

Chapitre 5 : Algorithmes dichotomiques

I- Recherche dichotomique d'un élément dans une liste triée

a) Principe

Le coût d'une recherche naïve d'un élément dans une liste L triée de longueur n peut nécessiter jusqu'à n comparaisons.

```
def recherche_naive(L,elt):
    for i in range(len(L)):
        if L[i]==elt:
            return print("l'elt {0} est à l'indice
{1}".format(elt,i))
    print("l'élément n'est pas dans la liste")
```

L'intérêt de la méthode dichotomique est d'accélérer grandement la recherche. Le principe est de comparer l'élément recherché avec celui se trouvant au milieu de la liste $L[n//2]$. S'ils sont égaux, c'est terminé, sinon on recommence avec la partie du tableau pouvant contenir l'élément.

b) Programmation (NYWYXN)

```
def recherche_dicho(L,elt):
    index_debut=0
    index_fin=len(L)-1
    while index_fin>=index_debut:
        index_milieu=(index_fin+index_debut)//2
        if L[index_milieu]==elt:
            return (print("l'elt {0} à
l'indice{1}".format(elt,index_milieu)))
        elif elt>L[index_milieu]:
            index_debut=index_milieu+1
        else :
            index_fin=index_milieu-1
    print("l'élément n'est pas dans la liste")
```

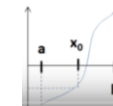
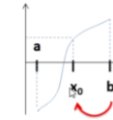
c) Commentaires

Considérons une liste L de longueur n telle que $n = 2^p - 1$ et examinons le cas de la recherche d'un élément qui n'est pas dans L . Le principe de la méthode dichotomique consiste à retirer l'élément centrale, il reste alors $2^p - 2$ éléments et à ne considérer que l'une des deux sous-listes (de longueur $2^{p-1} - 1$). Dans le pire cas, on continue k fois cette opération jusqu'à ce que $2^{p-k} - 1 = 0$ soit $k = p = \log_2(n + 1) \approx \log_2(n)$.

Si $n = 100$, 7 étapes suffisent, pour $n = 1000$, 10 étapes suffisent.

II- Résolution d'une équation du type $f(x) = 0$ par dichotomie (E7Y7EW)

Le principe de dichotomie s'applique en mathématiques pour trouver une solution approchée d'une équation du type $f(x) = 0$ sur un intervalle $[a, b]$ à epsilon près. La fonction f est supposée monotone et continue.



```
def solve_dicho(f, a, b, epsilon):
    debut=a
    fin=b
    while abs(debut-fin)>epsilon:
        x0=(debut+fin)/2
        if f(x0)*f(debut)<=0:
            fin=x0
        else :
            debut=x0
    return (debut+fin)/2
```

III- Exponentiation rapide (4454DP)

Il s'agit d'écrire une fonction puissance qui prend en paramètre un nombre x non nul et un entier naturel n et qui renvoie x^n .

Dans une version naïve, on calcule successivement x^2, x^3, \dots en se basant sur la définition $x^n = 1 \times x \times x \dots \times x$ où $n \geq 0$ est le nombre de facteur égaux à x (donc n multiplications):

```
def expo_naif(x,n):
    resultat=1
    for i in range(n):
        resultat=resultat*x
    return resultat
```

On peut procéder par dichotomie : $\begin{cases} n \text{ pair} \rightarrow x^n = (x^2)^{n//2} \\ n \text{ impair} \rightarrow x^n = x(x^2)^{n//2} \end{cases}$

```
def expo_dicho(x,n):
    resultat=1
    while n!=0:
        if n%2==1:
            resultat=resultat*x
        x=x**2
        n=n//2
    return resultat
```

Là encore si $n = 2^p - 1$ alors le nombre d'itérations est en $\log_2(n)$

Commenté [AM1]: Le mot dichotomie est formé sur le grec, dikha, en deux et tomia qui signifie division. Dichotomie = division en deux parties. C'est donc une méthode basée sur la stratégie de type diviser pour régner. En écriture binaire, si on écrit $n \approx 2^{MSB}$ Alors $MSB \approx \log_2(n)$ et donc le nombre de bits nécessaire pour écrire n est $\log_2(n) + 1$. Avec la méthode dichotomique, à chaque division : $n' = \frac{n}{2} \approx 2^{MSB'}$, soit $MSB' \approx \log_2(n') \approx \log_2(n) - 1$. A chaque étape de la division on perd un digit en binaire.

Commenté [AM2]: Si on cherche le 33 dans la liste ci-dessous :
 indice - debut = 0
 indice - fin = 10
 indice - milieu = 5
 [15,16,18,19,23,24,28,29,31,33]
 Comme $33 > 24$, on travaille ensuite avec la sous-liste de droite :
 indice - debut = 6
 indice - fin = 10
 indice - milieu = 8
 [28,29,31,33]
 Comme $31 < 33$, on travaille ensuite avec la sous-liste de droite :
 indice - debut = 9
 indice - fin = 10
 indice - milieu = 9
 [31,33]

On note la nécessité du $\gg=$ pour cette dernière étape : ...

Commenté [AM4]: On peut rapidement se persuader de l'efficacité de la méthode :
 $x^{100} = (x^2)^{50} = A^{50}$
 $x^{100} = (A^2)^{25} = B^{25}$
 $x^{100} = B(B^2)^{12} = BC^{12}$
 $x^{100} = B(C^2)^6 = B(D)^6$
 $x^{100} = B(D^2)^3 = B(E)^3$
 $x^{100} = BE(E^2)^1 = BE(F)^1$
 A cette dernière étape $n = 0$
 Soient 7 étapes, on retiendra que si n est impaire alors on stoache la valeur x dans le résultat

Commenté [AM3]: Cette preuve se fait en cherchant une propriété invariante lors des différentes itérations : on parle d'invariant de boucle