
MPSI - ITC - TD 6 : ÉTUDE D'ALGORITHMES

Exercice 1 (Deux éléments les plus proches)

On considère la fonction suivante :

```
1 def indices_elements_plus_proches(liste):
2     """
3     Entrée
4     -----
5     Une liste d'entiers contenant au moins 2 éléments
6
7     Sortie
8     -----
9     Une paire d'indices distincts (i, j) telle que liste[i] et liste[j]
10    sont les deux éléments les plus proches de liste.
11    """
12    cand_i = 0
13    cand_j = 1
14    for i in range(len(liste)):
15        for j in range(len(liste)):
16            ecart_actuel = abs(liste[cand_i] - liste[cand_j])
17            alternative = abs(liste[i] - liste[j])
18            if i != j and alternative < ecart_actuel :
19                cand_i = i
20                cand_j = j
21    return (cand_i, cand_j)
```

1. Comment s'appelle la chaîne de caractères des lignes 2 à 11 ?
2. Sur quelle pré-condition se base cette fonction ? Donnez un exemple de liste en entrée qui ne respecte pas cette pré-condition et pour laquelle cette fonction lève une erreur.
3. Énoncer, sans la prouver, une propriété invariante de la boucle `for` des lignes 14 à 20 qui permettrait de prouver la correction de ce programme.

Exercice 2 (Division euclidienne)

On considère la fonction suivante :

```
1 def division_euclidienne(dividende, diviseur):
2     """
3     Entrée
4     -----
5     dividende : un entier naturel
6     diviseur  : un entier naturel non-nul
7
8     Sortie
9     -----
10    Une paire (quotient, reste) telle que
11    quotient * diviseur + reste = dividende
12    et 0 <= reste < diviseur
13    """
14    quotient = 0
15    reste = dividende
16    while reste >= diviseur:
17        quotient = quotient + 1
18        reste = reste - diviseur
19    return (quotient, reste)
```

1. Donner un variant de la boucle `while` des lignes 16 à 18.
2. Donner, sans la démontrer, une propriété invariante de la boucle `while` des lignes 16 à 18 qui permet de prouver la correction de cette fonction.

Exercice 3 (Recherche de zéro d'une fonction)

On considère la fonction suivante :

```
1 def zero_fonction(f):
2     """
3     Entrée
4     -----
5     f : une fonction strictement croissante,
6         qui prend en entrée un entier entre 0 et 100;
7         telle que:
8         f(0) est strictement négatif,
9         f(100) est strictement positif
10        et il existe un entier n entre 0 et 100 pour lequel f(n) vaut 0
11
12    Sortie
13    -----
14    Un entier n compris entre 0 et 100 tel que f(n) vaut 0
15    """
16    gauche = 0
17    droite = 100
18    while True:
19        milieu = gauche + (droite - gauche)//2
20        if f(milieu) == 0:
21            return milieu
22        elif f(milieu) < 0:
23            gauche = milieu
24        elif f(milieu) > 0:
25            droite = milieu
```

1. Donnez un variant de la boucle `while` des lignes 18 à 25.

Exercice 4 (Bonus : Tri à bulles)

On se propose d'étudier un algorithme, qu'on appelle le *tri à bulles*, dont la description en français est la suivante

Tant que la liste n'est pas triée :

 Parcourir la liste de gauche à droite :

 Si on trouve deux éléments consécutifs tels que le premier est plus grand que le deuxième, on les échange entre eux.

1. Écrire une fonction `est_triee` qui prend en entrée une liste d'entiers, et renvoie un booléen indiquant si la liste en entrée est triée dans l'ordre croissant ou pas.
2. Écrire une fonction `tri_bulles` qui implémente l'algorithme *tri à bulles* décrit en introduction. La structure de votre fonction sera une boucle `while` à l'intérieur de laquelle il y a une boucle `for`.
3. Proposez un variant permettant de prouver la terminaison de la boucle `while`.
4. Grâce au variant proposé, écrire une fonction `tri_bulles_for` qui implémente l'algorithme du *tri à bulles* sans utiliser de boucle `while` (la structure de votre fonction sera deux boucles `for` l'une dans l'autre).
5. Proposer un invariant pour la boucle extérieure permettant de prouver que votre fonction résout correctement le problème du tri. Si vous manquez d'inspiration, affichez l'état de la liste à la fin de chaque itération. Que constatez-vous sur la fin de la liste ?
6. Grâce à votre invariant, écrire une fonction `tri_bulles_optimal` qui consiste en deux boucles `for` imbriquées, mais où les boucles `for` ne s'exécutent pas plus que nécessaire (en particulier, la boucle intérieure ne parcourt pas toute la liste à chaque fois).
7. Majorer le nombre d'affectations que fait votre fonction par un polynôme en la taille de la liste en entrée.

Cet exercice nous montre qu'étudier les algorithmes non seulement permet de montrer qu'ils terminent et qu'ils sont corrects, mais ça permet au passage de les rendre plus explicites (question 4) voir plus efficaces (question 6).