Listes chaînées
Christophe Viroulaud
Terminale - NSI
Archi 03

Listes chaînées

Christophe Viroulaud

Terminale - NSI

Archi 03

Quand un tableau est créé, le système lui alloue un espace contigu en mémoire.

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4								

FIGURE 1 – Le tableau est enregistré dans un espace libre

On accède à chaque élément du tableau **en temps constant**. Cependant insérer un nouvel élément devient problématique.

vrai en C, un peu différent en Python (haut-niveau)

Quand un tableau est créé, le système lui alloue un espace contigu en mémoire.

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4								

FIGURE 1 – Le tableau est enregistré dans un espace libre

On accède à chaque élément du tableau **en temps constant**. Cependant insérer un nouvel élément devient problématique.

Comment définir un autre type de structure de données qui pallie les limitations d'un tableau ?

Comment définir un autre type de structure de données qui pallie les limitations d'un tableau ?

Sommaire

Sommaire

Liste chaînée - principe

Chaque élément :
- prend une place libre quelconque en mémoire.

pointe vers

Liste chaînée - principe

Chaque élément :

- prend une place libre quelconque en mémoire.

pointe vers

Liste chaînée - principe

Chaque élément :
- prend une place libre quelconque en mémoire.
- connaît l'emplacement de l'élément suivant.

Liste chaînée - principe

Chaque élément :

- prend une place libre quelconque en mémoire.
- connaît l'emplacement de l'élément suivant.

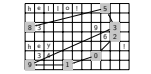


FIGURE 2 – Chaque élément pointe vers le suivant.

tête de liste = dernier élément ajouté (ici 2)

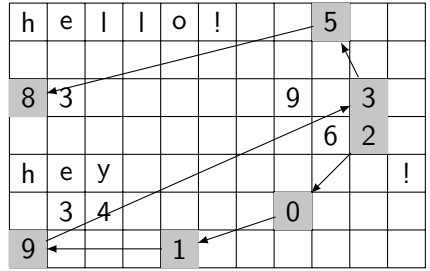


FIGURE 2 – Chaque élément pointe vers le suivant.

Sommaire

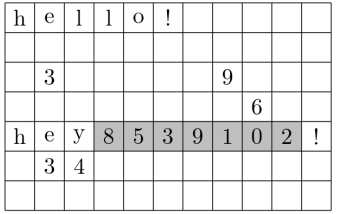
1. Liste chaînée
1.1 Principe
1.2 Comparaison avec un tableau
2. Implémentation
3. Manipuler une liste chaînée

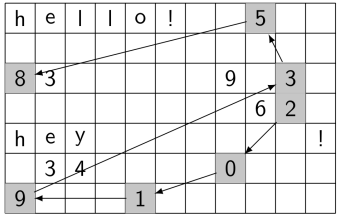
Sommaire

Comparaison avec un tableau

Activité 1 : Comparer l'efficacité des deux structures lors de :
- l'ajout d'un élément,
- l'accès à un élément,
- l'insertion d'un élément.

Comparaison avec un tableau





Activité 1 : Comparer l'efficacité des deux structures lors de :

- l'ajout d'un élément,
- l'accès à un élément,
- l'insertion d'un élément.

Ajouter un élément

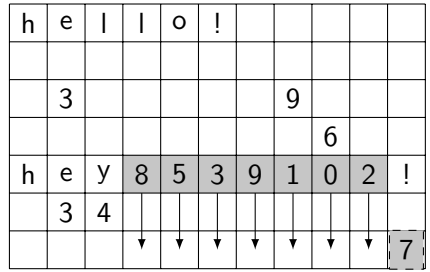


FIGURE 3 – Pour ajouter 7 au tableau il faut recopier entièrement ce-dernier dans un espace libre.

À retenir

L'ajout d'un élément à un tableau a une complexité **linéaire** dans le pire des cas.

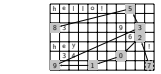


FIGURE 4 – Pour ajouter 7 à la liste, il suffit de modifier la tête de liste.

À retenir

L'ajout d'un élément à une liste chaînée se fait en temps **constant**.

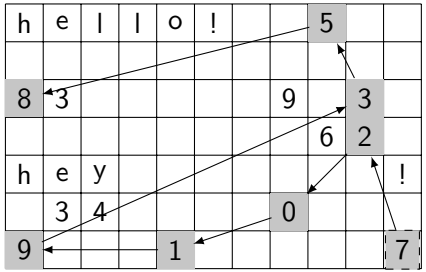
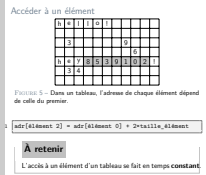


FIGURE 4 – Pour ajouter 7 à la liste, il suffit de modifier la tête de liste.

À retenir

L'ajout d'un élément à une liste chaînée se fait en temps **constant**.


Accéder à un élément

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4								

FIGURE 5 – Dans un tableau, l'adresse de chaque élément dépend de celle du premier.

```
1 adr[élément 2] = adr[élément 0] + 2×taille_élément
```

À retenir

L'accès à un élément d'un tableau se fait en temps **constant**.

Accéder à un élément

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4								

FIGURE 5 – Dans un tableau, l'adresse de chaque élément dépend de celle du premier.

```
1 adr[élément 2] = adr[élément 0] + 2×taille_élément
```

À retenir

L'accès à un élément d'un tableau se fait en temps **constant**.

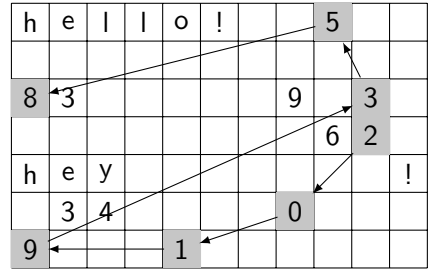


FIGURE 6 – Pour accéder à l'élément de rang n il faut partir de la tête et avancer 5 fois.

À retenir

L'accès à l'élément de rang n a une complexité **linéaire**.

Insérer un élément

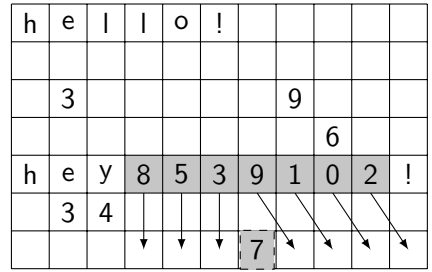


FIGURE 7 – Pour insérer 7 dans le tableau il faut recopier entièrement ce-dernier dans un espace libre.

À retenir

L'insertion d'un élément dans un tableau a une complexité **linéaire** dans le pire des cas.

Insérer un élément

FIGURE 7 – Pour insérer 7 dans le tableau il faut recopier entièrement ce-dernier dans un espace libre.

À retenir

L'insertion d'un élément dans un tableau a une complexité **linéaire** dans le pire des cas.

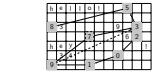


FIGURE 8 – Pour insérer 7 au rang i de la liste, il faut modifier le successeur de l'élément de rang i.

À retenir

L'insertion d'un élément à une liste chaînée se fait en temps **linéaire**.

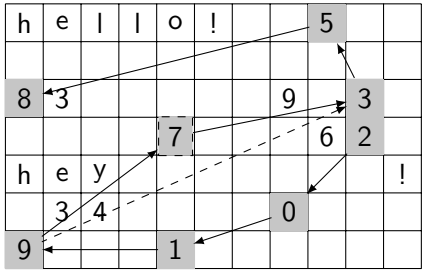


FIGURE 8 – Pour insérer 7 au rang i de la liste, il faut modifier le successeur de l'élément de rang i.

À retenir

L'insertion d'un élément à une liste chaînée se fait en temps **linéaire**.

Bilan

	tableau	liste
ajout	linéaire	constant
accès	constant	linéaire
insertion	linéaire	linéaire

Bilan

	tableau	**liste**
ajout	linéaire	constant
accès	constant	linéaire
insertion	linéaire	linéaire

Sommaire

1. Liste chaînée
2. Implémentation
3. Manipuler une liste chaînée

Sommaire

Implémentation - le maillon
Maillon
val = 3
suiv =
FIGURE 9 – Chaque maillon contient 2 informations
class Maillon:
"""
Crée un maillon de la liste chaînée
"""
def __init__(self, val: int, suiv: object):
self.valeur = val
self.suivant = suiv
Code 1 – Un objet Maillon

Implémentation - le maillon

Maillon
val = 3
suiv =

FIGURE 9 – Chaque maillon contient 2 informations

```
1  class Maillon:
2      """
3      Crée un maillon de la liste chaînée
4      """
5
6      def __init__(self, val: int, suiv: object):
7          self.valeur = val
8          self.suivant = suiv
```

Code 1 – Un objet **Maillon**

La liste

Figure 10 – La liste est une succession de maillons

La liste

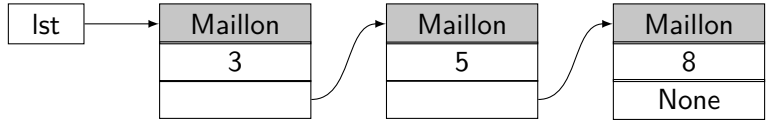


Figure 10 – La liste est une succession de maillons

1 lst = Maillon(3, Maillon(5, Maillon(8, None)))
Code 2 – Chaque maillon est le suivant d'un autre.

```
1 lst = Maillon(3, Maillon(5, Maillon(8, None)))
```

Code 2 – Chaque maillon est le suivant d'un autre.

La liste - seconde approche

FIGURE 11 – L'objet **Liste** contient une référence à la tête.

La liste - seconde approche

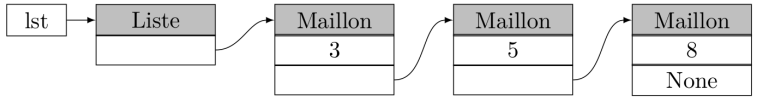


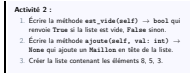
FIGURE 11 – L'objet **Liste** contient une référence à la tête.

Code 3 – Objet **Liste**

1. L'attribut *tete* représente le premier *Maillon*.
2. Une liste vide renvoie alors *None*.

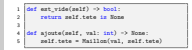
```python
class Liste:
    """
    Crée une liste chaînée
    """

    def __init__(self):
        self.tete: Maillon = None
```

Code 3 – Objet **Liste**



Activité 2 :

1. Écrire la méthode `est_vide(self)` → `bool` qui renvoie `True` si la liste est vide, `False` sinon.
2. Écrire la méthode `ajoute(self, val: int)` → `None` qui ajoute un `Maillon` en tête de la liste.
3. Créer la liste contenant les éléments 8, 5, 3.



```
def est_vide(self) -> bool:
    return self.tete is None

def ajoute(self, val: int) -> None:
    self.tete = Maillon(val, self.tete)
```

```
lst = Liste()
lst.ajoute(8)
lst.ajoute(5)
lst.ajoute(3)
```

Code 4 – Création de la liste

```
1  def est_vide(self) -> bool:
2      return self.tete is None
3
4  def ajoute(self, val: int) -> None:
5      self.tete = Maillon(val, self.tete)
```

```
1  lst = Liste()
2  lst.ajoute(8)
3  lst.ajoute(5)
4  lst.ajoute(3)
```

Code 4 – Création de la liste


La méthode __str__ permettra de visualiser l'état de la liste.

```
def __str__(self):
    m = self.tete
    s = ""
    while m is not None:
        s += str(m.valeur)+" - "
        m = m.suivant
    else:
        s += "fin"
    return s
```

```
print(lst)
```

Code 5 – Afficher la liste

La méthode **__str__** permettra de visualiser l'état de la liste.

```
def __str__(self):
    m = self.tete
    s = ""
    while m is not None:
        s += str(m.valeur)+" - "
        m = m.suivant
    else:
        s += "fin"
    return s
```

```
print(lst)
```

Code 5 – Afficher la liste

Sommaire

Sommaire

Manipuler une liste chaînée - taille

Pour connaître la taille de la liste il faut la parcourir entièrement.

Activité 3 :

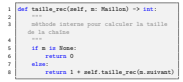
1. Écrire la méthode récursive `taille_rec(self, m: Maillon)` → `int` qui renvoie la taille de la liste démarrant à `m`
2. Écrire la méthode `taille(self)` → `int` qui renvoie la taille de la liste. Cette méthode utilisera `taille_rec`.
3. **Pour les plus avancés :** Écrire la méthode native (itérative) `__len__(self)` → `int` qui redéfinit la fonction `len` pour la classe `Liste`.

Manipuler une liste chaînée - taille

Pour connaître la taille de la liste il faut la parcourir entièrement.

Activité 3 :

1. Écrire la méthode récursive **`taille_rec(self, m: Maillon)`** → **`int`** qui renvoie la taille de la liste démarrant à **`m`**
2. Écrire la méthode **`taille(self)`** → **`int`** qui renvoie la taille de la liste. Cette méthode utilisera **`taille_rec`**.
3. **Pour les plus avancés :** Écrire la méthode native (itérative) **`__len__(self)`** → **`int`** qui redéfinit la fonction **`len`** pour la classe **`Liste`**.



```python
def taille_rec(self, m: Maillon) -> int:
    """
    méthode interne pour calculer la taille de la chaîne
    """
    if m is None:
        return 0
    else:
        return 1 + self.taille_rec(m.suivant)
```

```python
def taille(self) -> int:
    """
    appel principal de la méthode récursive
    pour mesurer
    la taille de la chaine
    """
    return self.taille_rec(self.tete)
```

```python
print(lst.taille())
```

Code 6 – Affichage de la taille de la liste

Listes chaînées

```python
def __len__(self) -> int:
    maillon = self.tete
    taille = 0
    while maillon is not None:
        maillon = maillon.suivant
        taille += 1
    return taille
```

```python
print(len(lst))
```

Code 7 – Appel de la fonction `len`


```python
def __len__(self) -> int:
    maillon = self.tete
    taille = 0
    while maillon is not None:
        maillon = maillon.suivant
        taille += 1
    return taille
```

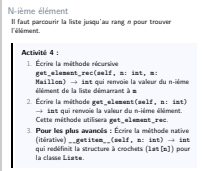
```python
print(len(lst))
```

Code 7 – Appel de la fonction `len`

Sommaire

1. Liste chaînée
2. Implémentation
3. Manipuler une liste chaînée
3.2 N-ième élément
3.3 Insertion

Sommaire


N-ième élément

Il faut parcourir la liste jusqu'au rang n pour trouver l'élément.

Activité 4 :

1. Écrire la méthode récursive `get_element_rec(self, n: int, m: Maillon)` → `int` qui renvoie la valeur du n-iéme élément de la liste démarrant à `m`
2. Écrire la méthode `get_element(self, n: int)` → `int` qui renvoie la valeur du n-iéme élément. Cette méthode utilisera `get_element_rec`.
3. **Pour les plus avancés :** Écrire la méthode native (itérative) `__getitem__(self, n: int)` → `int` qui redéfinit la structure à crochets (`lst[n]`) pour la classe `Liste`.

N-ième élément

Il faut parcourir la liste jusqu'au rang n pour trouver l'élément.

Activité 4 :

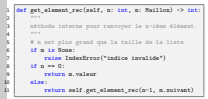
1. Écrire la méthode récursive `get_element_rec(self, n: int, m: Maillon)` → `int` qui renvoie la valeur du n-iéme élément de la liste démarrant à `m`
2. Écrire la méthode `get_element(self, n: int)` → `int` qui renvoie la valeur du n-iéme élément. Cette méthode utilisera `get_element_rec`.
3. **Pour les plus avancés :** Écrire la méthode native (itérative) `__getitem__(self, n: int)` → `int` qui redéfinit la structure à crochets (`lst[n]`) pour la classe `Liste`.

Correction

Correction

```python
def get_element_rec(self, n: int, m: Maillon) -> int:
    """
    méthode interne pour renvoyer le n-ième élément.
    """
    if n == 0:
        return m.valeur
    else:
        return self.get_element_rec(n-1, m.suivant)
```

Correction



Code 8 – Avec gestion du dépassement de taille

Correction

```python
def get_element_rec(self, n: int, m: Maillon) -> int:
    """
    méthode interne pour renvoyer le n-ième élément.
    """
    # n est plus grand que la taille de la liste
    if m is None:
        raise IndexError("indice invalide")
    if n == 0:
        return m.valeur
    else:
        return self.get_element_rec(n-1, m.suivant)
```

Code 8 – Avec gestion du dépassement de taille

Correction

```
def get_element(self, n: int) -> int:
    """
    appel principal de la méthode récursive pour renvoyer le n-ième élément
    """
    return self.get_element_rec(n, self.tete)
```

```
print(lst.get_element(3))
```

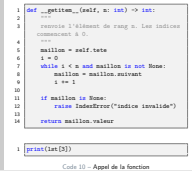
Code 9 – Appel de la fonction

Correction

```
def get_element(self, n: int) -> int:
    """
    appel principal de la méthode récursive pour renvoyer le n-ième élément
    """
    return self.get_element_rec(n, self.tete)
```

```
print(lst.get_element(3))
```

Code 9 – Appel de la fonction



```
def __getitem__(self, n: int) -> int:
    """
    renvoie l'élément de rang n. Les indices commencent à 0.
    """
    maillon = self.tete
    i = 0
    while i < n and maillon is not None:
        maillon = maillon.suivant
        i += 1

    if maillon is None:
        raise IndexError("indice invalide")

    return maillon.valeur
```

```
print(lst[3])
```

Code 10 – Appel de la fonction

```
def __getitem__(self, n: int) -> int:
    """
    renvoie l'élément de rang n. Les indices commencent à 0.
    """
    maillon = self.tete
    i = 0
    while i < n and maillon is not None:
        maillon = maillon.suivant
        i += 1

    if maillon is None:
        raise IndexError("indice invalide")

    return maillon.valeur
```

```
print(lst[3])
```

Code 10 – Appel de la fonction

Sommaire

1. Liste chaînée
2. Implémentation
3. Manipuler une liste chaînée
3.1 Taille de la liste
3.2 N-ième élément
3.3 Insertion

Sommaire

Insertion
L'insertion d'un élément au rang n peut être réalisée sur le même principe. On prendra le parti d'insérer l'élément en fin si la valeur de n dépasse la taille de la liste.

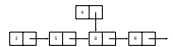


Figure 12 – Insertion de 4 au rang 2

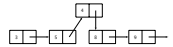


Figure 13 – Seconde étape

Insertion

L'insertion d'un élément au rang n peut être réalisée sur le même principe. On prendra le parti d'insérer l'élément en fin si la valeur de n dépasse la taille de la liste.

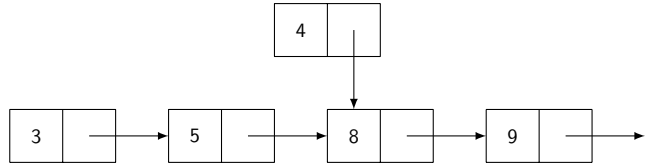


Figure 12 – Insertion de 4 au rang 2

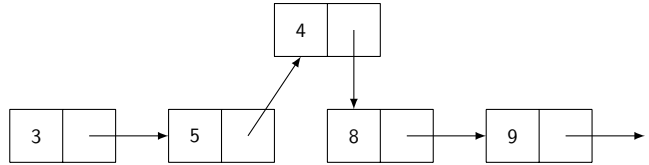
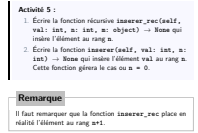


Figure 13 – Seconde étape



Activité 5 :

1. Écrire la fonction récursive `inserer_rec(self, val: int, n: int, m: object) → None` qui insère l'élément au rang `n`.
2. Écrire la fonction `inserer(self, val: int, n: int) → None` qui insère l'élément `val` au rang `n`. Cette fonction gérera le cas ou `n = 0`.

Remarque

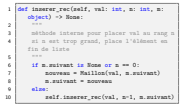
Il faut remarquer que la fonction `inserer_rec` place en réalité l'élément au rang `n+1`.

Activité 5 :

1. Écrire la fonction récursive **`inserer_rec(self, val: int, n: int, m: object)`** $\rightarrow$ **`None`** qui insère l'élément au rang **`n`**.
2. Écrire la fonction **`inserer(self, val: int, n: int)`** $\rightarrow$ **`None`** qui insère l'élément **`val`** au rang **`n`**. Cette fonction gérera le cas ou **`n = 0`**.

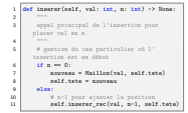
Remarque

Il faut remarquer que la fonction **`inserer_rec`** place en réalité l'élément au rang **`n+1`**.



Correction

```
1  def inserer_rec(self, val: int, n: int, m:
       object) -> None:
2      """
3      méthode interne pour placer val au rang n
4      si n est trop grand, place l'élément en
       fin de liste
5      """
6      if m.suivant is None or n == 0:
7          nouveau = Maillon(val, m.suivant)
8          m.suivant = nouveau
9      else:
10         self.inserer_rec(val, n-1, m.suivant)
```



```python
def inserer(self, val: int, n: int) -> None:
    """
    appel principal de l'insertion pour
    placer val en n
    """
    # gestion du cas particulier où l'
    insertion est en début
    if n == 0:
        nouveau = Maillon(val, self.tete)
        self.tete = nouveau
    else:
        # n-1 pour ajuster la position
        self.inserer_rec(val, n-1, self.tete)
```