1 Recherche en table par balayage

1.1 Problème de la recherche en table

Une **table** désigne une liste ou un tableau d'éléments.
Le **problème de la recherche en table** est celui de la recherche d'un élément appelé clef dans la table.
Avec le tri c'est un problème majeur en informatique puisque les capacités de stockage actuelles permettent de conserver des grandes quantités dans lesquelles on besoin de rechercher rapidement des informations.

1.2 Recherche en table par balayage (ou séquentielle)

Cet algorithme est simple : on parcourt le tableau du début à la fin en comparant chaque élément rencontré à la clef recherchée.
On peut parcourir tout le tableau ou stopper la recherche dès que la clef est trouvée.
Cet algorithme permet d'effectuer une recherche dans un tableau ordonné ou non (relation d'ordre sur les entiers, ordre alphabétique sur les chaînes de caractères ...). Si le tableau est ordonné, on peut stopper la recherche si la clef n'a pas été trouvée et que l'élément courant du tableau est supérieur (pour l'ordre croissant) à la clef selon la relation d'ordre considérée.
Les algorithmes de tri étudiés dans le chapitre suivant permettront d'ordonner les éléments d'une liste selon une relation d'ordre.

Exercice 1

Les trois fonctions ci-dessous implémentent la recherche séquentielle d'un caractère dans une chaîne de caractères. Elles fonctionnent aussi pour la recherche d'un élément dans une liste.

```
def appartient1(caractere,chaine):
    rep = False
    for c in chaine:
        if c==caractere:
            rep=True
    return rep
```

```
def appartient2(caractere,chaine):
    for c in chaine:
        if c==caractere:
            return True
    return False
```

```
def appartient3(caractere,chaine):
    i = 0
    rep = False
    while i<len(chaine) and not rep:
        if chaine[i]==caractere:
            rep=True
        i += 1
    return rep
```

1. Combien de comparaisons sont effectuées par chaque fonction pour rechercher le caractère 'a' dans la chaîne 'bbbba' ? dans la chaîne 'bbabb' ? dans la chaîne 'abbb' ?

2. Ecrire une fonction `maximum(tab)` qui retourne le maximum d'un tableau d'entiers (dans le désordre).
   Ecrire une fonction `pos_maximum(tab)` qui retourne le maximum d'un tableau d'entiers (dans le désordre)

et la liste des positions où il est atteint.
Peut-on éviter de parcourir tout le tableau pour rechercher le maximum d'un tableau d'entiers non trié ?

Exercice 2 *Recherche séquentielle dans un annuaire*

On pourra travailler avec le fichier `annuaireJunier.csv`.

1. Avec `LibreOffice`, créer un annuaire dans une nouvelle feuille de tableur en saisissant à partir de la ligne 1 : une série de prénoms dans la colonne A associée à une série de numéros de téléphones dans la colonne B. Enregistrer ce fichier au format `csv` sous le nom `annuaire.csv` en choisissant ',' comme séparateur de champ et rien comme séparateur de texte.

2. Compléter la fonction `extraire(fichier)` qui retourne une liste (ou tableau) de prénoms et une liste (ou tableau) de numéros extraites d'un fichier de type `annuaire.csv`.

3. Rajouter une fonction `recherche_sequentielle(clef,liste1,liste2)` au script précédent. Cette fonction prend pour paramètre une clef qu'elle recherche dans `liste1`.
Elle retourne le couple `(clef,liste2[i])` si clef apparaît en position `i` dans `liste1`.Si `liste1` et `liste2` sont respectivement la liste de prénoms et la liste de numéros extraites du fichier `annuaire.csv`, on peut ainsi faire une recherche dans l'annuaire par prénoms (on suppose que tous les prénoms sont distincts quitte à les numéroter comme 'Fred1', 'Fred2').
Tester cette fonction pour faire une recherche par prénom dans le fichier `annuaire.csv`.

4. Comment peut-on utiliser la fonction `recherche_sequentielle(clef,liste1,liste2)` pour faire une recherche inversée (par numéro) dans le fichier `annuaire.csv`.

5. Tester la fonction `recherche_sequentielle` pour rechercher les clefs 'Alphonse' puis 'Zied' dans le fichier `annuaireJunier.csv`. Combien de comparaisons avec les éléments du tableau `prenom` sont effectuées dans chaque cas ?

6. Si l'annuaire comporte n prénoms, combien cet algorithme effectue-t-il de comparaisons si le prénom cherché n'est pas dans l'annuaire ? est en première position (meilleur des cas) ? est en dernière position (pire des cas) ?.
Combien de comparaisons sont-elles effectuées en moyenne si on choisit un prénom au hasard parmi les prénoms dans l'annuaire ?

7. Estimer le temps nécessaire pour une recherche séquentielle dans un annuaire de 30 000 prénoms : en moyenne et dans le pire des cas.
Chaque tour de boucle de l'algorithme s'effectuant à coût constant en terme de comparaisons et d'affectations on peut mesurer sa **complexité** (ou son coût) par le nombre d'itérations de la boucle Tant Que.
Expliquer pourquoi on peut qualifier la **complexité** (ou coût) moyenne de cet algorithme de **linéaire**.

```python
def extraire(fichier):
    """Extrait d'un fichier texte annuaire avec des lignes du type
    prenom,0478010101\n l la liste des prenoms et celle des numéros de
    telephone"""
    f = open(fichier,'r')
    prenom,tel = [],[] #initialisation des listes de prenoms et de numeros
    for ligne in f:
        p,n =ligne.rstrip().split(',')
        #write your code here

def recherche_sequentielle(clef,liste1,liste2):
    """recherche de façon séquentielle si clef appartient à la liste1 de
    l'annuaire constitué des deux listes liste1 et liste2.
    Si clef est trouvée dans liste1 à la position i,
    retourne le couple clef,liste2[i],sinon retourne None"""
    #write your code here
```

exo2_rechercheannuaire.py

2 Recherche en table par dichotomie

2.1 Principe de la recherche en table par dichotomie

On considère désormais qu'on effectue une recherche dans un annuaire dont les noms sont rangés dans l'ordre alphabétique ou plus généralement une recherche dans une table triée.
Le nom *dichotomie* provient du grec ancien *tomia*, couper et *dikha*, en deux. Pour chercher le numéro correspondant à un nom dans cet annuaire on peut appliquer **l'algorithme de recherche par dichotomie** :

- si l'annuaire est vide on arrête l'algorithme
- sinon on ouvre l'annuaire au milieu et on compare avec le nom médian :
  - si le nom médian est égal au nom cherché on retourne le numéro de téléphone correspondant, la recherche est terminée ;
  - si le nom médian est après le nom cherché, on élimine la seconde moitié de l'annuaire et on applique la procédure de recherche dichotomique à la première moitié ;
  - si le nom médian est avant le nom cherché, on élimine la première moitié de l'annuaire et on applique la procédure de recherche dichotomique à la seconde moitié ;

Exercice 3 *Jeu du juste prix*

L'algorithme ci-après modélise un jeu qui consiste à deviner un entier choisi au hasard par l'ordinateur entre 0 et 100.

1. Programmer cet algorithme en Python puis le tester.

   Remarque :
   La fonction `randrange(0,101)` du module `random` retourne un entier au hasard entre 0 et 100 compris. (on peut aussi utiliser `randint(0,100)`).

2. Ecrire un nouveau programme où l'ordinateur détermine tout seul l'entier choisi au hasard en appliquant une recherche dichotomique.
Que peut-on dire du nombre maximal d'étapes nécessaires pour déterminer par dichotomie un entier choisi au hasard entre 0 et 100 ?

```
Variables :
   S,N,Essai, TestFin

Début Algorithme

Initialisation :
   S = randrange(0,101)
   Essai = 1
   TestFin = False

   Tant que Essai⩽ 6 ET TestFin==False Faire
      Saisir N
      Si N>S alors
            Affiche « C'est moins »
      Si N<S
            alors Affiche « C'est plus »
      Si N==S alors
            Affiche « C'est gagné »
            TestFin prend la valeur True
      Essai prend la valeur Essai+1

Si TestFin==False alors Affiche « C'est perdu »

Fin Algorithme
```

2.2 Pseudo-code de l'algorithme de recherche dichotomique (version itérative)

On considère la recherche d'une *valeur* dans une *liste* (ou tableau) de n éléments indexés de 0 à $n-1$, ces éléments étant rangés dans l'ordre croissant pour une relation d'ordre total.
L'algorithme de recherche dichotomique peut être codé sous la forme d'une fonction de paramètres *liste* et *clef*.
Dans le corps de la fonction on définit des variables locales : *bas* pour l'indice du plus petit élément et *haut* pour l'indice du plus grand élément de la sous-liste où s'effectue la recherche en cours.
La boucle **Tant Que** s'arrête lorsque $bas \geqslant haut$ c'est-à-dire lorsque la sous-liste de recherche est réduite à un élément ou vide. Ensuite l'algorithme retourne l'indice de l'élément correspondant lorsque la valeur est trouvée (on peut retourner le numéro de téléphone associé si la valeur est un nom recherché dans un annuaire), sinon elle retourne None (objet vide) ou "inconnu".
Voici le pseudo-code (les commentaires suivent les caractères #) :

Algorithme de recherche dichotomique

```
Recherche_Dichotomique(liste,clef)
    bas = 0
    haut = longueur(liste)-1    # longueur(liste) est la longueur de la liste
    Tant Que bas < haut
        med = ⌊ (bas + haut)/2 ⌋    # c'est la partie entière de (bas + haut)/2
        Si clef == liste[med]
            bas = med
            haut = med
        Sinon
            Si clef > liste[med]
                bas = med + 1
            Sinon
                haut = med - 1
    Si clef == liste[bas]
        Retourner bas    # ou Retourner la valeur d'une variable globale associée
    Sinon
        Retourner "Inconnu"    # ou Retourner None
```

Exercice 4

Soit un tableau de 17 entiers triés dans l'ordre croissant :

```
tab = [10, 13, 19, 28, 29, 32, 34, 37, 39, 42, 48, 49, 72, 81, 86, 93, 96]
```

1. Appliquer à la main l'algorithme de dichotomie pour rechercher l' entier 19 dans ce tableau. On fera un tableau d'évolution des variables `bas`, `haut` et `med`. Combien de tours de boucles ont-été effectués ?
2. Faire de même pour rechercher l'entier 20 dans ce tableau.

2.3 Preuve de terminaison et de correction de l'algorithme de recherche dichotomique

A chaque tour de boucle, on divise par 2 la longueur de la sous-liste où s'effectuera la recherche au prochain tour. Donc en un nombre d'itérations inférieur ou égal à $\log_2(n)$, la longueur de la sous-liste de recherche devient inférieure à 1 et la boucle Tant Que s'arrête puisque le test d'arrêt impose que la sous-liste contienne au moins deux éléments.
Par conséquent **l'algorithme se termine.**

Prouvons que l'algorithme est correct.
Pour cela, montrons que la propriété $\mathscr{P}_i$ suivante est un invariant de la boucle Tant Que :

$\mathscr{P}_i$: « Avant le tour de boucle i, si la clef recherchée appartient à la liste initiale alors elle appartient à la sous-liste comprise entre les positions *bas* et *haut* »

Initialisation

Avant la boucle Tant Que, la liste de recherche est la liste initiale complète donc il est évident que $\mathscr{P}_1$ est vraie.

Hérédité

Supposons que $\mathscr{P}_i$ est vraie pour un entier naturel i : avant le tour de boucle i, si la clef est dans la liste, elle se trouve entre les positions *bas* et *haut*.
Lors du tour de boucle i, on compare la clef avec la valeur en position médiane $\lfloor (bas + haut)/2 \rfloor$ entre *bas* et *haut*.

- Si la valeur médiane est égale à la clef alors *bas* et *haut* prennent la valeur *med* et la clef se trouve bien entre les nouvelles valeurs de *bas* et *haut* .
- Sinon, puisque les valeurs entre les positions *bas* et *haut* sont rangées dans l'ordre croissant, si la clef est dans la liste, sa comparaison avec la valeur médiane permet de changer les valeurs de *bas* ou de *haut* pour ne garder que la sous-liste susceptible de la contenir.

Dans tous les cas, si la clef appartient à la liste initiale, elle appartiendra nécessairement à la sous-liste allant des nouvelles positions bas et haut choisies par le tour de boucle i.
Ainsi avant le tour de boucle $i+1$, la propriété $\mathscr{P}_{i+1}$ sera vraie, c'est donc une propriété héréditaire.

Terminaison

Si on note r le numéro du dernier tout de boucle, la propriété $\mathscr{P}_{r+1}$ est vraie par hérédité et donc si la clef est dans la liste initiale elle se trouve entre les positions *bas* et *haut*.
Lorsqu'on sort de la boucle Tant Que on a $bas \geqslant haut$ c'est-à-dire $bas > haut$ ou $bas == haut$.

- Si $bas > haut$ alors la dernière sous-liste de recherche susceptible de contenir la clef est vide et donc la clef n'est pas dans la liste.
- Si $bas == haut$, la dernière sous-liste de recherche est réduite à un élément et la seule valeur susceptible d'être la clef est *liste*[bas].

Après la boucle Tant Que, la comparaison de *clef* avec *liste*[bas] permet donc de déterminer si la clef est dans la liste.
Ainsi la propriété $\mathscr{P}_i$ est un invariant de boucle et l'algorithme est correct.

2.4 Implémentation de l'algorithme

Exercice 5

1. Compléter le programme suivant avec une fonction `recherche_sequentielle` ainsi qu'une fonction `recherche_dichotomique` qui prennent pour paramètres une liste d'entiers et une clef (un entier) et qui retournent la position de la clef dans la liste si elle appartient à la liste ou un message d'erreur sinon. En utilisant le décorateur `timetest`, comparer le temps d'exécution des deux fonctions pour des listes d'entiers aléatoires triées de tailles 2^p avec $p \in \{5;8;10;12;15;20;25\}$.
2. Rajouter au programme `recherchenombre.py` une fonction `recherche_dichotomiquerec` qui implémente de façon récursive l'algorithme de rechercher dichotomique d'une clef dans une liste d'entiers triés.

```
import time ,from random import randint

def timetest(fonction):
    """exécute la fonction et affiche son temps d'exécution """
    def fonction_modifiee(*args,**kargs):
        tps_avant = time.time()
        ret= fonction(*args,**kargs)
        tps_apres = time.time()
        print("Le temps d'exécution de la fonction est de {:10.3e} s"
              .format(tps_apres-tps_avant))
        return ret
    return fonction_modifiee

@timetest
def recherche_sequentielle(liste,clef):
    ....

@timetest
def recherche_dichotomique(liste,clef):
    ....

n = int(input('Entrez la taille souhaitée pour la liste d\'entiers aléatoires
     : \n'))
clef = int(input('Entrez la clef qu\'il faut chercher : \n '))
liste = [randint(0,1000) for i in range(n)] #liste de test
liste.sort() #on trie dans l'ordre croissant la liste de test
copieliste = liste[:] #copie physique de la liste de tests
print('Résultat de la recherche séquentielle :')
print(recherche_sequentielle(liste,clef))
print('Résultat de la recherche dichotomique :')
print(recherche_dichotomique(copieliste,clef))
```

2.5 Analyse de complexité de l'algorithme de recherche dichotomique

Exercice 6

Avec l'algorithme de recherche dichotomique, on recherche un nom dans un annuaire de 30 000 noms qui sont rangés dans l'ordre alphabétique.

1. Compléter l'énumération suivante qui permet de déterminer le nombre d'étapes nécessaires pour trouver le nom ou découvrir que le nom n'est pas dans l'annuaire :

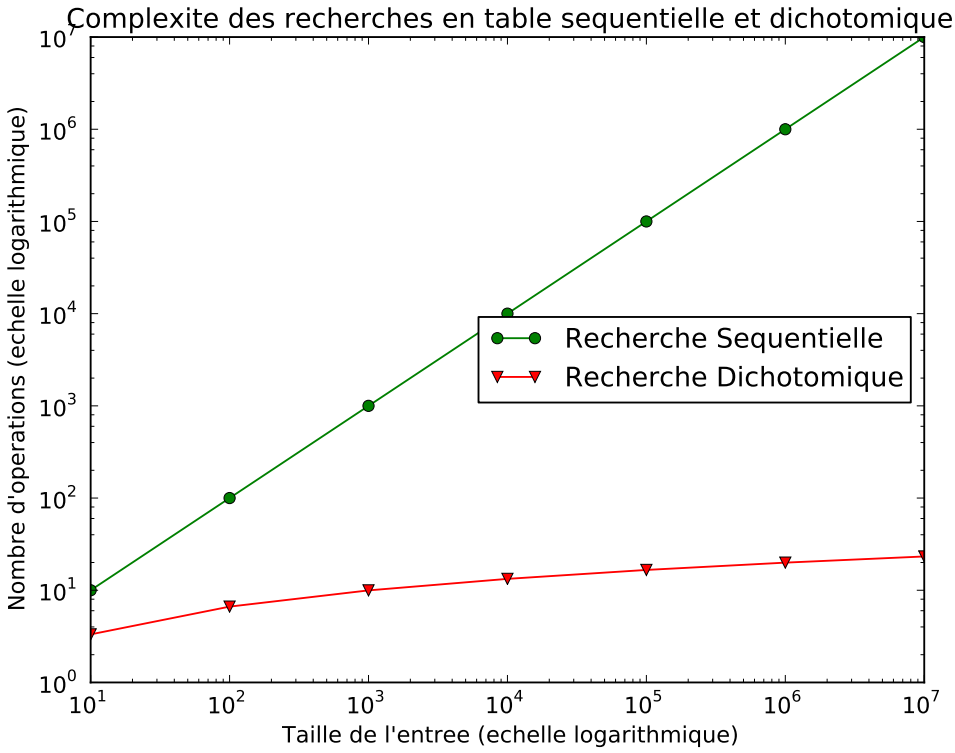
   - Après 1 comparaison, on le cherche dans un ensemble d'au plus 15 000 mots.
   - Après 2 comparaisons, on le cherche dans un ensemble d'au plus 7 500 mots.
   - Après 3 comparaison, on le cherche dans un ensemble d'au plus 3 750 mots.
   - Après 4 comparaisons, on le cherche dans un ensemble d'au plus 1 875 mots ...

2. Dans l'algorithme de recherche dichotomique, quelle est la relation entre la taille de l'ensemble de recherche à l'étape i et à l'étape $i+1$?

3. Si la taille de l'ensemble de recherche initial est 2^{16} majorer le nombre d'étapes nécessaire à l'exécution de la recherche dichotomique.
Même question si la taille de l'ensemble est 1000 puis si elle est de 2^p, de n .

4. Le logarithme entier d'un entier naturel n est le nombre de fois qu'il faut diviser n par 2 pour obtenir un nombre inférieur ou égal à 1, c'est $\log_2(n) = \lceil \frac{\ln(n)}{\ln(2)} \rceil$ c'est-à-dire l'arrondi à l'entier supérieur de $\log_2(n)$ le logarithme binaire de n.
Chaque étape de l'algorithme s'effectuant à coût constant en terme de comparaisons et d'affectations on peut mesurer sa **complexité** (ou son coût) par le nombre d'étapes nécessaires.
Expliquer pourquoi la **complexité** de l'algorithme de recherche dichotomique est au pire **logarithmique**.

5. Ecrire un programme en Python affiche les valeurs de 10^p et $\log_2(10^p)$ pour p variant entre 0 et 20.
En déduire une comparaison de l'efficacité des algorithmes de rechercher séquentielle ou dichotomique.



3 D'autres applications de la dichotomie

3.1 Conversion analogique-numérique

Les signaux électriques sont analogiques, ce sont des fonctions continues. Pour être utilisé par un appareil numérique comme un ordinateur, un appareil photo numérique ..., le signal analogique (par exemple une tension,) doit être discrétisé en une suite finie de bits représentant une valeur d'une échelle prédéfinie. L'appareil qui réalise une telle conversion est un convertisseur analogique-numérique. Pour déterminer une valeur

approchée de la tension analogique en la comparant aux valeurs de sa table de référence, le convertisseur peut utiliser un algorithme de recherche dichotomique.

3.2 Recherche d'un encadrement du zéro d'une fonction par dichotomie

Soit f une fonction continue et strictement monotone sur un intervalle $[a\,;b]$. Si f change de signe entre a et b, un corollaire du théorème des valeurs intermédiaires permet d'affirmer que l'équation $f(x)=0$ possède une unique solution α dans $[a\,;b]$.

Exercice 7

Soit f une fonction fonction continue, strictement monotone et changeant de signe sur un intervalle $[a\,;b]$.
On peut choisir par exemple f définie sur $[0\,;\pi]$ par $f(x)=\cos x - x$

1. Soit une fonction `dichotomie` qui prend en paramètre une fonction f,trois flottants a,b ,*précision*. Ecrire en Python le corps de cette fonction pour qu'elle détermine par dichotomie un encadrement d'amplitude strictement inférieure à *précision* de l'unique solution α de $f(x)=0$ dans $[a\,;b]$. Faire afficher également le nombre d'étapes nécessaires.

2. Justifier que l'équation $\cos x = x$ possède une unique solution α dans $[0\,;4]$.
Appliquer la fonction `dichotomie` précédente pour déterminer un encadrement de α d'amplitude strictement inférieure 0.001.

Algorithme *Approximation par dichotomie de la solution de $f(x)=0$ sur [a ;b]*

Entrée(s) $a, b, precision$
  tant que $b-a >$ precision **faire**
    $\frac{a+b}{2} \to m$
    si $f(a)\times f(m)<0$ **alors**
      $m \to b$
    sinon si $f(a)\times f(m)>0$ **alors**
      $m \to a$
    sinon
      $m \to a$
      $m \to b$
    fin du si
  fin du tant que
Sortie(s) a, b

4 Bibliographie

J'ai picoré des informations dans les ouvrages suivants pour préparer ce cours :

- *Informatique et sciences du numérique* de Gilles Dowek et Jean-Pierre Archambault aux éditions Eyrolles.
- *Informatique pour tous en classes préparatoires scientifiques* de Benjamin Wack, Gilles Dowek, Stéphane Gonnord et Judicael Courant aux éditions Eyrolles.
- *Algorithmique* de Cormen, Leiserson, Rivest, Stein aux éditions Dunod.

Table des matières