Capacités attendues
Spécifier une structure de données par son interface.
Distinguer interface et implémentation.
Écrire plusieurs implémentations d'une même structure de données.

III) Structure de données linéaires

Comme structure de données, vous connaissez la liste: l = [1,2,3] et les dictionnaires dic = 'nom':'Dupont, 'solde':20000. On dit qu'elles sont linéaires car les éléments sont ordonnées et à la suite les uns des autres. Ils sont accessibles par un indice ou une clé.

1) Définition

Une structure de données linéaire est un ensemble contenant une suite d'éléments. Elle est linéaire car elle forme une suite, c'est-à-dire une liste ordonnée d'éléments.
Les arbres et les graphes sont des structures non linéaires.

2) Exemples de sdd linéaires

- Pile
- File
- Liste chainée
- Dictionnaire

IV) Structure de donnée linéaire: la pile

1) Définitions

Une pile est une structure de données qui est un ensemble d'éléments accessibles selon une politique LIFO.
LIFO: Last In First Out = Dernier arrivé, premier sorti.
La structure de données pile peut se comparer à une pile d'assiettes: l'accès à un élément de la pile n'est possible qu'en dépilant le sommet de celle-ci jusqu'à l'élément recherché. L'ajout d'un élément ne peut se faire qu'au sommet de la pile.

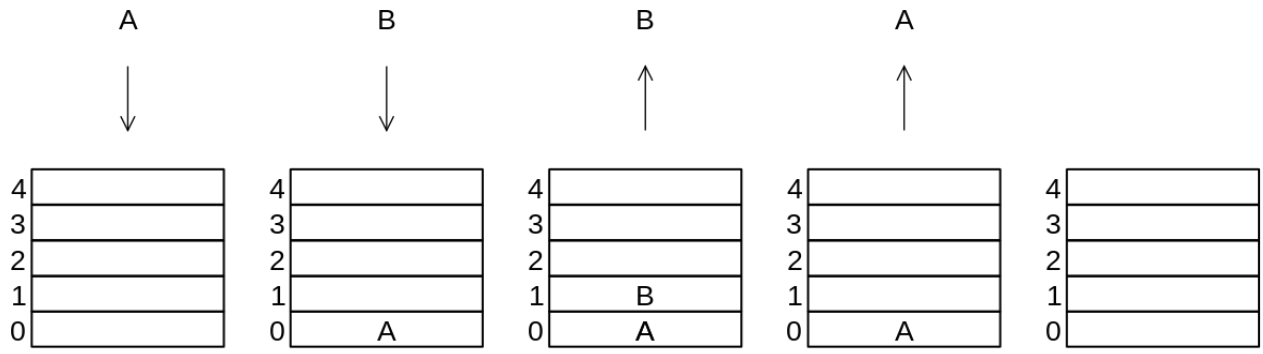

Figure 1: empilement (étapes 1 et 2) et dépilement (étapes 3 et 4) d'une pile

2) Implémentation de la pile

Implémenter une structure de données, c'est programmer cette structure avec son interface.
Voici un exemple en programmation orientée objet d'implémentation d'une pile en utilisant une liste python:

```
class Pile:
    def __init__(self):
        self.liste = []

    def empile(self, e):
        self.liste.append(e)

    def depile(self,e):
        self.liste.pop()

    def est_vide(self):
        return len(self.liste) == 0

    def __repr__(self):
        affichage = "Haut de la pile\n"
        longueur = len(self.liste)
        i = longueur - 1
        while i >= 0:
            affichage = affichage + str(self.liste[i]) +"\n"
            i -= 1
        affichage = affichage +"Bas de la pile"
        return affichage
```

3) Applications de la pile

- Ctrl + Z pour revenir en arrière
- Langage postscript (x y mul correspond à $x \times y$)

- notation polonaise inversée (10×5+4×2 devient, en RPN : 10 5 × 4 2 + /)
- Appel de fonctions dans un compilateur
- Parcours en profondeur d'un graphe

4) Interface de la pile

Une structure de données a une **interface** qui consiste en un ensemble de méthodes pour ajouter, effacer, accéder, réorganiser, etc. les données.

L'interface de la structure de données pile comporte 3 méthodes:

- est_vide(p) qui renvoie Vrai si la pile p est vide
- empiler(p,e) ajoute l'élément e au sommet de la pile p
- dépiler(p) extrait le dernier élément de la pile p et le renvoie

V) File

1) Définitions de la structure de données FILE

Une file est une structure de données qui est un ensemble d'éléments accessibles selon une politique FIFO.
FIFO: First In First Out = Premier arrivé, premier sorti.
La structure de données file peut se comparer à une file d'attente: l'accès à un élément de la file n'est possible qu'en retirant (en défilant) le premier élément. L'ajout d'un élément ne peut se faire qu'après le dernier élément de la file (enfiler).

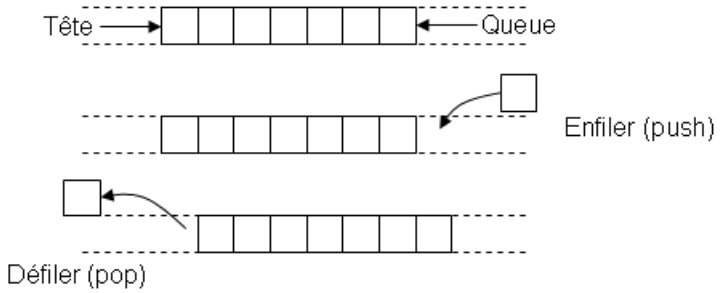

Figure 2: File: vocabulaire

2) Applications de la structure de données FILE

- en général, pour mémoriser temporairement des taches qui doivent attendre pour être traitées
- une imprimante, qui traite les requêtes dans l'ordre dans lequel elles arrivent, et les insère dans une file d'attente
- un algorithme de parcours en largeur d'un graphe utilise une file pour mémoriser les nœuds visités

3) Implémentation de la structure de données FILE

Voici un exemple en programmation orientée objet d'implémentation d'une file en utilisant une liste python:

```
class File:
    def __init__(self):
        # Initialiser une file vide (libre)
        self.file =[]

    def enfiler(self, element):
      """Inserer un element a la queue d une file"""
        return self.file.append(element)

    def defiler(self):
       """Supprimer l element de la tete une file et le retourner"""
        try:
            return self.file.pop(0)
        except Exception as e:
            print( "Error:"+ e)

    def afficherFile(self):
         """Afficher tous les elements de la file"""
        for i in range(len(self.file)):
            print (self.file[i])
```

Le problème de cette implémentation est que l'insertion d'un élément en tête de file est très couteux en terme de mémoire. Cette implémentation n'est donc pas efficace. Une autre implémentation contourne ce problème: c'est l'implémentation d'une file à partir de 2 piles (voir doc). Dans ce cas, les insertions se font toujours par le sommet d'une pile ce qui a un coût constant. Ensuite, la pile est renversée dans une autre pile. L'autre pile est dépilée.

VI) Une autre structure de données linéaire: la liste chainée

1) Description

Une liste chaînée (ou liste liée) est une structure de données composée d'une suite d'éléments de liste.
Chaque enregistrement ou élément d'une liste chaînée est souvent appelé nœud ou maillon.
Un maillon contient une valeur et l'adresse mémoire du maillon suivant:
schéma Une liste chainée aura donc cette représentation: schéma

La liste chainée pointe vers le premier maillon. L'adresse du dernier maillon est None ce qui marque la fin de la liste chaînée.

2) avantages et inconvénients

inconvénients:

- occupe plus de mémoire à cause des adresses en plus des valeurs
- pour accéder à un élément, il faut parcourir toue la liste

avantages:

- pas besoin d'allouer en mémoire plus de place que nécessaire