Structures : Piles, Files , Listes

On va ici étudier les structures linéaires : On peut définir un structure linéaire comme une structure où à chacun de ses éléments sauf le dernier, on peut associer un autre unique élément, qui est le suivant.

I. Les listes

Les listes sont une structure de données permettant de stocker différentes données et d'y accéder librement. On parle de structures de données séquentielles

Cette notion de listes a été mise en valeur par le langage de programmation Lisp (list processing) en 1958.

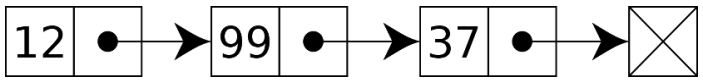
Les spécifications sur les listes sont :

- Tester si la liste est vide
- Ajouter un élément
- Supprimer un élément
- Compter le nombre d'élément.

Une liste est composée de deux éléments : Sa tête et sa queue. Sa tête est le dernier élément ajouté et sa queue contient tous les autres éléments.

Il existe deux types principaux de listes : Les tableaux et les listes chainées.

- **Les tableaux** : on accède à un élément par sa position, comme on l'a vu en python. Le problème des tableaux est que dans certains langages, comme le C, la taille est définie d'entrée. Ainsi tous les éléments du tableau sont stockés dans des espaces mémoires voisins. Dès que l'on veut ajouter un élément, on est coincé. Ainsi , si l'accès à un élément se fait à coût constant, l'insertion ou la suppression d'un élément nécessite la recopie de tout le tableau et donc le coût va être en O(n)
  En langage python, l'utilisation de tableau dynamique permet d'éviter ce problème.

- **<u>Comment fonctionne un tableau dynamique</u>** ? Lors de la création de la liste, on réserve un espace mémoire supérieur à la longueur de la liste, ce qui permet un certain nombre d'ajouts. Lorsque l'espace mémoire est plein et que l'on veut encore ajouter des éléments, il faut réallouer un espace mémoire, que l'on prendra là aussi plus grand que les besoins du moment, et y recopier notre tableau. La construction des tableaux dynamiques s'appuie sur les listes chaînées.
- **Les listes chaînées :** Ici , il n'y a plus de limitation de taille car chaque élément de la liste peut « pointer vers un autre élément » (en fait vers son adresse mémoire)
- 

(source wikipédia)

Chaque élément de la liste possède une valeur et un pointeur vers une autre valeur. Ainsi , si l'on veut insérer un élément dans une liste chaînée, il faut donner sa valeur et faire pointer vers elle l'élément de la liste que devra le précéder.

Exemple : Ecrire un algorithme qui permette d'insérer un élément entre les valeurs 2 et 5 de cette liste chaînée.

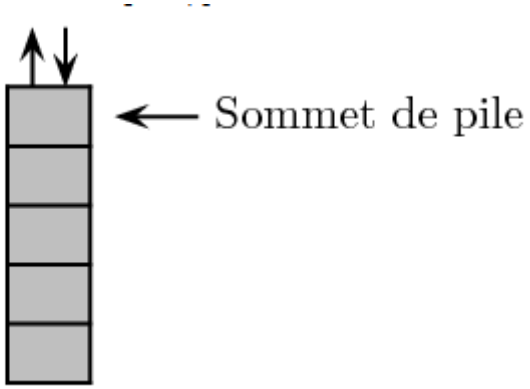
1 →	2 →	5 →	4 →
Adresse 1000	Adresse 1001	Adresse 1010	Adresse 1011

……

Algorithme :

II. Les piles

Dans une pile, on ne peut manipuler que le dernier élément introduit. C'est un modèle de structures de données où l'on peut ajouter des éléments mais où on ne peut traiter que le dernier élément entré. C'est le principe du **<u>LIFO</u>** : ***Last in First out.***
On peut se ramener à une pile d'assiettes : La dernière assiette rangée sera la première à être sortie.



Les spécifications des piles sont :

- Tester si elle est vide
- Ajouter un élément (push) sur le sommet de la pile **: Empiler**
- Supprimer un élément (pop) et le garder en mémoire **: Dépiler**
- Compter son nombre d'éléments
- Accéder à l'élément situé au sommet de la pile.

Lorsque l'on travaille sur les listes en python, la méthode append ajoute un élément sur le sommet de la liste et la méthode pop enlève le dernier élément entré

```
In [1]: l= [1,2,3,4,5,]

In [2]: l.append(6)

In [3]: print(l)
[1, 2, 3, 4, 5, 6]

In [4]: l.pop()
Out[4]: 6

In [5]: print(l)
[1, 2, 3, 4, 5]

In [6]:
```

La notion de pile est utilisée lors de l'appel à des fonctions.

Voici un exemple qui devrait vous permettre de comprendre le fonctionnement de la notion de pile lors de l'appel à plusieurs fonctions .

```
def fonction(n):
    if n>0:
        fonction(n-1)
    print(n)
```

Exécutons ce code avec n= 2

On appelle fonction(1) puis fonction(0) on sort de la condition :

- Affichage de la valeur de n :0
- Exécution de fonction(0) : affichage 1
- Exécution de fonction(1) : affichage de 2

```
In [7]: fonction(2)
0
1
2
```

On dépile : Affichage de 0
Affichage de 1
Affichage de 2

		fonction(0)		
	fonction(1)	fonction(1)	fonction(1)	
fonction(2)	fonction(2)	fonction(2)	fonction(2)	fonction(2)

Remarque : Lorsqu'une fonction s'appelle elle-même , on parle de récursivité. On reverra cet exemple en détail

III. Les Files

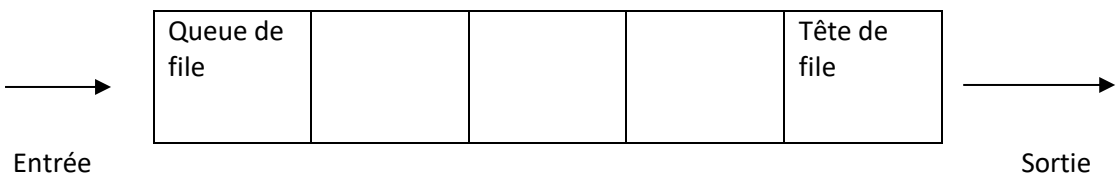
La notion de file en tant que structure de données est à rapprocher de la liste d'attente. Le premier arrivé sera le premier servi, soit ici, premier entré, premier sorti. La notion de queue à une caisse de magasin est aussi explicite.

C'est un modèle où l'on accumule les informations les unes après les autres et où les éléments sont traités selon l'ordre d'arrivée . C'est le principe du **FIFO** : ***First in first out.***

Ce système est utilisé par le processus pour gérer par exemple les impressions. Elles s'exécutent suivant lors ordre d'arrivée.

On peut ici voir le lien avec les listes chainées, chaque élément de la file pointant vers l'adresse mémoire du suivant.

Les spécifications sur les files sont semblables à celles des piles : tester si elle est vide, ajouter un élément en queue de file, retirer, compter, accéder au premier élément.



IV. Les dictionnaires

On a étudié en première les dictionnaires comme structure de données. Pour rappel, un dictionnaire est composé d'u couple clé, valeur.
Les dictionnaires sont des structures mutables : On peut ajouter, modifier, enlever des éléments.
Contrairement aux listes, les dictionnaires ne sont pas des séquences, ce qui signifie que l'on n'accède pas à leur élément par un indice de position.
Les dictionnaires ont un autre petit nom, moins romantique, les tables de hachage.
Sans entrer dans les détails, les tables de hachages sont obtenues à l'aide d'une fonction de …hachage.
Comme une fonction classique, une fonction de hachage prend un élément et calcule son image…qui correspond à une adresse mémoire …où l'on trouve la valeur de la clé.
On reviendra par la suite sur l'intérêt d'une telle construction.
Les spécifications d'un dictionnaire sont :

- Ajouter un élément (qui correspond à un couple « clé, valeur »)
- Supprimer un élément
- Modifier un élément
- Rechercher un élément.

.

<u>Différence entre la recherche dans une liste et un dictionnaire</u>

Pour chercher un élément dans une liste de taille n, on est complexité O(n).
La construction des dictionnaires, de part l'utilisation d'une fonction de hachage permet, connaissant la clé, de trouver immédiatement sa valeur puisque la fonction *« calcule l'image de la clé ».* On est dans en complexité O(1) : le temps de recherche d'un élément ne dépend pas de la taille de la structure .

Une vidéo sur la fonction de hachage.

V. Choix de la structure de données suivant le contexte

Voici différents contextes. A vous de préciser quelles structures de données est la plus adaptée pour traiter le problème.

1. Réaliser un annuaire téléphonique
2. Annuler la dernière opération effectuée sur un ordinateur (crtl Z)
3. Gérer les impressions
4. Aller à la page précédente dans un navigateur web
5. Stocker une liste de nombre que l'on veut modifier à souhait.