Exercices sur la récursivité

NSI Terminale - Programmation

qkzk - 2020/05/12

Généralités sur les algorithmes récursifs

Que calculent les fonctions `even` et `odd` ?

Voici la déclaration en Python de deux fonctions :

```
def even(n):
    if n == 0:
       return True
    else:
       return not odd(n)

def odd(n):
    if n == 0:
       return False
    else:
       return not even(n)
```

Que pensez-vous des expressions calculées par chacune des deux fonctions dans les deux cas de base et récursif ?

Encore la fonction `even`

Voici la déclaration en Python de la fonction `even` :

```
def even(n):
    if n == 0:
       return True
    else:
       return even(n - 2)
```

Que pensez-vous de cette fonction ?

Algorithmes simples

1. Proposer un algorithme récursif qui calcule la **somme** des entiers de `1` à `n`.
2. **Factorielle.**

   `n! = 1 * 2 * 3 * ... * n` et `0! = 1`. Proposer un algorithme récursif qui calcule `n!`
3. **Algorithme d'Euclide**

   L'algorithme d'Euclide permet de calculer le pgcd de deux nombres entiers, c'est-à-dire le plus grand entier positif divisant ces deux nombres, par des divisions successives.

   Voici le déroulement de cet algorithme pour le calcul du pgcd de *a=119* et *b=544*

a		b		q		r
119	=	544	*	0	+	119
544	=	119	*	4	+	68
119	=	68	*	1	+	51

a		b		q		r
68	=	51	*	1	+	**17**
51	=	**17**	*	3	+	0

Le pgcd de 119 et 544 est le dernier reste non nul, c'est-à-dire 17.

Le pgcd n'est pas défini lorsque les deux nombres sont nuls.

Exprimez de manière récursive cet algorithme. Vous pourrez supposer que les deux entiers a et b sont positifs ou nuls, et que l'un au moins de ces deux entiers n'est pas nul.

4. **Suite de Fibonacci**

   Proposer une fonction récursive pour calculer le terme d'indice `n` de la suite de Fibonacci.

   Les premiers termes sont `1, 1, 2, 3, 5, 8, 13, 21...`

   Le termes suivant est calculé en ajoutant les deux derniers termes.

   Dessiner le graphe des appels successifs de `fib(4)`.

5. **Puissance d'un nombre**

   Proposer un algorithme récursif qui calcule la puissance d'un nombre.

   ```
   puissance(3, 2)  ---> 9
   ```

Algorithmes sur les tableaux et les chaînes

1. **Palindromes**

   Un *palindrome* est un mot dont les lettres, lues de droites à gauche sont les mêmes que celles lues de gauche à droite. Les mots `radar`, `elle`, `été`, `ici` sont des palindromes. `DraD` n'est pas un palindrome.

   Proposez un prédicat récursif qui teste si un mot est un palindrome.

2. **Somme d'une liste**

   Proposez une fonction récursive `somme` qui calcule la somme des éléments d'une liste d'entiers passée en paramètre.

   On supposera que la somme d'une liste vide est 0.

3. Retourner une chaîne de caractères.

   Proposez une fonction `retourner` paramétrée par une chaîne de caractère et qui retourne la même chaîne écrite de droite à gauche.

   ```
   >>> retourner('robert')
   'trebor'
   ```

4. **Concaténer**

   Proposez une fonction qui concatène deux listes passées en paramètres.

   - Votre fonction doit être récursive,
   - Les seules méthodes sur les listes auxquelles vous avez droit sont :
     - `ajouter` qui ajoute un élément à la fin de la `liste`,
     - `retirer` qui retourne le premier élément de la `liste` et supprime cet élément de la liste.

5. **Occurrences**

   Proposez une fonction récursive qui compte les occurences d'une lettre dans un mot. Elle prend deux paramètres, la lettre et le mot.

   Elle retourne le nombre de fois où cette lettre apparaît dans le mot.

Autres algorithmes récursifs

1. **Coloriage**

   Supposons données les coordonnées (entières) *(x ; y)* d'un pixel situé à l'intérieur d'une région du plan délimitée par une courbe fermée. Les points sur la courbe délimitant la région sont de couleur noire, et ceux à l'intérieur sont blancs. On souhaite donner la couleur rouge à tous les points à l'intérieur de la région.

   Proposez un algorithme récursif pour effectuer ce coloriage. (Vous pourrez utiliser deux fonctions `get_color(x, y)` qui renvoie la couleur du pixel de coordonnées *(x ; y)* et `set_color(x, y, color)` qui fixe la couleur du pixel de coordonnées *(x ; y)*.

   Pour simplifier on peut supposer ici que les couleurs sont définies par des valeurs globales `BLACK`, `WHITE` et `RED`.

2. **Permutations des caractères**

   Dans cet exercice, on appelle *permutation* d'une chaîne de caractères *s* toute chaîne de même longueur que *s* contenant les mêmes caractères que *s*. Par exemple, la chaîne `'eadbc'` est une permutation de la chaîne `'abcde'`.

   Réalisez une fonction récursive qui construit la liste de toutes les permutations possibles d'une chaîne *s*.

   NB il sera probablement nécessaire de définir des fonctions auxiliaires, on essaiera de les coder récursivement aussi.

3. **Conversion des nombres romains**

   On suppose défini le dictionnaire :

   ```
   VALEUR_ROMAIN = { 'M' : 1000, 'D' : 500, 'C' : 100, 'L' : 50,
                     'X' : 10, 'V' : 5, 'I' : 1}
   ```

   Réalisez une fonction récursive `romain_to_arabe` qui prend en paramètre une chaîne de caractères représentant un « nombre romain » et dont le résultat est l'entier correspondant.

   ```
   >>> romain_to_arabe('X')
   10
   >>> romain_to_arabe('XCI')
   91
   >>> romain_to_arabe('MMXIX')
   2019
   ```

   NB Il est nécessaire de prendre en compte le cas où la valeur correspondante au second caractère est supérieure à celle du premier.

4. **Mélanger deux lises**

   Réalisez récursivement une fonction nommée `shuffle` paramétrée par deux listes `l1` et `l2` qui renvoie une liste dont les éléments sont ceux de ces deux listes dans un ordre alterné, tant que c'est possible.

   Si l'une des listes est plus courte que l'autre, on termine avec les éléments non utilisés de la plus longue liste.

   ```
   >>> shuffle([1,5,3,9,7],[8,2,6])
   [1,8,5,2,3,6,9,7]
   ```

5. **Les tours de Hanoï**

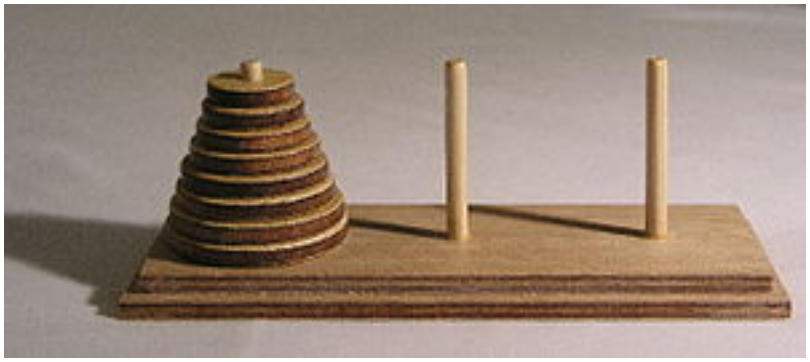

Figure 1: Hanoï

Il s'agit d'un exemple très classique d'algorithme récursif.

Voici ce qu'en dit Wikipedia

> Les tours de Hanoï sont un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :
>
> - on ne peut déplacer plus d'un disque à la fois ;
> - on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

Figure 2: Hanoï animation

Pour ceux qui souhaitent s'entrainer.

- choisir : "play",
- discs : "5",
- pegs : "3"

Trouver (puis programmer) un algorithme pour résoudre ce problème n disques. On pourra se contenter d'afficher sur la sortie standard les déplacements réalisés au cours de la résolution (par exemple *"dique de taille 3 déplacer de la tour 1 à la tour 2"*).

6. **Tri par insertion**

   Réalisez une version récursive du *tri par insertion* vu en première

7. **Permutations**

   Cet exercice est similaire à celui du calcul des permutations sur les chaînes de caractères.
   On cherche ici à produire la liste de toutes les listes obtenues par permutation des éléments d'une liste donnée.

   On peut alors reproduire l'exemple donné sur Wikipedia pour produire toutes les permutations de la liste `["Belle Marquise","vos beaux yeux","me font mourir","d'amour"]` dont le texte est tiré du Bourgeois gentilhomme (Acte II Scène IV) de Molière.

8. **Organisation des rencontres d'un championnat**

   Dans le cadre d'un championnat sportif (ou autre) on dispose de la liste de tous les joueurs (ou équipes) concernés. On souhaite organiser la liste de toutes les rencontres possibles entre ces joueurs. Chaque joueur devant rencontrer tous les autres une et une seule fois.

   On considère que chaque joueur est identifié par un nombre (qui peut par exemple correspondre à une clef dans une table qui permet d'accéder aux informations détaillées sur le joueur). Une liste de joueurs est donc en fait la liste des nombres associés à ces joueurs.

   Une rencontre est représentée par un couple (tuple) dont les deux composantes sont les numéros des deux joueurs impliqués.

   Donnez et codez un algorithme récursif qui produit, à partir d'une liste de joueurs, la liste de toutes les parties possibles entre ces joueurs.

```
>>> rencontres([1,2,3,4])
[ (1,2) , (1,3) , (1,4), (2,3), (2,4), (3,4) ]
```

9. `zip` et `unzip`

   En Python ces deux fonctions permettent d'itérer sur plusieurs objets ou, inversement, de séparer plusieurs itérables.

   ```
   >>> zip([1, 2, 3], ['a', 'b', 'c'])
   (1, 'a'), (2, 'b'), (3, 'c')
   ```

   Attention : en pratique, `zip` retourne *un itérable*. Si vous faites `for x, y in zip([1, 2, 3], ['a', 'b', 'c']): ...` ça fonctionne, mais si vous voulez accéder au premier élément, il faut d'abord convertir en *tuple*.

   1. Proposer un algorithme récursif pour `zip`.
   2. Proposer un algorithme récursif qui fait le contraire.

      ```
      >>> x, y = unzip(((1, 'a'), (2, 'b'), (3, 'c')))
      >>> x
      [1, 2, 3]
      >>> y
      ['a', 'b', 'c'])
      ```