DICHOTOMIE, TERMINAISON (TP EN UNE HEURE)

1 Recherche dans un tableau trié

Principe. Au TP 4, on a vu un algorithme pour vérifier si une valeur donnée se trouve dans un tableau. Lorsque le tableau est de taille n, cela donnait une complexité d'ordre n dans le pire cas, car il faut tester les éléments du tableau un par un. Mais, lorsque le tableau est *trié* (on supposera par ordre croissant), on peut aller beaucoup plus vite avec la *méthode par dichotomie*. Elle peut s'écrire en pseudo-code de la manière suivante :

```
Entrées : T, x

Tant que T est non vide, faire:
    m ← indice du "milieu" de T
    Si x = T(m):
        retourner m
    Sinon si x < T(m):
        T ← la première moitié de T (sans l'indice m)
    Sinon
        T ← la seconde moitié de T (sans l'indice m)
retourner ∅
```

Autrement dit, après chaque itération, on écarte au moins 50% des valeurs de T et on cherche x uniquement dans les valeurs restantes. On verra que grâce à cela, la complexité est grandement réduite par rapport à la complexité linéaire vue en TP 4.

L'algorithme ci-dessus a toutefois des désavantages : il présente un effet de bord sur T, et surtout il ne permet pas de retourner la position de x dans le tableau de départ T. Ainsi, plutôt que de changer T à chaque itération, on préfère introduire deux nouvelles variables a et b. Elles représentent les deux indices extrêmes du sous-tableau de T : à chaque itération, on cherche x dans le sous-tableau $[T_a, T_{a+1}, \cdots, T_b]$.

Exemple. On recherche la valeur $x = 9$ dans le tableau `T = [0,1,3,5,6,9,12,14,17]`. Ci-dessous, les valeurs soulignées correspondent à T_m.

Itération	a	b	m	Sous-tableau `T[a:b+1]` et indices	x	T_m	Comparaison
1	0	8	4	0 1 3 5 **6** 9 12 14 17 0 1 2 3 4 5 6 7 8	9	6	$x > T_m$ modification de a
2	5	8	6	9 **12** 14 17 5 6 7 8	9	12	$x < T_m$ modification de b
3	5	5	5	**9** 5	9	9	$x = T_m$ on retourne $m = 5$

A la deuxième itération, le milieu du tableau ne tombe pas juste car il y a un nombre pair d'éléments : quand ça arrive, on prend l'élément juste avant le milieu, donc ici 12. C'est un choix, on aurait pu faire le contraire. *Note* : si on avait cherché $x = 8$ dans le tableau `T`, on aurait commencé une 4ème itération avec $a = 5$ et $b = 4$. Mais alors, le sous-tableau `T[a:b+1]` serait vide, et donc l'algorithme se termine : x n'est pas dans T.

Question 1. Réaliser à la main, étape par étape, l'algorithme de dichotomie lorsqu'on cherche $x = 4$ dans le tableau `T` ci-dessus. Combien d'itérations sont nécessaires pour conclure ?

Voici le code Python correspondant à cette implémentation de l'algorithme :

```
def dichotomie(T,x):
    '''si x est dans T, retourne la position de x. Sinon, retourne None'''

    a = 0              # indices extrêmes : on cherche x dans T[a:b+1]
    b = len(T)-1
    while ... :        # le tableau T[a:b+1] n'est pas vide

        m = ...        # indice du "milieu" entre a et b
        if T[m] == x:
            return m
        elif T[m] < x:
            a = ...    # on cherche x dans la seconde moitié
        else:
            b = ...    # on cherche x dans la première moitié (car x > T[m] )

    return None        # x n'est pas dans T
```

Exercice 1. Compléter la fonction.

Exercice 2. Écrire un jeu de tests pour cette fonction comportant au moins 4 cas.

Exercice 3 (**Terminaison de l'algorithme `dichotomie`**)**.** Déterminer un variant de boucle pour la fonction `dichotomie` ci-dessus. Conclure quant à sa terminaison.

Démonstration.

□

2 Exercices d'approfondissement

Exercice 4. On a vu dans l'exemple que lorsqu'un tableau est de taille paire, le milieu ne « tombe pas juste » : avec l'instruction `m = (a+b)//2`, on choisit l'élément juste à gauche. On veut modifier le code pour prendre à la place l'élément juste à *droite*.

1. Un élève modifie la ligne `m = (a+b)//2` en `m = (a+b)//2 + 1` . Tester la fonction avec un jeu de test.
2. Quelle est la bonne modification à faire ? Vérifier avec un jeu de test.

Exercice 5 (Trouver le zéro d'une fonction par dichotomie)**.** On appelle zéro d'une fonction h tout réel x_0 tel que $h(x_0) = 0$ (si h est un polynôme, on parle aussi de racine). Soit f la fonction définie par :

$$\begin{aligned} f : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto x^5 + x + 1 \end{aligned}$$

1. Justifier que f admet un unique zéro sur $\mathbb{R}$.

On cherche à déterminer une valeur approchée du zéro de f.

2. Trouver $a, b \in \mathbb{R}$ tels que $f(a) < 0 < f(b)$.
3. Implémenter en Python l'algorithme suivant, qui calcule de manière approchée le zéro d'une fonction f donnée en entrée, avec a et b deux valeurs telles que $f(a)$ et $f(b)$ sont de signes (stricts) opposés. Enfin, on donne également en entrée une précision (variable `prec`), qui est un seuil en dessous duquel on considère qu'on a trouvé une valeur de f « suffisamment proche de zéro ».

```
Entrées : f,a,b,prec

Si f(a) et f(b) ne sont pas de signes opposés :
    retourner "Erreur sur le signe de f(a) et f(b)"

Si f(a)>0 :
    Échanger les valeurs de b et a # ainsi f(a) < 0 < f(b)

trouve ← FAUX

Tant que trouve = FAUX faire:
    c ← "milieu" de a et b

    Si |f(c)| ≤ prec:
        trouve ← VRAI
    Si f(c) < 0:
        Modifier a et b de sorte que f(a)<0, f(b)>0 et que la nouvelle
           distance |b-a| soit divisée par deux par rapport à l'ancienne.
    Si f(c) > 0:
        Modifier a et b de sorte que f(a)<0, f(b)>0 et que la nouvelle
           distance |b-a| soit divisée par deux par rapport à l'ancienne.

retourner c
```

4. Implémenter la fonction f de l'énoncé. Utiliser l'algorithme ci-dessus pour déterminer le zéro de f à la précision 10^{-12} près.
5. Vérifier que si on cherche le zéro de f à une précision de 10^{-16} près, la boucle est infinie.