T NSI **Structures de données linéaires**

1. Généralités

Les structures de données abstraites

Lorsqu'on crée un algorithme, on s'aperçoit assez vite que, pour manipuler des données, l'utilisation de simples variables de type "nombres" (entiers ou flottants) ou "booléens" n'est pas suffisant. Par exemple, on a vu que le regroupement de données dans un tableau permet d'accéder à ces données simplement à l'aide d'un index.
Au stade de l'algorithme, et indépendamment d'une quelconque implémentation, on peut envisager des structures de données plus complexes, selon les problèmes à traiter.
On se donne une notation pour décrire ces structures ainsi que l'ensemble des opérations que l'on peut appliquer et les propriétés de ces opérations. On parle alors de **type abstrait de données**.
La notion de type abstrait permet de définir des types de données "non primitifs", c'est-à-dire non disponibles dans les langages de programmation courants.
Les types de données "primitifs" sont par exemple les entiers, les flottants, les booléens.

Nous étudierons au cours de l'année 4 types de structures de données abstraites :
- Les structures **linéaires** : les listes, les piles et les files.
- Les structures à **accès par clés** : les dictionnaires.
- Les structures **hiérarchiques** : les arbres.
- Les structures **relationnelles** : les graphes.

Remarque : Ces structures de données sont parfois "prévues" nativement dans certains langages de programmation comme type de données, mais ce n'est pas toujours le cas !

Une structure de données possède un ensemble de routines (procédures ou fonctions) permettant d'ajouter, d'effacer, d'accéder aux données. Cet ensemble de routines est appelé **interface**.

Les opérations élémentaires

L'interface est généralement constituée des 4 routines élémentaires dites CRUD :
Create : ajout d'une donnée.
Read : lecture d'une donnée.
Update : modification d'une donnée.
Delete : suppression d'une donnée.

Derrière les opérations de lecture, de modification ou de suppression d'une donnée se cache une autre routine tout aussi importante : la **recherche** d'une donnée. Car il faut d'abord trouver la donnée dans la structure avant de pouvoir la lire, la modifier ou la supprimer.

2. Les listes

Définition : Une **liste** est une structure de données permettant de regrouper des données et dont l'accès est séquentiel. Elle correspond à une suite finie d'éléments repérés par leur rang (index). Les éléments sont ordonnés et leur place a une grande importance.
Une liste est évolutive, on peut ajouter ou supprimer n'importe quel élément.

Voici deux exemples d'opérations qui peuvent être effectuées sur une liste créée :

INSERER(L, x, i) : repère le donnée de rang i-1 puis décale de 1 rang à droite tous les éléments situés derrière elle et insère x au rang i.
SUPPRIMER(L, i) : qui repère la donnée de rang i (en commençant par le premier élément), puis décale de 1 rang à gauche tous les éléments situés derrière elle.

Type abstrait Liste :

Type abstrait : Liste.
Données : éléments de type T.
Opérations :

CREER_LISTE_VIDE() qui retourne un objet de type Liste.
La liste existe et elle est vide.
INSERER(L, e, i)
L'élément e est inséré à la position i dans la liste L.
SUPPRIMER(L, i)
L'élément situé à la position i est supprimé de la liste L.
RECHERCHER(L, e) qui retourne un objet de type Entier.
L'élément e est recherché dans la liste L et on retourne son index (sa position).
LIRE(L, i) qui retourne un objet de type T.
L'élément situé à la position i dans la liste L est retourné.
MODIFIER(L, i, e)
L'élément situé à la position i dans la liste L est écrasé par le nouvel élément e.
LONGUEUR(L) qui retourne un objet de type Entier.
Le nombre d'éléments présents dans la liste L est retourné.

Conditions :

LIRE(L, i), SUPPRIMER(L, i), MODIFIER(L, i, e) sont définis si, et seulement si :
$1 \leq i \leq$ LONGUEUR(L).

INSERER(L, e, i) est défini si, et seulement si, $1 \leq i \leq$ LONGUEUR(L) + 1.

Exemple d'application de ce type abstrait :

Ecrire l'évolution de la liste L lors de l'exécution de la suite d'instructions suivante :

```
L = CREER_LISTE_VIDE()
INSERER(L,'A', 1)
INSERER(L,'O', 2)
INSERER(L,'B', 1)
INSERER(L,'V', 3)
INSERER(L,'R', 2)
```

Représentation d'une liste avec un tableau
Tous les langages de programmation permettent de réaliser des tableaux.
On utilise un tableau de taille fixe n + 1 dont les indices vont de 0 à n.
●La première case du tableau (indice 0) contient le nombre d'éléments présents dans la liste.

●Les cases suivantes du tableau (indices 1 à n) contiennent les éléments de la liste ou sont vides.
Dans cette représentation, une liste a donc une taille maximale : n. Lorsque la liste atteint sa taille maximale, on dit qu'elle est pleine.
Il est utile de définir deux opérations en plus de celles vues précédemment :
EST_VIDE(L) qui retourne un objet de type Booléen.
Retourne Vrai si la liste L est vide c'est-à-dire si LONGUEUR(L) == 0.
EST_PLEINE(L) qui retourne un objet de type Booléen.
Retourne Vrai si la liste L est pleine c'est-à-dire si LONGUEUR(L) == n.

Exemple :
On représente une liste à l'aide d'un tableau de taille fixe 6.
Lors de la création de la liste vide, toutes les cases du tableau sont initialisées à 0.
Ecrire le contenu du tableau à chaque étape :
L = CREER_LISTE_VIDE()
INSERER(L, 3, 1)

INSERER(L, 5, 2)

INSERER(L, 8, 1)

SUPPRIMER(L, 2)

Exercice 1 :

On donne la séquence d'instructions suivante : (on supposera que les listes ne sont jamais pleines)

```
L1 = CREER_LISTE_VIDE()
L2 = CREER_LISTE_VIDE()
INSERER(L1, 1, 1)
INSERER(L1, 2, 2)
INSERER(L1, 3, 3)
INSERER(L1, 4, 4)
INSERER(L2, LIRE(L1, 1), 1)
INSERER(L2, LIRE(L1, 2), 1)
INSERER(L2, LIRE(L1, 3), 1)
INSERER(L2, LIRE(L1, 4), 1)
```

1. Illustrer le résultat de chaque étape de cette séquence.
2. Quelle est l'opération effectuée sur L1 permettant d'obtenir L2 ?

Exercice 2 :

Implémenter en Python chaque fonction du type abstrait Liste.