Rechercher

INFORMATIQUE COMMUNE - TP n° 1.5 - Olivier Reynet

À la fin de ce chapitre, je sais :

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

L'objectif de ce TP est d'étudier les algorithmes qui recherchent un élément dans un tableau.

A Recherche séquentielle

A1. Écrire une fonction de prototype `seq_search(t : list[int], elem : int)-> int` qui impémente l'algorithme de recherche séquentielle d'un élément dans un tableau (cf. algorithme 1). Lorsque l'élément n'est pas présent dans le tableau, la fonction 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.

Algorithme 1 Recherche séquentielle d'un élément dans un tableau

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

A2. Dans le pire des cas, combien d'opérations élémentaires seront nécessaires pour rechercher séquentiellement un élément dans un tableau de taille n ?

B Recherche dichotomique

On suppose maintenant que le tableau dans lequel la recherche doit être effectuée est **trié**.

B1. Écrire une fonction de prototype `dichotomic_search(t : list[int], elem : int)-> int` qui implémente l'algorithme de recherche d'un élément par dichotomie (cf. algorithme 2). Lorsque l'élément n'est pas présent dans le tableau, la fonction 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.

Algorithme 2 Recherche d'un élément par dichotomie dans un tableau **trié**

```
1 : Fonction RECHERCHE_DICHOTOMIQUE(t, elem)
2 :     n ← taille(t)
3 :     g ← 0
4 :     d ← n-1
5 :     tant que g ⩽ d répéter                ▷ ⩽ cas où valeur au début, au milieu ou à la fin
6 :         m ← (g+d)//2                      ▷ Division entière : un indice est un entier!
7 :         si t[m] = elem alors              ▷ avoir de la chance n'est pas exclu!
8 :             renvoyer m                    ▷ l'élément a été trouvé
9 :         sinon si t[m] < elem alors
10 :            g ← m + 1                     ▷ l'élément devrait se trouver dans t[m+1, d]
11 :        sinon
12 :            d ← m - 1                     ▷ l'élément devrait se trouver dans t[g, m-1]
13 :    renvoyer l'élément n'a pas été trouvé
```

B2. On suppose que la longueur du tableau est une puissance de 2, c'est à dire $n = 2^p$ avec $p \geqslant 1$. Combien d'itérations la boucle `tant que` de l'algorithme 2 comporte-t-elle ? En déduire le nombre d'opérations élémentaires effectuées dans le cas où l'élément est absent (c'est-à-dire le pire des cas), en fonction de n. Comparer avec l'algorithme de recherche séquentielle.

B3. Tracer le graphique des temps d'exécution des algorithmes précédents en fonction de n, la taille du tableau. Les tracés sont-ils cohérents avec les calculs des coûts effectués précédemment ? On prendre des tailles de tableau n de 10 à 100000 exclu par pas de 500.

B4. La recherche dichotomique fonctionne-t-elle sur les listes non triées ? Donner un contre-exemple si ce n'est pas le cas.

B5. Soit `t` un tableau de chaînes caractères trié dans l'ordre lexicographique. Peut-on utiliser la recherche dichotomique programmée ci-dessus pour rechercher une chaîne de caractère ? Pourquoi ? On pourra prendre par exemple le tableau `['', 'A', 'ACCTA', 'ACGT', 'AT', 'CACG', 'CTCACGA', 'GGTCA', 'GTCAAA', 'TAGCTGA', 'TT']`.

C Rechercher dans une liste imbriquée et jouer avec…

C1. Construire une liste imbriquée de listes d'entiers choisis aléatoirement en 0 et 100 exclu sont la taille des sous-listes est variable.

C2. Écrire une fonction de prototype `flatten(L : list[list[int]])-> list[int]` qui renvoie la liste mise à plat. Par exemple, pour la liste `[[39, 89], [], [51, 24, 84, 27], [], [39, 44]]` cette fonction renvoie `[39, 89, 51, 24, 84, 27, 39, 44]`.

C3. Appliquer la recherche dichomotique à une liste imbriquée en la mettant à plat et en la triant.

C4. Écrire une fonction de prototype `n_sum(L : list[list[int]])-> list[int]` qui renvoie la somme des éléments des sous-listes d'une liste imbriquée.

C5. Écrire une fonction de prototype `size_sl(L : list[list[int]])-> list[int]` qui renvoie la liste des taille des sous-listes de la liste imbriquée. Par exemple, pour la liste `[[39, 89], [], [51, 24, 84, 27], [], [39, 44]]` cette fonction renvoie `[2, 0, 4, 0, 2]`.