Algorithmique
Master MEEF, préparation CAPES Maths
Feuille d'exercices, 2018–2019

1 Algorithmes non récursifs

Dans cette section, on demande des algorithmes *non récursifs*.

Exercice 1.1

1. Écrire en pseudo langage un algorithme prenant en argument un entier n et qui renvoie $\sum_{k=0}^{n} k^3$ si $n \geqslant 0$, et -1 si $n < 0$.
2. On peut montrer que pour $n \geqslant 0$, on a :
$$\sum_{k=0}^{n} k^3 = \left(\sum_{k=0}^{n} k\right)^2. \tag{1.1}$$
On veut vérifier cette égalité pour tous les entiers n jusqu'à un certain rang N. Écrire en pseudo langage un algorithme prenant en argument un entier $N > 0$, et qui renvoie Vrai si l'égalité 1.1 est vraie pour tout entier n compris entre 0 et N, et Faux sinon.
3. Écrire des fonctions en langage Python correspondant aux algorithmes des questions précédentes.

Exercice 1.2

1. Écrire en Python une fonction prenant en entrée un entier n, et qui retourne `True` si n est un entier strictement positif et premier, et `False` sinon.
2. Évaluer la complexité de cette fonction en fonction de *la taille de la représentation de n en base 2*.

Exercice 1.3 On considère la fonction suivante, qui suppose que son entrée n est un entier strictement positif.

```
1: MYSTERE(n : entier)
2: tant que n ≠ 100 faire
3:     si n < 100 alors
4:         n ← 2 * n
5:     sinon
6:         n ← n − 4
7:     fin si
8: fin tant que
9: retourner n
```

1. Simuler et expliquer le comportement de cet algorithme sur les entrées $n = 100$, $n = 101$, et $n = 99$.
2. La fonction peut-elle calculer autre chose que la valeur 100 ?
3. La fonction s'arrête-t-elle toujours ?
4. Reprendre les deux questions précédentes en changeant la ligne 6 par
$$n \leftarrow n - 3$$
en démontrant la réponse proposée.
5. Écrire en Python les deux versions et tester les réponses aux questions précédentes. Peut-on obtenir une garantie que vos réponses sont correctes grâce à ces tests ?

Exercice 1.4 On considère la suite définie par $u_0 = 1$, $u_1 = 2$ et $u_{n+2} = 2u_{n+1} + 3u_n$ pour $n \geqslant 0$.

1. Écrire un algorithme itératif prenant en argument $n \geqslant 0$ et qui calcule u_n.
2. Évaluer le nombre d'opétations arithmétiques effectuées par l'algorithme.
3. Quel est le comportement de la suite u_n quand n tend vers $+\infty$?
4. Écrire un algorithme itératif qui prend en argument $k > 0$ et qui calcule le plus petit entier n tel que $u_n > k$.

Exercice 1.5 On considère des tableaux dont chaque case contient un caractère, soit `'('`, soit `')'`. On dit qu'un tel tableau représente un *mot bien parenthésé* si :

- il contient globalement autant de parenthèses ouvrantes `'('` que de parenthèses fermantes `')'`.
- lorsqu'on le lit de gauche à droite, on ne rencontre jamais plus de `')'` que de `'('`.

Écrire en Python une fonction qui prend un tableau t et sa taille n et qui renvoie `True` si t représente un mot bien parenthésé et `False` sinon.

2 Récursivité

Dans cette section, on demande des algorithmes *récursifs*.

Exercice 1.6
1. Écrire en langage Python une version récursive des deux fonctions de l'exercice 1.
2. Simuler la fonction calculant la somme des cubes pour $n = 3$, en expliquant l'exécution étape par étape.
3. Évaluer en fonction de n le nombre d'appels récursifs effectués par cette fonction.

Exercice 1.7
1. Écrire une fonction récursive prenant en argument un tableau t et deux indices i et j de t, et qui renvoie
   - -1 si $i > j$, et sinon,
   - un indice $k \in [i\,,j]$ tel que $t[k]$ est maximal parmi les éléments du tableau situés entre les positions i et j.
2. Utiliser la fonction précédente pour écrire un algorithme de tri prenant en argument un tableau t et sa taille n, qui
   - place d'abord la valeur maximale de t en fin de tableau, et
   - trie ensuite récursivement les $n-1$ premières cases de t.
3. Évaluer le nombre de comparaisons effectuées sur un tableau de n cases, dans le cas le pire.

Exercice 1.8
1. Rappeler le principe de la recherche dichotomique dans un tableau trié.
2. Écrire l'algorithme en pseudo langage, puis en Python. Les arguments sont un tableau t, une valeur x recherchée, et deux indices $i\,,j \geqslant 0$ entre lesquels on cherche la valeur x.
3. Évaluer le nombre de comparaisons effectuées par cet algorithme sur un tableau de n cases.

Exercice 1.9 On considère la suite définie par $u_0 = 1$, $u_1 = 2$ et $u_{n+2} = 2u_{n+1} + 3u_n$ pour $n \geqslant 0$.
1. Écrire un algorithme *récursif* prenant en argument $n \geqslant 0$ et qui calcule u_n.
2. Évaluer la complexité de l'algorithme en fonction de n en supposant que les opérations arithmétiques coûtent $O(1)$.
3. Trouver une matrice M de taille 2×2 telle que $\begin{pmatrix} u_{n+2} \\ u_{n+1} \end{pmatrix} = M \begin{pmatrix} u_{n+1} \\ u_n \end{pmatrix}$ pour tout $n \geqslant 0$.
4. En déduire un autre algorithme de calcul du n^{e} terme de la suite $(u_k)_{k\in\mathbb{N}}$ faisant $O(n)$ opérations arithmétiques.
5. Écrire un algorithme récursif prenant en entrée un entier n et calculant M^n, faisant $O(\log_2(n))$ appels récursifs.
6. En déduire un algorithme de calcul de u_n effectuant $O(\log_2(n))$ opérations arithmétiques.

Exercice 1.10 — Fonction de McCarthy. On considère la fonction Python suivante :

```
def f(n):
  if n < 0:
    return -1
  if n > 100:
    return n-10
  return f(f(n+11))
```

1. Calculer $f(n)$ pour $90 \leqslant n \leqslant 100$.
2. Calculer $f(n)$ pour $0 \leqslant n \leqslant 100$.