Université de Bordeaux
Semestre 1

2021-2022

TD3 : Fonctions récursives (1)

Compétences

— Prevoir le résultat d'une fonction récursive
— Évaluer la terminaison d'une fonction récursive

Exercices

* **Ex. 1 —** On considère les fonctions `foo1` et `foo2` suivantes :

```python
def foo1(n):
    if n == 0:
        print(0)
    else:
        print(n)
        foo1(n - 1)
```

```python
def foo2(n):
    if n == 0:
        print(0)
    else:
        foo2(n - 1)
        print(n)
```

Décrire précisément la pile d'exécution des fonctions `foo1` et `foo2` pour $n = 3$

* **Ex. 2 —** Soit la fonction récursive suivante :

```python
def f(n):
    if n > 100:
        return n - 5
    return f(n + 90)
```

Montrer que cette fonction se termine bien et simuler son exécution pour $n = 5$

** **Ex. 3 —** Soit la fonction récursive suivante :

```python
def f(n):
    if n > 100:
        return n - 5
    return f(f(n + 90))
```

Discuter de la terminaison de cette fonction et simuler son exécution pour $n = 5$

** **Ex. 4 —** 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, en voilà un exemple :

```python
def dicho(x, t):
    if len(t) == 0:
        return False
    k = len(t) // 2
    if x == t[k]:
        return True
    elif x < t[k]:
        return dicho(x, t[:k])
    else:
        return dicho(x, t[k+1:])
```

1. Expliquer pourquoi ce code est mauvais bien que fonctionnel et facile à lire.
2. Faire une évaluation de la complexité dans le pire cas de cet algorithme. Conclure.
3. Modifier le code pour que l'algorithme s'exécute avec une complexité temporelle et spatiale en $O(\log(n))$