

6 – Exemple de structures de données linéaire implémentées avec des tableaux ou des listes. Applications

I- Structures linéaires

a- Les tableaux

- Définition :

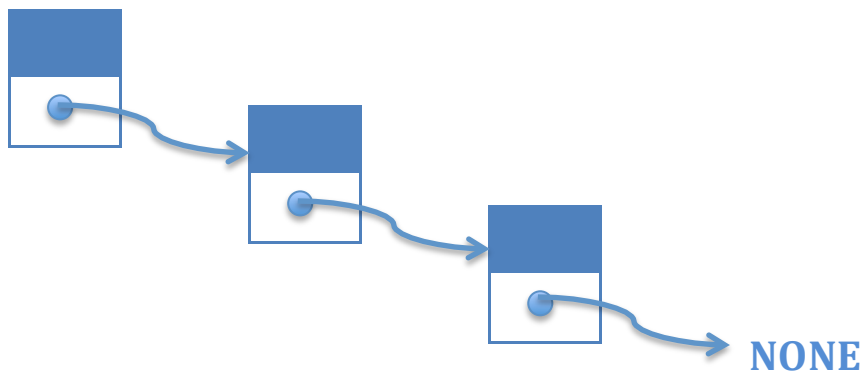
Généralement, n case de même taille pour stocker un même type



- Taille = nombre d'éléments / de cases
- Accès aux éléments : rapide car taille de case fixé. L'accès se fait avec un saut de mémoire au partir du début du tableau jusqu'à l'indice $i \times \text{taille}$ d'une case. Cela se fait donc d'une manière arithmétique.
- Mémoire contigüe : les cases du tableau sont les unes à la suite des autres dans la mémoire.




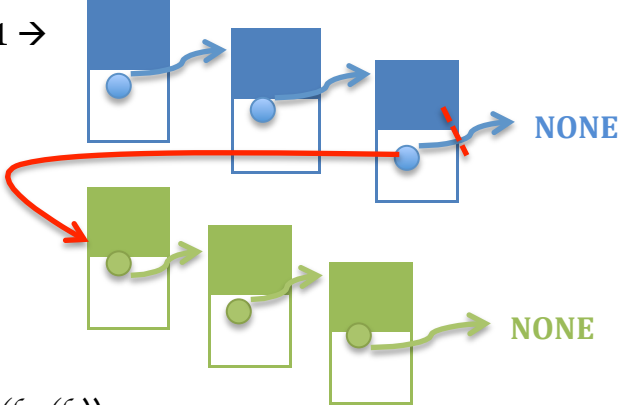
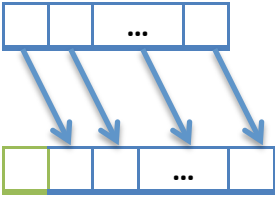
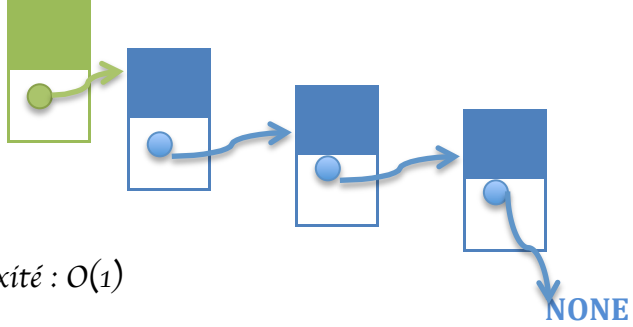

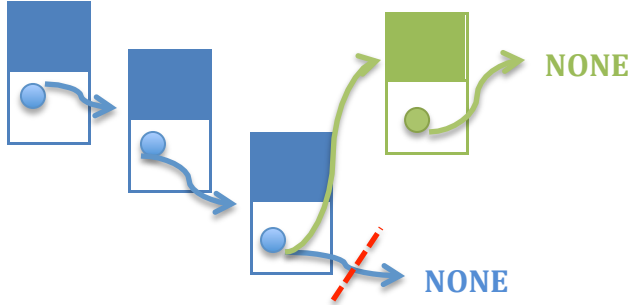
b- Les listes chaînées

- Définition

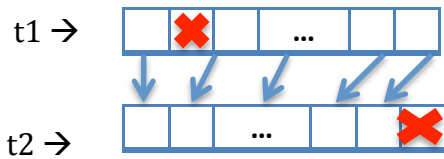


- Mémoire avec indirection : on suit un chemin
- Longueur : calcul en parcourant toute la liste : $O(n)$
- Accès aux éléments : plus pénible, parcours de la liste
- Mémoire discontinue et sauts aléatoires

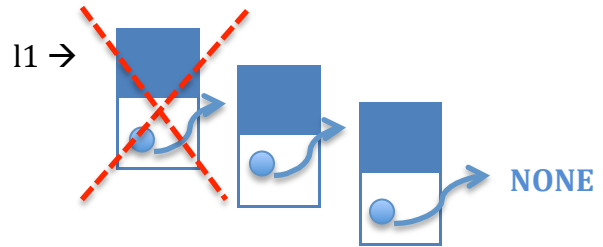
c- Opérations sur les structures linéaires

Mémoire contigüe	Mémoire avec indirection
Concaténation	
<p>Arithmétique sur les adresses possible</p> <p>t1 → </p> <p>t2 → </p> <p>En supposant qu'il y ait de la place en mémoire derrière t1, on recopie t2 à la suite sinon on recopie la concaténation à un autre endroit mémoire.</p> <p></p> <p>Au pire : $O(\text{len}(t1)+\text{len}(t2))$</p>	<p>Pas d'arithmétique -> on stock les adresses</p> <p>l1 → </p> <p>$O(\text{len}(l1))$</p>
Ajout en tête	
<p>t1 → </p> <p>Complexité : $O(1)$</p>	<p>l1 → </p> <p>Complexité : $O(1)$</p>
Ajout en fin	
<p>t1 → </p> <p>Complexité : $O(1)$</p> <p>Si recopie : $O(\text{len}(t1))$</p>	<p>l → </p> <p>Complexité : $O(\text{len}(l))$</p>

Suppression en tête / milieu

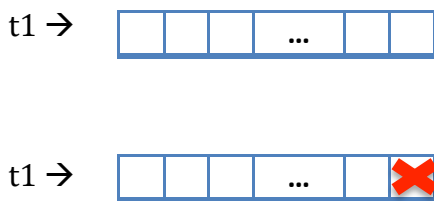


Complexité au pire : $O(\text{len}(t1))$

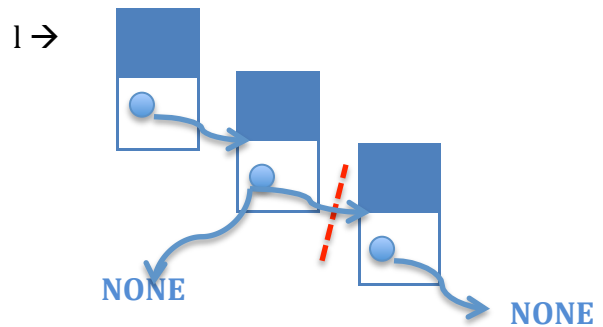


Complexité en tête : $O(1)$
au milieu : $O(i)$

Suppression en fin



Complexité : $O(1)$



Complexité : $O(\text{len}(l))$

d- Complément : les listes Python

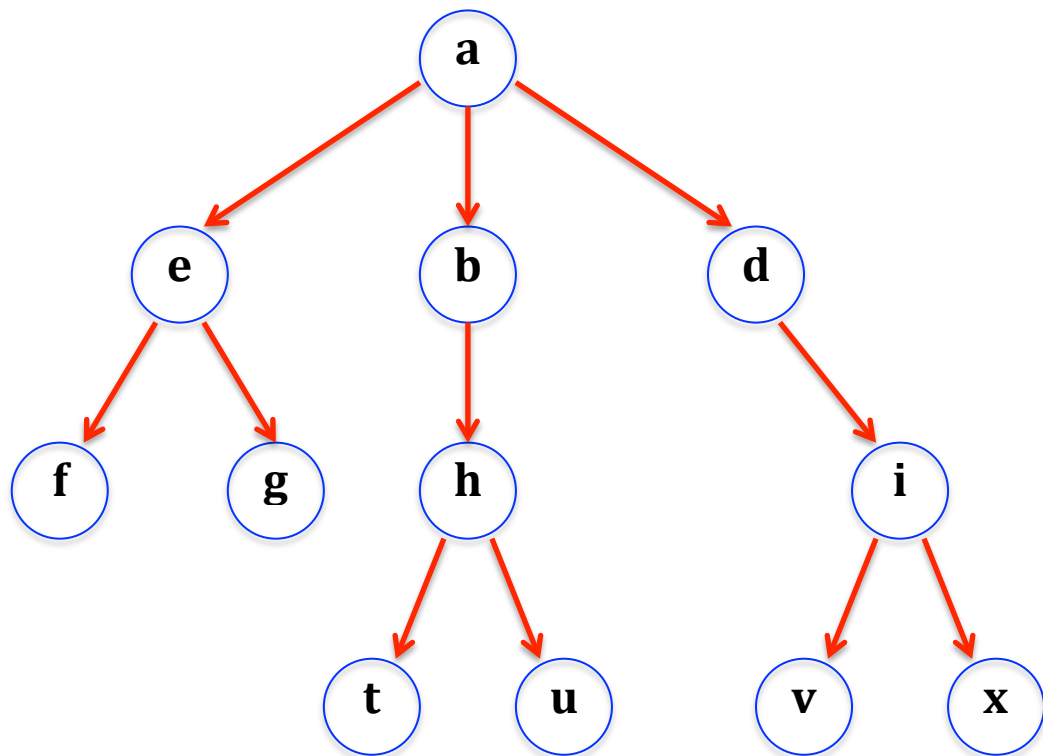
Les listes python sont des tableaux d'objets

[12, None, [5,2], (« bonjour », 3.14)]

La liste python stocke les adresses mémoires de ces objets et non les objets en tant que tel.

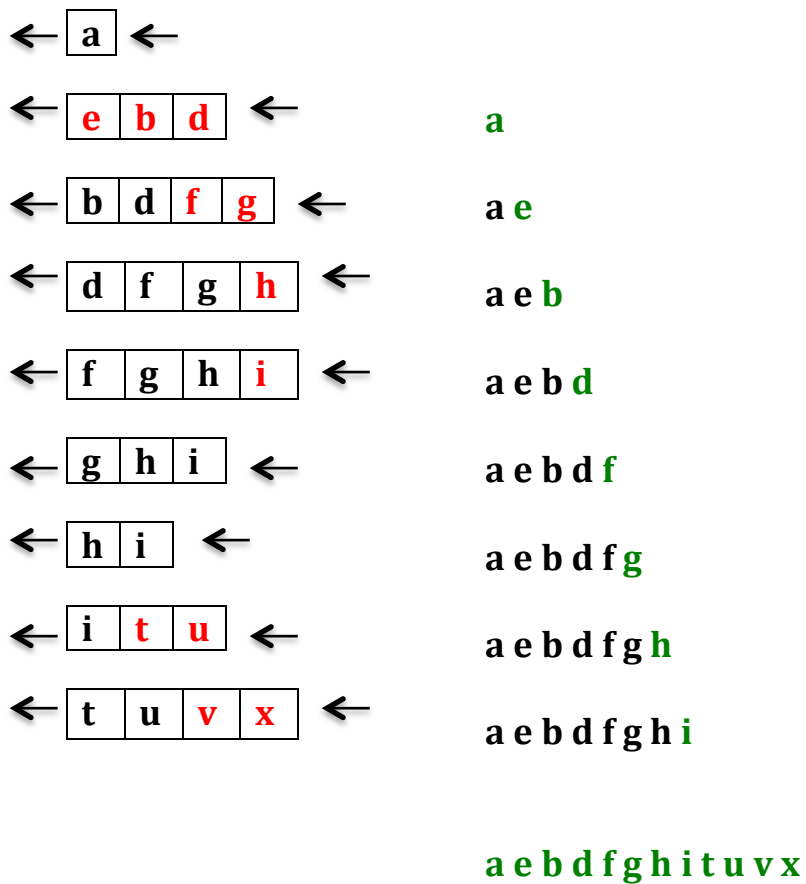
AdresseObjet1	AdresseObjet2	...	AdresseObjetn
---------------	---------------	-----	---------------

- Parcours en largeur d'un graphe avec une file



Principe :

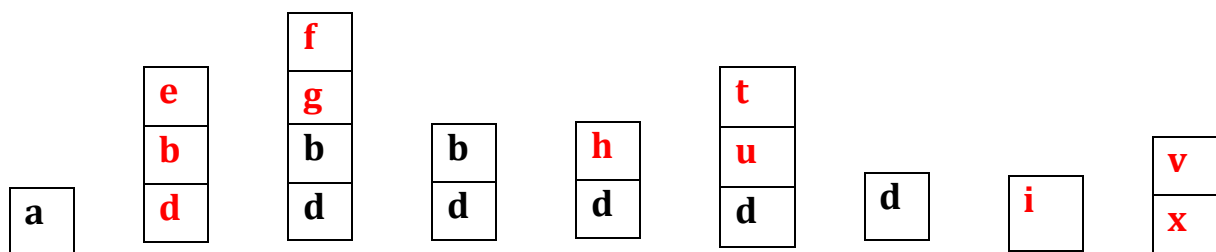
On place dans une file l'élément le plus en haut de l'arbre (ici « a »), puis on l'enlève en ajoutant dans la file ses enfants de gauche à droite.



➤ Parcours en profondeur d'un graphe avec une pile

Principe :

On place dans un pile l'élément le plus en haut de l'arbre (ici « a »), puis on l'enlève en ajoutant, de droite à gauche, ses enfants dans la pile. On continue jusqu'à ce que la pile soit vide.



a

a e

a e f

a e f g

a e f g b

a e f g b h

a e f g b h t u

a e f g b h t u d

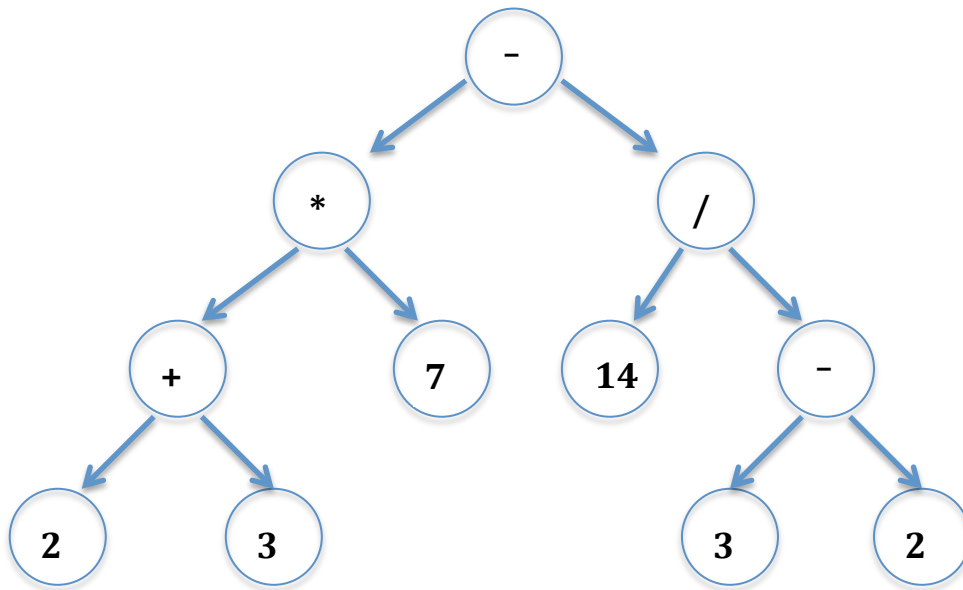
a e f g b h t u d i

a e f g b h t u d i v x

➤ Calcul d'expression algébrique post-fixé avec un pile

$$(2 + 3) * 7 - 14 / (3 - 2)$$

Création d'un graphe pour transformer l'écriture infixée en post fixé



Ecriture post-fixée : $2\ 3\ +\ 7\ *\ 14\ 3\ 2\ -\ /\ -$

Principe :

Tant qu'on lit un chiffre, on l'ajoute dans la pile. Quand on lit un opérateur, on fait l'opération sur les 2 nombres en haut de la pile et on ajoute le résultat à la pile tout en retirant les 2 nombres.

