Parcours séquentiel

Christophe Viroulaud

Première - NSI

Algo 01

Le langage Python propose des outils de recherche dans les structures de données :

- `max`
- `min`

Peut-on implémenter efficacement les outils natifs de Python ?

Sommaire

Algorithme

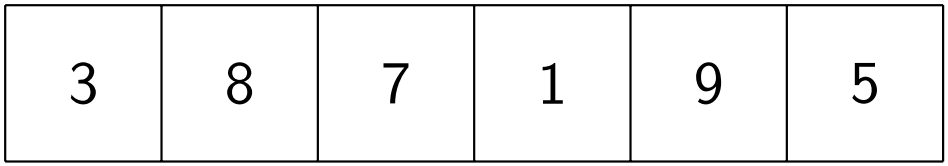
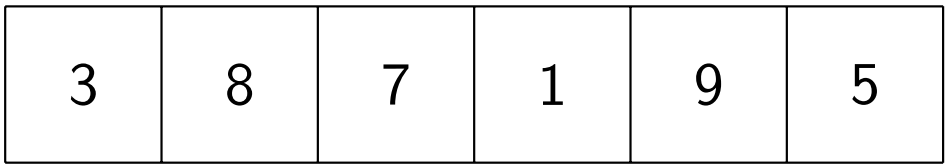


FIGURE 1 – Recherche du maximum

À retenir

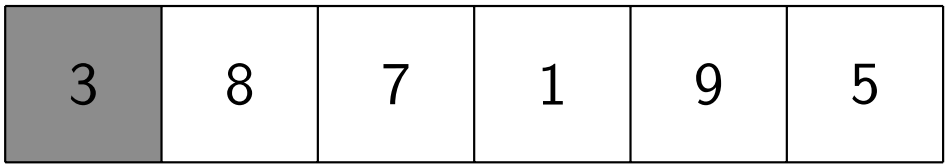
Pour trouver une valeur dans un tableau il faut le parcourir séquentiellement (élément après élément).

new_maxi = 0

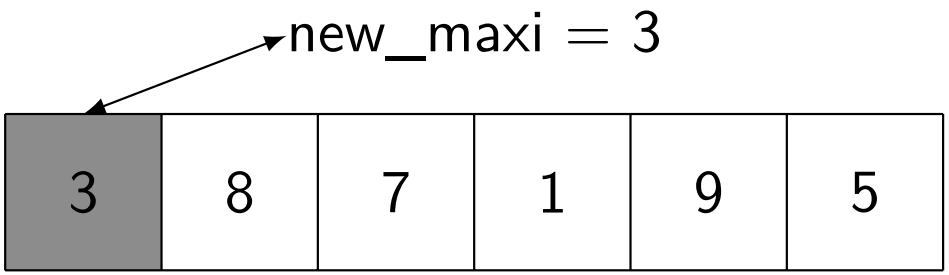


1. Créer une variable de stockage du maximum provisoire.

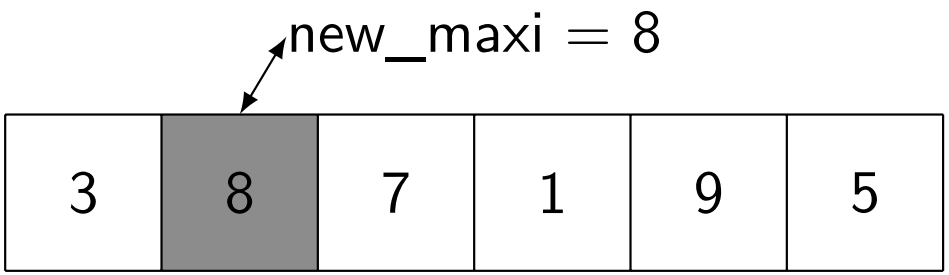
new_maxi = 0



1. Créer une variable de stockage du maximum provisoire.
2. Comparer la valeur en cours au maximum provisoire et mettre éventuellement à jour ce maximum.



1. Créer une variable de stockage du maximum provisoire.
2. Comparer la valeur en cours au maximum provisoire et mettre éventuellement à jour ce maximum.



1. Créer une variable de stockage du maximum provisoire.
2. Comparer la valeur en cours au maximum provisoire et mettre éventuellement à jour ce maximum.
3. Répéter l'étape 2 jusqu'à la fin du tableau.

Algorithme de recherche du maximum :

1. Créer une variable de stockage du maximum provisoire.
2. Comparer la valeur en cours au maximum provisoire et mettre éventuellement à jour ce maximum.
3. Répéter l'étape 2 jusqu'à la fin du tableau.

Activité 1 :

1. Construire par compréhension un tableau de 10 entiers aléatoires compris entre 1 et 100.
2. Écrire la fonction `maximum(tab: list)` $\rightarrow$ `int` qui respecte l'algorithme et renvoie la valeur maximale du tableau.

Correction


```python
def maximum(tab: list) -> int:
    """
    valeur maximale de tab

    Args:
        tab (list): tableau

    Returns:
        int: valeur max
    """
    new_maxi = 0
    for val in tab:
        if val > new_maxi:
            new_maxi = val
    # fin de la boucle, renvoie le maxi
    return new_maxi
```

Code 1 – Itérer sur le tableau

```python
def maximum(tab: list) -> int:
    """
    valeur maximale de tab

    Args:
        tab (list): tableau

    Returns:
        int: valeur max
    """
    new_maxi = 0
    for i in range(len(tab)):
        if tab[i] > new_maxi:
            new_maxi = tab[i]
    # fin de la boucle, renvoie le maxi
    return new_maxi
```

Code 2 – Itérer sur les indices

```
1 # création du tableau
2 tab = [randint(1, 100) for _ in range(10)]
3 # affichage du tableau
4 print(tab)
5 # affichage du maximum
6 print(maximum(tab))
```

Code 3 – Appel de la fonction

Sommaire

Complexité

À retenir

Le **complexité temporelle** représente le nombre d'étapes que l'algorithme doit réaliser pour exécuter sa tâche.

Remarque

La durée d'exécution dépend de la puissance de la machine, mais le nombre d'étapes reste le même.

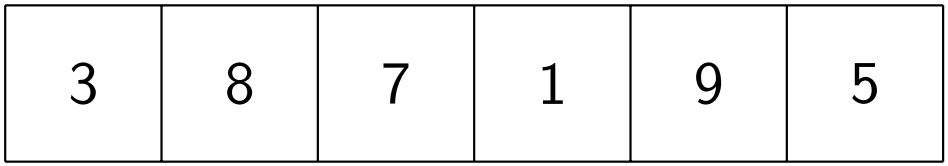


FIGURE 2 – Recherche du maximum

Pour trouver le maximum du tableau, l'algorithme doit visiter chaque cellule une fois. Le nombre d'étapes dépend de la taille du tableau.

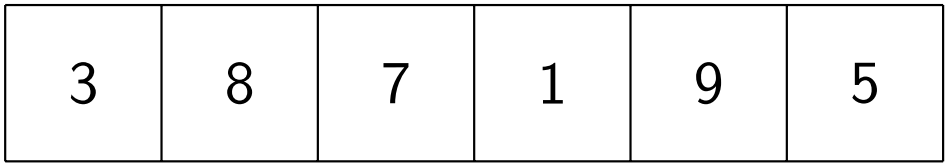


FIGURE 3 – Recherche du maximum

À retenir

La complexité de la recherche du maximum d'un tableau dépend de la taille `n` du tableau. On dit que la complexité est **linéaire** et on la note :

$$O(n)$$

Activité 2 :

1. Écrire la fonction `est_present(tab: list, e: int)` $\rightarrow$ `bool` qui renvoie `true` si l'entier `e` est présent dans le tableau.
2. Que peut-on dire à propos de la complexité temporelle de cette fonction si l'élément est présent en première position du tableau ? En dernière position ?
3. Quelle est la complexité temporelle si l'élément n'est pas présent dans le tableau ?

Correction


```
1  def est_present(tab: list, e: int) -> bool:
2      """
3      vérifie si e est dans le tableau
4      """
5      for val in tab:
6          if val == e:
7              return True
8      # On est sorti de la boucle sans avoir
       trouvé
9      return False
```

3	8	7	1	9	5

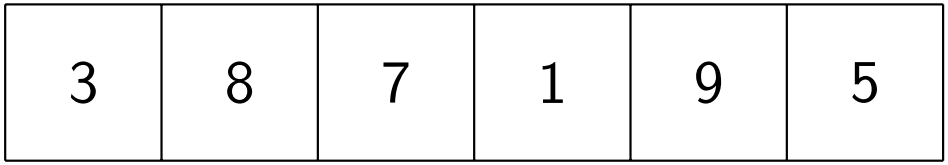
Complexités :

- **dans le meilleur des cas :** l'élément cherché est le premier du tableau. La complexité est **constante** ($O(1)$).

3	8	7	1	9	5

Complexités :

- **dans le meilleur des cas :** l'élément cherché est le premier du tableau. La complexité est **constante** ($O(1)$).
- **dans le pire des cas :** l'élément cherché n'est pas présent. La complexité est **linéaire** ($O(n)$).



Complexités :

- **dans le meilleur des cas :** l'élément cherché est le premier du tableau. La complexité est **constante** $(O(1))$.
- **dans le pire des cas :** l'élément cherché n'est pas présent. La complexité est **linéaire** $(O(n))$.
- **moyenne :** l'élément est dans le tableau. La complexité est **linéaire** $(O(n))$.

Sommaire

Terminaison

À retenir

La **terminaison** d'un programme est le fait de vérifier qu'il finit par s'arrêter. Dans une boucle on cherche **un variant de boucle** : une expression qui change à chaque itération, jusqu'à un cas limite.

```
1  def maximum(tab: list) -> int:
2      new_maxi = 0
3      for i in range(len(tab)):
4          if tab[i] > new_maxi:
5              new_maxi = tab[i]
6      # fin de la boucle, renvoie le maxi
7      return new_maxi
```

Code 4 – La variable `i` est un variant de la boucle

La variable `i` augmente à chaque itération. Elle finira toujours par atteindre **`len(tab)`**.

Activité 3 :

```
1 i = 0
2 while i < 10:
3     print(i)
```

Ce code termine-t-il ?

Correction

Il n'y a pas de variant de boucle. Le code ne termine pas.

Activité 4 :

```
1 i = 10
2 while i != 1:
3     if i%2 == 0:
4         i = i//2
5     else:
6         i = 3*i + 1
```

Code 5 – Suite de Syracuse

Ce code termine-t-il ?

Correction

Pour `i = 10` la suite termine. Mais personne n'a encore prouvé que la propriété est vérifiée pour toutes les valeurs de `i`.

Sommaire

Correction

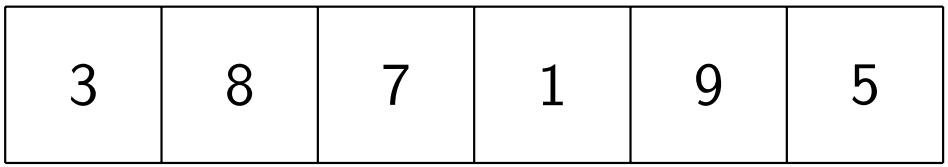
À retenir

La **correction** d'un programme est le fait de vérifier qu'il réalise effectivement ce qui était prévu. Dans une boucle on cherche **un invariant de boucle :** une expression qui est vraie avant chaque itération.

```
1  def maximum(tab: list) -> int:
2      new_maxi = 0
3      for i in range(len(tab)):
4          if tab[i] > new_maxi:
5              new_maxi = tab[i]
6      # fin de la boucle, renvoie le maxi
7      return new_maxi
```

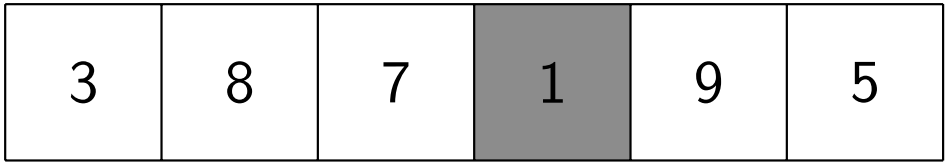
Code 6 – « **new_maxi contient le maximum du début du tableau** » est un invariant.

new_maxi $= 0$



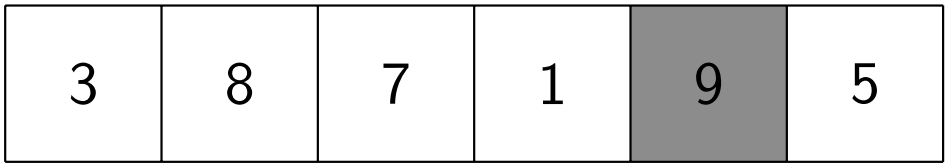
- Avant la première itération, la propriété est vraie.

new_maxi = 8



- Avant la première itération, la propriété est vraie.
- On suppose que la propriété est vraie pour l'itération `n`.

new_maxi $= 9$



- Avant la première itération, la propriété est vraie.
- On suppose que la propriété est vraie pour l'itération **`n`**.
- À l'itération suivante **`new_maxi`** est comparé et mis à jour.