Rendu cryptographie

Stefan RADOVANOVIC - 21/10/2022

Paradoxe des anniversaires

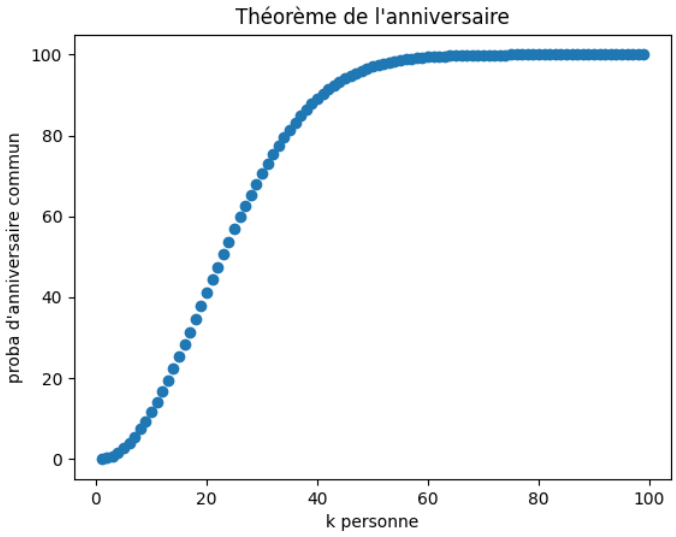
Le paradoxe des anniversaires se base sur le principe des tiroirs. C'est-à-dire que l'on va chercher à partir de combien de personne nous pouvons avoir une probabilité de 50% que deux personne du groupe soit nés le même jours, ou que deux objets partagent le même tiroir.

Pour le code, on va prendre le problème à l'envers et chercher quelle est la probabilité qu'aucune personne ne partage la même date de naissance (puis inverser la probabilité pour le graphe):

$$p(k) = (1 - \frac{1}{N})...(1 - \frac{k-1}{N})$$

```
def anniv_non_commun(k, N):
    probability = 1  # 100% au départ
    for i in range(1, k):
        probability *= 1 - (i / N)

    return probability
```



Graphe d'évolution de la probabilité de collision pour k entité selon N jours

Pour revenir au paradoxe, on constate qu'à partir de 23 personne, on à 50% de probabilité qu'il y est dans le groupe au moins deux personne nés le même jour. Et on approche les 100% à partir de 57 personne.

La preuve de travail

Le principale objectif d'une preuve de travail est de ralentir l'accès à un service. Une version moderne serait le captcha, son utilisation prouve que l'on est bien humain et permet également de bloquer les

spams, la preuve de travail est contournable mais c'est son temps d'exécution, qui parait insignifiant pour un réel humain, qui va décourager les spammeurs.

le puzzle

Le puzzle va obliger les utilisateurs à passer du temps sur un problème, on fait cela avec la fonction `Sha256` .

tel que :

$F(A, D, x) = True$

avec la concaténation : $A|D|x$

$A =$ String Aléatoire

$D =$ un entier représentant la difficulté du problème

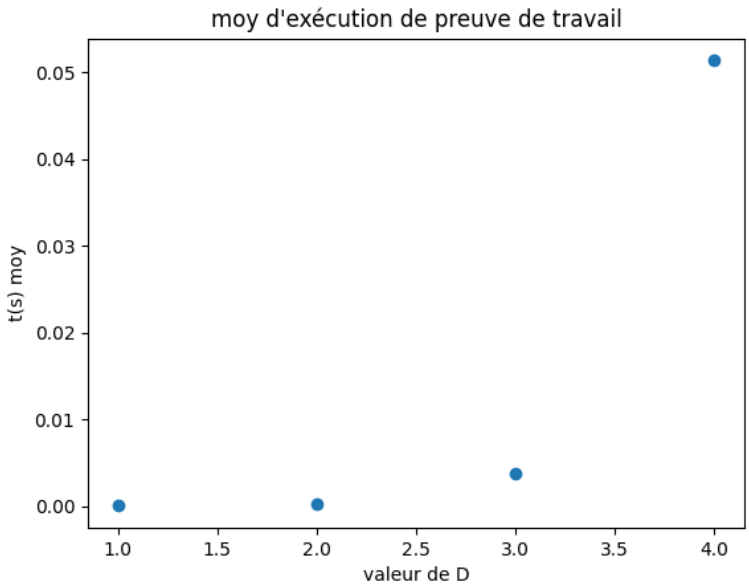
$x =$ variable

La seule façon de résoudre ce puzzle est le `brut force` , par exemple : "obtenez un hash qui commence avec 6 zéros", il faudra tester un grand nombre de x pour y arriver (on ne peut pas le prédire).

```
def proof_of_work(A, D):
    #  renvoi le temps passé à trouver une preuve de travail
    d_zero = ''.join('0' for n in range(D))
    x = 0
    start_time = time.time()

    while True:
        # concatenation A | D | x
        if hashlib.sha256((A + d_zero + str(x)).encode()).hexdigest().startswith(d_zero):
            break
        x += 1

    return time.time() - start_time
```



Graphe de l'évolution de la complexité en temps de la résolution d'une preuve de travail

Notre simulation affiche le temps moyen pour obtenir des preuves de travail avec une difficulté qui s’incrémente. On constate que la résolution d’une preuve de travail suit une loi exponentielle, plus la difficulté est grande plus le temps de résolution est grand. C’est pourquoi on dit que la résolution est sans mémoire.