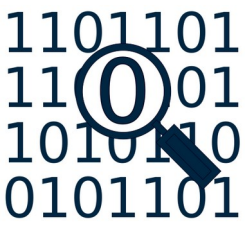
Algorithme de recherche d'une donnée



Des volumes importants de données sont susceptibles d'être traitées par les ordinateurs. Des algorithmes efficaces sont alors nécessaires pour réaliser ces opérations comme, par exemple, la sélection et la récupération des données. Les algorithmes de recherche entrent dans cette catégorie. Leur rôle est de déterminer si une donnée est présente et, le cas échéant, d'en indiquer sa position, pour effectuer des traitements annexes.

La recherche d'une information dans un annuaire illustre cette idée. On cherche si telle personne est présente dans l'annuaire afin d'en déterminer l'adresse. Plus généralement, c'est l'un des mécanismes principaux des bases de données : à l'aide d'un identifiant, on souhaite retrouver les informations correspondantes.

Dans cette famille d'algorithmes, la recherche dichotomique permet de traiter efficacement des données représentées dans un tableau de façon ordonnée.

Recherche séquentielle d'une valeur (méthode naïve)

Recherche par une méthode naïve

Une première façon de rechercher la présence d'une valeur dans un tableau est d'effectuer une recherche naïve à l'aide d'un parcours de tableau, que l'on peut programmer ainsi :

```
def recherche_naive(tab, val):
    for i in range(len(tab)):
        if tab[i] == val:
            return i
    return -1
```

Ici, la boucle for balaye la liste de gauche à droite et renvoie la position de la valeur recherchée dans la liste (entier positif ou nul) en cas de succès et -1 en cas d'échec.

Q1. Indiquer la complexité de cet algorithme. **Justifier** la réponse.

Recherche d'une valeur par dichotomie

L'idée centrale de cette approche repose sur l'idée de réduire de moitié l'espace de recherche à chaque étape. On regarde la valeur du milieu et si ce n'est pas celle recherchée, en tenant compte du caractère trié du tableau on sait qu'il faut continuer de chercher dans la première moitié ou dans la seconde. Plus précisément, la procédure de recherche est la suivante :

- ✗ on détermine l'élément m au milieu du tableau ;
- ✗ si c'est la valeur recherchée, on s'arrête avec un succès ;
- ✗ sinon, deux cas sont possibles :
  - ➔ si m est plus grand que la valeur recherchée, comme la tableau est trié, cela signifie qu'il suffit de continuer à chercher dans la première moitié du tableau ;
  - ➔ sinon, il suffit de chercher dans la moitié droite.

✗ On répète cela jusqu'à avoir trouvé la valeur recherchée, ou bien avoir réduit l'intervalle de recherche à un intervalle vide, ce qui signifie que la valeur recherchée n'est pas présente.
✗ A chaque étape, on coupe l'intervalle de recherche en deux, et on en choisit une moitié. On dit que l'on procède par **dichotomie**. Un exemple animé de l'exécution de cet algorithme est disponible à l'adresse
https://professeurb.github.io/articles/dichoto/

L'implémentation Python d'un tel algorithme est la suivante :

```
def recherche_dichotomique(tab, val):
    gauche = 0
    droite = len(tab) - 1
    while gauche <= droite :
        milieu = (gauche + droite) // 2
        if tab[milieu] == val: # val dans le tableau
            return milieu
        elif tab[milieu] > val:
            droite = milieu - 1 # Réduction du tableau par la droite
        else:
            gauche = milieu + 1 # Réduction du tableau par la gauche
    return -1  # val pas dans le tableau (-1)
```

Analyse de l'algorithme de recherche par dichotomie

Q2. Dérouler à la main l'exécution attendue de la fonction recherche_dichotomique(t, 11), 11 étant la valeur à chercher dans la table t initialisée avec :

t = [1, 3, 5, 6, 8, 11, 13, 14, 14, 17, 19, 21, 23]

Q3. Compléter la table d'exécution suivante afin de détailler l'état des différentes variables au début de chaque itération de la boucle while lors de l'appel recherche_dichotomique(t , 11). **Repérer** sur chaque liste t du tableau, par une flèche de couleur, la position des indices gauche (vert), droite (bleu), milieu (rouge).

Liste t	gauche	droite	milieu
1 3 5 6 8 11 13 14 14 17 19 21 23	0	12	6
1 3 5 6 8 11 13 14 14 17 19 21 23			
1 3 5 6 8 11 13 14 14 17 19 21 23			
1 3 5 6 8 11 13 14 14 17 19 21 23			

Complexité de l'algorithme de recherche par dichotomie

Quelque soit la méthode utilisée (méthode naïve ou par dichotomie), la recherche de présence d'une valeur dans une liste est effectuée en comparant la valeur recherchée à tout ou partie des éléments de la liste.

Q4. Pour une liste de 13 valeurs (cas étudié précédemment), **déterminer** dans le meilleur puis dans le pire des cas, le nombre de comparaisons effectuées par l'algorithme de dichotomie.

Q5. **Indiquer** sur le tableau suivant, le nombre de comparaisons effectuées dans le pire des cas, par l'algorithme de recherche séquentielle et par celui de recherche par dichotomie. **Comparer** les performances de ces deux algorithmes. **Conclure** quant à l'intérêt d'une approche par dichotomie.

Taille de la liste	1	2	4	8	16	32	64	128
Recherche séquentielle								
Recherche par dichotomie								

Q6. **Exprimer** sous la forme de puissances de 2, les résultats obtenus pour l'algorithme de recherche séquentielle. **En déduire** l'expression de la complexité N de l'algorithme de recherche séquentielle en fonction de la complexité n de l'algorithme de recherche par dichotomie.

Conclusion

Quand on écrit un nombre N en puissance de 2 (par exemple $256 = 2^8$), la puissance est appelée logarithme en base 2 de N et on la note $\log_2(N)$. D'après le tableau précédent on peut conclure que :

La complexité de l'algorithme de recherche par dichotomie est $\log_2(N)+1$. On dit qu'il est de l'ordre de $\log_2(N)$.