TD4 : Écriture de fonctions récursives

Compétences

- Évaluer la terminaison d'une fonction récursive
- Passer d'une fonction itérative à une fonction récursive

Exercices

** **Ex. 1 —** Une banque centrale a mis en circulation n types de billets de banque, dont les valeurs sont notées $v_1, v_2, \ldots, v_n$. On suppose que ces valeurs sont des entiers strictement positifs. On fait en sorte que ces valeurs forment une suite strictement décroissante. Enfin, on suppose $n > 1$ et $v_1 = 500$. Dans cet exercice nous considérons que le distributeur dispose de billets de 500, 200, 100, 50, 20 et 10 euros. On a $n = 6$ et $v_1, v_2, \ldots, v_6$
Si s est un entier positif multiple de 10 ou nul, on note p(s) le nombre minimal de billets (de tous types) nécessaires pour former la somme s. Par exemple, dans la zone euro, $p(80)$ est égal à 3, car on peut obtenir 80 euros à l'aide de 3 billets (de 10, 20, et 50 euros) et il est impossible d'obtenir cette somme à l'aide de deux billets seulement. Le problème du distributeur de billets consiste, étant donné s, à calculer $p(s)$. On cherche un algorithme simple et efficace capable d'exhiber une manière, pas nécessairement optimale, d'atteindre la somme s.

1. Écrire une fonction itérative `giveChange(give, n)` qui demande la somme désirée, le nombre de billets de valeur différente (6 dans l'exemple de départ) et renvoie une répartition sous la forme d'un tableau avec le moins de billets possible. (Le rang dans le tableau designe une valeur de billet par exemple le rang 0 indique un billet de 500, le rang 1 indique un billet de 200 et ainsi de suite.)

   L'appel `giveChange(80)` doit renvoyer `[0, 0, 0, 1, 1, 1]`. C'est à dire $s = 50 + 20 + 10$.

   L'appel `giveChange(490)` doit renvoyer `[0, 2, 0, 1, 2, 0]`. C'est à dire $s = 2 \times 200 + 1 \times 50 + 2 \times 20$
2. Écrire la fonction récursive `giveChangeRec(give, solution, i)` qui demande la somme a rendre (`give`), un tableau `solution` qu possède le même nombre d'éléments que le tableau des billets et une variable `i` pour stocker le nombre de billets d'une valeur donnée. La fonction renvoie une répartition avec le moins de billets possible.

   L'appel `giveChangeRec(490, [0,0,0,0,0,0], 0)` doit renvoyer `[0,2,0,1,2,0]`. Soit $s = 2 \times 200 + 1 \times 50 + 2 \times 20$
3. Que se passe-t-il si on rajoute un billet de 40 pour le rendu de monnaie de 80 euros ?

** **Ex. 2 —** Monsieur ça marche ! Cette réponse est un classique chez les étudiants. Les principales difficultés sont souvent liées à de mauvaises évaluations des coûts tant temporels que spatiaux. Partons d'un exemple connu, l'algorithme de recherche dichotomique. Étant donné un tableau `t` de taille `n` contenant une `liste triée` par ordre croissant d'éléments et un élément `x`, on cherche à déterminer si `x` se trouve dans `t`. Cette algorithme se prête facilement à une programmation récursive.

1. Proposer une version récursive de la fonction `binarySearch(x, t, g, d)`
2. Faire une évaluation de la complexité dans le pire cas de cet algorithme. Conclure.

** **Ex. 3 —** On considère la fonction `checkNumberTwo` suivante :

```
1  def checkNumbersTwo( L1, n1, L2, n2) :
2      if n1 == 0:
3          return L2
4      else :
5          elt = L1[n1-1]
6          if elt not in L2:
7              L2[n2 - n1] = elt
8      return checkNumbersTwo(L1, n1-1, L2, n2)
9
10 L1 = [3,5,3,4,2,1,1,3,9]
11 L2 = [None]*len(L1)
12 checkNumbersTwo(L1, len(L1), L2, len(L2)))
```

1. Décrire précisément la pile d'exécution lors de l'appel de `checkNumbersTwo`
2. Quel est le rôle de cette fonction?
3. Modifier la fonction `checkNumbersTwo` pour que le résultat soit `[9,3,1,2,4,5,None,None,None]`
4. Écrire une version permettant de conserver l'ordre initial des éléments.
5. Quelle est la complexité de cet algorithme?