IEFS Informatique et Société Numérique 2
FC, AS, EUR, AMER
Juin 2023

Durée: 1h30
Documents et calculatrice interdits

Un programme **non documenté**, ou avec de **mauvais choix de noms de variables** sera **sanctionné**. Mais, il est inutile de recopier les DocStrings fournies.

Exercice 1 Questions de cours (2pts)

(Q1.1) Qu'est-ce que le DNS ? (2 lignes maximum - 0,5pt) Correction:

Domain Name Service. Annuaire distribué permettant de trouver l'IP associée à un nom de domaine.

(Q1.2) Que représente `127.0.0.1` *? (2 lignes maximum - 0,5pt)* Correction:

C'est l'adresse IP du localhost (l'ordinateur local). Tous les ordinateurs disposent de cette IP.

(Q1.3) Les concepts ou expressions suivants ont été présentés ou évoqués en cours pour traiter du modèle économique du Web et de ses implications sociales. Choisissez ***un*** *concept* ***ou une*** *expression, puis donnez en la définition et expliquez son lien avec l'économie du Web en 5 lignes maximum. (1pt)*

- Marché biface / marché à deux versants
- Inaction écologique
- Plateforme numérique structurante
- L'engagement des abonnés ou utilisateurs
- Désintermédiation de la chaine de valeur
- Oligopole

Exercice 2 Choix algorithmiques (2pts)

Nous disposons de la liste triée par ordre croissant des âges de la population française (listes de 70 millions d'entiers, avec beaucoup d'éléments égaux). On veut connaître le nombre de personnes ayant un age donné.

(Q2.1) Quel ou quels algorithmes sont efficaces dans cette situation et seraient à privilégier pour résoudre ce problème ? Décrivez en français, *en* quelque mots, *un* algorithme *général que vous pourriez mettre en œuvre pour résoudre efficacement ce problème.*

Correction:
Pour faire une recherche dans une liste triée la dichotomie est l'algo le plus efficace que nous ayons vu en cours.
Exemple de solution mettant en œuvre cet algo :

- Par dichotomie, rechercher le premier indice, puis le dernier indice de l'age cherché.
- Soustraire les 2 indices.

Exercice 3 Codage et décodage de texte (6pts)

Nous voulons coder un texte sous la forme d'une liste de motifs, comme illustré dans l'exemple ci-dessous. On y voit que la liste `txt` décrit un texte sous la forme d'une liste de numéros de motifs (`samples`). Dans cet exemple `txt` encode un texte commençant par "Bien le bonjour !".

Dans la suite, nous vous guidons pour écrire des fonctions qui décodent ou encodent un texte à partir d'une liste de motifs donnée.

```python
samples = ["Bien ", "jour !", "ont", "bon", "le", "s", " "]      # Liste de motifs
txt = [0, 4, 6, 3, 1, 6, 4, 5, 6, 3, 3, 5, 6, 5, 2, 6, 3, 5]     # Texte encodé
# Début du message encodé dans txt : "Bien le bonjour ! (...)"
# Détail :
#    samples[txt[0]] -> samples[0] -> "Bien "
#    samples[txt[1]] -> samples[4] -> "le"
#    samples[txt[2]] -> samples[6] -> " "
#    samples[txt[3]] -> samples[3] -> "bon"
#    samples[txt[4]] -> samples[1] -> "jour !"
```

Hypothèse : dans tout cet exercice, nous ferons toujours l'hypothèse que la liste des motifs est suffisante par rapport au texte à décoder ou à encoder, et qu'elle est triée par ordre décroissant de longueur de motifs.

Partie 1: Décodage (2pts)

(Q3.1) Terminez de décoder à la main le texte de l'exemple. Écrivez le texte en clair. (1pts)

Correction:
« Bien le bonjour ! les bonbons sont bons »

(Q3.2) Écrivez la fonction python `decode(txt, samples)` *spécifiée ci-dessous (1pt)*

```python
def decode(t, samples) :
    """Prend en paramètre un texte encodé t (une liste de numéros de motifs, ex : txt)
       et une liste de motifs (Une liste de textes, ex : samples dans l'exemple précédent)
       Renvoie le texte décodé sous forme de chaîne de caractères.
    """
```

Correction:

```python
def decode(txt, samples) :
    res = ""
    for v in txt :
        res += samples[v]
    return res
```

Partie 2: Encodage (4pts)

Nous voulons écrire une fonction `encode(texte, samples)` qui encode le texte pris en paramètre, en utilisant les motifs `samples`. Voici un exemple d'appel voulu :

```python
samples = ["abcdefg", "efgh", "bcd", "a","b","c","d","e","f","g","h"]
txt="efghabcdbbcd"
print(encode(txt, samples))  # Affiche : [1, 3, 2, 4, 2]
```

Pour ce faire, il peut-être utile de passer par des fonctions intermédiaires. Vous êtes libres de vos choix, mais voici 2 fonctions pouvant vous inspirer **(et que vous devrez écrire vous-même si vous voulez les utiliser)** :

(Q3.3) Écrivez une fonction python `encode(texte, samples)` *qui prend en paramètre un* `texte` *en clair et une liste de motifs* `samples` *respectant les contraintes décrites ci-dessus, et qui renvoie une liste d'entiers encodant le* `texte` *en utilisant les motifs* `samples`*.*

Correction:

```python
def est_le_bon_motif(t1, start, t2) :
    """Prend en paramètre un texte t1, un indice start dans ce texte et un second texte t2.
       Retourne un booléen. Il vaut vrai ssi t2 est présent dans t1 en position start.
    """
```

```
def est_le_bon_motif(t1, start, t2) :
    """Prend en paramètre un texte t1, un indice start dans ce texte et un motif t2.
       Retourne un booléen. Il vaut vrai ssi t2 est présent dans t1 en position start.
    """

def get_num_motif(txt, start, samples) :
    """Prend en paramètre un texte txt, un indice start de ce texte et une liste de motifs.
       Retourne le numéro du motif (son indice dans samples) le plus long commençant à
       start dans le texte txt.
       Hypothèse : samples est trié par longueurs décroissantes de motifs et contient
                   forcément un motif qui correspond.
    """
```

```
    res = False
    if len(t2)+start <= len(t1) :
        i = 0
        while i<len(t2) and t1[start+i] == t2[i] :
            i+=1
        if i==len(t2) :
            res = True

    return res

def get_num_motif(t, start, samples) :
    """Retourne le numéro de samples le plus long commençant à start dans le texte t.
       Hypothèse : samples est triés par longueurs décroissantes de textes."""
    i = 0
    while i<len(samples) and not est_le_bon_motif(t, start, samples[i]):
        i += 1

    return i

def encode(texte, samples) :
    """Hypothèse : la liste samples est triée (du motifs le plus long au motif le plus court) et permet
       nécessairement d'encoder le texte "texte" (tous les caractères uniques sont présents en fin de
       liste).
       renvoie une liste de valeurs numérique encodant le texte reçu en paramètre en utilisant le
       dictionnaire samples.
    """
    res =[]
    i = 0
    while i<len(texte) :
        num_sample = get_num_motif(texte, i, samples)
        res.append(num_sample)
        i = i+len(samples[num_sample])

    return res
```

Exercice 4 Récursivité (10pts)

Partie 1: Question de cours (3pts)

(Q4.1) Écrivez une fonction python ***récursive*** *de recherche dichotomique d'un élément dans une liste triée.*

```
def dichotomie(valeur, liste, debut, fin):
    """
       Entrées :
                valeur, un entier
                liste, une liste d'entiers, triée par ordre croissant
                debut, fin : les indices de début et fin de recherche (fin est inclus)
       Sortie :
                True si valeur est dans liste entre les indices debut et fin, False sinon
    """
```

Correction:

```
def dichotomie(valeur, liste, debut, fin):
        if debut > fin :
                resultat = False
        else :
                milieu = (debut+fin)//2
                if valeur==liste[milieu]:
                        resultat = True
                elif valeur < liste[milieu]:
                        resultat = dichotomie(valeur, liste, debut, milieu-1)
                else:
                        resultat = dichotomie(valeur, liste, milieu + 1, fin)
        return resultat
```

Partie 2: Arbres binaires (4pts)

Dans cet exercice, on va définir la notion d'**arbre binaire** contenant des entiers.

Un arbre est une structure telle qu'illustrée dans la Figure 1 : chaque *cercle* est appelé un **sommet**. Il contient une valeur. Ici on se restreindra à des entiers. Un sommet particulier (tout en haut) s'appelle la **racine** de l'arbre. Les flèches indiquent des **sous-arbres**. Dans le cas des arbres binaires (arbres ayant 2 sous-arbres par sommet), on parlera du **sous-arbre de gauche** et du **sous-arbre de droite**. Si on regarde, par exemple, le sous-arbre de gauche de la racine, c'est aussi un arbre, avec une racine et des sous-arbres. Il s'agit d'une structure récursive.

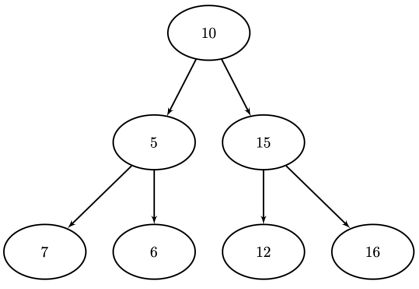

FIGURE 1 – Exemple d'arbre binaire

On définit donc un arbre binaire de manière récursive comme suit : un arbre binaire AB est soit vide, soit un triplet constitué d'une valeur, d'un sous-arbre de gauche et d'un sous-arbre de droite. Ces deux sous-arbres sont des arbres binaires. Mathématiquement, cela donne : $AB := \begin{cases} \emptyset \\ x, AB_{gauche}, AB_{droit} \end{cases}$

Dans ce sujet, un arbre binaire est représenté en python comme une liste. Elle est soit vide (de longueur 0), soit de longueur 3 : le premier élément de la liste est la valeur du sommet, le deuxième élément est le sous-arbre de gauche, le troisième le sous-arbre de droite. L'exemple ci-dessus est ainsi implémenté comme suit.

```
ex_arbre = [10,
                [5,
                    [7, [], []],
                    [6, [], []]
                ],
                [15,
                    [12,  [], []],
                    [16, [], []]
                ]
            ]
```

*(Q4.2) Écrivez **en français** la fonction **récursive** qui vérifie si une valeur `val` est contenue dans un arbre binaire `t`. (2pts)* Correction:

def is_in_tree(val, t) :
  Si t est vide, alors val n'est pas présente
  Sinon,
    si la valeur de t est val, alors val est présente
    Sinon,
      Si val est présente dans le sous arbre gauche ou droit, c'est que val est présente
      Sinon,
        val n'est pas présente

(Q4.3) Écrivez la fonction python correspondante (2pts)

```
def is_in_tree(val, t):
    """
        Test si val est dans t
        Entrée :
                        val un entier
                        t une liste représentant un arbre binaire
        Sortie : True si valeur est dans t, False sinon
    """
```

Correction:

```
def is_in_tree(valeur,t):
    if len(t)==0:
        resultat = False
    elif valeur == t[0]: # trouvé !
            resultat = True
    else: #cas général
            resultat = is_in_tree(valeur,t[1]) or  is_in_tree(valeur,t[2])
    return resultat
```

Partie 3: Arbre binaire de recherche (3pts)

Un arbre binaire de recherche (ABR) est un arbre binaire respectant trois propriétés.

1. tous les éléments du sous-arbre de gauche d'un arbre sont inférieurs ou égaux à la racine
2. tous les éléments du sous-arbre de droite sont strictement supérieurs à la racine
3. les sous-arbres de gauche et de droite sont des ABR

En particulier un arbre vide est un ABR.

(Q4.4) L'arbre binaire de la Figure 1 n'est pas un ABR. Expliquez pourquoi. (1pt)

Correction:
Le noeud de valeur 7 est dans le sous-arbre gauche du sommet de valeur 5, ce qui est interdit.
À l'inverse, l'arbre représenté dans la Figure 2 est un ABR.

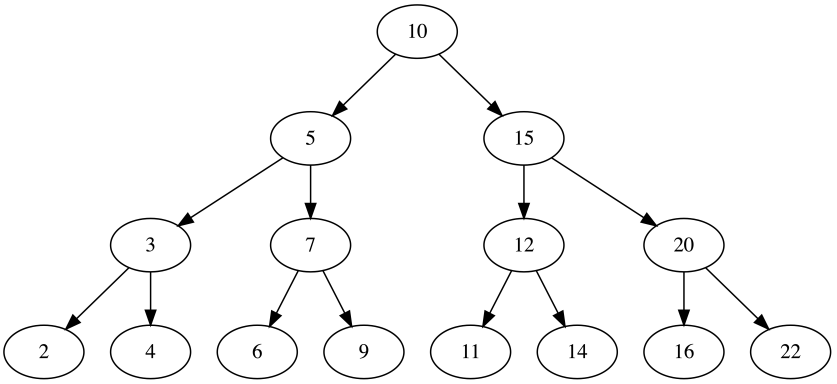

FIGURE 2 – Exemple d'arbre binaire de recherche

(Q4.5) Expliquez en quelques mots pourquoi on trouve plus vite la valeur dans un ABR que dans un arbre binaire normal *(1pt).*

Correction:
Vu que tous les éléments de gauche sont plus petits que les éléments de droite on peut faire une sorte de dichotomie et diviser par deux la taille de l'arbre à chaque récursion

(Q4.6) Écrivez une fonction récursive qui prend en paramètre une valeur et un arbre binaire de recherche et renvoie True si cette valeur est dans l'arbre, False sinon. (1pts)

```
1   def is_in_abr(valeur, abr):
2     """
3     Entree :
4       valeur, un entier
5       abr un arbre binaire
6     Sortie : True si valeur est dans abr, False sinon
7     """
```

Correction:

```
1  def is_in_abr(x,t):
2      if len(t)==0:
3          resultat = False
4      elif x == t[0]: # trouvé !
5              resultat = True
6      else: #cas général
7          if x < t[0]: # il ne peut être qu'à gauche
8              resultat = is_in_tree(x,t[1])
9          else : # il ne peut être qu'à droite
10             resultat = is_in_tree(x,t[2])
11     return resultat
```