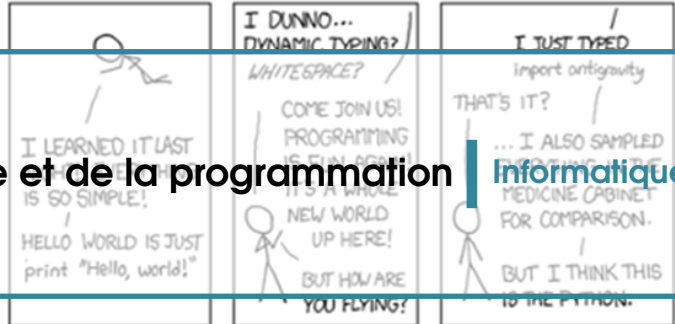
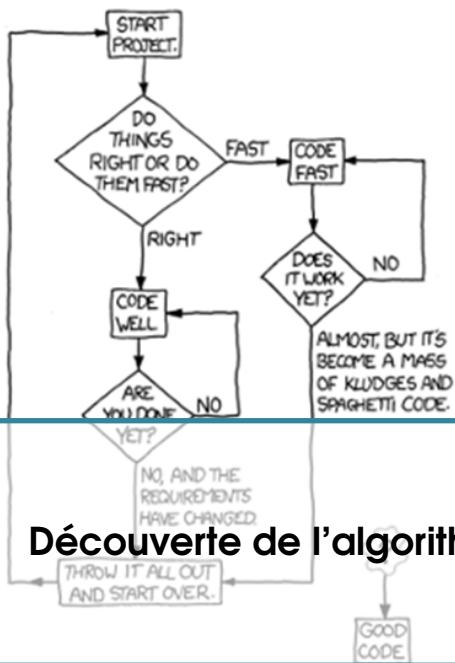


HOW TO WRITE GOOD CODE:



Découverte de l'algorithmique et de la programmation Informatique

Chapitre 5

Algorithmes dichotomiques et algorithmes gloutons

Savoirs et compétences :

- Algorithmes dichotomiques.

1	Algorithmes dichotomiques	2
2	Présentation des algorithmes Gloutons	4
3	Éléments de la stratégie gloutonne	4

1 Algorithmes dichotomiques

1.1 Introduction

Les méthodes de résolutions par un algorithme dichotomique font partie des algorithmes basés sur le principe de "diviser pour régner".

Elles utilisent la définition du terme *dichotomie* qui signifie diviser un tout en deux parties "opposées".

Certains algorithmes de tris sont basés sur ce principe de diviser pour régner.

Ce cours vous présente deux algorithmes dichotomiques :

- la recherche d'un élément dans une liste triée;
- la détermination de la racine d'une fonction quand elle existe.

1.2 Recherche dichotomique dans une liste triée : Principe

Lorsque vous cherchez le mot « hippocampe » dans le dictionnaire, vous ne vous amusez pas à parcourir chaque page depuis la lettre a jusqu'à tomber sur le mot « hippocampe »...

Dans une liste triée, il y a plus efficace! Par exemple dans le dictionnaire, vous ouvrez à peu près au milieu, et suivant si le mot trouvé est « inférieur » ou « supérieur » à « hippocampe » (pour l'ordre alphabétique), vous poursuivez votre recherche dans l'une ou l'autre moitié du dictionnaire.

Propriété On se donne une liste L de nombres de longueur n , triée dans l'ordre croissant, et un nombre x_0 .

Pour chercher x_0 , on va couper la liste en deux moitiés et chercher dans la moitié intéressante et ainsi de suite.

On appelle g l'indice de l'élément du début de la sous-liste dans laquelle on travaille et d l'indice de l'élément de fin.

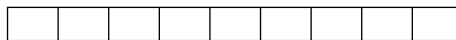
Au début, $g = 0$ et $d = n - 1$

On souhaite construire un algorithme admettant l'invariant suivant :

si x_0 est dans L alors x_0 est dans la sous-liste $L[g : d]$ (g inclus et d exclu).

On va utiliser la méthode suivante :

- On compare x_0 à "l'élément du milieu" : c'est $L[m]$ où $m = (g + d) // 2$ son indice est $m = n // 2$ (division euclidienne)



- Si $x_0 = L[m]$, on a trouvé x_0 , on peut alors s'arrêter.
- Si $x_0 < L[m]$, c'est qu'il faut chercher dans la première moitié de la liste, entre $L[g]$ et $L[m-1]$ ($L[m]$ exclu).
- Si $x_0 > L[m]$, c'est qu'il faut chercher dans la seconde moitié de la liste, entre $L[m+1]$ et $L[d]$ ($L[m]$ exclu).

On poursuit jusqu'à ce qu'on a trouvé x_0 ou lorsque l'on a épuisé la liste L .

1.3 Exemples d'application

Indiquer pour les deux exemples suivants les valeurs successives de g et d :

1. $x_0 = 5$ et $L = [-3, 5, 7, 10, 11, 14, 17, 21, 30]$

$$\begin{cases} g = 0 \\ d = 8 \\ m = 4, L[m] > x_0 \end{cases} \quad \begin{cases} g = 0 \\ d = 3 \\ m = 1, L[m] = x_0 \end{cases} .$$

C'est fini, on a bien trouvé x_0 dans la liste.

2. $x_0 = 11$ et $L = [-2, 1, 2, 7, 8, 10, 13, 16, 17]$

$$\begin{cases} g = 0 \\ d = 8 \\ m = 4, L[m] < x_0 \end{cases} , \begin{cases} g = 5 \\ d = 8 \\ m = 6, L[m] > x_0 \end{cases} \quad \begin{cases} g = 5 \\ d = 5 \\ m = 5, L[m] < x_0 \end{cases} \quad \begin{cases} g = 6 \\ d = 5 \end{cases} .$$

C'est fini, on a épuisé la liste L et on n'a pas trouvé x_0 .

1.4 Implémentation en Python

La fonction `recherche_dichotomie` d'arguments une liste L et un élément x_0 renvoyant un booléen disant si x_0 est dans la liste L est proposée :

```
def recherche_dichotomie(L:list, x0:int)-> bool:
    n = len(L)
    g_ind = 0 # c'est l'indice de gauche
```

```
d_ind = n - 1 # c'est l'indice de droite
rep = False
while g_ind <= d_ind and rep == False:
    # si x0 est dans L alors L[g_ind] <= x0 <= L[d_ind] {invariant}
    m = (d_ind + g_ind) // 2
    if x0 == L[m]:
        rep = True
    elif x0 < L[m]:
        d_ind = m - 1
    else:
        g_ind = m + 1
    # si x0 est dans L alors L[g_ind] <= x0 <= L[d_ind] {invariant}
return(rep)
```

Remarque : La terminaison de l'algorithme est obtenue avec $d - g$ qui est un entier positif qui décroît strictement à chaque passage dans la boucle `while` et joue le rôle de variant.

1.5 Détermination de la racine d'une fonction par dichotomie

1.5.1 Principe théorique de la méthode par dichotomie

On considère une fonction f vérifiant :

f continue sur $[a, b]$; $f(a)$ et $f(b)$ de signes opposés.

Le théorème des valeurs intermédiaires nous assure que f possède au moins un zéro ℓ entre a et b . La preuve, vue en cours de mathématiques, repose sur la méthode de dichotomie. Prenons le cas $f(a) < 0$ et $f(b) > 0$ et posons

$g_0 = a$, $d_0 = b$. On considère $m_0 = \frac{g_0 + d_0}{2}$ et on évalue $f(m_0)$:

- Si $f(m_0) \geq 0$, on va poursuivre la recherche d'un zéro dans l'intervalle $[g_0, m_0]$
On pose donc : $g_1 = g_0$; $d_1 = m_0$

- Sinon, la recherche doit se poursuivre dans l'intervalle $[m_0, d_0]$
On pose donc : $g_1 = m_0$; $d_1 = d_0$

- On recommence alors en considérant $m_1 = \frac{g_1 + d_1}{2}$...

1. Par quoi peut-on remplacer la condition " $f(m_k) \geq 0$ " dans le cas général où $f(a)$ et $f(b)$ sont de signes contraires (pas forcément $f(a) < 0$ et $f(b) > 0$) ?
2. À quelle condition sur g_n et d_n s'arrête-t-on, si l'on souhaite que g_n et d_n soient des solutions approchées de ℓ à une précision ε ?
3. Au lieu de renvoyer g_n et/ou d_n comme valeurs approchées de ℓ , que pourrait-on prendre ?
Que mettre comme condition d'arrêt pour avoir une précision ε ?
4. Un étudiant propose de tester si $f(m_k) = 0$. Qu'en pensez-vous ?

1.6 Implémentation en Python et avec scipy

Écrivons une fonction `zero_dichotomie(f: function, a: float, b: float, epsilon: float) -> float` d'arguments une fonction f , des flottants a et b (tels que $a < b$), et la précision voulue ϵ (flottant strictement positif). Cette fonction renverra une valeur approchée à ϵ près d'un zéro de f , compris entre a et b , obtenue par la méthode de dichotomie.

```
def zero_dichotomie(f: function, a: float, b: float, epsilon: float):
    val g = a # c'est un flottant
    val d = b # c'est un flottant
    while val d - val g > 2 * epsilon :
        m = (val g + val d) / 2
        if f(val g) * f(m) <= 0:
            val d = m
        else:
            val g = m
    return ((val g + val d) / 2)
```

Effectuons un test avec la fonction $f : x \mapsto x^2 - 2$ sur l'intervalle $[1, 2]$, avec une précision de 10^{-6} :

```
def f(x):
    return(x ** 2 - 2)
print (zero_dichotomie(f, 1, 2, 10**(-6)))

# il s'affichera : 1.4142141342163086
```

Une telle fonction est déjà prédéfinie dans la bibliothèque `scipy.optimize`, la fonction `bisect` (la méthode de dichotomie s'appelle aussi la méthode de la *bisection*) :

```
import scipy.optimize as spo\\[2mm]
print (spo.bisect(f, 1, 2)) \\
# il s'affichera : 1.4142135623724243
```

La précision est un argument optionnel (à mettre après `f`, `a` et `b`) et vaut 10^{-12} par défaut.

2 Présentation des algorithmes Gloutons

En informatique, un *algorithme glouton* (*greedy algorithm* en anglais, parfois appelé aussi *algorithme gourmand*) est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local. Dans certains cas, cette approche permet d'arriver à un optimum global, mais dans le cas général c'est une heuristique¹ (Wikipédia).

Les algorithmes gloutons sont utilisés dans des problèmes d'optimisation. Un problème d'optimisation consiste à déterminer les valeurs des paramètres permettant de :

- minimiser ou maximiser une fonction objectif;
- satisfaire une ou des fonctions contraintes (il existe des problèmes avec ou sans contrainte).

2.1 Choix d'itinéraire

On peut citer par exemple le problème du choix d'itinéraire dans un réseau routier : il faut déterminer le trajet de sorte à minimiser le temps de parcours, tout en respectant un ensemble de contraintes : circuler sur les routes (on ne coupe pas à travers champs!) et respecter les sens de circulation.

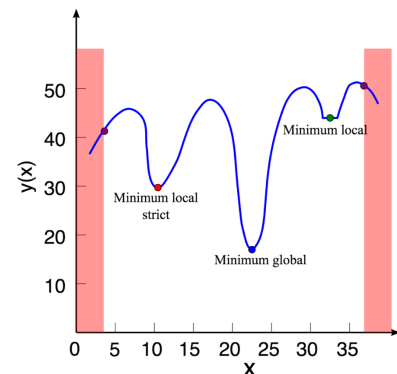
2.2 Recherche d'un minimum d'une fonction

Voici un second exemple :

- La recherche du minimum de la fonction $f : x \mapsto x^2 - x - 2$ est un problème d'optimisation sans contrainte.
- La recherche du minimum de la fonction $f : x \mapsto x^2 - x - 2$ sous la contrainte $x^3 \geq 8$ est un problème d'optimisation sous contrainte (contrainte d'inégalité). L'optimum est alors $x = 2$ et $f(2) = 0$. L'ensemble des solutions vérifiant les contraintes sont appelées solutions valides ou valables.

On distingue également plusieurs types d'optimum :

- optimum global;
- optimum local strict (unique dans un intervalle réduit);
- optimum local (non unique dans un intervalle réduit).



La méthode gloutonne est très puissante et fonctionne bien pour toutes sortes de problèmes. L'algorithme de Dijkstra, que l'on étudiera en fin d'année, qui calcule des plus courts chemins à origine unique, peut être vu comme une application de la méthode gloutonne.

2.3 Éléments de la stratégie gloutonne

Le processus pour mettre au point un algorithme glouton présente plusieurs étapes :

1. Détermination de la sous-structure optimale du problème;
2. Développement d'une solution formulée par récurrence;
3. Démonstration que, si nous avons fait un choix glouton, il ne reste qu'un seul sous-problème;
4. Démonstration qu'il est toujours sûr de faire le choix glouton. (Les étapes 3 et 4 peuvent se faire dans n'importe quel ordre.)
5. Écriture d'un algorithme récursif ou itératif.

1. une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile.

2.3.1 Propriété de choix glouton

La première caractéristique principale est la propriété de choix glouton : on peut assembler une solution globalement optimale en effectuant des choix localement optimaux (gloutons). Autrement dit, quand on considère le choix à faire, on fait le choix qui paraît le meilleur pour le problème courant, sans tenir compte des résultats des sous-problèmes.

2.3.2 Exemple : un problème d'organisation

source : algorithmes gloutons - eduscol

Des conférenciers sont invités à présenter leurs exposés dans une salle. Mais leurs disponibilités ne leur permettent d'intervenir qu'à des horaires bien définis. Le problème est de construire un planning d'occupation de la salle avec le plus grand nombre de conférenciers.

Désignons par n , entier naturel non nul, le nombre de conférenciers. Chacun d'eux, identifié par une lettre C_i , où i est un entier compris entre 0 et $n - 1$, est associé à un intervalle temporel $[d_i, f_i[$ où d_i et f_i désignent respectivement l'heure de début et l'heure de fin de l'intervention. Afin de dégager une tactique de résolution du problème, commençons par analyser plusieurs situations.

2.3.3 Les créneaux ne se chevauchent pas

Une telle situation est simple puisque tous les conférenciers peuvent intervenir sur des créneaux horaires disjoints.

2.3.4 Les créneaux se chevauchent

Dans cette situation, les intervalles ne sont plus disjoints. Nous dirons que ces intervalles ne sont pas compatibles. Des choix doivent être faits et certains conférenciers peuvent ne pas être retenus pour construire un planning.