4TPM205U: Algorithmique des tableaux: feuille 8

Récursivité (1)

Travaux dirigés sur machine

Exercice 1 – Dans cet exercice vous allez utiliser pythontutor pour visualiser le déroulement de fonctions récursives.

1. Créez le fichier `tp08.py` et ajoutez-y la fonction `factorielleRecursive` vue en `TD`.
2. Ouvrez l'url `https://services.emi.u-bordeaux.fr/pythontutor/visualize-initinfo.html`. Copiez-collez votre fonction `factorielleRecursive` dans la fenêtre d'exécution. Ajoutez une ligne d'appel de la fonction pour `n` valant 4. Cliquez sur Lancer exécution, puis plusieurs fois sur le bouton Avant pour exécuter pas à pas la fonction. Observez et interprétez l'évolution de la **pile d'exécution**.
3. Toujours sous pythontutor, comparez le déroulement de `test1` et `test2`, puis `test3` avec `test4` (exercice 5 du TD) en observant bien l'empilement des instructions.

Exercice 2 – Écrire et tester les fonctions `sommeEltRec(t, n)` et `estPalindromeTab(t, n)` étudiées en TD.

Exercice 3 – Écrire et tester la fonction récursive `estStrictCroissantRec(t,n)` qui renvoie `True` si le tableau `t` de `n` entiers ($n > 0$) est strictement croissant, et renvoie `False` sinon.
Par convention, un tableau contenant 1 élément est considéré comme strictement croissant.

Exercice 4 – Écrire et tester fonction `renverser(t,n)` qui utilise une fonction récursive pour *renverser* la suite d'éléments contenus dans un tableau, c'est-à-dire que le premier élément devient le dernier, le deuxième devient l'avant-dernier, ..., le dernier devient le premier.

On ne devra pas utiliser de tableau auxiliaire et on pourra s'inspirer de la fonction qui teste de manière récursive si un tableau contient une suite palindrome.

Exercice 5 – Pour écrire une version récursive de l'algorithme de recherche dichotomique :

- écrire la fonction **récursive** `positionRec(t,debut,fin,x)` (utilisant la méthode de recherche dichotomique) qui retourne un indice `i` compris entre `debut` et `fin` tel que `t[i]` soit égal à `x` si cet indice existe et `-1` sinon ;
- écrire ensuite la fonction `position(t,n,x)` qui appellera `positionRec` et renverra la même chose que la fonction itérative `rechercheDichotomique` étudiée dans la feuille de TD 6 ;
- tester.

Exercice 6

1. Écrire une fonction récursive qui calcule 2^n par multiplications successives. À partir de quelle valeur de n le programme échoue-t-il ?

L'objectif est maintenant d'expliquer la cause de cette erreur.

2. Quelle est la valeur par défaut de la profondeur maximale de l'arbre des appels dans votre environnement `python` ? Pour répondre à cette question, vous pouvez utiliser la fonction `getrecursionlimit` du module `sys`.
3. Pour supprimer l'erreur, augmentez cette valeur avec la fonction `setrecursionlimit` et testez avec des valeurs de `n` de plus en plus grandes. À partir de quelle valeur de `n` n'obtenez vous plus de résultat ? Pour obtenir un message d'erreur explicite, testez votre programme (avec cette valeur de `n`) dans un TERMINAL avec la commande `python3 tp08.py`. Quel est la cause de l'échec du calcul constaté précédemment ?
4. Comparez les temps de calcul des versions itérative et récursive du calcul de 2^n (toujours par multiplications successives). Vous ferez cette comparaison pour la plus grande valeur de `n` possible. Comment s'explique la différence des deux temps de calcul ?
5. La fonction récursive `f` ci-dessous calcule également 2^n :

```
def f(n):
  if n == 0:
    return 1
  return f(n-1) + f(n-1)
```

   On veut comparer les temps de calcul des deux fonctions récursives calculant 2^n. Comment évolue le temps de calcul de `f` quand `n` augmente ? Quel est/serait le temps de calcul de `f(100)` ? Mêmes questions pour la fonction récursive de la question 1.