

Liste chaînée

Principe  
Comparaison avec un  
tableau

Implémentation

Manipuler une liste  
chaînée

Taille de la liste  
N-ième élément  
Insertion

# 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.

#### Liste chaînée

Principe

Comparaison avec un tableau

#### Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

Liste chaînée

Principe

Comparaison avec un  
tableau

Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

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

## 1. Liste chaînée

### 1.1 Principe

### 1.2 Comparaison avec un tableau

## 2. Implémentation

## 3. Manipuler une liste chaînée

### Liste chaînée

Principe

Comparaison avec un  
tableau

### Implémentation

### Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

# Liste chaînée - principe

Chaque élément :

- ▶ prend une place libre quelconque en mémoire.

Liste chaînée

**Principe**

Comparaison avec un tableau

Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

# Liste chaînée - principe

Chaque élément :

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

Liste chaînée

**Principe**

Comparaison avec un tableau

Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

## Liste chaînée

## Principe

Comparaison avec un tableau

## Implémentation

## Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

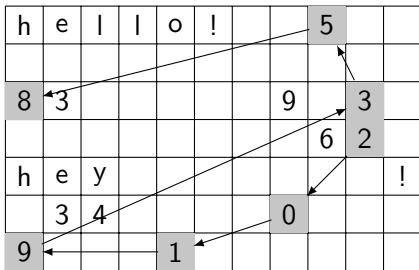


FIGURE 2 – Chaque élément pointe vers le suivant.

## 1. Liste chaînée

### 1.1 Principe

### 1.2 Comparaison avec un tableau

## 2. Implémentation

## 3. Manipuler une liste chaînée

### Liste chaînée

Principe

**Comparaison avec un  
tableau**

### Implémentation

### Manipuler une liste chaînée

Taille de la liste

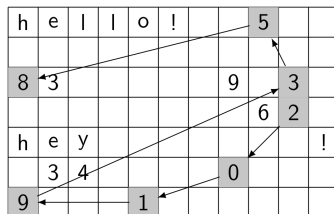
N-ième élément

Insertion



# Comparaison avec un tableau

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



**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.

## Liste chaînée

Principe

Comparaison avec un tableau

## Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

# Ajouter un élément

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4	↓	↓	↓	↓	↓	↓	↓	↓
			↓	↓	↓	↓	↓	↓	↓	7

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.

### Liste chaînée

Principe

Comparaison avec un tableau

### Implémentation

### Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

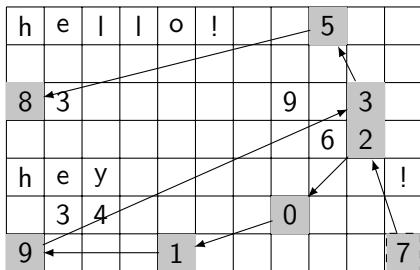


FIGURE 4 – Pour ajouter 7 à la liste, il suffit de modifier la tête de liste.

## À retenir

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

### Liste chaînée

Principe

Comparaison avec un tableau

### Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

# 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.

$$\text{adr}[\text{élément } 2] = \text{adr}[\text{élément } 0] + 2 \times \text{taille\_élément}$$

## À retenir

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

### Liste chaînée

Principe

Comparaison avec un tableau

### Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

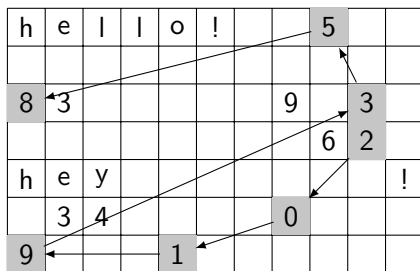


FIGURE 6 – Pour accéder à l'élément de rang  $n$  il faut partir de la tête et avancer 5 fois.

## À retenir

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

### Liste chaînée

Principe

Comparaison avec un tableau

### Implémentation

Manipuler une liste chaînée

Taille de la liste

$N$ -ième élément

Insertion

# Insérer un élément

h	e	l	l	o	!					
	3						9			
								6		
h	e	y	8	5	3	9	1	0	2	!
	3	4								
						7				

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.

### Liste chaînée

Principe

Comparaison avec un tableau

### Implémentation

### Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

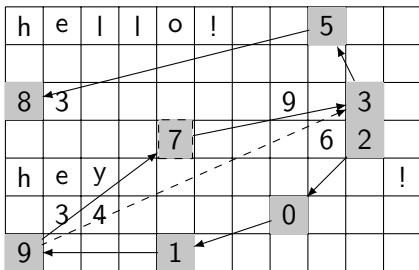


FIGURE 8 – Pour insérer 7 au rang  $i$  de la liste, il faut modifier le successeur de l'élément de rang  $i$ .

## À retenir

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

### Liste chaînée

Principe

Comparaison avec un  
tableau

### Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

	<b>tableau</b>	<b>liste</b>
<b>ajout</b>	linéaire	constant
<b>accès</b>	constant	linéaire
<b>insertion</b>	linéaire	linéaire



## 1. Liste chaînée

## 2. Implémentation

## 3. Manipuler une liste chaînée

### Liste chaînée

Principe

Comparaison avec un tableau

### Implémentation

### Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

# 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

## Liste chaînée

Principe

Comparaison avec un tableau

## Implémentation

## Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

## Liste chaînée

Principe  
Comparaison avec un  
tableau

## Implémentation

### Manipuler une liste chaînée

Taille de la liste  
N-ième élément  
Insertion

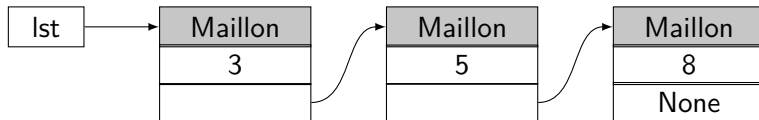


FIGURE 10 – La liste est une succession de maillons

## Liste chaînée

Principe  
Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste  
N-ième élément  
Insertion

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

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

# La liste - seconde approche

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

### Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

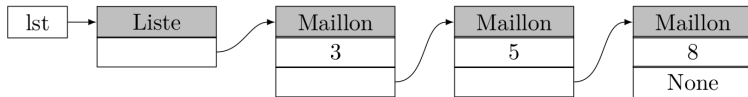


FIGURE 11 – L'objet **Liste** contient une référence à la tête.

## Liste chaînée

Principe  
Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste  
N-ième élément  
Insertion

```
1 class Liste:
2     """
3     Crée une liste chaînée
4     """
5
6     def __init__(self):
7         self.tete: Maillon = None
```

## Code 3 – Objet Liste

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

**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.

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

```
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.

## Liste chaînée

Principe

Comparaison avec un tableau

## Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

```
1 def __str__(self):
2     m = self.tete
3     s = ""
4     while m is not None:
5         s += str(m.valeur)+" - "
6         m = m.suivant
7     else:
8         s += "fin"
9     return s
```

```
1 print(lst)
```

Code 5 – Afficher la liste

### 1. Liste chaînée

### 2. Implémentation

### 3. Manipuler une liste chaînée

#### 3.1 Taille de la liste

#### 3.2 N-ième élément

#### 3.3 Insertion

#### Liste chaînée

Principe

Comparaison avec un  
tableau

#### Implémentation

#### Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

# 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`.

Liste chaînée

Principe

Comparaison avec un tableau

Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée**Taille de la liste**

N-ième élément

Insertion

```
1 def taille_rec(self, m: Maillon) -> int:
2     """
3     méthode interne pour calculer la taille
4     de la chaîne
5     """
6     if m is None:
7         return 0
8     else:
9         return 1 + self.taille_rec(m.suivant)
```

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée**Taille de la liste**

N-ième élément

Insertion

```
1 def taille(self) -> int:
2     """
3     appel principal de la méthode récursive
    pour mesurer
4     la taille de la chaîne
5     """
6     return self.taille_rec(self.tete)
```

```
1 print(lst.taille())
```

Code 6 – Affichage de la taille de la liste

## Liste chaînée

Principe  
Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste  
N-ième élément  
Insertion

```
1 def __len__(self) -> int:
2     maillon = self.tete
3     taille = 0
4     while maillon is not None:
5         maillon = maillon.suivant
6         taille += 1
7     return taille
```

```
1 print(len(lst))
```

Code 7 – Appel de la fonction `len`

## 1. Liste chaînée

## 2. Implémentation

## 3. Manipuler une liste chaînée

### 3.1 Taille de la liste

### 3.2 N-ième élément

### 3.3 Insertion

#### Liste chaînée

Principe

Comparaison avec un tableau

#### Implémentation

#### Manipuler une liste chaînée

Taille de la liste

**N-ième élément**

Insertion

# 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  $n$ -ième élément de la liste démarrant à  $m$
2. Écrire la méthode `get_element(self, n: int) → int` qui renvoie la valeur du  $n$ -ième 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`.

Liste chaînée

Principe

Comparaison avec un tableau

Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion



## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

```
1 def get_element_rec(self, n: int, m: Maillon) -> int:
2     """
3     méthode interne pour renvoyer le n-ième élément.
4     """
5     if n == 0:
6         return m.valeur
7     else:
8         return self.get_element_rec(n-1, m.suivant)
```

## Liste chaînée

Principe  
Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste  
**N-ième élément**  
Insertion

```
1 def get_element_rec(self, n: int, m: Maillon) -> int:
2     """
3     méthode interne pour renvoyer le n-ième élément.
4     """
5     # n est plus grand que la taille de la liste
6     if m is None:
7         raise IndexError("indice invalide")
8     if n == 0:
9         return m.valeur
0     else:
1         return self.get_element_rec(n-1, m.suivant)
```

Code 8 – Avec gestion du dépassement de taille

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

```
1 def get_element(self, n: int) -> int:
2     """
3     appel principal de la méthode récursive pour
4     renvoyer le n-ième élément
5     """
6     return self.get_element_rec(n, self.tete)
```

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

Code 9 – Appel de la fonction

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

```
1 def __getitem__(self, n: int) -> int:
2     """
3     renvoie l'élément de rang n. Les indices
4     commencent à 0.
5     """
6     maillon = self.tete
7     i = 0
8     while i < n and maillon is not None:
9         maillon = maillon.suivant
10        i += 1
11
12    if maillon is None:
13        raise IndexError("indice invalide")
14
15    return maillon.valeur
```

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

## 1. Liste chaînée

## 2. Implémentation

## 3. Manipuler une liste chaînée

### 3.1 Taille de la liste

### 3.2 N-ième élément

### 3.3 Insertion

#### Liste chaînée

Principe

Comparaison avec un  
tableau

#### Implémentation

#### Manipuler une liste chaînée

Taille de la liste

N-ième élément

**Insertion**

# 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.

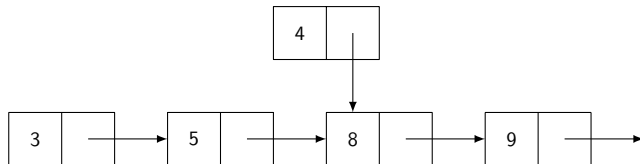


FIGURE 12 – Insertion de 4 au rang 2

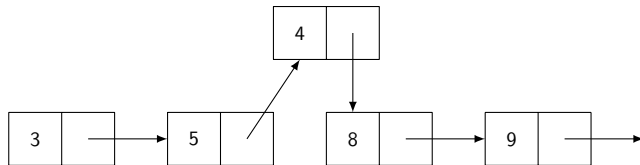


FIGURE 13 – Seconde étape

## Liste chaînée

### Principe

Comparaison avec un tableau

## Implémentation

## Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

## Activité 5 :

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

## Remarque

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

Liste chaînée

Principe

Comparaison avec un tableau

Implémentation

Manipuler une liste chaînée

Taille de la liste

N-ième élément

Insertion

## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

```
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)
```



## Liste chaînée

Principe

Comparaison avec un  
tableau

## Implémentation

Manipuler une liste  
chaînée

Taille de la liste

N-ième élément

Insertion

```
1 def inserer(self, val: int, n: int) -> None:
2     """
3     appel principal de l'insertion pour
4     placer val en n
5     """
6     # gestion du cas particulier où l'
7     insertion est en début
8     if n == 0:
9         nouveau = Maillon(val, self.tete)
10        self.tete = nouveau
11    else:
12        # n-1 pour ajuster la position
13        self.inserer_rec(val, n-1, self.tete)
```