TP INFO — Révisions sur la récursivité et les dictionnaires

Partie I Révisions - Récursivités

Rappelons qu'une fonction récursive est une fonction qui s'appelle elle-même dans sa définition. Deux éléments sont indispensables à la terminaison d'une fonction récursive :

- il est nécessaire qu'il y ait une condition d'arrêt ;
- il ne doit pas y avoir de suite infinie d'appels récursifs.

I.A - Figures alphanumériques

Dans cette partie, on écrit quelques fonctions récursives permettant d'afficher à l'écran des figures composées de caractères.

Question 1 : Écrire une fonction récursive `triangle1(n:int) -> None` qui prend en argument un entier naturel `n` et affiche un triangle isocèle « croissant » de hauteur `n` composé de `#`. Voici un exemple ci-dessous.

```
>>> triangle1(4)
#
##
###
####
```

Question 2 : Écrire une fonction récursive `triangle2(n:int) -> None` qui prend en argument un entier naturel `n` et affiche un triangle isocèle « décroissant » de hauteur `n` composé de `#`. Voici un exemple ci-dessous.

```
>>> triangle2(4)
####
###
##
#
```

Pour terminer, on souhaite généraliser à l'aide de la récursivité la célèbre relation $0 = (0+0)$. Par exemple, les premières relations sont les suivantes.

$$0 = (0+0), \quad 0 = ((0+0)+(0+0)), \quad 0 = (((0+0)+(0+0))+((0+0)+(0+0))).$$

Question 3 : Écrire une fonction récursive `toto(n:int) -> None` qui prend un argument un entier naturel non nul `n` et affiche la `n`-ième relation décrite ci-dessus.

I.B - Renverser une chaîne de caractères

Question 4 : Écrire une fonction récursive `renverser(chaine:str) -> str` qui prend en argument une chaîne de caractères `chaine` et renvoie une version renversée de la chaîne de caractères. Par exemple, l'exécution de `renverser("abcd")` doit renvoyer `"dcba"`.

I.C - Représentation binaire d'un entier positif

On rappelle que tout entier positif s'écrit comme une somme de puissances de 2. Par exemple, on a

$$11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = [1,0,1,1]_2.$$

Question 5 : Écrire une fonction récursive `binaire(n:int) -> list` qui prend en argument un entier positif `n` et renvoie la liste des chiffres de sa représentation binaire.

I.D - Recherche dichotomique dans un tableau trié

L'algorithme de recherche dichotomique permet de savoir si un élément `x` appartient à un tableau `t` d'entiers triés dans l'ordre croissant. Rappelons succinctement son principe. On note c un indice pointant (approximativement) sur un élément au milieu du tableau, puis on distingue deux cas :

- si `x < t[c]`, alors on poursuit la recherche dans la première partie du tableau ;
- sinon on poursuit la recherche dans la seconde partie du tableau.

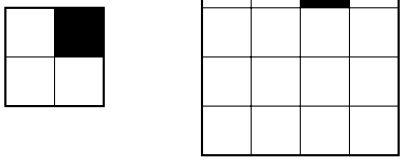
La recherche s'arrêtera et pourra renvoyer une réponse lorsqu'on tombera sur un tableau à un élément.

Question 6 : Écrire une fonction récursive `recherche_dicho(t:tuple, x:int) -> bool` prenant en argument un tuple d'entiers rangés dans l'ordre croissant et implémentant l'algorithme décrit ci-dessus.

Question 7 : Évaluer la complexité temporelle de votre fonction `dicho` en fonction de la taille de la liste `tab`.

I.E - Pavage d'un carré

Dans ce problème, on considère un carré quadrillé par un nombre de lignes et de colonnes qui est une puissance de 2. On choisit arbitrairement une case du quadrillage que l'on noircit. Voici deux exemples.



On souhaite démontrer à l'aide d'un algorithme que l'on peut toujours recouvrir les cases blanches d'un tel carré avec des pièces composées de trois cases en forme de L. Par exemple, les deux carrés ci-dessus se recouvrent de la manière suivante.

1	
1	1

3	3		4
3	2	4	4
1	2	2	5
1	1	5	5

Question 8 : Proposer un algorithme récursif permettant de recouvrir un carré comme décrit ci-dessus par des pièces en L. On ne demande pas d'écrire un code Python.

Partie II Révisions - Dictionnaires

Un dictionnaire présente de nombreuses similitudes avec les listes, si ce n'est qu'au lieu d'accéder aux éléments individuels par le biais d'un indice, on y accède par le biais d'une clé. Les clés du dictionnaire doivent être des objets immuables : `str`, `int`, `tuple`, `float`,...

La création d'un dictionnaire se réalise en suivant la syntaxe `{c1: v1, ... , cn: vn}` où $c_1,\dots,c_n$ sont des clés (nécessairement deux à deux distinctes) et $v_1,\dots,v_n$ les valeurs qui leur sont associées.

```
# Création d'un dictionnaire vide
D = {}

# Création d'un dictionnaire avec quatre paires clé/valeur
D = {'cle1': 'valeur1', -8: [1,3], (3,5):'ptet', 0.5: 12}
```

Si `D` est un dictionnaire et `c` est une clé, alors

- l'expression `D[c] = v` crée une nouvelle association si la clé n'est pas présente dans le dictionnaire et modifie l'association précédente sinon ;
- l'expression `del D[c]` supprime une association si la clé est présente dans le dictionnaire et déclenche l'exception `KeyError` sinon ;
- l'expression `c in D` renvoie un booléen indiquant si la clé est présente ou non dans le dictionnaire ;
- l'expression `D[c]` renvoie la valeur associée à la clé si celle-ci est présente dans le dictionnaire et déclenche l'exception `KeyError` sinon.

Remarque 1 : On considère que les quatre opérations de base décrites ci-dessus se réalisent avec une complexité temporelle constante (même si ce n'est pas tout à fait exact en réalité).

II.A - Nombre d'occurrences d'un mot dans un texte

Dans cette partie, on ne considère que des chaînes de caractères contenant uniquement des caractères alphanumériques et des espaces. On pourra également utiliser la méthode `lower()` de la classe `str`, dont on pourra trouver le fonctionnement avec la commande `help(str.lower)`.

Question 9 : Écrire une fonction `car_max(chaine:str) -> str` qui renvoie un caractère ayant le nombre maximal d'occurrences dans la chaîne de caractères `chaine`. Par exemple, l'exécution de `car_max("ananas")` renvoie le caractère `"a"`. On pourra utiliser un dictionnaire pour compter le nombre d'occurrences de chaque caractère.

Question 10 : Utiliser votre fonction pour déterminer le mot avec le nombre maximal d'occurrences dans le texte se trouvant dans le fichier `Zola_Jaccuse.txt`. On commencera par ouvrir ce fichier avec Python pour sauvegarder son contenu dans une chaîne de caractères.

II.B - Nombre d'éléments distincts dans un tableau

Dans cette partie, on souhaite écrire une fonction déterminant efficacement le nombre d'éléments distincts dans un tableau d'entiers ou de chaînes de caractères.

Question 11 : Écrire une fonction `nb_elements(lst:list) -> int` qui prend en argument une liste d'entiers `lst` et renvoie le nombre d'éléments distincts dans la liste `lst`. Par exemple, si `lst=[1, 0, 0, 1, 2, 1, 1, 2]`, l'exécution de `nb_elements(lst)` renvoie `3`. On pourra utiliser un dictionnaire mémorisant les éléments déjà rencontrés.

Question 12 : Évaluer la complexité temporelle de la fonction `nb_elements` en fonction de la taille de la liste `lst`.

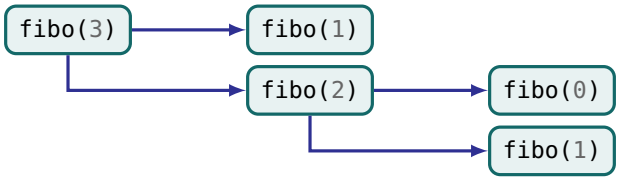
Question 13 : Quel est l'intérêt d'utiliser un dictionnaire à la place d'une liste pour mémoriser les éléments déjà rencontrés ?

II.C - Un premier pas vers la programmation dynamique

La simplicité de rédaction d'un algorithme récursif peut cacher une complexité non satisfaisante. Dans cette partie, on s'intéresse à la suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour tout $n \in \mathbb{N}$.

Question 14 : Écrire une fonction récursive `fibo(n:int) -> int` basée directement sur la définition ci-dessus qui prend en argument un entier naturel `n` et renvoie le nombre F_n.

Si l'on souhaite calculer le nombre F_3 avec notre fonction `fibo`, on peut représenter les appels successifs de la fonction `fibo` sous la forme d'un arbre comme ci-dessous.



Question 15 : Représenter l'arbre associé à l'exécution de `fibo(5)`.

Question 16 : Combien d'appels à la fonction `fibo` ont été effectués pour calculer F_5 ?

Si on note a_n le nombre d'appels à la fonction `fibo` pour calculer F_n pour tout $n \in \mathbb{N}$, on peut démontrer que

$$a_n \underset{n\to+\infty}{\sim} \frac{2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}.$$

On en déduit que la complexité temporelle de la fonction `fibo` est exponentielle.

Les arbres précédents nous permettent d'observer l'origine du problème : l'ordinateur recalcule plusieurs fois les mêmes valeurs. Pour contourner cette difficulté, nous allons utiliser une nouvelle méthode : la **mémoïsation**. Cette dernière consiste à mémoriser l'ensemble des résultats intermédiaires afin que l'ordinateur ne refasse pas plusieurs fois des calculs identiques.

Question 17 : En utilisant un dictionnaire pour mémoriser les résultats intermédiaires, écrire une fonction récursive `fibo_dico(n:int) -> int` effectuant plus efficacement la même tâche que `fibo`.

Remarque 2 : On pourra vérifier que $F_{100} = 354\,224\,848\,179\,261\,915\,075$.