Structures de données

Listes chaînées - Piles et Files - Arbres - Graphes

M. Tellene

Structures de données

Les structures de données jouent un rôle essentiel en informatique. Elles permettent d'organiser l'information pour la traiter ensuite efficacement.

Pour une même information, de nombreuses structures de données peuvent être utilisées. Le choix dépend des opérations que l'on souhaite effectuer et de leur efficacité.

Structures de données

Vous connaissez déjà des structures de données.

Structures de données

Vous connaissez déjà des structures de données.

$\rightarrow$ les tableaux : `L = []`

$\rightarrow$ les tableaux associatifs : `d = {}`

Structures de données

Dans ce cours divisé en plusieurs parties, nous verrons 5 autres structures de données :

- les listes chaînées
- les piles et les files
- les arbres
- les graphes

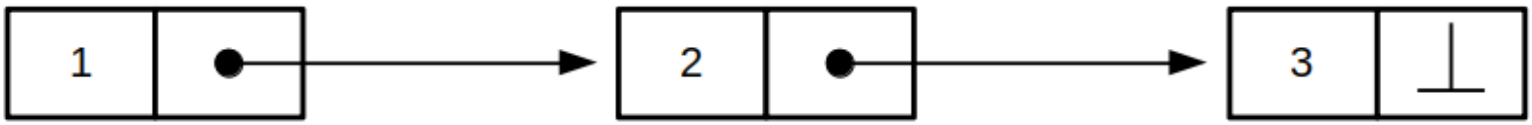
Structures de données - Listes chaînées

Une **liste chaînée** permet de représenter une séquence finie de valeurs, par exemple des entiers

Cette structure est caractérisée par le fait que les éléments sont chaînés entre eux, permettant le passage d'un élément à l'élément suivant

Chaque élément est stocké dans un petit bloc alloué quelque part dans la mémoire que l'on pourra appeler **maillon** ou **cellule**

Structures de données - Listes chaînées



Ici, nous avons une liste contenant trois élément : 1, 2 et 3

Chaque élément (maillon) de la liste est caractérisé par une valeur (case de gauche) et l'adresse mémoire du maillon suivant dans la liste (case de droite)

Le symbole $\perp$ indique que le maillon n'a pas de suivant

Structures de données - Listes chaînées

La liste chaînée prend plus, autant ou moins de mémoire qu'un tableau pour stocker un même nombre d'éléments ?

Structures de données - Listes chaînées

La liste chaînée prend plus, autant ou moins de mémoire qu'un tableau pour stocker un même nombre d'éléments ?

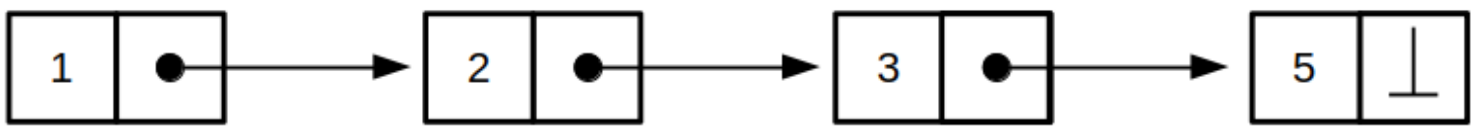
$\rightarrow$ Plus, car il faut stocker la valeur du maillon et l'adresse mémoire du suivant

Structures de données - Listes chaînées

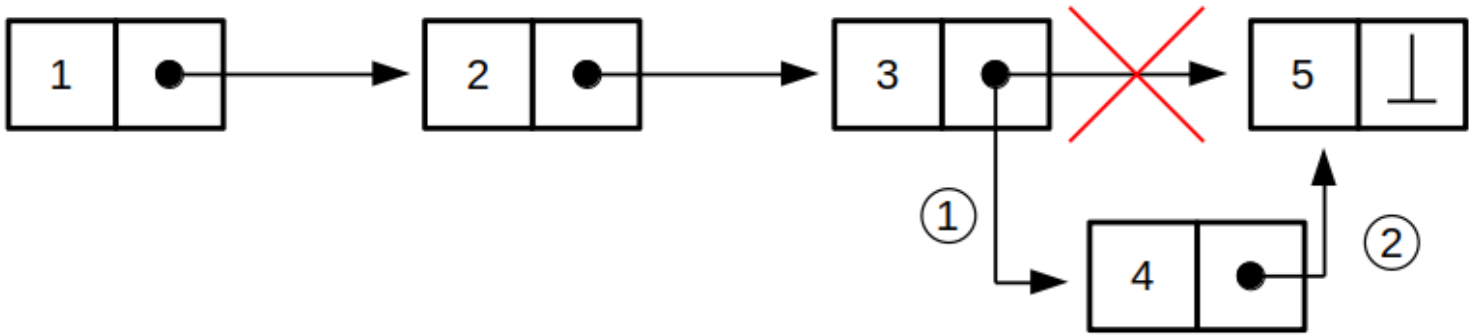
Bien que l'on prend plus de mémoire, utiliser une liste chaînée permet de réaliser certaines opérations plus efficacement qu'avec un tableau.

Exemple de l'insertion d'un élément entre deux éléments consécutifs

Insérer la valeur 4 entre la valeur 3 et 5 :

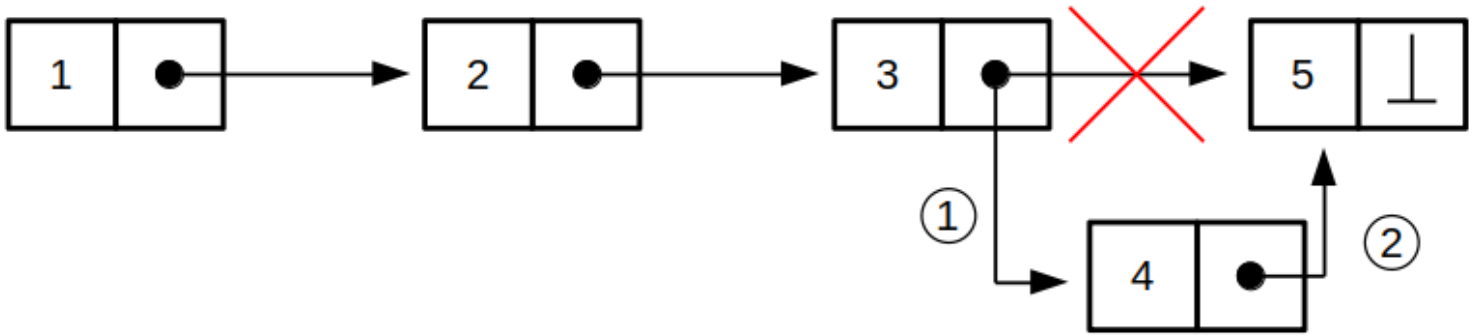


Structures de données - Listes chaînées



Pour insérer 4 entre 3 et 5, on fait pointer 3 vers 4 et 4 vers 5

Structures de données - Listes chaînées



Pour insérer 4 entre 3 et 5, on fait pointer 3 vers 4 et 4 vers 5

De la même façon, on peut supprimer un maillon avec seule affectation : il suffit de faire pointer le maillon précédent vers le maillon suivant, pour « sauter » par-dessus le maillon supprimé

Structures de données - Listes chaînées

La classe `Maillon`

```
1 class Maillon:
2
3     def __init__(self, info, suivant):
4         self.info = info
5         self.suivant = suivant
```

Structures de données - Listes chaînées

Méthodes de la classe `ListeChainee`

- constructeur `ListeChainee()`
  - Postcondition : on crée deux attributs : `tete` et `queue` initialisés à `None`
- Méthode `est_vide` $\rightarrow$ `bool`
  - Résultat : renvoie `True` si la liste est vide, `False` sinon
- Méthode `vider`
  - Postcondition : la liste ne contient plus aucun maillon
- Méthode `insertion_en_tete(x)`
  - Postcondition : ajoute un maillon ayant la valeur `x` en tête de liste
- Méthode `insertion_en_queue(x)`
  - Postcondition : ajoute un maillon ayant la valeur `x` en queue de liste
- Méthode `rechercher_element(x)` $\rightarrow$ `int`
  - Résultat : renvoie l'indice du maillon ayant pour valeur `e`

Structures de données - Listes chaînées

- Méthode `ieme_element(i)` $\rightarrow$ tous types
  - Précondition : $0 \leq i \leq$ *nombre éléments*
  - Résultat : renvoie la valeur du maillon à l'indice `i`
- Méthode `modifier_ieme_element(x, i)`
  - Précondition : $0 \leq i \leq$ *nombre éléments*
  - Postcondition : la valeur du maillon à l'indice `i` devient `x`
- Méthode `afficher()`
  - Postcondition : les valeurs des maillons sont affichées
- Méthode `suppression_en_tete()`
  - Postcondition : supprime le maillon en tête de liste
- Méthode `suppression_en_queue()`
  - Postcondition : supprime le maillon en queue de liste
- Méthode `insertion_element(x, i)`
  - Précondition : $0 \leq i \leq$ *nombre éléments*
  - Postcondition : un maillon ayant pour valeur `x` est ajouté à l'indice `i`

Structures de données - Listes chaînées

Exemple d'utilisation d'une liste chaînée

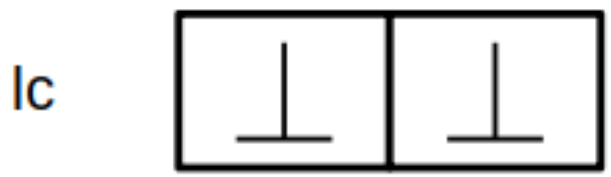
```
1 lc = ListeChainee()
2 lc.insertion_en_tete(12)
3 lc.insertion_en_tete(5)
4 lc.insertion_en_queue(3)
5 lc.suppression_en_tete()
6 lc.insertion_element(8,1)
```

Nous pouvons remarquer que lors de l'utilisation de la classe `ListeChainee`, nous ne créons pas de maillon, ceci s'appelle la **barrière d'abstraction**

Structures de données - Listes chaînées

Exemple d'utilisation d'une liste chaînée

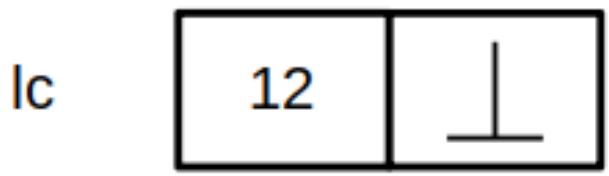
```
1 *lc = ListeChainee()
2 lc.insertion_en_tete(12)
3 lc.insertion_en_tete(5)
4 lc.insertion_en_queue(3)
5 lc.suppression_en_tete()
6 lc.insertion_element(8,1)
```



Structures de données - Listes chaînées

Exemple d'utilisation d'une liste chaînée

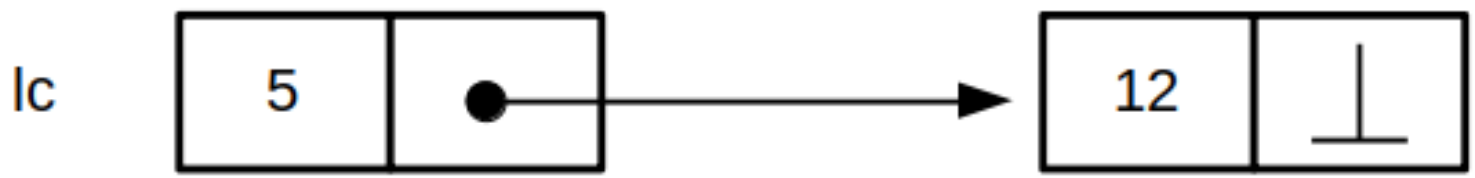
```
1 lc = ListeChainee()
2 *lc.insertion_en_tete(12)
3 lc.insertion_en_tete(5)
4 lc.insertion_en_queue(3)
5 lc.suppression_en_tete()
6 lc.insertion_element(8,1)
```



Structures de données - Listes chaînées

Exemple d'utilisation d'une liste chaînée

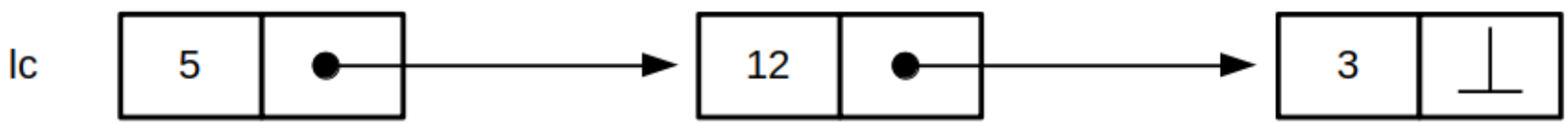
```
1 lc = ListeChainee()
2 lc.insertion_en_tete(12)
3 *lc.insertion_en_tete(5)
4 lc.insertion_en_queue(3)
5 lc.suppression_en_tete()
6 lc.insertion_element(8,1)
```



Structures de données - Listes chaînées

Exemple d'utilisation d'une liste chaînée

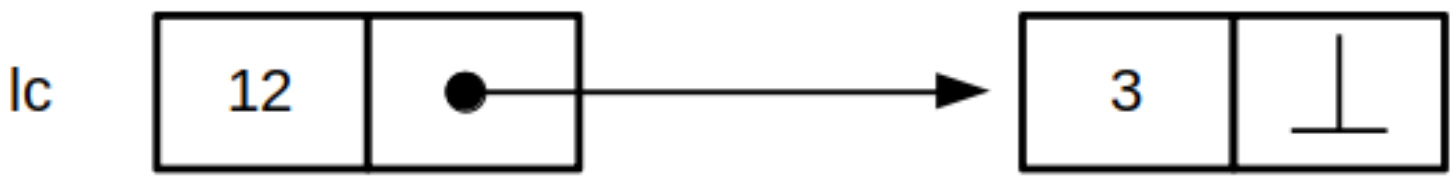
```
1 lc = ListeChainee()
2 lc.insertion_en_tete(12)
3 lc.insertion_en_tete(5)
4 *lc.insertion_en_queue(3)
5 lc.suppression_en_tete()
6 lc.insertion_element(8,1)
```



Structures de données - Listes chaînées

Exemple d'utilisation d'une liste chaînée

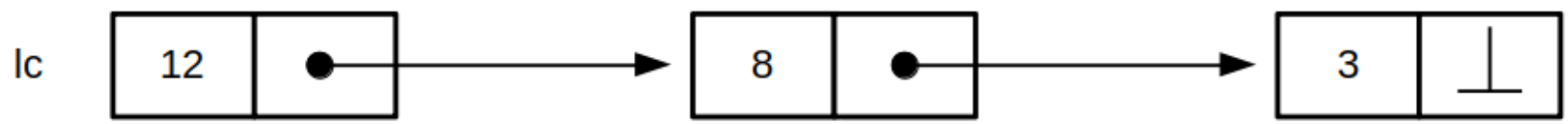
```
1 lc = ListeChainee()
2 lc.insertion_en_tete(12)
3 lc.insertion_en_tete(5)
4 lc.insertion_en_queue(3)
5 *lc.suppression_en_tete()
6 lc.insertion_element(8,1)
```



Structures de données - Listes chaînées

Exemple d'utilisation d'une liste chaînée

```
1 lc = ListeChainee()
2 lc.insertion_en_tete(12)
3 lc.insertion_en_tete(5)
4 lc.insertion_en_queue(3)
5 lc.suppression_en_tete()
6 *lc.insertion_element(8,1)
```



Structures de données - Listes chaînées

Comparaison de complexité (dans le pire des cas) entre les tableaux et les listes chaînées

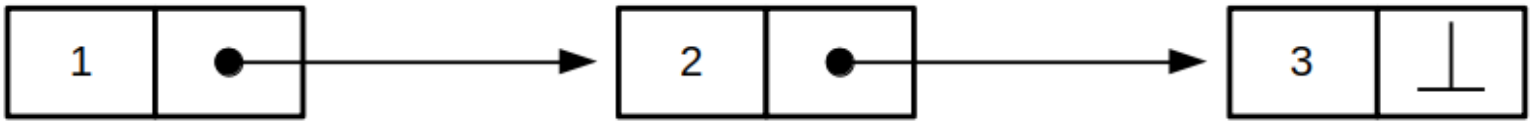
Structure de données	Accès	Recherche	Insertion	Suppression
Tableaux	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Listes chaînées	$O(n)$	$O(n)$	$O(1)$	$O(1)$

n représente le nombre d'éléments dans la structure de données

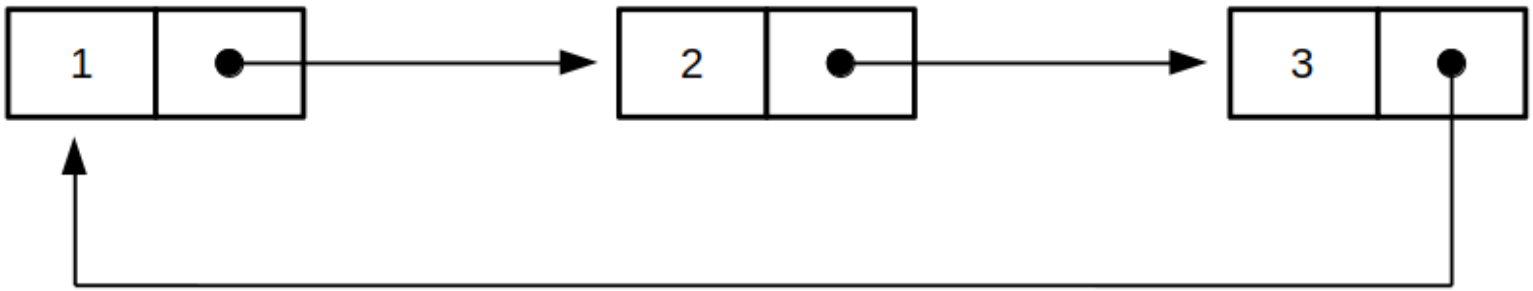
Structures de données - Listes chaînées

Il existe d'autres types de listes

Liste simplement chaînée

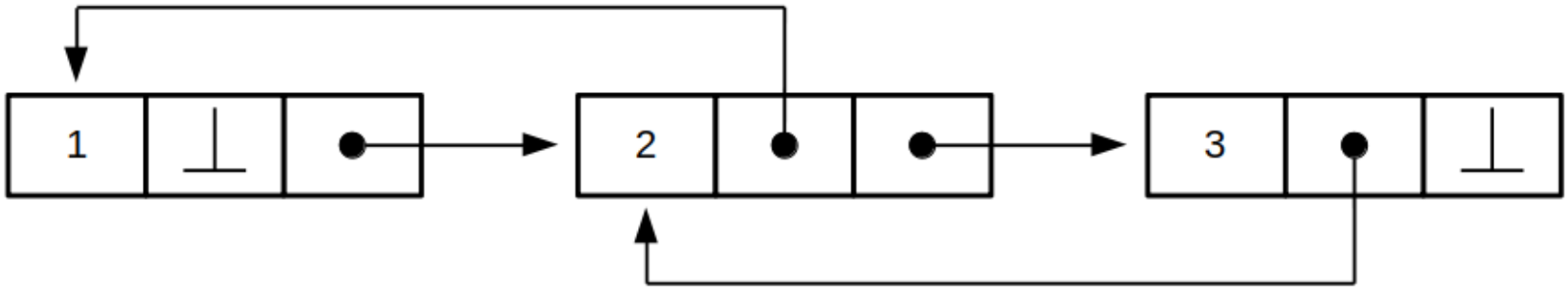


Liste simplement chaînée circulaire



Structures de données - Listes chaînées

Liste doublement chaînée



Liste doublement chaînée circulaire