NSI - Première

TD : Recherche dichotomique

qkzk

2021/05/26

pdf pour impression

Recherche dichotomique dans un tableau trié.

1. Recherche dichotomique

1. Peut-on appliquer la recherche dichotomique au tableau `[-4, 1, 3, 0, 5, 8]` pour rechercher le nombre 2 ? Justifier.
2. Combien d'itérations vont être nécessaires à la recherche dichotomique vue en cours pour trouver `15` dans le tableau `[1, 2, 15, 17, 30]` ?

Même question pour trouver l'élément 2. 2. Même question pour `[-4, -2, 0, 2, 10, 15, 18]` et les éléments `15` puis `2`. 3. Quand on multiplie par 2 la taille du tableau, de combien augmente le nombre d'itération d'une recherche dichotomique ?

2. Faire tourner à la main

On rappelle l'algorithme de recherche dichotomique dans un tableau trié.

On dispose d'un tableau d'éléments triés par ordre croissant.

```
fonction appartient(tableau, element)

    gauche = 0
    droite = dernier indice
    trouvé = faux
    # AVANT

    Tant que gauche <= droite et que trouvé = faux
        milieu = (gauche + droite) // 2
        # ICI
        si tableau[milieu] = element alors trouvé = vrai
        sinon si tableau[milieu] < element alors gauche = milieu + 1
        sinon droite = milieu - 1
        # LA
    Retourner trouvé
```

1. a. Que retourne l'appel `appartient ([1, 3, 5, 17, 17, 19], 3)` ?

   b. Compléter le tableau suivant pour l'appel précédent.

	gauche	milieu	droite	tableau[milieu]	trouvé
# AVANT	0	-	?	-	?
# ICI	0	2	?	?	?
# LA	0	2	1	5	?
# ICI	0	0	1	1	?
# LA	?	?	?	?	?
# ICI	?	?	?	?	?

	gauche	milieu	droite	tableau[milieu]	trouvé
# LA	?	?	?	?	?

c. Dans l'exemple précédent, pourquoi sort-on de la boucle ?

2. On exécute l'appel `appartient ([1, 3, 5, 17, 17, 19], 4)` .

   a. Que renvoie cet appel ?
   b. Construire un tableau similaire au précédent.
   c. Pourquoi sort-on de la boucle ?
   d. Combien de fois passe-t-on par la ligne `# ICI` ?

3. On exécute l'appel `appartient ([1, 13, 5, 17, 17], 13)`

   a. Quelle devrait-être la sortie de la fonction ?
   b. Qu'obtient-on en pratique ? On s'aidera d'un tableau similaire aux précédents.
   c. Expliquer le problème.

3. Programmer

1. Traduire en Python l'algorithme proposé dans l'exercice 1. On n'oubliera pas les indications de type et la documentation.

2. En vous aidant de la fonction précédente, écrire le code de la fonction suivante :

```
def indice(tableau: list, element: int) -> int:
    '''
    Retourne l'indice `i` tel que `tableau[i] == element` s'il y en a un
    sinon retourne -1.
    Pré condition : ... A FAIRE
    '''
    # À FAIRE

assert indice([1, 5, 17, 17, 19], 3) == 1
assert indice([1, 5, 17, 17, 19], 4) == -1
assert indice([1, 5, 17, 17, 19], 17) == 3 or \
       indice([1, 5, 17, 17, 19], 3) == 4
```

4. Dichotomie et présence d'ex aequo

La fonction `bisect_left(t, v)` de Python (`from bisect import bisect_left`) est une variante de la recherche dichotomique vue en cours qui renvoie systématiquement le premier indice `i` de `t` tel que `t[i] >= v` (alors que la recherche dichotomique renvoie un indice quelconque `i` tel que `t[i] == v` et -1 s'il n'en existe pas). La complexité de `bisect_left` est logarithmique.

Soit `t` le tableau `[1, 3, 7, 9]`

1. Quel sera le résultat de `bisect_left(t, 5)` ?

2. Même question avec le tableau `t = [1, 1, 3, 3, 3, 7, 9]` et l'instruction `bisect_left(t, 3)`.

3. On enregistre dans la variable `indice` le résultat de l'instruction précédente. Quel sera la valeur de `t` après l'instruction : `t.insert(indice, 5)` ?

4. De manière générale, on considère un tableau `t` et les deux instructions suivantes :

```
indice = bisect_left(t, valeur)
t.insert(indice, valeur)
```

   Que peut-on dire de l'état de `t` après ces instructions ?

5. Quel-est le rôle de la fonction `bisect_left` ?

4. Résoudre numériquement une équation

Les méthodes numériques permettent de calculer de manière effective des solutions numériques à divers problèmes, souvent liés à la physique. Nous abordons ici la méthode dichotomique utilisée pour résoudre des équations.

Énoncé

La méthode dichotomique est employée dans la résolution de nombreux problèmes. En particulier, elle permet de trouver une solution à l'équation $f(x) = 0$ où f est une fonction continue (= dont la courbe n'a pas de *trou*) d'une variable réelle.

Cette méthode peut être utilisée dans les cas où une solution analytique n'est pas connue.

Dans toute la suite, on supposera que f est une fonction continue.
On admet que s'il existe $a < b$ tels que $f(a) \times f(b) < 0$ alors, il existe $c \in [a; b]$ tel que $f(c) = 0$.

1. L'algorithme de recherche de solution par la méthode dichotomique consiste, à partir d'un intervalle $[a; b]$ contenant une solution x à l'équation $f(x) = 0$ à trouver un nouvel intervalle, deux fois plus petit, qui contient aussi une solution.

   Pour cela, on évalue $f(m)$ avec $m = \dfrac{a+b}{2}$. Alors, au moins un des deux intervalles, $[a; m]$ ou $[m; b]$, contient une solution.

   Écrire une fonction, nommé `iteration_dicho` qui prend en paramètres la fonction f, les deux valeurs a et b et renvoie les bornes du nouvel intervalle (plus petit) contenant la solution.

2. La probabilité pour que la valeur soit trouvée exactement est très faible (et il faut tenir compte des erreurs de calcul sur les nombres flottants). Nous allons donc nous contenter de considérer que si l'intervalle est suffisamment petit ($b - a$ inférieur à ε fixé), alors une solution approchée est $\dfrac{a+b}{2}$, ce qui garantit une erreur sur la solution inférieure à $\dfrac{\varepsilon}{2}$.

   Écrire une fonction nommée `dichotomie` qui prend en paramètre f, les bornes d'un intervalle a et b, la valeur ε et qui renvoie la solution approchée de l'équation $f(x) = 0$, ou *None* s'il n'y a *a priori* pas de solution dans l'intervalle considéré (car $f(a)f(b) > 0$ par exemple, ce qui ne signifie pas qu'il n'y a pas de racine dans l'intervalle, mais ne permet pas non plus d'assurer qu'il y en a une).

3. Tester la méthode de résolution pour une équation dont la solution exacte est connue par exemple :

   ```
   def fonction(x):
       return (x-1) * (x+3) ** 2 * (x-4)

   dichotomie(fonction, 0, 2, 1e-2)
   ```

   La seule racine entre 0 et 2 est 1. Vérifier que la valeur approchée de la racine est correcte.

4. Utiliser la fonction dichotomie pour trouver une solution de $\cos(\sqrt{x}) = 0$ comprise entre 10 et 30.

5. Le nombre d'étapes de l'algorithme dépend uniquement de la largeur de l'intervalle et de la borne ε choisie. En supposant que $a = 0$, $b = 8$, et $\varepsilon = 10^{-2}$, combien d'étapes (tours de boucle) réalisera la fonction dichotomie ?

6. À supposer que l'appel précédent renvoie une solution en $0,1$ milliseconde, à combien pourrait-on estimer le temps de calcul si la valeur de ε etait maintenant 10^{-6} ?

Feuille de route

1. **Passer une fonction en paramètre d'une autre fonction**

   On peut passer une fonction en paramètre d'une autre fonction en Python. Par exemple :

```
def carre(x):
    return x ** 2

def appliquer(f, x):
    return f(x)

>>> appliquer(carre, 3)
9
```

2. et 3. **Programmer un algorithme itératif**

   Après avoir vérifié que $f(a)f(b) \leq 0$ (si ce n'est pas le cas, la fonction devra renvoyer `None`), écrire une boucle qui tournera tant que la largeur de l'intervalle (initialement l'intervalle est $[a; b]$) est supérieure à ε. À chaque tour de boucle, a ou b (un seul des deux) sera modifié, et deviendra égal à $m = \dfrac{a+b}{2}$.

3. **Utiliser un programme de résolution numérique**

   Les fonctions `cos` et `sqrt` (racine carrée) sont dans le module `math` qu'on importe avec `import math`. On préfixe les commandes, par exemple : `math.cos(9)` et `math.sqrt(1)`

4. **Évaluer le nombre d'étapes d'un algorithme itératif**

   À chaque tour de boucle, la largeur de l'intervalle est divisé par 2. La question est de savoir combien de fois l'intervale doit être divisé par 2 avant que la largeur ne devienne inférieure à ε

5. **Évaluer la complexité en temps d'un algorithme**

   Reprendre la question précédente avec la nouvelle valeur de ε et trouver une lien entre le temps de calcul et le nombre de tours de boucle.