print output (drag lower right corner to resi

```
1000000000  
1000000000
```

Frames

Objects

Global frame

a	100
b	10000
c	1000000000

暗号を作成する Python プログラム



```
In [2]: def DH_exchange(p):  
        """ generates a shared DH key """  
        g = randrange(1,p-1)  
        a = randrange(1,p-1) # Alice's secret  
        b = randrange(1,p-1) # Bob's secret  
        x = pow(g,a,p)  
        y = pow(g,b,p)  
        key_A = pow(y,a,p)  
        key_B = pow(x,b,p)  
        return g, a, b, x, y, key_A, key_B
```

```
In [3]: DH_exchange(190125101)
```

```
Out[3]: (138828318, 163869277, 133309977, 103589429, 144268455, 9912066, 9912066)
```

Diffie-Hellman法で暗号化 (Pythonを使用)

参考ウェブページ

<http://nbviewer.jupyter.org/github/yoavram/CS1001.py/blob/master/recitation4.ipynb>

暗号を解読する Python プログラム



```
In [4]: def crack_DH(p, g, x):
        """find secret "a" that satisfies  $g^a \equiv x \pmod{p}$ 
        Not feasible for large  $p$ """
        for a in range(1,p-1):
            if a % 100000 == 0:
                print("Iteration",a) # progress bar
            if pow(g,a,p) == x:
                return a
        return None
```

```
In [5]: def find_prime(n):
        """ find random n-bit long prime (no leading zeros:  $2^{n-1} < N < 2^n$ ) """
        while True: #here we're optimistic, but actually we have a good reason to be:
            # after  $O(1/n)$  iterations we expect to find a prime and halt
            candidate = randrange(2**(n - 1), 2**n)
            if is_prime(candidate):
                return candidate
```

```
In [6]: from math import log
        p = find_prime(10)
        print(p,log(p,2))
        g,a,b,x,y,key_A,key_B = DH_exchange(p)
        print('g',g,'a',a,'b',b,'x',x,'y',y,'key_A',key_A,'key_B',key_B)
        print('a',crack_DH(p,g,x))
```

```
983 9.94104760634058
g 867 a 598 b 417 x 519 y 75 key_A 393 key_B 393
a 107
```

Diffie-Hellman法の暗号を解読するプログラム

参考ウェブページ

<http://nbviewer.jupyter.org/github/yoavram/CS1001.py/blob/master/recitation4.ipynb>

暗号作成より，暗号解読の方が手間がかかる

- **暗号解読**の**算法（アルゴリズム）**は**発見済み** ということが多い
→ プログラムを作成可能
- しかし，プログラムが答えを出すまでに**時間がかかる**場合がある

すでに，2012年に，このようなレポートが.

<https://www.dit.co.jp/service/security/report/03.html>

英大小文字+数字+記号を組み合わせた ZIP のパスワード

6桁の解読時間： 2分24秒 = 超危険

8桁の解読時間： 14日 = 危険

10桁の解読時間： 341年

- **算法（アルゴリズム）の作成が不可能**という問題は、すでに発見済みである

人間にもコンピュータにも解けない問題

チューリングマシンの停止判定

- **算法 (アルゴリズム)**

※ コンピュータは、ある定まった手順により問題を解く
プログラムの見通しの良さ、性能向上、バグの防止に役立つ

- **算法 (アルゴリズム) を駆使しても、解くのに時間がかかりすぎるため難しいという問題がある**

このことは、**暗号**の基礎として役立つ