TP : Recherche dichotomique

L'objectif de cette activité est de visualiser expérimentalement l’efficacité de l'algorithme de recherche dichotomique en le comparant avec l'algorithme de recherche séquentielle.

Algorithme de *recherche séquentielle*

La fonction `recherche_sequentielle(v, T)` qui implémente l'algorithme de recherche séquentielle de la valeur `v` dans le tableau `T`. Cette fonction renvoie renvoie `True` si `v` appartient à `T`, et `False` sinon.

```
def recherche_sequentielle(v, T):
    """Renvoie True si v appartient au tableau T, False sinon."""
    for elt in T:
        if elt == v:
            return True
    return False
```

Protocole

On veut mesurer le temps dans le pire cas pour rechercher séquentiellement un élément dans un tableau. Pour cela, on va créer un tableau constitué uniquement de zéros et y chercher une valeur absente.

✍ **Question 1** : Pourquoi cette situation relève d'un pire cas pour cet algorithme ?

`Réponse :` _____

_____

Voici le protocole mis en place que vous pouvez analyser pour répondre aux questions qui suivent.

```
import time # le module time pour mesurer des temps avec Python.

def fabrique_tableau_de_zeros(taille):
    """Fabrication d'un tableau de zéros de taille `taille`."""
    return [0]*taille

def mesure_pire_cas_recherche_sequentielle(taille):
    """Mesure du temps dans le pire cas pour une recherche séquentielle."""

    # création du tableau (est long si taille est trop grand)
    tab = fabrique_tableau_de_zeros(taille)

    # messure du temps pour la recherche sequentielle
    tps_total = 0
    for _ in range(10):
        t0 = time.time()
        recherche_sequentielle(1, tab)
        t1 = time.time()
        delta_t = t1 - t0
        tps_total = tps_total + delta_t
    tps_moyen = tps_total / 10

    print(f"temps (pire cas) pour une recherche séquentielle dans un tableau de taille
{taille} : {tps_moyen} secondes")
```

✍ **Question 2** : Analysez la fonction `mesure_pire_cas_recherche_sequentielle`. Quelle est la valeur cherchée dans le tableau de zéros créé ?

Réponse : _____

_____

✍ **Question 3** : Que représente la valeur de la variable `tps_moyen` à la fin de cette fonction ? (autrement dit, que fait cette fonction ?).

Réponse : _____

_____

✍ **Question 4** : Exécutez les différentes cellules ci-dessous et expliquez en quoi les temps affichés justifient que l'algorithme de recherche séquentielle a un **coût linéaire**.

```
mesure_pire_cas_recherche_sequentielle(1000)
```

```
mesure_pire_cas_recherche_sequentielle(10000)
```

```
mesure_pire_cas_recherche_sequentielle(10**5)
```

```
mesure_pire_cas_recherche_sequentielle(10**6)
```

```
mesure_pire_cas_recherche_sequentielle(3*10**6)
```

Réponse : _____

_____

✍ **Question 5** : En utilisant le temps obtenu pour rechercher dans un tableau de taille 10**6 , combien de temps faudra-t-il pour rechercher dans un tableau de taille 10**9.
Attention, on ne demande pas d'appeler la fonction `mesure_pire_cas` car ce serait beaucoup trop long !

Réponse : _____

_____

Algorithme de *recherche dichotomique*

Implémentation de l'algorithme

La fonction `recherche_dichotomique(v, T)` suivante implémente l'algorithme de recherche dichotomique :

```
def recherche_dichotomique(v, T):
    """
    T : tableau d’entiers trié dans l’ordre croissant
    v : nombre entier
    La fonction renvoie True si T contient v et False sinon
    """

    debut = 0
    fin = len(T) - 1
    while debut <= fin:
        m = (debut + fin) // 2
        if v == T[m]:
            return True
        if v > T[m]:
            debut = m + 1
        else:
            fin = m - 1
    return False
```

Efficacité de l'algorithme

⚠ **Attention** : on rappelle que la recherche dichotomique s'applique sur un tableau **trié** (par ordre croissant avec notre implémentation).

Comme précédemment, on va prendre un tableau constitué uniquement de zéros, **qui est donc trié**, et y chercher la valeur 1 pour se placer dans un pire cas.

💻 **Question 6** : Adaptez très légèrement le protocole vu plus haut pour créer une fonction `mesure_pire_cas_recherche_dichotomique` qui mesure le temps mis (dans le pire cas) pour rechercher une valeur par dichotomie.

Indice : il n'y a qu'une ligne à modifier !

💻 **Question 7** : Testez cette fonction et vérifiez que pour un tableau de taille 1 million puis de taille 10 millions la recherche dichotomique est instantanée.

> ⚠ **Attention**, la création du tableau est beaucoup plus longue que la recherche, donc ne pas se fier au temps à attendre l'affichage mais seulement au temps affiché !)

⚙ Pour des tailles supérieures de tableau, on dépasse rapidement les capacités mémoire d'une machine classique. Par exemple, sur un système 64 bits, chaque élément d'un tableau Python est codé sur 8 octets. Donc un tableau de 1 milliard de valeurs nécessite 8 milliards d'octets pour être mémorisé, soit 8 Go de RAM nécessaire. Beaucoup d'ordinateurs n'ont pas cette capacité de mémoire.

Bilan

La recherche dichotomique est quasi instantanée, même pour des tableaux très grand. Cependant, il est nécessaire de l'appliquer sur un tableau trié. Si le tableau n'est pas trié on est obligé d'effectuer une recherche séquentielle, dont le coût en temps est linéaire ($O(n)$) donc plus lente.

On pourrait alors se demander s'il n'est pas intéressant de trier un tableau pour pouvoir ensuite chercher par dichotomie la présence de valeurs. Cela dépend, car le tri est coûteux !

Les meilleurs algorithmes de tri sont en $O(n\log 2n)$, soit plus coûteux qu'une recherche séquentielle qui est en $O(n)$.
Il est donc plus coûteux de trier un tableau pour appliquer l'algorithme de dichotomie que de chercher directement dans le tableau non trié de manière séquentielle.
Cependant, cela devient très intéressant si on sait que l'on a beaucoup de valeurs à chercher : on trie le tableau une fois pour toutes au départ (un peu long) mais ensuite toutes les recherches seront instantanées par dichotomie.

Notez enfin qu'il est n'est pas rare que des données soient triées dès le départ et donc que l'on puisse faire directement des recherches dichotomiques.

Sources :

https://info-mounier.fr/premiere_nsi/algorithmique/index.php#chap3

https://capytale2.ac-paris.fr/web/c/5b4a-3570909