Recherche dichotomique
Christophe Viroulaud
Première - NSI

Recherche dichotomique

Christophe Viroulaud

Première - NSI

Sommaire
1. Problématique
2. Recherche classique dans un tableau
3. Recherche dans un tableau trié

Sommaire

Problématique

Rechercher un élément dans un tableau est une opération courante. Cette tâche a un coût qui dépend de la taille du tableau. Cependant, si le tableau est déjà trié il est possible d'accélérer grandement la recherche.

Comment implémenter une recherche efficace dans un tableau trié ?

Problématique

Rechercher un élément dans un tableau est une opération courante. Cette tâche a un coût qui dépend de la taille du tableau. Cependant, si le tableau est déjà trié il est possible d'accélérer grandement la recherche.

Comment implémenter une recherche efficace dans un tableau trié ?

Sommaire

Sommaire

Génération des données

On demande à dix mille personnes de penser à un nombre entre 0 et 1000000 et on stocke les résultats dans un tableau au fur et à mesure des réponses.

Activité 1 : Construire par compréhension un tableau de dix mille entiers compris entre 0 et 1000000.

Génération des données

On demande à dix mille personnes de penser à un nombre entre 0 et 1000000 et on stocke les résultats dans un tableau au fur et à mesure des réponses.

> **Activité 1 :** Construire par compréhension un tableau de dix mille entiers compris entre 0 et 1000000.

Correction

Correction

```
1 entiers = [randint(0, 1000000) for _ in range(100000)]
```

Jeu de données

Sommaire

Sommaire

Recherche dans les données

Pour vérifier la présence d'une valeur dans les données, il faut parcourir le tableau élément par élément.

3 | 180 | 1007 | 56 | | | 2178 | 8

Figure 1 – Parcours séquentiel

Recherche dans les données

Pour vérifier la présence d'une valeur dans les données, il faut parcourir le tableau élément par élément.

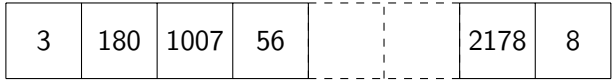


Figure 1 – Parcours séquentiel

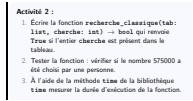
À retenir
Dans le pire des cas la complexité temporelle de la recherche dépend du nombre d'éléments.
$O(n)$

cas où l'élément n'est pas présent

À retenir

Dans le pire des cas la complexité temporelle de la recherche dépend du nombre d'éléments.

$$O(n)$$


Activité 2 :
1. Écrire la fonction recherche_classique(tab: list, cherche: int) → bool qui renvoie True si l'entier cherche est présent dans le tableau.
2. Tester la fonction : vérifier si le nombre 575000 a été choisi par une personne.
3. À l'aide de la méthode time de la bibliothèque time mesurer la durée d'exécution de la fonction.

Activité 2 :

1. Écrire la fonction **`recherche_classique(tab: list, cherche: int)`** $\rightarrow$ **`bool`** qui renvoie **`True`** si l'entier **`cherche`** est présent dans le tableau.
2. Tester la fonction : vérifier si le nombre 575000 a été choisi par une personne.
3. À l'aide de la méthode **`time`** de la bibliothèque **`time`** mesurer la durée d'exécution de la fonction.

Correction

Correction

```python
def recherche_classique(tab: list, cherche: int) -> bool:
    """
    Renvoie True si 'cherche' est dans 'tab'
    """
    for element in tab:
        if element == cherche:
            return True
    # à la fin de la boucle on n'a pas trouvé 'cherche'
    return False
```

Correction

```
deb = time()
print(recherche_classique(entiers, 575000))
fin = time()
print(fin-deb)
```

```
False
0.0066792964935302734
```

Exemple de résultat

Correction

```
deb = time()
print(recherche_classique(entiers, 575000))
fin = time()
print(fin-deb)
```

```
False
0.0066792964935302734
```

Exemple de résultat

Sommaire

Sommaire

Des données ordonnées

Imaginons maintenant la même expérience mais prenons la peine de trier les éléments au fur et à mesure de leur ajout dans le tableau de données.

3 | 8 | 56 | 180 | | | 1007 | 2178

Figure 2 – Tableau trié

Des données ordonnées

Imaginons maintenant la même expérience mais prenons la peine de trier les éléments au fur et à mesure de leur ajout dans le tableau de données.

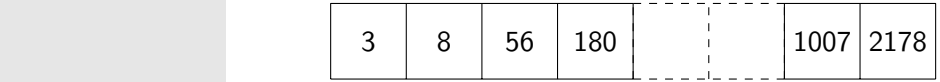


Figure 2 – Tableau trié

Activité 3 : Pour simplifier nous allons utiliser la méthode `sort` pour trier les données.
1. Construire par compréhension un tableau de dix mille entiers compris entre 0 et 1000000.
2. Trier le tableau.

Correction

```
entiers = [randint(0, 1000000) for _ in range(100000)]
entiers.sort()
```

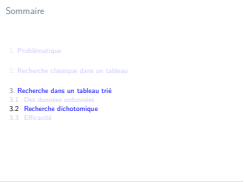
Jeu de données

Correction


```
entiers = [randint(0, 1000000) for _ in range(100000)]
entiers.sort()
```


Jeu de données

Sommaire



Sommaire

Recherche dichotomique

Les données étant triées, le principe de la dichotomie, pour chercher la présence d'un élément, consiste à :
- couper le tableau en deux parties égales,
- ne garder que la partie contenant l'élément,
- répéter l'opération jusqu'à trouver l'élément ou avoir une partie vide.

à peu près égales selon parité

Recherche dichotomique

Les données étant triées, le principe de la dichotomie, pour chercher la présence d'un élément, consiste à :

- couper le tableau en deux parties égales,
- ne garder que la partie contenant l'élément,
- répéter l'opération jusqu'à trouver l'élément ou avoir une partie vide.

Cherchons 765 dans le tableau suivant :

3	8	56	180	256	302	765	1007	2178

Figure 3 – Séparons les données en deux parties

Cherchons 765 dans le tableau suivant :

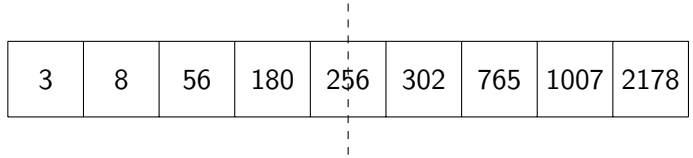


Figure 3 – Séparons les données en deux parties

3 8 56 180 256 302 765 1007 2178
FIGURE 4 – 256 n'est pas le nombre recherché et il est inférieur à 765

3	8	56	180	256	302	765	1007	2178

FIGURE 4 – 256 n'est pas le nombre recherché et il est inférieur à 765

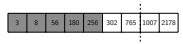


FIGURE 5 – Séparons les données restantes en deux parties

3 8 56 180 256 302 765 1007 2178

FIGURE 5 – Séparons les données restantes en deux parties

FIGURE 6 – Nous pouvons éliminer la partie supérieure à 765

FIGURE 6 – Nous pouvons éliminer la partie supérieure à 765

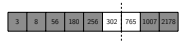


FIGURE 7 – Dernière séparation

3 8 56 180 256 302 765 1007 2178

FIGURE 7 – Dernière séparation

3 | 8 | 56 | 180 | 256 | 302 | 765 | 1007 | 2178
FIGURE 8 – 765 a été trouvée en trois itérations

3	8	56	180	256	302	765	1007	2178

FIGURE 8 – 765 a été trouvée en trois itérations

En pratique, on utilise les indices pour trouver le milieu.

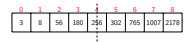


FIGURE 9 – $\frac{8+0}{2} = 4$ l'indice médian est 4

- 256 n'est pas le nombre recherché,
- il est inférieur au nombre recherché.

En pratique, on utilise les indices pour trouver le milieu.

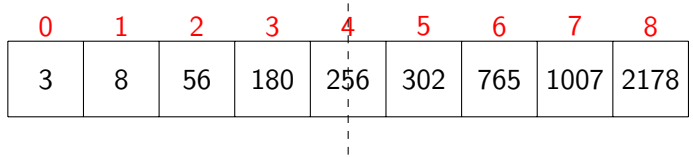


FIGURE 9 – $\frac{8+0}{2} = 4$ l'indice médian est 4

- 256 n'est pas le nombre recherché,
- il est inférieur au nombre recherché.

1. l'indice est un entier

FIGURE 10 – $\frac{8+5}{2} = 6$ l'indice médian est 6

- 765 est le nombre recherché,
- la recherche s'arrête.

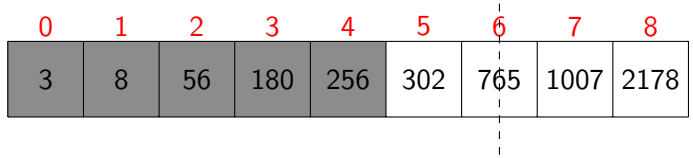


FIGURE 10 – $\frac{8+5}{2} = 6$ l'indice médian est 6

- 765 est le nombre recherché,
- la recherche s'arrête.

Activité 4 : Écrire la fonction `recherche_dicho(tab: list, cherche: int)` → `bool` qui applique le principe de la dichotomie. Pour séparer les données en deux parties (à peu près) égales il faudra calculer l'indice médian de la partie encore valide.

Correction

Correction

```
1  def recherche_dicho(tab: list, cherche: int) -> bool:
2      i_debut = 0
3      i_fin = len(tab)-1
```

Correction

Correction

```
1   while i_fin >= i_debut:
2       i_milieu = (i_debut+i_fin) // 2
```

Correction

```
if cherche == tab[i_milieu] :
    return True
```

Correction

```
1        if cherche == tab[i_milieu] :
2            return True
```

Correction

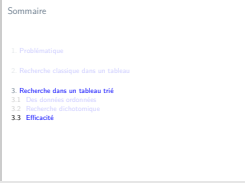
Correction

```
1        elif cherche < tab[i_milieu] :
2            i_fin = i_milieu-1
3        else:  # cherche > tab[i_milieu]
4            i_debut = i_milieu+1
```

Correction

Correction

```python
def recherche_dicho(tab: list, cherche: int) -> bool:
    i_debut = 0
    i_fin = len(tab)-1
    while i_fin >= i_debut:
        i_milieu = (i_debut+i_fin) // 2
        if cherche == tab[i_milieu] :
            return True
        elif cherche < tab[i_milieu] :
            i_fin = i_milieu-1
        else:  # cherche > tab[i_milieu]
            i_debut = i_milieu+1
    # à la fin de la boucle on n'a pas trouvé 'cherche'
    return False
```


Sommaire

1. Problématique

2. Recherche classique dans un tableau

3. Recherche dans un tableau trié
3.1 Des données ordonnées
3.2 Recherche dichotomique
3.3 Efficacité

Sommaire

0 = pas trouvé

Efficacité

À chaque itération la quantité de données (notée n) à étudier est divisée par deux. Dans le pire des cas, on divise jusqu'à ce que la taille de la partie restante soit inférieure ou égale à 1.

$$\frac{n}{2^x} = 1$$
$$\Leftrightarrow n = 2^x$$

Efficacité

À chaque itération la quantité de données (notée **n**) à étudier est divisée par deux. Dans le pire des cas, on divise jusqu'à ce que la taille de la partie restante soit inférieure ou égale à 1.

$$\frac{n}{2^x} = 1$$
$$\Leftrightarrow n = 2^x$$

Activité 5 :
1. Encadrer la valeur de x entre deux entiers, si le tableau contient $n = 10000$ éléments.
2. Mesurer la durée d'exécution de la fonction et la comparer à celle de la recherche classique.

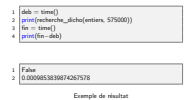
Correction
$2^{13} = 8192 < x < 2^{14} = 16384$

Correction

$$2^{13} = 8192 < x < 2^{14} = 16384$$

facteur 10

Correction



Correction

```
1 deb = time()
2 print(recherche_dicho(entiers, 575000))
3 fin = time()
4 print(fin−deb)
```

```
1 False
2 0.0009853839874267578
```

Exemple de résultat

À retenir
La complexité temporelle de la recherche dichotomique est :
$\log_2 n = x$

$$\log_2 n = \frac{\ln n}{\ln 2}$$

À retenir

La complexité temporelle de la recherche dichotomique est :

$$\log_2 n = x$$

Code complet

Le code complet se trouve ici.

Code complet

Le code complet se trouve ici.