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

#### I- Structures linéaires

- a- Les tableaux
  - Définition :

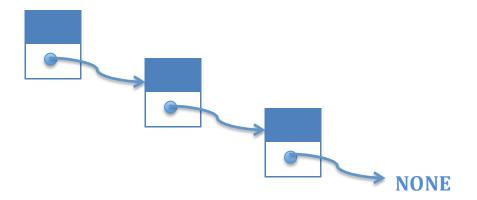
Généralement, n case de même taille pour stocker un même type

...

- 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\*taille d'une case. Cela ce 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 chainées

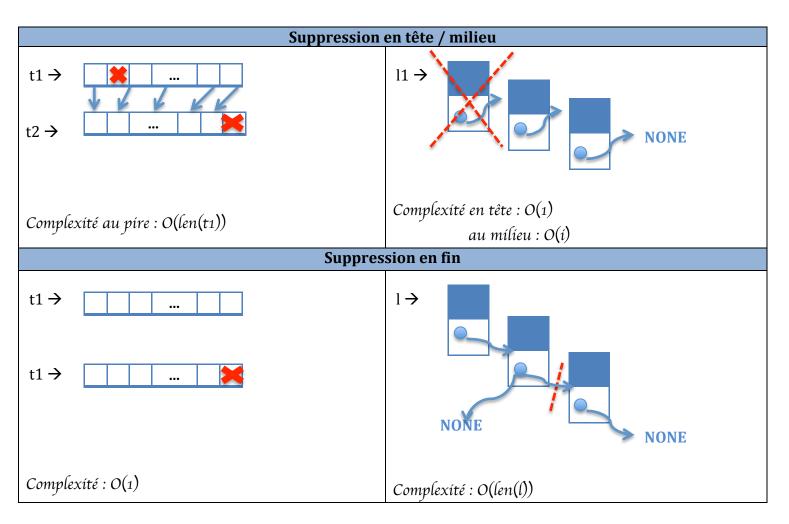
Définition



- 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- <u>Opérations sur les structures linéaires</u>

Mémoire contigüe	Mémoire avec indirection
Conca	ténation
Arithmétique sur les adresses possible	Pas d'arithmétique -> on stock les adresses
$t1 \rightarrow \qquad \dots \\ t2 \rightarrow \qquad \dots$	l1 → NONE
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.	NONE
Au pire : O(len(t1)+len(t2))	O(len(l1))
Ajout	en tête
t1 →	11 ->
Complexité : O(len(t1))	Complexité : O(1) NONE
Ajou	t en fin
t1 →  Complexité : O(1)  Si recopie : O(len(t1))	l → NONE  NONE
	Complexité : O(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	/ Adragga()hiat')		AdresseObjetn
Auresseometr	Auresseubietz	l	i Auresseobieur
		I ===	

#### II- Piles et Files

#### a- Pile

- Définition : Last in, first out
- Pile d'exécutions

#### b- File

Définition : first in , first outParcours largeurs d'un arbre

## III- Applications pour les piles et les files

- Double Files d'attente pouvant représenter une queue à un téléski, avec une file « école de ski » et une file « reste de monde ». On place dans le téléski une personne de chaque file, on gère les cas où l'une des deux files est vide.
- Pile d'exécution (ex : Fibonacci )

```
1 ▼ def fibo(n)
2    if n<=1:
3    return 1
4    return fibo(n-1)+fibo(n-2)</pre>
```

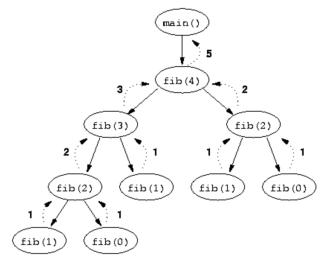
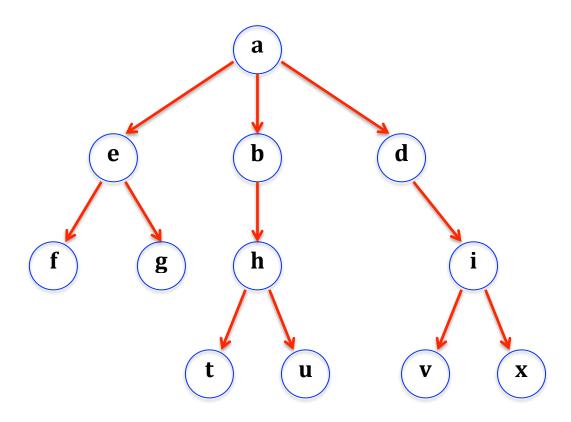


Figure 2 : Arbre d'invocation de la suite de Fibonacci

		4	3 4	2 3 4	1 2 3 4	2 3 4	0 2 3 4	2 3 4	3 4	1 3 4	3	4	2 4	1 2 4	2 4	0 2 4	2 4	4		
	π	π	π	π	π	π	π	π	π	π	π	π	π	π	π	π	π	π	π	

Figure 1 : Pile d'exécution pour la suite de Fibonacci

> Parcours en largeur d'un graphe avec une file



## **Principe:**

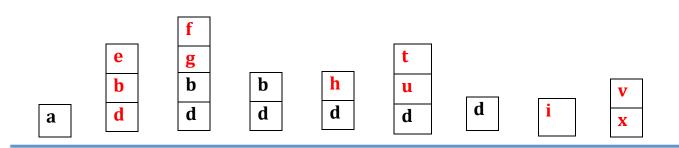
On place dans une file l'élément le plus en haut de l'arbre (ici « a »), puis on l'enlève en ajouter dans la file ses enfants de gauche à droite.

aebdfghituvx

> 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

aefg

aefgb

aefgbh

aefgbhtu

aefgbhtud

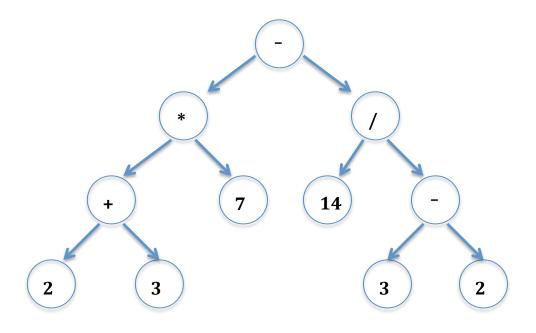
aefgbhtudi

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 : 23 + 7 \* 1432 - / -

## 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.

