



Développement Efficace

2. Structures de Données de base

BUT INFO 2A

David Auger

IUT de Vélizy - UVSQ



2.2- Tableaux (rigides)

Définition

Un tableau est une structure de données de taille fixe, indexée par les entiers 0, 1, 2, ... Dans la mémoire les éléments sont stockés de façon contigüe dans des espaces mémoire de même taille, ce qui permet de calculer l'adresse correspondant à un indice par simple multiplication.

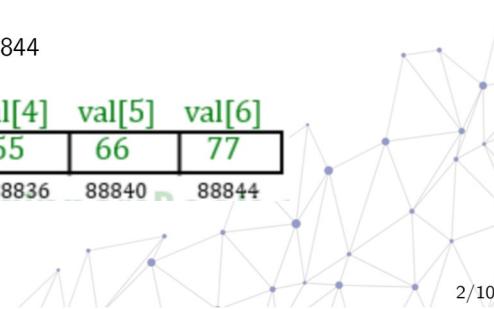
Exemple en C :

Dans un tableau d'int (4 octets par int) dont le premier élément (indice 0) est à l'adresse fictive 88820, l'élément d'indice 6 se trouve à l'indice

$$88820 + 6 \times 4 = 88844$$

val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]
11	22	33	44	55	66	77

88820 88824 88828 88832 88836 88840 88844



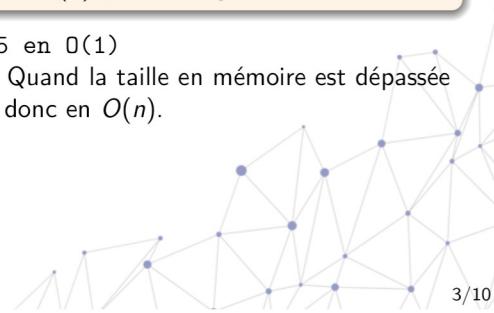
2.3- Tableaux

Complexités principales

- ▶ Accès direct à une case : $O(1)$
- ▶ Modification d'une case : $O(1)$
- ▶ Recherche d'une valeur : $O(n)$
- ▶ Insertion/suppression : non adaptée (taille fixe) – sinon $O(n)$ par décalage

Exemple Python : `T = [0]*100` puis `T[45] = 25` en $O(1)$

Attention : le type list de python est un tableau rigide. Quand la taille en mémoire est dépassée il faut tout déplacer en mémoire. L'opération append est donc en $O(n)$.



2.4- Tableaux rigides triés

Définition : Un tableau rigide dont les éléments sont rangés par ordre croissant.

Complexités principales

- ▶ Recherche dichotomique : $O(\log n)$
 - ▶ Accès/modification : $O(1)$
 - ▶ Insertion/suppression : $O(n)$ (décalage nécessaire)
-
- ▶ **Point fort** : la recherche dichotomique est rapide
 - ▶ **Point faible** : non adapté à l'insertion/suppression
- 

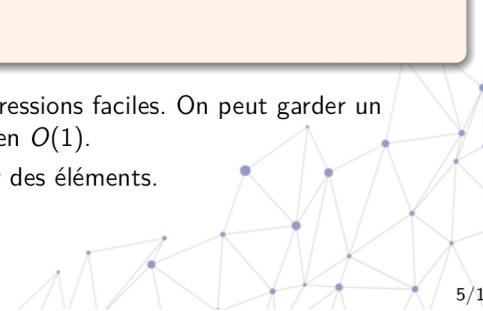
4/10

2.5- Listes chaînées

Définition : Une suite de cellules, chaque cellule contient une valeur et un pointeur vers la suivante.
On garde une référence sur la première (et éventuellement dernière) cellule.

Complexités principales

- ▶ Accès à l'élément i : $O(i)$
 - ▶ Insertion/suppression en tête : $O(1)$
 - ▶ Insertion/suppression en position i : $O(i)$ sauf si on a un pointeur ou référence directe sur la position, c'est alors $O(1)$
 - ▶ Recherche : $O(n)$
-
- ▶ **Point fort** : la structure permet des insertions/suppressions faciles. On peut garder un pointeur sur la dernière cellule pour ajouter à la fin en $O(1)$.
 - ▶ **Point faible** : Il faut parcourir la liste pour chercher des éléments.



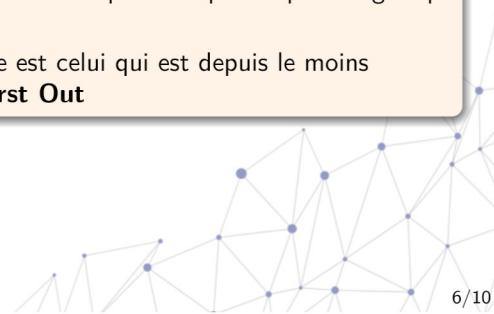
2.6- Piles et Files

Les piles et files sont des structures de données où on peut uniquement

- ▶ ajouter des éléments
- ▶ extraire un élément, mais sans choisir lequel.

Fonctionnement

- ▶ Dans le cas des files, l'élément que l'on peut extraire est celui qui est depuis le plus longtemps dans la file. **principe FIFO : First In, First Out**
- ▶ Dans le cas des piles, l'élément que l'on peut extraire est celui qui est depuis le moins longtemps dans la pile. **principe LIFO : Last In, First Out**



2.7- Files FIFO (queue)

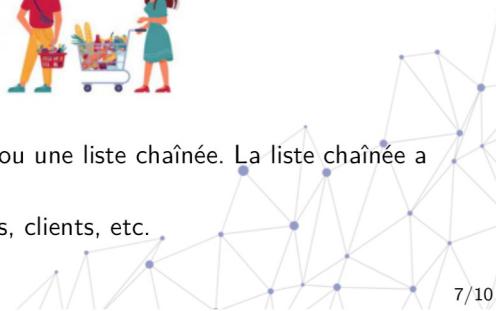
Définition : Structure **First In, First Out**. Insertion en **queue**, suppression en **tête**.

Complexités principales

- ▶ Enfiler (enqueue) : $O(1)$
- ▶ Défiler (dequeue) : $O(1)$
- ▶ Accès direct : non adapté



- ▶ Deux implémentations principales : dans un tableau ou une liste chaînée. La liste chaînée a l'avantage d'avoir une taille variable.
- ▶ **Applications** : processus, gestions de sockets, tâches, clients, etc.



2.8- Files FIFO (queue)

Les files en Python :

```
1 from collections import deque // c'est une liste chaine
2
3
4 f = deque()
5 f.append(1) // c'est un appendright
6 f.append(2)
7 f.append(3)
8 e = f.popleft() // e vaut 1, il reste 2,3 dans la file
```



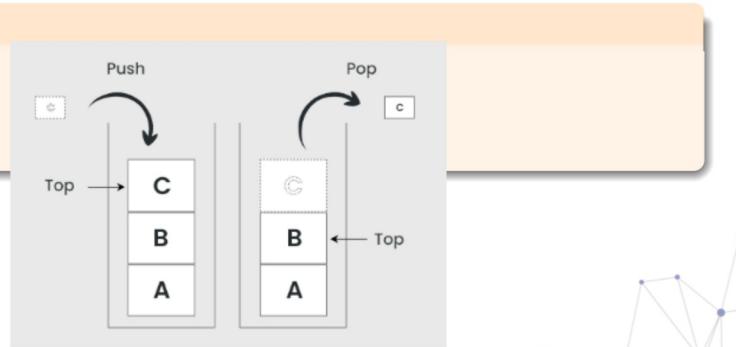
8/10

2.9- Piles LIFO (stack)

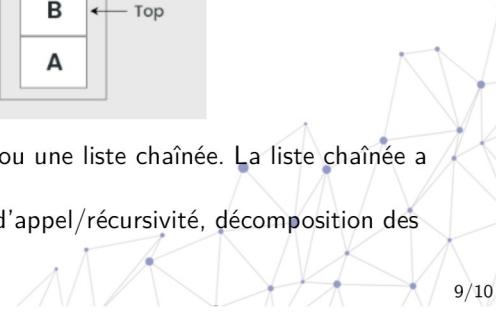
Définition : Structure **Last In, First Out**. Les insertions et suppressions se font au **sommet de la pile**.

Complexités principales

- Empiler (push) : $O(1)$
- Dépiler (pop) : $O(1)$
- Accès direct : non adapté



- Deux implémentations principales : dans un tableau ou une liste chaînée. La liste chaînée a l'avantage d'avoir une taille variable.
- **Applications** : navigateurs internet, CTRL+Z, pile d'appel/récursivité, décomposition des parenthèses.



2.10-. Piles LIFO (queue)

Les piles en Python :

```
1 from collections import deque // c'est une liste chaine
2
3 f = deque()
4 f.append(1) // c'est un appendright
5 f.append(2)
6 f.append(3)
7 e = f.popright() // e vaut 3, il reste 1,2 dans la pile
```

10/10