Recherche dichotomique dans un tableau trié

Comment chercher un mot dans un dictionnaire ?
Parcourt-on le dictionnaire du début à la fin ?
Surtout pas car le dictionnaire est **ordonné** et en utilisant le fait qu'il soit ordonné on peut procéder de manière plus efficace
On procède plutôt **par dichotomie**, (couper en deux) c'est à dire on ouvre le dictionnaire au milieu et on compare le mot du milieu au mot recherché
Si le mot recherché "est plus petit" que le mot du milieu on réitère la recherche dans la partie du dictionnaire qui précède le mot du milieu, sinon on réitère la recherche dans la partie qui suit le mot recherché

On traduit l'idée ci-dessus sous une forme mathématique plus précise
si $x \in [a, b[$ alors **ou bien** $x \in [a, \frac{a+b}{2}[$ **ou bien** $x \in [\frac{a+b}{2}, b[$

Bien observer comment on garde **invariant** l'encadrement de v dans un intervalle du type [..,...[

L'idée ci-dessus nous amène à poser comme **invariant de boucle** :

$I(a,b) = \{0 \leqslant a < b \leqslant n-1 \;\; T[a] \leqslant x < T[b]\}$

1. L'assertion I(a,b) doit être vraie au début de la boucle (ligne 10). Compléter les lignes 4 et 7 de l'algorithme ci-dessous pour que cela soit ainsi
2. Il nous reste à trouver la condition d'arrêt de la boucle et compléter la ligne 11 de l'algorithme

   Pour comprendre comment arrêter la boucle traiter les cas particuliers

   (a) $T = \{E, G, I, K, M, M, M\}$ et $v = H$

   (b) $T = \{E, G, I, K, M, M, M\}$ et $v = K$
3. Une fois que vous avez compris comment s'arrête la boucle il faut conclure. Compléter les lignes 20 et 21

Algorithme 1 : Recherche dichotomique

`estDansTableau` (T,v)
Données : Un tableau T ayant $n \geqslant 2$ éléments comparables **trié dans l'ordre croissant**,une valeur v
Résultat : un indice de v si v est dans T, -1 sinon

```
1  début
2      a ← 0
3      b ← n − 1
4      si ..... alors
5          retourner -1
6      fin
7      si ..... alors
8          retourner n-1
9      fin
10     {I(a,b)}
11     tant que ..... faire
12         m ← E((a+b)/2)
13         si v < T[m] alors
14             b ← m
15         sinon
16             a ← m
17         fin
18         {I(a,b)}
19     fin
20     si ..... alors
21         retourner .....
22     sinon
23         retourner -1
24     fin
25 fin
```

1 Preuve

Algorithme 2 : Recherche dichotomique

`estDansTableau` (T,v)
Données : Un tableau T ayant $n \geqslant 2$ éléments comparables **trié dans l'ordre croissant**,une valeur v **Précondition**
Résultat : un indice de v si v est dans T, -1 sinon **Postcondition**

```
1  début
2      a ← 0
3      b ← n − 1
4      si v < T[0] ou v > T[n-1] alors
5          retourner -1
6      fin
7      si v = T[n-1] alors
8          retourner n-1
9      fin
10     tant que a+1 < b faire
11         m ← E((a+b)/2)
12         si v < T[m] alors
13             b ← m
14         sinon
15             a ← m
16         fin
17     fin
18     si T[a] = v alors
19         retourner a
20     sinon
21         retourner -1
22     fin
23 fin
```

1. **Terminaison** : la grandeur $b - a \geqslant 1$, **l'amplitude de l'encadrement de v** , décroit à chaque tour de boucle car est à peu près divisée par 2 à chaque tour de boucle jusqu'à ce que $b - a = 1$
   Or la condition d'arrêt de la boucle est justement que $a + 1 \geqslant b$ (négation de $a + 1 < b$), c'est à dire $b - a \geqslant 1$ donc la boucle se termine au bout d'un nombre fini de tours
2. **Invariant de boucle** :
   A l'entrée de la boucle I(a,b) est vraie pour a = 0 et b = n-1 car à cause du test précédant la boucle on sait que $T[a] \leqslant v < T[b]$
   Montrons que I(a,b) reste vraie après une itération :
   si I(a,b) est vraie, c'est à dire $\{0 \leqslant a < b \leqslant n-1 \quad T[a] \leqslant v < T[b]\}$ alors les nouvelles valeurs après une itération de a et b notées a' et b' sont :
   soit $a' = E(\frac{a+b}{2})$ et $b' = b$ si $v \geqslant T[E(\frac{a+b}{2})]$ ou $a' = a$ et $b' = E(\frac{a+b}{2})$ si $v < T[E(\frac{a+b}{2})]$

Donc dans le premier cas et le deuxième cas $T[a] \leqslant v < T[b]$ avec $0 \leqslant a < b \leqslant n-1$

3. **Sortie de boucle** la boucle s'arrête lorsque b = a+1 pour $0 \leqslant a \leqslant n-2$ donc I(a,a+1) est vraie autrement dit $T[a] \leqslant v < T[a+1]$ donc :
   Soit $v = T[a]$ et on retourne a ou $T[a] < v < T[a+1]$ et on retourne -1

2 Complexité

Pour évaluer la complexité de la recherche dichotomique dans le pire des cas on va supposer que la longueur du tableau T est une puissance de 2
Regardons sur un cas particulier :
T = {E,G,I,K,M,O,Q,S} avec v = J
A chaque tour de boucle il y a 2 affectations et 3 comparaisons, combien de tours de boucles ?
On se rend compte qu'il y a 3 tours de boucle pour 2^3 valeurs
Plus généralement on a n tours de boucles pour 2^n valeurs
Il existe une fonction mathématique, appelée le logarithme en base 2
la fonction $n \to \ln_2(n)$ telle que $\ln_2(2^n) = n$

Par conséquent si la taille d'un tableau est 2^n alors la complexité de la recherche dichotomique est en $O(\ln_2(n))$
Plus généralement pour toute taille m, il existe n tel que
$2^n \leqslant m < 2^{n+1}$ avec $n = \ln_2(m)$
Par conséquent la complexité de la recherche dichotomique est en $O(\ln_2(n))$
Quelques valeurs expérimentales : On a vu en TP que dans le pire des cas pour une liste de un million d'éléments la recherche séquentielle prenait à peu près 4 centièmes de secondes alors que la recherche dichotomique sur ce même tableau trié prend ...40 micro - secondes , 1000 fois moins de temps !
Encore plus fort on verra en exercice que si la taille du tableau est multiplié par un facteur 10^6 le temps d'exécution ne fera que **doubler** pour atteindre 80 micro secondes !

3 Exercices - Recherche dichotomique

Ex 1

Exécuter l'algorithme de la recherche dichotomique avec $T = [2, 4, 6, 8, 10, 12, 14, 16]$ pour les valeurs suivantes

1. v = 1
2. v = 3
3. v = 14
4. v = 16

Ex 2

Si une valeur est dans le tableau, l'algorithme retourne-t-il forcément le plus petit indice de cette valeur ?

Ex 3

Ecrire en Python la fonction `estDansTableauDicho(liste,valeur)`

Ex 4

1. En TP on a chronométré que le tri par sélection prenait à peu près 4 centièmes de secondes pour un tableau de 1000 éléments. Justifier le temps de 1000 jours pour un tableau de 100 millions d'éléments
2. En TP on a chronométré que dans le pire des cas la recherche séquentielle prenait à peu près 4 centièmes de secondes pour un tableau d'un million d'éléments. Combien de temps prendra la recherche séquentielle dans le pire des cas pour un tableau de mille milliards d'éléments ?
3. En TP on a chronométré que dans le pire des cas la recherche dichotomiques prenait à peu près 40 micro secondes pour un tableau d'un million d'éléments. Combien de temps prendra la recherche dichotomique pour un tableau de mille milliards d'éléments ?(Indication : on admettra que $T(n) = c \ln_2(n)$ et on admettra la relation $\ln_2(n^2) = 2 \ln_2(n)$ pour $n > 0$

Ex 5

Peut-on écrire a $<$ b à la place de a $+ 1 <$ b dans le programme vu en cours ?

Ex 6

Il existe d'autres façons d'écrire la recherche dichotomique, voici un autre algorithme :

1. Prouver que la boucle s'arrête
2. Prouver que $\{0 \leqslant a < b \leqslant n-1 \mid$ v n'est pas dans le tableau pour les indices dans [0,a[et]b,n-1]$\}$ est un invariant de boucle
3. Prouver qu'à la sortie de boucle l'algorithme résoud le problème de la recherche dichotomique

Algorithme 3 : Recherche dichotomique

`estDansTableau` (T,v)
Données : Un tableau T ayant $n \geqslant 2$ éléments comparables **trié dans l'ordre croissant**,une valeur v
Résultat : un indice de v si v est dans T, -1 sinon

```
1  début
2      a ← 0
3      b ← n − 1
4      {I(a,b)}
5      tant que a < b faire
6          m ← E((a + b)/2)
7          si v < T[m] alors
8              b ← m − 1
9          fin
10         sinon si v > T[m] alors
11             a ← m + 1
12         fin
13         sinon
14             retourner m
15         fin
16     fin
17     {I(a,b)}
18     si v = T[a] alors
19         retourner a
20     sinon
21         retourner -1
22     fin
23 fin
```

Ex 7

1. Montrer que $\{\ 0 \leqslant i < n\ f = i!\ \}$ est l'invariant de boucle
2. Justifier la terminaison de la boucle
3. Montrer que le programme calcule bien n !
4. Que se passe-t-il si on permute les deux instructions à l'intérieur de la boucle ?

Algorithme 4 : Calcul de n !

```
début
    i ← 0
    f ← 1
    tant que i < n faire
        i ← i + 1
        f ← i * f
    fin
fin
```