Algorithmes et Pensée Computationnelle

Probabilistic Algorithms

Le but de cette séance est de comprendre les algorithmes probabilistes. Ceux-ci permettent de résoudre des problèmes complexes en relativement peu de temps. La contrepartie est que le résultat obtenu est généralement une solution approximative du problème initial. Néanmoins, ces algorithmes demeurent très utiles pour beaucoup d'applications.

1 Monte-Carlo

Question 1: (🕓 *10 minutes*) **Un jeu de hasard : Python**
Supposez que vous lanciez une pièce de monnaie **l** fois et que vous voulez calculez la probabilité d'avoir un certain nombre de piles. Vous devez programmer un algorithme probabiliste, permettant de calculer cette probabilité. Pour ce faire, vous devez compléter la fonction *proba(n,l,iter)* contenue dans le fichier **Piece.py** (Dans le dossier **Ressources**). La fonction *Piece(l)* permet de créer une liste contenant des 0 et des 1 aléatoirement avec une probabilité $\frac{1}{2}$. Considérez un chiffre 1 comme une réussite (pile) et 0 comme un échec (face).

Conseil

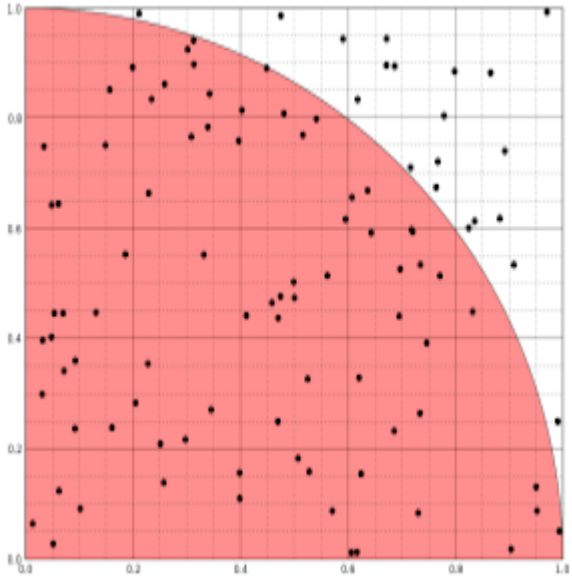
1. Pour estimer empiriquement la probabilité d'un événement, comptez le nombre de fois que l'événement en question se produit en effectuant un nombre d'essai. Puis divisez le nombre d'occurrence de l'événement par le nombre total d'essai. Par exemple, si vous voulez estimer la probabilité d'obtenir un 2 avec un dé. Lancez le dé 1000 fois, comptez le nombre de fois que vous obtenez 2, et divisez le résultat par 1000.
2. Pour générer des nombres pseudo-aléatoires en Python, vous pouvez utliser la méthode **random.random()** après avoir importé le module **random**.

Solution

```
1  import random
2
3
4  # La fonction Piece retourne une liste contenant des 0 et des 1, considérez un 1 comme un succès, i.e. une fois où
       la pièce tombe sur pile, et 0 comme un échec
5  def Piece(l):
6      return [random.randint(0, 1) for i in range(l)]
7
8
9  def proba(n, l, iter=10000):
10     # n correspond au nombre de succès et l au nombre d'essais. Iter correspond au nombre d'expérience que vous
           allez
11     # réaliser pour obtenir la réponse. Cela devrait être grand mais pas trop (sinon le programme prendra trop de
12     # temps à s'exécuter). 10000 est un bon nombre d'itérations.
13     proba = 0
14     for i in range(iter):
15         temp = Piece(l)  # On simule une expérience de l lancés.
16         count = sum(temp)  # On compte le nombre de fois que l'on obtient pile
17         if count == n:  # Si le nombre de pile obtenu correspond à la probabilité que l'on veut estimer
18             proba += 1  # On ajoute 1 à notre estimateur de probabilité
19     return proba / iter  # Divise notre estimateur de probabilité par le nombre total d'expériences réalisées.
20
21
22 n = 5
23 l = 10
24 print("La probabilité d'avoir {} pile en {} lancés de pièce est approximativement égale à {}".format(n, l,
       proba(n, l,10000)))
```

Question 2: (🕓 *20 minutes*) **Une approximation de π : Python**

L'objectif de cet exercice est de programmer un algorithme probabiliste permettant d'approximer le chiffre π. Imaginez un plan sur lequel $0 < x < 1$ et $0 < y < 1$. Sur ce dernier, nous allons dessiner un quart de cercle centré en (0,0) et avec un rayon de 1. Par conséquent, un point dans cet espace se trouve à l'intérieur du cercle si $x^2 + y^2 < 1$. Vous trouverez ci-dessous une illustration de la situation :



La première étape de cet exercice consiste à créer une fonction permettant de déterminer si un point est à l'intérieur (zone rouge) ou à l'extérieur du cercle. Puis, générez 10000 points dans cet espace (x et y devrait appartenir à l'intervalle [0,1]). Pour ce faire, vous pouvez utliser la fonction *random.random()* après avoir importé le module **random**. Vous pouvez obtenir l'approximation de π à partir de la formule suivante : $\pi \approx [\frac{\text{Nombre de points dans le cercle}}{\text{Nombre total de points}}] \cdot 4$. Votre réponse devrait être assez proche du vrai chiffre π.

Conseil

La fonction **random.random()** génère aléatoirement un chiffre compris entre 0 et 1. Etant donné que vous devez simuler des points en 2 dimensions, vous devrez utiliser 2 fois cette fonction.

Solution

```
1  import random
2
3  def inside(point):#Point définit sous la forme d'un tuple
4      # Cette fonction permet de vérifier si un point se trouve à l'intérieur du cercle
5      if (point[0]**2+point[1]**2) < 1:
6          return 1
7
8      else:
9          return 0
10
11 def app():
12     count = 0 #On initialise le nombre de points dans le cercle
13     for i in range(10000):
14         temp1 = random.random()#Génére la première coordonnée
15         temp2 = random.random()#Génère la deuxième coordonnée
16         temp = [temp1,temp2]#Crée le point
17
18         count += inside(temp)#On appelle la fonction. Si le point est dans le cercle, elle retourne 1, par conséquent
             on ajoute 1 au compteur. Sinon elle retourne 0, on ajoute donc rien.
19
20     return count/10000*4#Retourne selon la formule donnée dans l'exercice.
21
22 print("L'approximation du chiffre pi est : {}".format(app()))
```

2 Fingerprinting

Question 3: (*20 minutes*) **Fingerprinting : Une mission pour l'agente secrète Alice : Python**
Dans cet exercice, vous prendrez le rôle de l'agente secrète Alice. Cette dernière enquêtait sur la disparition de son collègue, l'agent Bob, et se doutait que l'indice clé qui la mènerait à la vérité se trouvait dans la boîte mail de Bob. Alice arriva à trouver un bout de papier avec écrit dessus : "*Mon mot de passe est l'empreinte de ceciestmonmotdepasse*". Aidez Alice à trouver l'empreinte du mot de passe !

Pour cela, vous devez compléter deux fonctions :

1. **is_a_prime_number(num)** qui vérifie que **num** est un nombre premier ou pas. Un nombre premier est un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs. Ces deux diviseurs sont 1 et le nombre considéré, puisque tout nombre a pour diviseurs 1 et lui-même, les nombres premiers étant ceux qui n'en possèdent aucun autre.
2. **fingerprinting(p, message)** qui implémente l'algorithme de fingerprinting suivant :
   (a) Si **p** est un nombre premier, calculez la valeur de hachage de la chaîne à l'aide de la fonction **hash(...)**, puis calculez le modulo du résultat du hachage.
   (b) Sinon, imprimez un message qui dit que le nombre n'est pas un nombre premier.

Si vous réussissez à implémenter les deux fonctions correctement, le code vous imprimera : **Connection réussie? True**.
À vos ordis, détectives !

```
import base64

# num est un nombre entier
def is_a_prime_number(num):
    # Partie à compléter


# p est un nombre premier et message est une chaine de caractères
def fingerprinting(p, message):
    # Partie à compléter


# password est une chaine de caractères et your_details est un tuple avec le
# format suivant (nombre premier, hash du mot de passe)
def login(password, your_details):
    return your_details[1] % your_details[0] == fingerprinting(your_details[0], password)



# Début de votre programme
password = "ceciestmonmotdepasse"
your_details = (19, hash(password))
success = login(password, your_details)

print("Connection réussie? " + str(success))
if success:
    message = '''SmUgc2VyYWlzIGNvbmZpbsOpIGNoZXogbWVzIHBhcmVudHMgw
                6AgbGEgY2FtcGFnbmUgbGVzIGRldXggcHJvY2hhaW5lIHNlbWF
                pbmVzLCBldCBqZSBuJ2F1cmFpcyBwYXMgYWNjw6hzIMOgIEludGV
                ybmV0LiDDgCBiaWVudMO0dCE='''
    print(base64.b64decode(message).decode())
```

> **Conseil**
>
> Pour vérifier si un nombre n est premier, il faut parcourir tous les nombres à partir de 2 à $n/2$ et vérifier si chaque nombre divise n. Si un nombre qui divise n est trouvé, il faut retourner **False**. Si aucun diviseur est trouvé alors cela signifie que n est premier et il faut retourner **True**.

Solution

```python
import base64

# num est un nombre entier
def is_a_prime_number(num):
    if num <= 1:
        return False
    for i in range(2, int(num**.5)):
        if num % i == 0:
            return False
    return True

# p est un nombre premier et message est une chaine de caractères
def fingerprinting(p, message):
    if is_a_prime_number(p):
        result = hash(message) % p
        return result
    print(str(p) + " is not a prime number!")

# password est une chaine de caractères et your details est un tuple avec le
# format suivant (nombre premier, hash du mot de passe)
def login(password, your_details):
    return your_details[1] % your_details[0] == fingerprinting(your_details[0], password)

if __name__ == "__main__":
    password = "ceciestmonmotdepasse"
    your_details = (19, hash(password))
    success = login(password, your_details)

    print("Connection réussie? " + str(success))
    if success:
        message = '''SmUgc2VyYWlzIGNvbmZpbsOpIGNoZXogbWVzIHBhcmVudHMgw
                6AgbGEgY2FtcGFnbmUgbGVzIGRldXggcHJvY2hhaW5lIHNlbWF
                pbmVzLCBldCBqZSBuJ2F1cmFpcyBwYXMgYWNjw6hzIMOgIEludGV
                ybmV0LiDDgCBiaWVudMO0dCE='''
        print(base64.b64decode(message).decode())
```

3 Las Vegas

Question 4: (*10 minutes*) **Un point dans un cercle unitaire : Python**
Les algorithmes de Monte Carlo sont des algorithmes probabilistes dont la sortie peut être incorrecte avec une certaine probabilité, qui est généralement faible. En revanche, un algorithme de Las Vegas est un algorithme probabiliste qui trouve toujours le bon résultat lorsqu'il existe. Son inconvénient est que sa complexité temporelle ne peut être garantie à l'avance car elle dépend des données passées en paramètres.

L'objectif de cet exercice est de programmer un algorithme probabiliste permettant de donner un point contenu dans un cercle unitaire.

Conseil

Vous pouvez vous inspirer de l'exercice 2 *(Approximation de π)*.

Solution

```
1   import random
2
3
4   def inside(point):  # Point définit sous la forme d'un tuple
5       # Cette fonction permet de vérifier si un point se trouve à l'intérieur du cercle
6       if (point[0] ** 2 + point[1] ** 2) < 1:
7           return 1
8
9       else:
10          return 0
11
12
13  def app():
14      count = 0  # On initialise le nombre de points dans le cercle
15      for i in range(10000):
16          temp1 = random.random()  # Génére la première coordonnée
17          temp2 = random.random()  # Génère la deuxième coordonnée
18          temp = [temp1, temp2]  # Crée le point
19
20          if (inside(temp)) :
21              return temp # Retourne le point trouvé.
22
23
24  print("Voilà un point dans un cercle unitaire  : {}".format(app()))
```

Question 5: (*10 minutes*) **Quicksort - Algorithme de Las Vegas : Python**
Dans cet exercice, vous allez implémenter un algorithme de tri rapide (quicksort) sur une liste d'éléments avec l'algorithme de Las Vegas.

L'algorithme de tri rapide applique un paradigme *divide-and-conquer* afin de trier un ensemble de nombres A. Il fonctionne en trois étapes :

1. il choisit d'abord un élément pivot, A[q], en utilisant un générateur de nombres aléatoires (d'où sa nature d'algorithme dit probabiliste) ;
2. puis il réorganise le tableau en deux sous-tableaux $A[p...q-1]$ et $A[q+1...r]$, où les éléments des premier et deuxième tableaux sont respectivement plus petits et plus grands que $A[q]$.
3. L'algorithme applique ensuite récursivement les étapes de tri rapide ci-dessus sur les deux tableaux indépendants, produisant ainsi un tableau entièrement trié.

Complétez le code suivant :

```
1   import random
2
3   # lst représente la liste à trier, l l'index 0 et r la taille de la liste -1
4   def sort(lst, l, r):
5       # mettre une condition pour arrêter la récursivité
6
```

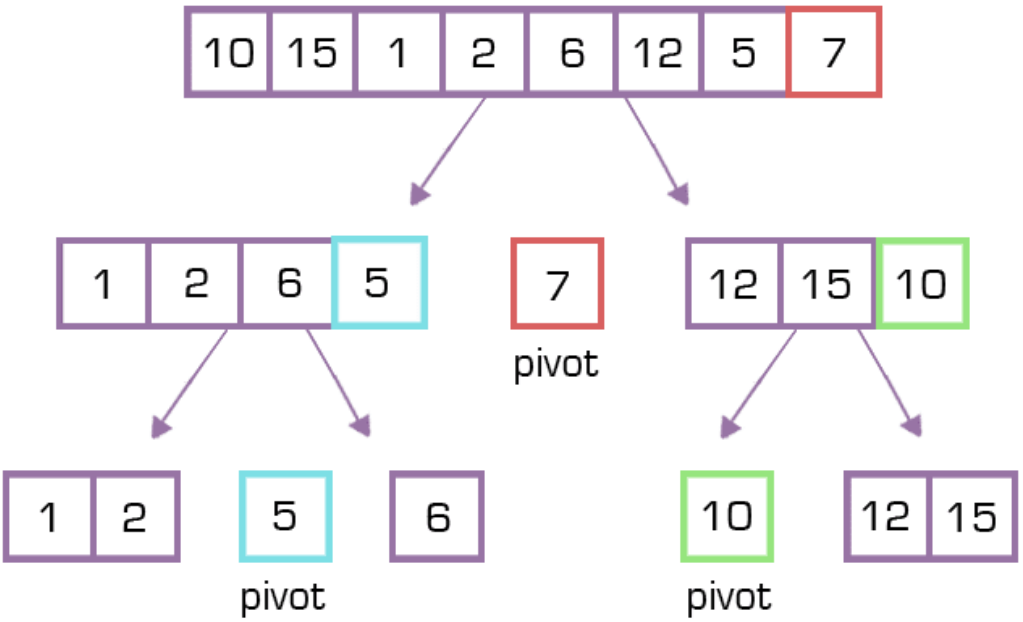


FIGURE 1 – Illustration de l'algorithme de tri rapide

```
7
8       pivot_index = ... # Partie à compléter: Choisissez un pivot compris entre 0 et la longueur de votre liste - 1
9
10      # Déplacer votre pivot dans votre liste
11
12      # Partitionnez votre liste de telle sorte que les éléments plus petits que le pivot soient placés avant celui-ci et les é
          léments plus grands soient placés après
13
14      # Replacer votre pivot à l'endroit adéquat
15
16      # Effectuez le tri de fa con récursive sur les parties gauches et droites de la liste
17
18  def quicksort(items):
19      if items is None or len(items) < 2:
20          return
21      sort(items, 0, len(items) - 1)
22
23  l = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
24  quicksort(l)
25  print('Liste triée:   ', l)
```

Conseil

Pour le choix de votre élément pivot, pensez à utiliser la méthode **randint()** de la librairie **random**.

Solution

```
1  import random
2
3  # lst représente la liste à trier, l l'index 0 et r la taille de la liste -1
4  def sort(lst, l, r):
5      # Dans le meilleur des cas, on arrête la récursivité
6      if r <= l:
7          return
8
9      # Choix du pivot
10     pivot_index = random.randint(l, r)
11
12     # On déplace le pivot au premier élément
13     lst[l], lst[pivot_index] = lst[pivot_index], lst[l]
14
15     # partition
16     i = l
17     for j in range(l+1, r+1):
18         if lst[j] < lst[l]:
19             i += 1
20             lst[i], lst[j] = lst[j], lst[i]
21
22     # On place le pivot à la position adéquate
23     lst[i], lst[l] = lst[l], lst[i]
24
25     # On effectue le tri de fa con récursive sur les parties gauches et droites de la liste
26     sort(lst, l, i-1)
27     sort(lst, i+1, r)
28
29 def quicksort(items):
30     if items is None or len(items) < 2:
31         return
32     sort(items, 0, len(items) - 1)
33
34 l = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
35 quicksort(l)
36 print('Liste triée:   ', l)
```