TP N°12 : Les méthodes de recherche

Les algorithmes de recherche sont très utiles en informatique. Leur rôle est de :

- Vérifier l'existence d'un ou de plusieurs éléments, satisfaisant une condition donnée, dans un tableau ;
- Chercher le nombre d'occurrences d'un élément ;
- Chercher la ou les positions d'un élément donné dans un tableau ; etc.

Problématique : On considère une liste `L` et un objet `e`. On souhaite tester l'appartenance de `e` à `L`.
Des primitives de Python réalisent ce travail (`e in L`), mais notre objectif est de comprendre comment on les réalise.

Il existe essentiellement deux stratégies de recherche : la *recherche séquentielle* ou encore *linéaire* et la *recherche dichotomique*.

1 Recherche séquentielle

La recherche séquentielle d'un élément dans un tableau consiste à parcourir le tableau séquentiellement jusqu'à trouver l'élément recherché ou atteindre la fin du tableau. La recherche peut se faire sur un tableau trié ou non, la différence réside dans le parcours du tableau et dans la condition d'arrêt de la recherche.

Principe :

1. Parcourir la liste.
2. Comparer chaque élément parcouru à l'élément recherché.
3. Si l'élément est trouvé, quitter la boucle et renvoyer `True` ou l'indice de l'élément (plus précisément de la 1ère occurrence).
4. Arrivé à la fin de la liste, quitter la boucle et renvoyer `False` ou `None` (aucune occurrence).

Implémentation en Python

```
def recherche_seq(L,x):
    n =len(T)
    for i in range(n):
        if L[i] == x:
            return i
    return None
```

Complexité : le temps d'exécution de ce programme est proportionnel à la taille de la liste, on dit que la complexité de calcul est linéaire, on la note : $O(n)$ où n est la longueur de la liste.

Exercice 1

1. Écrire une fonction qui renvoie l'indice de la première occurrence de l'élément maximal d'une liste d'entiers. Évaluer la complexité de cette fonction.
2. Écrire une fonction qui renvoie les deux plus grands éléments d'une liste d'entiers. On supposera que la liste est de longueur au moins 2 ; en revanche, on veillera à ne la parcourir qu'une seule fois.

Exercice 2: Recherche d'un mot (une séquence) dans un texte (tableau)

Un problème classique en informatique consiste à rechercher, non pas une seule valeur, mais une séquence de valeurs dans un tableau. Cela revient à chercher une occurrence d'un tableau dans un autre ou, pour les chaînes de caractères, une occurrence d'un mot dans un texte.

1. Écrire une fonction `recherche_motif(m, t)` qui, étant donnés deux tableaux m et t, détermine la position de la première occurrence de m dans t, si elle existe, et qui renvoie `None` sinon.
   Exemple : pour les tableaux `m=[1,2,3]` et `t=[2,1,4,1,2,6,1,2,3,7]`, la fonction renvoie 6.
2. Modifier la fonction `recherche_motif(motif, texte)` pour qu'elle retourne toutes les occurrences d'une chaîne de caractères `motif` de longueur m dans une chaîne de caractères `texte` de longueur n. On supposera $m \leq n$.

Existe-t-il un algorithme plus efficace que la recherche naïve ?
OUI si la liste est triée. C'est la *recherche dichotomique*.

2 Recherche dichotomique

L'algorithme que nous proposons ici suppose que la liste dans laquelle on recherche un élément est **triée** en ordre croissant. Quand la liste (tableau) est triée, la recherche d'un élément dans une liste peut être réalisée de manière plus efficace en appliquant le principe *"diviser pour régner"*.

L'idée est que l'on compare l'élément cherché au terme du milieu de la liste, ce qui conduit, tant qu'on ne le trouve pas, à le rechercher soit dans la première partie soit dans la seconde partie de la liste.

Principe :

1. Parcourir la liste.
2. Comparer chaque élément parcouru à l'élément recherché.
3. Si l'élément est trouvé, quitter la boucle et renvoyer `True` ou l'indice de l'élément (plus précisément de la 1ère occurrence).
4. Arrivé à la fin de la liste, quitter la boucle et renvoyer `False` ou `None` (aucune occurrence).

Implémentation en Python

```
def rech_dicho(x,T):
    a, b = 0, len(T)-1
    while (a <= b):
        m = (a+b)//2
        if x > T[m]:
            a = m + 1
        elif x < T[m]:
            b = m - 1
        else : # ici, x == T[m], on retoune la position m
            return m
    return None
```

Complexité : La complexité de la recherche dichotomique est $O(log_2(n))$ où n est la taille du tableau.

Exercice 3

1. Proposer une fonction itérative `recherche_dicho(x,L)` qui renvoie le booléeen indiquant la présence (`True`) ou non (`False`) de x dans L par dichotomie.
2. Vérifiez votre code en cherchant 13, puis 2 et enfin 20 dans `[1,3,5,7,8,13,14,17,19]`.
3. Écrire une fonction `rechDicho1(t,n,x)` qui effectue une recherche dichotomique de x (entier) parmi les n premiers éléments d'un Tableau t trié et qui renvoie l'indice d'une occurrence (pas forcément la première) de x. La fonction renvoie -1 en cas de recherche infructueuse.
4. Exécuter votre fonction sur les données suivantes : `n=7`, `t=[1,7,8,9,12,15,15,22,30,31]` et `x=15`.
5. Écrire une fonction `rechDicho2(t,n,x)` version récursive de `rechDicho1(t,n,x)`.