Probabilités et informatique I

Aléatoire et machine

INFO2 - semaines 36 à 37

Guillaume CONNAN

septembre 2015

Iut de Nantes - Dpt d'informatique

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/moi/.ssh/id_rsa): /tmp/id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /tmp/id_rsa.
Your public key has been saved in /tmp/id_rsa.pub.
```

```
----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAyq/yDe2K8EGOEYfj3YBqtjPWeVEUcEiNv6EFxEf17bQlUr1z
UBPxs5uWPFkR8lKKoW+pipAmrM9mgJc/wQz3VoNO6xqic7XAjeTcHRyUmfOkHFAb
OFDzpAzDQMguLzhoKm5ac6u91kxL8pKLvrgDc1HVhHXild8pkjtVgGJFpOYu2av+
PwbAdOIdgIyfatOtlQJjjNi/X+jlcP7+SmEkdutft7ZieBkEYoJddYQBllzGsgVS
vvwZFwDY3DcVoCndeSDy7vdEXdXGw7HQGZd2y2xlxeoTNSTCxUibjWtBkQTuFK52
9g5HCg3PDJk+qBObOZ5FT2beomimpsJdLq5BhQIDAQABAoIBABUVQidiux4Mo5t9
OB2VOwRi1f9eiLK5SU1SkYf+OD2WjH9Tx4ff2zdkircSSMSdA2CRfeAO6GD3XeEo
WnBZSeOMOYuxBI7mq/XWhXLdwzNVWok7Z/k+QmmhDjpR2EP/KQ6o1e3MQn3lB81e
ovyepXU8YOpbXMtcIDXeyCEHl9jODQii1RgqL1BKxQz2ubcIWBlx4Foyvlgmx0Vs
us3f2T5yKBPgOdP3reZ4OoSJFq4vWQxIzOHl59lPNU4zGOz9vEOsNAbRbjZkqm4W
3dtPDVnUonGn+kSiDOuI+P7jShHVKVSahEmMMoxUGoPkUbIfTq9gKSCs8IkOLqlp
mUajVSECgYEA9l97nC2OaFMdywrKPunitI43+MR8g4KVmAeeZYIoMEtasrg/twRt
TiVuhu6j+NxaZoRuZxaIHPvc9YoPhyrjidnSwBeM23wVU3GSObr3pCcwBZniuWJa
WMjMKTArqdWMPqTFWhfHb7IgOYpxLtWWy5Lo4+Cv2hHVZoZlWkbE4JkCgYEAOpt3
VMZHNO+aE3StvBremrNsbdv1nkIAOAlHWju/w+c9qeE4yxB5SG1w7suJgOLw8Tjm
E4OCeJYzJiZ29nBtOq/ZTo4OoOrO9NnUVTVRuZkVRcLdLnbPnPYkxxbmjkOrCblL
lM+JpQo9xeRwetUUAkJIKDUks+aVI4avmjx+/8OCgYEA3UDcemIzHJ9gByLWg+94
55s8ysxx4YyZGWHmLtTXkJ9rgUqT9aElJGPbQZP4b5o8GQrfNvb2H8lUIrHsxf+P
PhH/AQFlPwrNABXjMrMkiCelKbKL+N5JsqgJKQgYfjnH683/aawnL7jxXhyNDnlE
Jef1/AehBR6O3ZDeXldloSkCgYEAxFUoBoCdCwO+flqsR4po8KDJORTvwWvX1mVX
/hdrKs6cXozWK/+cgNd9iy91bRCi+s1HNvyiTBYBctQy3IRs7Rb/psqpenvuhX+4
AQMmv7Wk7ha9qL4bxOBzgqGnpwIQJY12lzBuKcwFUEEoVmnjMEPoDouUPp17dNXv
SAKThhUCgYEAkqDr3LeWh6tLxcfWlcdE5/ukZcaKNKH1w2ShyBbHB9I7UJFlOyoZ
uGUp9yLftgg2eBECwzeRisdoDQSRtPofF8zMBTBV43xV58RdAOBgJ3NZxq7f2Stv
vJW53PNT8udYFLh5lnaVfliO5J11lboO3KWGfFfWM3bYcVMvbda16zQ=
-----END RSA PRIVATE KEY-----
```

- réseau
- cryptographie
- test
- jeux
- infographie
- tri – recherche
- langues naturelles
- ...

- réseau
- cryptographie
- test
- jeux
- infographie
- tri – recherche
- langues naturelles
- ...

- réseau
- cryptographie
- test
- jeux
- infographie
- tri – recherche
- langues naturelles
- ...

- réseau
- cryptographie
- test
- jeux
- infographie
- tri - recherche
- langues naturelles
- ...

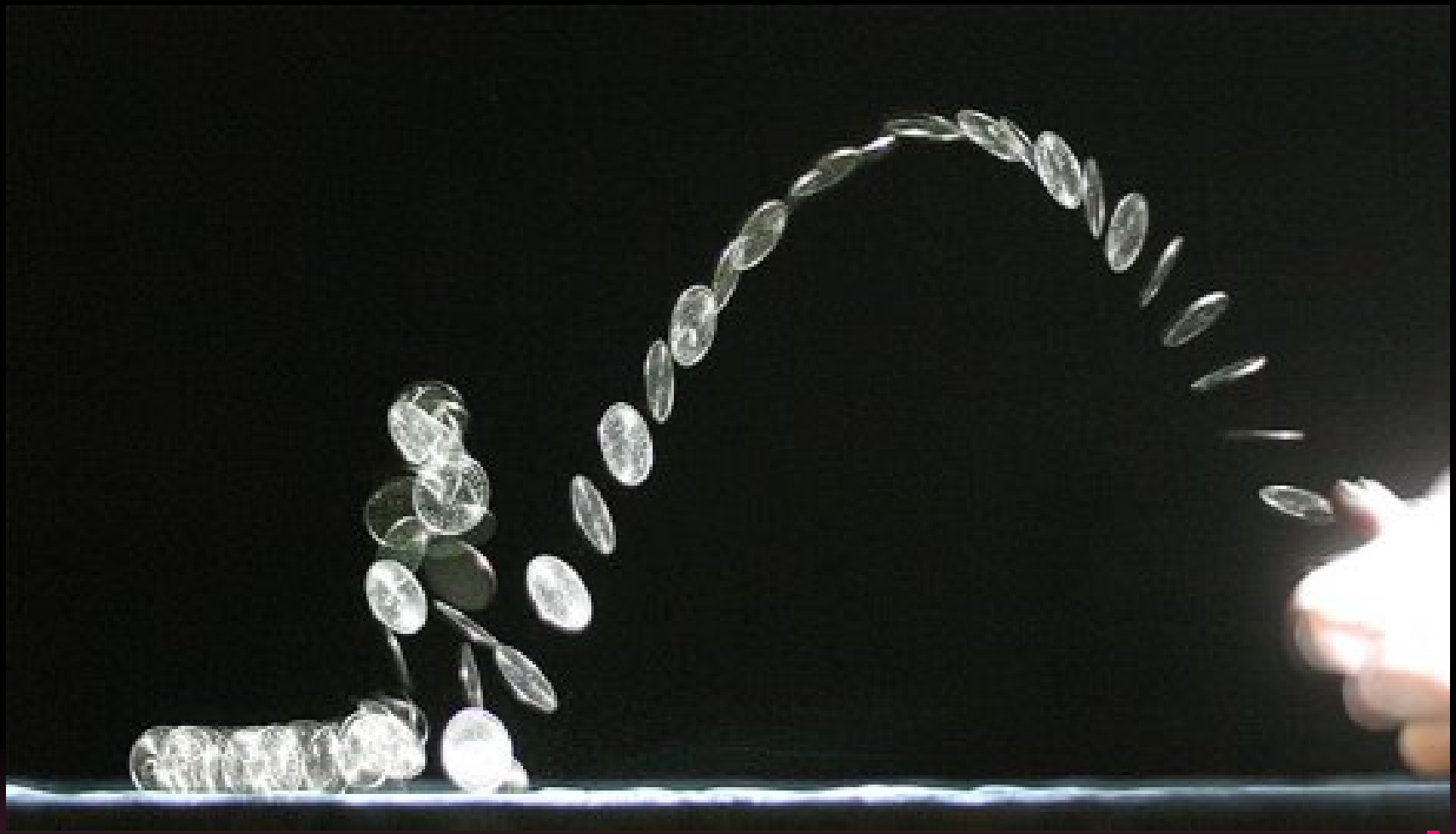
- réseau
- cryptographie
- test
- jeux
- infographie
- tri – recherche
- langues naturelles
- ...

- réseau
- cryptographie
- test
- jeux
- infographie
- tri – recherche
- langues naturelles
- ...

- réseau
- cryptographie
- test
- jeux
- infographie
- tri – recherche
- langues naturelles
- ...

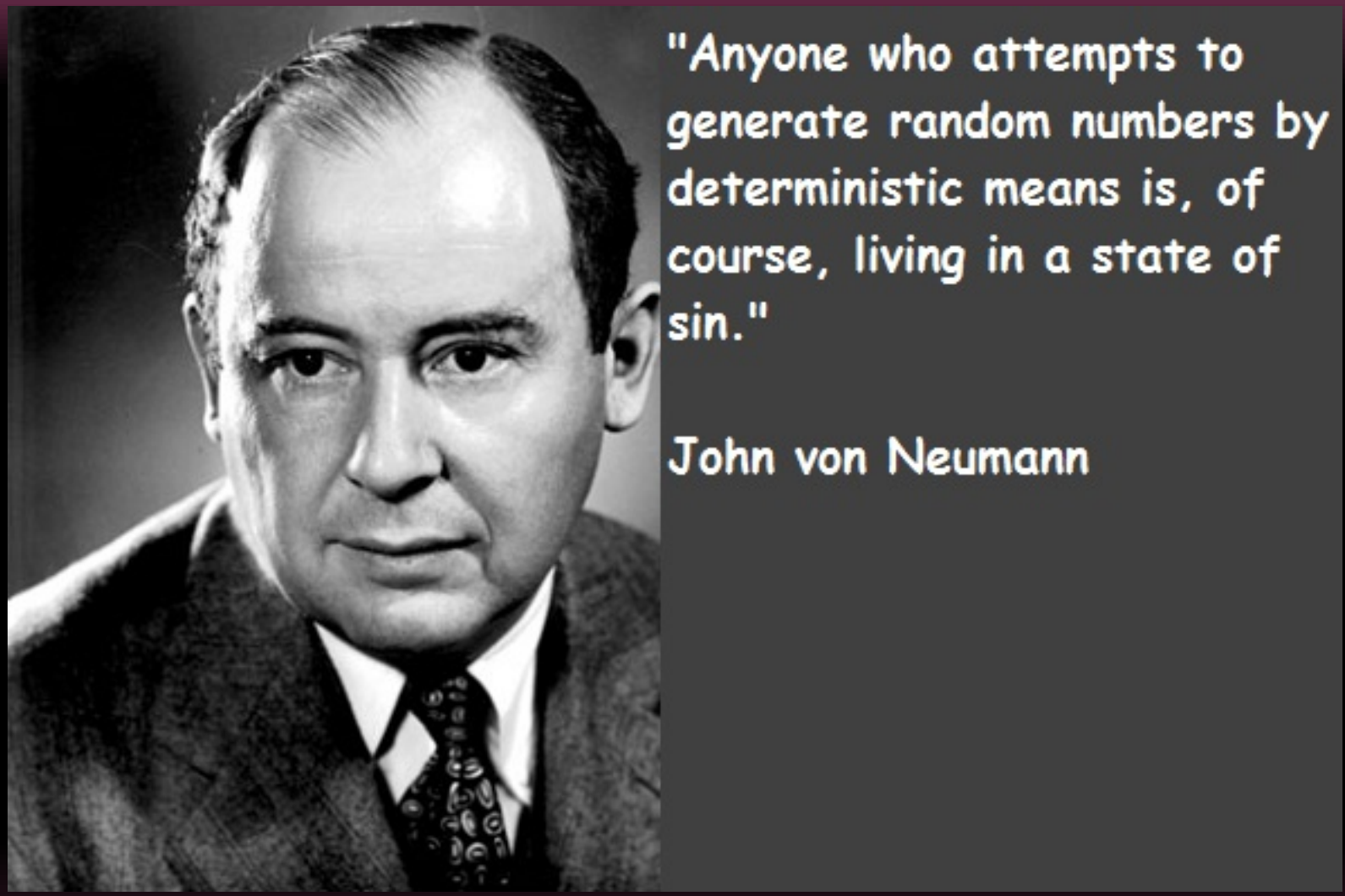
- réseau
- cryptographie
- test
- jeux
- infographie
- tri – recherche
- langues naturelles
- ...

Aléatoire ?



Sur machine ?

```
int getRandomNumber()
{
      return 4 ; // J'ai trouvé 4 en lançant un dé
                 // Hasard assuré, j'le jure
}
```


"Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin."
John von Neumann

- On veut générer une suite pseudo-aléatoire de 10 chiffres
- L'actuel est 5 772 156 649
- Son carré vaut 33 317 792 380 594 909 201
- Le nouveau nombre est donc 7 923 805 949

- On veut générer une suite pseudo-aléatoire de 10 chiffres
- L'actuel est 5 772 156 649
- Son carré vaut 33 317 792 380 594 909 201
- Le nouveau nombre est donc 7 923 805 949

- On veut générer une suite pseudo-aléatoire de 10 chiffres
- L'actuel est 5 772 156 649
- Son carré vaut 33 317 792 380 594 909 201
- Le nouveau nombre est donc 7 923 805 949

- On veut générer une suite pseudo-aléatoire de 10 chiffres
- L'actuel est 5 772 156 649
- Son carré vaut 33 317 792 380 594 909 201
- Le nouveau nombre est donc 7 923 805 949

Python

```python
def prendMilieu(nb):
    lnb = len(str(nb))
    return((nb**2 // 10**(max(1,lnb // 2)) ) % 10**lnb)
```

```python
def longVonNeumann(graine):
    L = []
    k = graine
    while k not in L:
        L.append(k)
        k = prendMilieu(k)
    return(len(L))
```

```python
def prendMilieu(nb):
    lnb = len(str(nb))
    return((nb**2 // 10**(max(1,lnb // 2)) ) % 10**lnb)
```

```python
def longVonNeumann(graine):
    L = []
    k = graine
    while k not in L:
        L.append(k)
        k = prendMilieu(k)
    return(len(L))
```

```
In [45]: prendMilieu(5772156649)
Out[47]: 7923805949
```

```
In [48]: max([longVonNeumann(k) for k in range(900000,1000000)])
Out[48]: 119
```

```
In [45]: prendMilieu(5772156649)
Out[47]: 7923805949
```

```
In [48]: max([longVonNeumann(k) for k in range(900000,1000000)])
Out[48]: 119
```

D. H. Lehmer
1905–1991

1949 :

$$X_{n+1} = (aX_n + c) \mod m$$

- m, le module, vérifiant $0 < m$,
- a, le facteur, vérifiant $0 \leqslant a < m$,
- c, l'incrément, vérifiant $0 \leqslant c < m$,
- X_0, la graine, vérifiant $0 \leqslant X_0 < m$,

1949 :

$$X_{n+1} = (aX_n + c) \mod m$$

- m, le module, vérifiant $0 < m$,
- a, le facteur, vérifiant $0 \leqslant a < m$,
- c, l'incrément, vérifiant $0 \leqslant c < m$,
- X_0, la graine, vérifiant $0 \leqslant X_0 < m$,

1949 :

$$X_{n+1} = (aX_n + c) \mod m$$

- m, le module, vérifiant $0 < m$,
- a, le facteur, vérifiant $0 \leqslant a < m$,
- c, l'incrément, vérifiant $0 \leqslant c < m$,
- X_0, la graine, vérifiant $0 \leqslant X_0 < m$,

1949 :

$$X_{n+1} = (aX_n + c) \mod m$$

- m, le module, vérifiant $0 < m$,
- a, le facteur, vérifiant $0 \leqslant a < m$,
- c, l'incrément, vérifiant $0 \leqslant c < m$,
- X_0, la graine, vérifiant $0 \leqslant X_0 < m$,

Python c'est facile

```python
def gcm(m,a,c,x):
    while True:
        yield x
        x = (a*x + c) % m
```

```
In [105]: gen = gcm(10,7,7,7)

In [106]: gen
Out[106]: <generator object gcm at 0x7fc4dfd56678>

# Sur Python 2.7 next est une méthode : gen.next()
In [107]: next(gen)
Out[107]: 7

In [108]: next(gen)
Out[108]: 6

In [109]: next(gen)
Out[109]: 9

In [110]: [next(gen) for k in range(20)]
Out[110]: [0, 7, 6, 9, 0, 7, 6, 9, 0, 7, 6, 9, 0, 7, 6, 9, 0, 7, 6, 9]

In [111]: [x for x in itertools.islice(gen,10)]
Out[111]: [0, 7, 6, 9, 0, 7, 6, 9, 0, 7]
```

Théorème 1

Le GCM est de période m si, et seulement si :

1. *c est premier avec m ;*
2. *$a-1$ est un multiple de tous les diviseurs premiers de m ;*
3. *$a-1$ est un multiple de 4 si m est un multiple de 4.*

Théorème 1

Le GCM est de période m si, et seulement si :

1. *c est premier avec m ;*
2. *$a - 1$ est un multiple de tous les diviseurs premiers de m ;*
3. *$a - 1$ est un multiple de 4 si m est un multiple de 4.*

Théorème 1

Le GCM est de période m si, et seulement si :

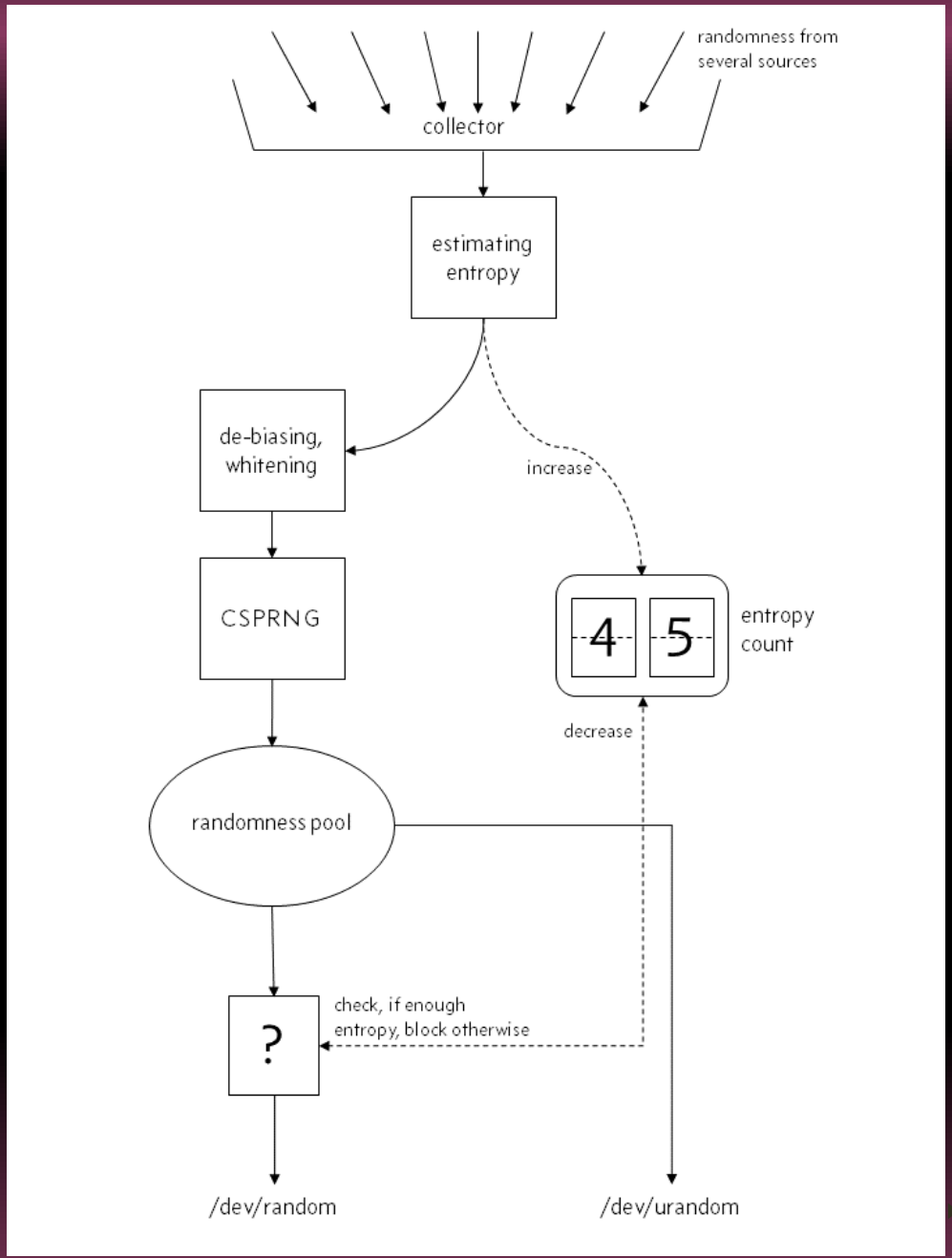
1. *c est premier avec m ;*
2. *$a - 1$ est un multiple de tous les diviseurs premiers de m ;*
3. *$a - 1$ est un multiple de 4 si m est un multiple de 4.*

Théorème 1

Le GCM est de période m si, et seulement si :

1. *c est premier avec m ;*
2. *$a - 1$ est un multiple de tous les diviseurs premiers de m ;*
3. *$a - 1$ est un multiple de 4 si m est un multiple de 4.*

- /dev/random
- /dev/urandom


randomness from
several sources
collector
estimating
entropy
de-biasing,
whitening
increase
CSPRNG
4
5
entropy
count
decrease
randomness pool
?
check, if enough
entropy, block otherwise
/dev/random
/dev/urandom

π

$3,14159\ 26\ 53589\ 79\ 32\ 38\ 46\ 26\ 43\ 38\ 32\ 79\ \ldots$

```
In [9]: ?Counter
Type:            type
String form:     <class 'collections.Counter'>
File:            /usr/local/lib/python3.4/collections/__init__.py
Init definition: Counter(self, iterable=None, **kwds)
Docstring:
Dict subclass for counting hashable items.  Sometimes called a bag
or multiset.  Elements are stored as dictionary keys and their counts
are stored as dictionary values.
```

```
stats1Pi = Counter(str(N(pi,10**5)))
```

```
In [8]: stats1Pi
Out[8]: Counter({'1': 10137, '6': 10028, '5': 10026, '3': 10026, '7':
        ↪  10025, '0': 9999, '8': 9978, '4': 9971, '2': 9908, '9':
        ↪  9902, '.': 1})
```

```
stats1Pi = Counter(str(N(pi,10**5)))
```

```
In [8]: stats1Pi
Out[8]: Counter({'1': 10137, '6': 10028, '5': 10026, '3': 10026, '7':
        ↪  10025, '0': 9999, '8': 9978, '4': 9971, '2': 9908, '9':
        ↪  9902, '.': 1})
```

```python
def parPaquets(iterable, n):
    """
      iter transforme un itérable en itérateur
     * n répète n fois la liste
     * iters découpe la liste iters en une suite d'arguments
     zip crée une liste de n-uplets à partir de n listes
    """
    iters = [iter(iterable)] * n
    return zip_longest(* iters)

stats2Pi = Counter( parPaquets( str( N(pi, 10**5) ), 2 ) ).values()
```

```
In [59]: stats2Pi
Out[59]: dict_values([506, 440, 483, 499, 523, 469, 522, 517, 556, 491,
        ↪  469, 511, 491, 538, 498, 489, 510, 488, 503, 509, 558, 536,
        ↪  509, 511, 494, 523, 505, 499, 504, 498, 474, 459, 518, 492,
        ↪  485, 494, 462, 507, 487, 491, 536, 558, 529, 499, 493, 537,
        ↪  549, 465, 522, 487, 472, 498, 517, 521, 482, 483, 482, 494,
        ↪  503, 498, 494, 452, 501, 490, 491, 512, 514, 470, 456, 484,
        ↪  515, 527, 479, 512, 502, 490, 473, 468, 530, 509, 484, 1,
        ↪  488, 513, 520, 556, 445, 491, 1, 505, 475, 494, 463, 463,
        ↪  468, 507, 501, 510, 551, 512, 534, 507])
```

Voici un code naïf :

```
def chercheMotif(motif,chaine):
    en = enumerate(chaine)
    deb = motif[0]
    n = len(motif)
    for i,k in en:
        if k == deb and chaine[i:i+n] == motif:
            return i
    return 'Pas trouvé'
```

```
In [10]: chercheMotif('007',str(N(pi,10**7)))
Out[10]: 2807
```

```
In [12]: chercheMotif('1234567',str(N(pi,10**7)))
Out[12]: 9470345
```

Voici un code naïf :

```
def chercheMotif(motif,chaine):
    en = enumerate(chaine)
    deb = motif[0]
    n = len(motif)
    for i,k in en:
        if k == deb and chaine[i:i+n] == motif:
            return i
    return 'Pas trouvé'
```

```
In [10]: chercheMotif('007',str(N(pi,10**7)))
Out[10]: 2807
```

```
In [12]: chercheMotif('1234567',str(N(pi,10**7)))
Out[12]: 9470345
```

Voici un code naïf :

```
def chercheMotif(motif,chaine):
    en = enumerate(chaine)
    deb = motif[0]
    n = len(motif)
    for i,k in en:
        if k == deb and chaine[i:i+n] == motif:
            return i
    return 'Pas trouvé'
```

```
In [10]: chercheMotif('007',str(N(pi,10**7)))
Out[10]: 2807
```

```
In [12]: chercheMotif('1234567',str(N(pi,10**7)))
Out[12]: 9470345
```