CH 3 : STRUCTURES DE DONNEES, PILES ET FILES

I. DIFFERENTES TYPES DE STRUCTURES DE DONNEES

Les algorithmes opèrent sur des données qui peuvent être de différentes natures.

A un premier niveau, les données sont considérées de manière abstraite, on se donne une notation pour les décrire et on se donne également un ensemble d'opérations que l'on peut leur appliquer. On parle de **type abstrait de données**.

Ceci permet de créer des algorithmes indépendants du langage, et de définir des types de données « non primitifs », c'est-à-dire non disponibles dans les langages de programmation courants. (Les types « primitifs » sont par exemple en Python int, str, float ...).

Nous étudierons cette année quatre types de structures de données abstraites :

- **Les structures linéaires** : listes, piles et files qui sont l'objet de ce chapitre
- Les structures par accès à clé : les dictionnaires, vus en première
- Les structures hiérarchiques : les arbres
- Les structures relationnelles : les graphes

Une structure de données possède un ensemble de routines (fonctions ou procédures) permettant d'ajouter, d'effacer, d'accéder aux données. Cet ensemble de routines est appelé **interface**. L'**implémentation** de la structure de données contient le code de ces routines.

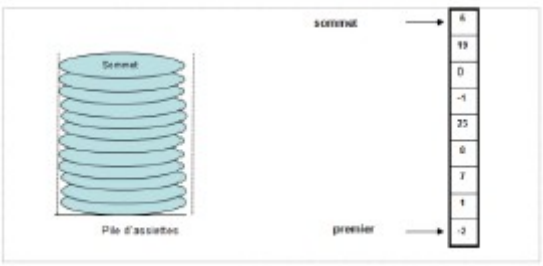
L'interface classique est constituée des quatre routines élémentaires dites CRUD :

- Create : ajout d'une donnée
- Read : lecture d'une donnée
- Update : modification d'une donnée
- Delete : suppression d'une donnée.

introduction

II. LES PILES

On retrouve dans les piles une partie des propriétés vues sur les listes. Dans les piles, il est uniquement possible de manipuler le dernier élément introduit dans la pile. On prend souvent l'analogie avec une pile d'assiettes : dans une pile d'assiettes la seule assiette directement accessible est la dernière assiette qui a été déposée sur la pile.



Les piles sont basées sur le principe LIFO (Last In First Out : le dernier rentré sera le premier à sortir). On retrouve souvent ce principe LIFO en informatique.

Le principe de pile intervient très souvent, dans la vie quotidienne et en programmation :

- Charger/décharger un camion
- Le ctrl-Z de l'ordinateur
- La gestion de l'historique de navigation sur internet

Voici les opérations de base sur une pile :

- CREER_PILE_VIDE() : crée une pile qui est vide
- EMPILER(P, e) : insère l'élément e au sommet de la pile
- DEPILER(P) : l'élément au sommet de la pile est supprimé et est retourné
- EST_VIDE(P) : retourne vrai si la pile est vide, faux sinon

On peut ajouter les opérations suivantes :

- LONGUEUR(P) : taille de la liste
- TETE(P) : retourne le premier élément de la pile sans le dépiler

Exemple : expliquer ce que fait la suite d'instructions suivante :

P = CREER_PILE_VIDE()

EMPILER (P, 3)

EMPILER (P, 2)

N = DEPILER (P)

EMPILER (P, 5)

EMPILER (P, N)

EMPILER (P, 9)

On peut représenter une pile de taille $n$en Python par une liste de n éléments, le dernier élément représentant le sommet de la pile. Empiler consiste donc à ajouter un élément en fin de liste, dépiler à supprimer le dernier élément de la liste.

Ecrire l'algorithme des fonctions EMPILER et DEPILER, puis les coder en Python.

Les tester sur la suite d'instructions précédente.

Coût : le coût de chacune des fonctions EMPILER et DEPILER est O(1).

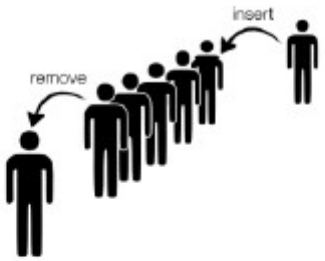
Une insertion d'un élément dans une pile de taille n a un coût en O(n).

Exercices d'application 1 à 3

III. LES FILES

Une file est une structure de données dans laquelle on accède aux éléments suivant la règle du « premier arrivé, premier sorti » : FIFO (First In, First Out). Autrement dit, on ne peut accéder qu'au premier élément de la liste.

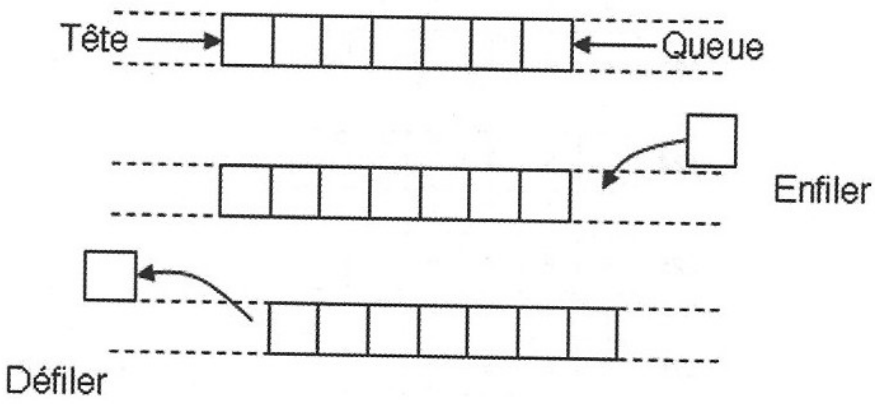
Ceci correspond à la situation concrète d'une file d'attente à un guichet.



Une file a deux attributs : une tête (premier arrivé) et une queue (dernier arrivé).

Les opérations élémentaires de cette structure sont :

- ENFILER(F, x) : on insère la donnée x à la queue de la file F
- DEFILER(F) : on retire la donnée de tête de la file F et on la retourne



On peut également utiliser les opérations suivantes :

- CREER_FILE_VIDE() : renvoie un objet de type file (vide)
- EST_VIDE(F) : retourne un booléen : Vrai si la file est vide, Faux sinon.

Expliquer ce que fait la suite d'instructions suivante :

F = CREER_FILE_VIDE()

ENFILER (F , 21)

ENFILER (F, 22)

ENFILER (F , 23)

N = DEFILER(F)

ENFILER (F , 24)

ENFINLER (F , N)

N = DEFILER (F)

On peut représenter une file de taille n en Python par une liste de n éléments, le premier élément représente la tête et le dernier élément la queue. Enfiler consiste donc à ajouter un élément à la fin de la liste, défiler à supprimer et retourner le premier élément de la liste.

Ecrire l'algorithme des fonctions ENFILER et DEFILER, puis les coder en Python.

Les tester sur la suite d'instructions précédente.

Coût : le coût de chacune des fonctions EMFILER et DEFILER est O(1).

Une insertion d'un élément dans une file de taille n a un coût en O(n).

ACT File d'attente

Exercices d'application 4 et 5

Exercices de synthèse : exos 6 à 8

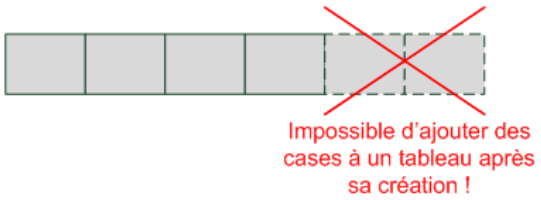
IV. LES LISTES CHAINEES

1. Définition

En programmation, les données peuvent être structurées en tableaux.

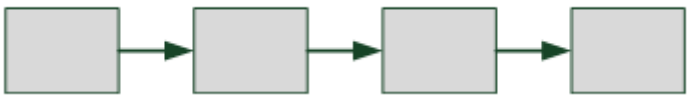
Un tableau est un ensemble de « cases » de nombre fini :

(ici 4 cases)

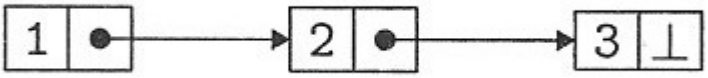


On ne peut pas augmenter la taille du tableau après sa définition

Pour remédier à ce problème, on définit des objets de type **« listes chaînées »**, ou chaque élément « pointe » sur le suivant.



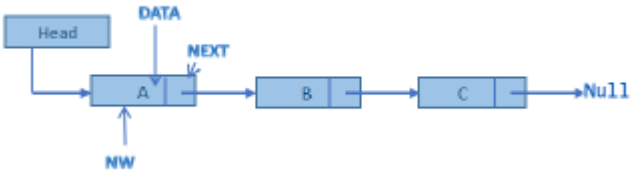
Chaque élément est stocké dans un bloc alloué dans la mémoire, que l'on appellera cellule, et est accompagné d'une seconde information : l'adresse mémoire où se trouve l'élément suivant de la liste.



Une liste chaînée se décompose en :

- Sa tête (le premier maillon de la chaîne)
- Sa queue (les autres maillons)

Une liste vide a une tête qui contient ***Null***. Le successeur du dernier élément vaut aussi ***Null*** (il ne pointe nulle part)



Note : *En python les tableaux de type* ***list()*** *sont déjà des listes chaînées, donc … donc de taille variable !*

2. Opérations sur les listes chaînées

Les opérations de base sur les listes chaînées sont :

- CREER_LISTE_VIDE() qui retourne un objet de type liste
- INSERER(L, e, i) : l'élément e est inséré à la position i dans la liste L
- SUPPIMER(L, i) : supprime l'élément situé à la position i dans la liste L

On peut ajouter les opérations suivantes :

- RECHERCHER(L, e) : l'élément e est cherché dans la liste L et on retourne sa position
- LIRE(L, i) : retourne l'élément situé à la position i
- MODIFIER(L, i, e) : l'élément situé à la position i est écrasé par l'élément e.
- LONGUEUR(L) : retourne le nombre d'éléments dans la liste

Exemple : Que contient la liste L après exécution des instructions suivantes ?

L = CREER_LISTE_VIDE

INSERER(L, 'A', 1)

INSERER(L, 'O', 2)

INSERER(L, 'B', 1)

INSERER(L, 'V', 3)

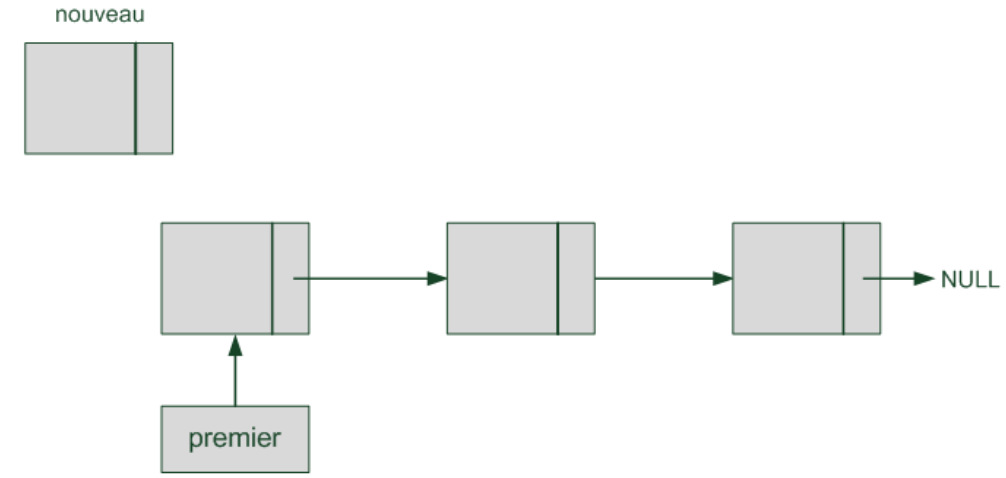
INSERER(L, 'R', 2)

On peut représenter une liste à l'aide d'un tableau (donc en Python … d'une liste …), dont chaque élément est identifié par son indice. On utilise donc en Python une liste contenant n éléments pour représenter une liste de n éléments.

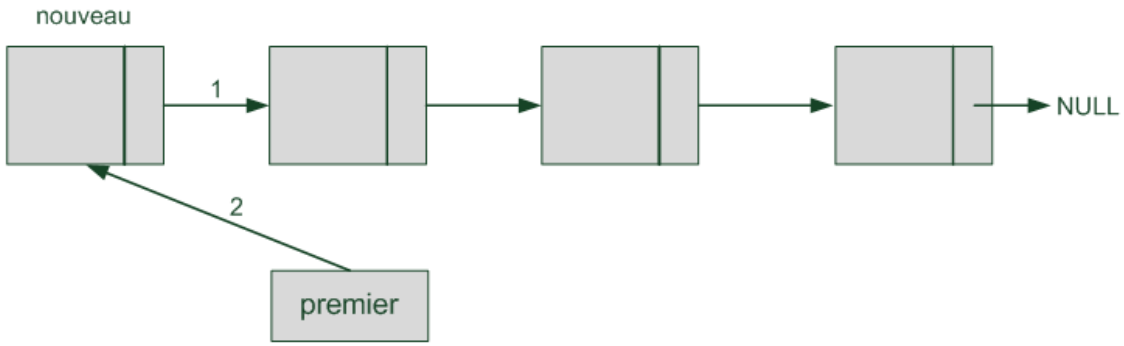
Exemples d'implémentation

Fonction insérer

Un élément nouveau est considéré.



L'insertion de cet élément consiste à changer le pointeur :



On donne ci-dessous le pseudo-code de la fonction INSERER :

```
Fonction INSERER(L, x, i) :
    Si i > longueur(L) +1 :
        Retourner Faux
    Sinon :
        Pour k allant de longueur(L) à i +1 par pas de -1 :
            L[k] = L[k-1]
        L[i + 1] = x
```

Expliquer cet algorithme et l'implémenter en Python.

Fonction supprimer

Ecrire de même une fonction SUPPRIMER(L , i) qui supprime l'élément en position i de la liste L.

Donner le contenu de Ma_liste après chacune des instructions suivantes :

```
Ma_liste = []
INSERER(Ma_liste, 3, 1)
INSERER(Ma_liste, 5, 2)
INSERER(Ma_liste, 8, 1)
SUPPRIMER(Ma_liste, 2)
```

Coût : le coût de chacune des fonctions INSERER et SUPPRIMER est O(n).