

TD 20 – Algorithme de recherche dichotomique dans une liste triée

On dispose d'un vecteur ou d'une liste d'éléments, L contenant N éléments. On se place ici dans le cas le plus simple qui est le cas d'une liste triée par ordre **strictement croissant**, c'est-à-dire :

$$\forall i \in \llbracket 0, N-1 \rrbracket, L_i < L_{i+1}$$

(la suite des (L_i) est strictement croissante). On cherche ici un L_k en particulier, c'est-à-dire l'indice k correspondant à un élément, unique, de la liste.

```
def dichotomie(a, L): # on admet que l'élément a appartient effectivement à L
    g = 0 # le point qui correspondra à l'indice le plus à gauche
    d = np.size(L) - 1 # le point qui correspondra à l'indice le + à droite
    m = int((g+d)/2) # m point milieu entre g et d (on arrondit à gauche)
    while d > g :
        if L[m] < a:
            g = m + 1 # g légèrement décalé à droite par rapport à m
        else :
            d = m
        m = int((g+d)/2) # on met à jour m
    return m # renvoie m la position (index) de a dans L
```

B / Démonstrations

Notons, dans cette partie, k le numéro de l'itération en cours d'exécution (que nous avons noté j dans le cours).

Q1 – ✎ Démontrer par récurrence que la suite $(d_k - g_k)$ est une suite d'entier.

Q2 – ✎ Démontrer que, tant qu'on reste dans la boucle, à l'issue d'une boucle on a : $d_k - g_k < d_{k-1} - g_{k-1}$
Que peut-on en déduire ?

Q3 – ✎ Démontrer la validité de cet algorithme. On suppose **l'invariant de boucle** : $L_g \leq a \leq L_d$.

Aide : vous aurez besoin du fait que la suite (L_k) est strictement croissante. On admet que a est contenu dans L , on notera K son indice (que l'on cherche), c'est-à-dire $L_K = a$.

Q4 – ✎ Démontrer qu'à la fin de toute itération k , on a l'inégalité suivante :

$$\frac{d_{k-1} - g_{k-1}}{2} - 1 \leq d_k - g_k < \frac{d_{k-1} - g_{k-1}}{2}$$

Point mathématique : on pourrait alors démontrer que ceci permet l'encadrement suivant :

$$\frac{d_0 - g_0 + 2}{2^k} - 2 \leq d_k - g_k < \frac{d_0 - g_0}{2^k}$$

- Pour l'inégalité de droite, on peut intuitivement cette inégalité en faisant l'analogie avec une suite géométrique $u_k = q \cdot u_{k-1}$ (avec la raison $q = \frac{1}{2}$ dans notre cas), on sait que $u_k = u_0 \cdot q^k$. C'est-à-dire ici $u_k = \frac{u_0}{2^k}$. La démonstration se fait par récurrence.

- Pour l'inégalité de gauche, on l'intuitue en faisant l'analogie avec une suite arithmético-géométrique :

$$v_k = a \cdot v_{k-1} + b$$

(avec, important, ici $a = 1/2 \neq 1$ sinon ce serait une suite arithmétique simple ! et $b = -1$)

On aurait alors $v_k = (v_0 - r) \cdot a^k + r$, avec $r = \frac{b}{1-a}$

Dans notre cas, $a = \frac{1}{2}$ et $r = \frac{-1}{1-1/2} = \frac{-1}{1/2} = -2$, d'où $v_k = (v_0 + 2) \cdot a^k - 2$ (démonstration par récurrence aussi).

Q5 – ✎ Démontrer alors que l'ordre de complexité de l'algorithme de dichotomie dans le pire des cas (c'est-à-dire le nombre d'itération k_r nécessaire pour que l'algorithme renvoie la solution recherchée) est $k_r = O(\log_2(N))$.

Remarque : N étant la taille de la liste en entrée. Ce résultat sera alors admis comme étant un résultat de cours.

C / Comparaison relative face à une méthode « naïve »

Si l'on n'avait pas réfléchi à l'algorithme optimisé de dichotomie (qui est un algorithme du type **diviser pour régner**), on aurait potentiellement écrit une version de recherche de l'élément a dans la liste L par balayage. C'est-à-dire une fonction qui parcourt un à un tous les éléments de la liste L jusqu'à trouver l'élément recherché. On quittera la fonction dès que l'élément a été trouvé.

Q6 – 📄 Ecrire une telle fonction `recherche(a, L)` qui renvoie l'index de l'élément a dans la liste L .

Q7 – ✎ Quelle aurait été l'ordre de complexité de cette fonction

- dans le pire des cas ?
- dans le meilleur des cas ?
- en moyenne ?