Trier et rechercher

INFORMATIQUE COMMUNE - TP nº 1.4 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ coder un algorithme de tri simple et explicité
- ☞ évaluer le temps d'exécution d'un algorithme avec la bibliothèque time
- ☞ rechercher un élément dans un tableau séquentiellement ou par dichotomie itérative
- ☞ générer un graphique légendé avec la bibliothèque matplotlib

A Trier un tableau

A1. On souhaite trier des listes Python, considérées ici comme des tableaux, avec des algorithmes différents (cf. algorithmes 1, 2 et 3). Chaque algorithme de tri est implémenté par une fonction Python. Le prototype de ces fonctions est `my_sort(t)`, où `t` est un paramètre formel qui représente le tableau à trier.

```
def my_sort(t):
  # tri du tableau
  # for i in range(len(t))
  #     t[i] = ...
```

Cette fonction, une fois réalisée, trie le tableau `t` passé en paramètre mais ne renvoie rien (i.e. pas de `return`). Expliquer pourquoi.

A2. Coder les algorithmes de tri par sélection, par insertion et par comptage en respectant le prototype défini à la question précédente [1].

A3. Tester ces algorithmes sur une **même** liste Python de longueur 20 et contenant de types `int` choisis aléatoirement entre 0 et 100.

A4. Peut-t-on trier des listes de chaînes de caractères avec ces mêmes codes ? Tester cette possbilité à l'aide de la liste `["Zorglub", "Spirou", "Fantasio", "Marsupilami", "Marsu", "Samovar", "Zantafio"]`. Analyser les résultats. Pourquoi est-ce possible ? Pourquoi n'est-ce pas possible ?

La bibliothèque `matplotlib` permet de générer des graphiques à partir de données de type `list` qui constituent les abscisses et les ordonnées associées. La démarche à suivre est de :

- importer la bibliothèqe `from matplotlib import pyplot as plt`
- créer une figure `plt.figure()`
- tracer une courbe `plt.plot(x,y)` si `x` et `y` sont les listes des abscisses et des ordonnées associées. La bibliothèque trace les points `(x[i], y[i])` sur le graphique.

1. On a le droit de collaborer, de se répartir les algorithmes et de s'échanger les codes s'ils sont corrects !

Algorithme 1 Tri par sélection

```
Fonction TRIER_SELECTION(t)
    n ← taille(t)
    pour i de 0 à n − 1 répéter
        min_index ← i                          ▷ indice du prochain plus petit
        pour j de i + 1 à n − 1 répéter        ▷ pour tous les éléments non triés
            si t[j] < t[min_index ] alors
                min_index ← j                  ▷ c'est l'indice du plus petit non trié !
        échanger(t[i], t[min_index])           ▷ c'est le plus grand des triés !
```

Algorithme 2 Tri par insertion

```
Fonction TRIER_INSERTION(t)
    n ← taille(t)
    pour i de 1 à n-1 répéter
        à_insérer ← t[i]
        j ← i
        tant que t[j-1] > à_insérer et j>0 répéter
            t[j] ← t[j-1]                      ▷ faire monter les éléments
            j ← j-1
        t[j] ← à_insérer                       ▷ insertion de l'élément
```

Algorithme 3 Tri par comptage

```
Fonction TRIER_COMPTAGE(t, v_max)              ▷ v_max est le plus grand entier à trier
    n ← taille(t)
    comptage ← un tableau de taille v_max + 1 initialisé avec des zéros
    résultat ← un tableau de taille n
    pour i de 0 à n − 1 répéter                ▷ compter chaque valeur du tableau.
        key ← t[i]
        comptage[key] ← comptage[key] + 1
    i ← 0                                      ▷ Indice du tableau résultat
    pour key de 0 à v_max − 1 répéter          ▷ pour chaque valeur du tableau, dans l'ordre)
        pour j de 0 à comptage[key] répéter    ▷ autant de fois que la valeur est présente
            résultat[i] ← key
            i ← i+1
    renvoyer résultat
```

- ajouter les éléments de légende et de titre,
- montrer la figure ainsi réalisée `plt.show()`.

La bibliothèque `time` permet notamment de mesurer le temps d'exécution d'un code. Un exemple de code utilisant ces deux bibliothèques est donné ci-dessous. Le graphique qui en résulte est montré sur la figure 1.

Code 1 – Example d'utilisation des bibliothèques time et matplotlib

```python
import time
from matplotlib import pyplot as plt


def to_measure(d):
    time.sleep(d)  # Do nothing, wait for d seconds


# Simple use
tic = time.perf_counter()
to_measure(0.1)
toc = time.perf_counter()

print(f"Execution time : {toc - tic} seconds")

# Plotting results
timing = []
delay = [d / 1000 for d in range(1, 100, 5)]
for d in delay:
    tic = time.perf_counter()
    to_measure(d)
    toc = time.perf_counter()
    timing.append(toc - tic)

plt.figure()
plt.plot(delay, timing, color='cyan', label='fonction to_measure')
plt.xlabel('Delay', fontsize=18)
plt.ylabel("Execution time", fontsize=16)
plt.legend()
plt.show()
```

A5. À l'aide de la bibliothèque matplotlib, tracer les temps d'exécution nécessaires au tri d'un même tableau d'entiers par les algorithmes implémentés. On pourra également les comparer à la fonction `sorted` de Python. Analyser les résultats. Essayer de qualifier les coûts des algorithmes en fonction de la taille du tableau d'entrée.

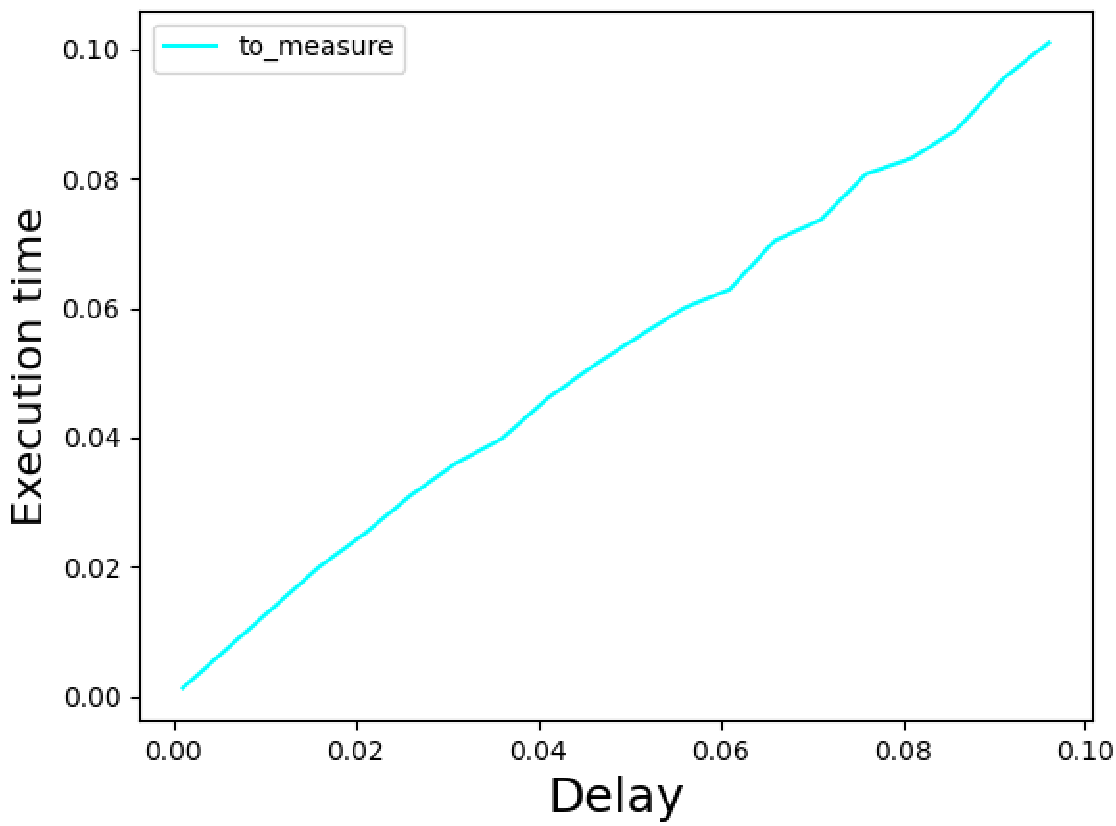

FIGURE 1 – Figure obtenue à partir des bibliothèques matplotlib et time et du code 1

B Tri bulle (Facultatif)

Le tri bulle est un tri par comparaisons et échanges, stable et en place. Il n'est pas efficace et sa vocation reste pédagogique.

Le principal avantage du tri bulle est qu'il est simple à expliquer. Son principe est le suivant : on prend chaque élément du tableau et on le fait monter d'une place dans le tableau si celui-ci est plus grand que son successeur en échangeant leurs places. Sinon, on fait prend le successeur et on opère à l'identique.

Algorithme 4 Tri bulle

```
Fonction BUBBLE_SORT(t)
    n ← taille(t)
    pour i de 0 à n-2 répéter
        pour j de 0 à n-i-2 répéter
            si t[j] > t[j+1] alors
                echange(t[j], t[j+1])                ▷ faire monter la bulle
```

C Recherche d'un élément dans un tableau

On considère une liste `L` contenant des éléments de type `int`. Cette liste est **triée** par ordre croissant de ses éléments. On veut savoir si un élément `x` est présent dans `L` et comparer les performances des approches séquentielles et la dichotomiques. On dispose des algorithmes 5, 6 et 7.

Algorithme 5 Recherche séquentielle d'un élément dans un tableau

```
Fonction RECHERCHE_SÉQUENTIELLE(t, elem)
    n ← taille(t)
    pour i de 0 à n − 1 répéter
        si t[i]= elem alors
            renvoyer i               ▷ élément trouvé, on renvoie sa position dans t
    renvoyer l'élément n'a pas été trouvé
```

C1. Coder l'algorithme de recherche séquentielle d'un élément dans un tableau. Lorsque l'élément n'est pas présent dans le tableau, la fonction Python renvoie `None`. Sinon, elle renvoie l'indice de l'élément trouvé dans le tableau. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement. Dans le pire des cas, quel est le coût d'une recherche séquentielle en fonction de la taille du tableau ?

C2. Coder l'algorithme 6. Lorsque l'élément n'est pas présent dans le tableau, la fonction Python renvoie `None`. Sinon, elle renvoie l'indice de l'élément trouvé dans le tableau. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement et trié.

C3. Coder l'algorithme 7. Vérifier que cet algorithme fonctionne sur un tableau d'entiers de 20 éléments rempli aléatoirement et trié et que l'indice renvoyé est bien l'indice minimal de la première occurrence de l'élément recherché.

Algorithme 6 Recherche d'un élément par dichotomie dans un tableau trié

```
Fonction RECHERCHE_DICHOTOMIQUE(t, elem)
    n ← taille(t)
    g ← 0
    d ← n-1
    tant que g ⩽ d répéter                 ▷ ⩽ cas où valeur au début, au milieu ou à la fin
        m ← (g+d)//2                       ▷ Division entière : un indice est un entier!
        si t[m] < elem alors
            g ← m + 1                      ▷ l'élément devrait se trouver dans t[m+1, d]
        sinon si t[m] > elem alors
            d ← m - 1                      ▷ l'élément devrait se trouver dans t[g, m-1]
        sinon
            renvoyer m                     ▷ l'élément a été trouvé
    renvoyer l'élément n'a pas été trouvé
```

Algorithme 7 Recherche d'un élément par dichotomie dans un tableau trié, renvoyer l'indice minimal en cas d'occurrences multiples.

```
Fonction RECHERCHE_DICHOTOMIQUE_INDICE_MIN(t, elem)
    n ← taille(t)
    g ← 0
    d ← n-1
    tant que g < d répéter                 ▷ attention au strictement inférieur!
        m ← (g+d)//2                       ▷ Un indice de tableau est un entier!
        si t[m] < elem alors
            g ← m + 1                      ▷ l'élément devrait se trouver dans t[m+1, d]
        sinon
            d ← m                          ▷ l'élément devrait se trouver dans t[g, m]
    si t[g] = elem alors
        renvoyer g
    sinon
        renvoyer l'élément n'a pas été trouvé
```

C4. On suppose que la longueur de la liste est une puissance de 2, c'est à dire $n = 2^p$ avec $p \geqslant 1$. Combien d'étapes l'algorithme 7 comporte-t-il ? En déduire le nombre de comparaisons effectuées, dans le cas où l'élément est absent, en fonction de p puis de n, et comparer avec l'algorithme de recherche séquentielle.

C5. Tracer le graphique des temps d'exécution des algorithmes précédent en fonction de n. Les tracés sont-ils cohérents avec les calculs des coûts effectués précédemment ?

C6. La recherche dichotomique fonctionne-t-elle sur les listes non triées ? Donner un contre-exemple si ce n'est pas le cas.