Chapitre 8 : Algorithmes dichotomiques

Le mot dichotomie vient du grec et signifie division en deux parties. Dans ce chapitre, nous allons voir le principe de la dichotomie appliqué à des algorithmes. Nous commencerons par illustrer son intérêt dans le cas de la recherche d'un élément dans une liste. Puis, nous verrons comment il est possible d'envisager une méthode dichotomique pour remplacer un algorithme naïf. D'une manière générale, utiliser une méthode dichotomique constitue une stratégie de type *diviser pour régner*.

1. Recherche dichotomique

1.1.Position du problème et algorithme naïf

On dispose d'une liste `L` contenant `n` éléments et on veut tester la présence d'un élément `e` dans cette liste.

Écrire à l'aide d'une boucle `for` une fonction `presence` renvoyant un booléen traduisant la présence ou non d'un élément `e` dans une liste `L` passés en argument

Si on considère le coût de cet algorithme dit naïf, on constate que, dans le pire des cas, l'élément `e` recherché est en dernière position ; il y aura donc `n` tests de comparaison effectués. On parle alors de coût linéaire.

1.2.Principe de la recherche dichotomique

Il est possible d'accélérer grandement la recherche en utilisant une stratégie dichotomique. En revanche, cette méthode nécessite que la liste soit triée au préalable.
Le principe est de comparer l'élément recherché `e` avec l'élément situé au milieu du tableau. Si les deux éléments sont égaux alors la recherche est terminée. S'ils sont différents, on peut réduire la « zone de recherche » à la partie du tableau susceptible de contenir l'élément `e`.

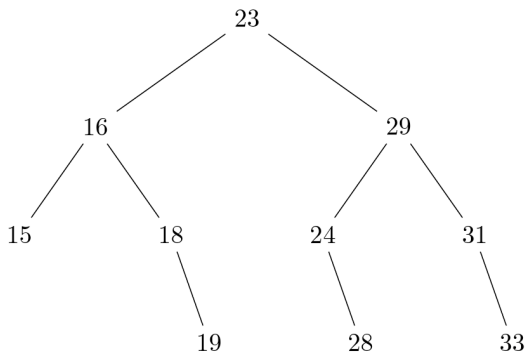
Considérons l'exemple suivant :
on cherche si l'élément `28` est présent dans la liste `[15,16,18,19,23,24,28,29,31,33]`.
Pour chaque étape, on a souligné les éléments constituant les extrémités de la partie du tableau considérée et l'élément « milieu » utilisé pour le test est grisé et désigné par une flèche verticale.

				↓						
15	16	18	19	23	24	28	29	31	33	$28 > 23$
							↓			
15	16	18	19	23	24	28	29	31	33	$28 < 29$
					↓					
15	16	18	19	23	24	28	29	31	33	$28 > 24$
						↓				
15	16	18	19	23	24	28	29	31	33	$28 = 28$

Si on cherche l'élément `27`, le processus reste le même ; seule la conclusion change.

				↓						
15	16	18	19	23	24	28	29	31	33	$27 > 23$
							↓			
15	16	18	19	23	24	28	29	31	33	$27 < 29$
					↓					
15	16	18	19	23	24	28	29	31	33	$27 > 24$
						↓				
15	16	18	19	23	24	28	29	31	33	$27 \neq 28$

Afin de visualiser l'intérêt de cette méthode, on peut effectuer une représentation sous forme d'arbre où l'on fait figurer les éventuels milieux successifs.



On constate que n'importe quel élément de la liste peut être atteint en au maximum quatre étapes. Il faut par exemple trois étapes pour trouver `31` alors qu'il en faudrait neuf avec l'algorithme naïf.

1.3.Algorithme de recherche dichotomique

On souhaite écrire une fonction dichotomie permettant de tester la présence d'un élément `e` dans une liste triée `L` passés en arguments. La fonction doit retourner un booléen.

Si on s'appuie sur l'exemple précédent, on peut imaginer créer deux entiers `g` et `d` correspondant aux indices des éléments situés aux extrémités de la partie de tableau considérée ainsi qu'un entier `m` qui correspondra à l'indice de l'élément situé milieu (ou juste avant dans la cas d'un nombre pair d'éléments dans la partie du tableau considérée).

À chaque étape, on compare `e` à l'élément « milieu » ; dans le cas où ils ne correspondent pas, on calcule les nouvelles valeurs de `g` et `d` définissant la partie du tableau à considérer pour l'étape suivante.

Écrire une fonction presence_`dichotomie` répondant aux critères précédents.

2. Résolution d'équation par une méthode dichotomique

2.1. Description du problème et principe

On cherche à résoudre de manière approchée une équation du type $f(x)=0$ sur un intervalle $[a,b]$ à ε près. La fonction est supposée monotone et continue.

Dans le cas où une solution à cette équation existe ($f(a)\times f(b)<0$), il s'agit tout comme pour la recherche dans une liste de comparer à zéro la valeur de $f(m)$ où m est le milieu de l'intervalle. S'il y a égalité alors la solution est obtenue de manière exacte. Dans le cas contraire, on réduit la recherche au sous-intervalle dans lequel l'annulation de la fonction est possible.

Si la solution exacte n'est jamais obtenue, le processus est interrompu lorsque la largeur de l'intervalle est inférieure à ε.

2.2. Programmation

Écrire une fonction `solution` permettant la recherche d'une solution de l'équation `f(x)=0` sur l'intervalle `[a,b]` avec une précision `eps`. `f`, `a`, `b` et `eps` étant passés en arguments. On pourra noter `g` et `d` les bornes de l'intervalle considéré pour la recherche.

Le programme devra tester si l'annulation de la fonction est possible sur l'intervalle considéré et retourner un message d'erreur dans le cas contraire.

3. Exponentiation rapide

3.1.Description et algorithme naïf

Il s'agit d'écrire une fonction puissance qui, pour un flottant x et entier n passés en arguments, renvoie la valeur de x^n sans passer par la commande `x**n`.
Dans une version naïve, on calcule successivement x^2, x^3 en s'appuyant sur la définition mathématique $x^n = 1 \times x \times ... \times x$ où n est le nombre de facteur égaux à x.
Écrire une fonction `puissance` en s'appuyant sur la définition mathématique précédente.

Cet algorithme effectue n multiplications et $n+1$ affectations. On a donc un coût linéaire en fonction de n.

3.2.Méthode dichotomique

Le principe d'un algorithme dichotomique est de diviser un problème en deux sous-problèmes. Ici, on constate qu'on peut rencontrer deux situations différentes suivant la parité de n.

- n pair ($n = 2k$) : on a alors : $x^n = x^{2k} = \left(x^2\right)^k$
- n impair ($n = 2k+1$) : on a alors : $x^n = x^{2k+1} = x\left(x^2\right)^k$

Dans les deux cas, on a $k = n//2$. Le calcul de $(x)^k$ peut également faire l'objet de deux situations différentes suivant la parité de k et ainsi de suite.
Écrire une fonction puissance2 en s'appuyant sur la méthode décrite précédemment.