Chapitre 1

Listes, piles, files

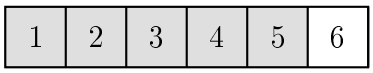
1.1 Listes

1.1.1 Le type abstrait liste

Une **liste** est une structure abstraite de données organisées sous une forme linéaire et séquentielle. Elle est constituée d'éléments d'un même type, chacun possédant un rang. Une liste est évolutive : on peut ajouter ou supprimer n'importe lequel de ses éléments. On considère généralement qu'elle est constituée de deux parties :

— la **tête** qui correspond au dernier élément ajouté à la liste (en blanc ci-dessous) ;

— la **queue** qui correspond au reste de la liste (en gris ci-dessous).



Les opérations possibles avec les listes sont les suivantes :

— Créer une liste vide, tester si une liste est vide.

— Afficher les éléments d'une liste.

— Ajouter des éléments en début (tête), fin (queue) de liste ou à une position donnée.

— Supprimer des éléments en début, fin de liste ou à une position donnée.

On peut aussi considérer d'autres opérations particulières comme trier, fusionner deux listes, rechercher une valeur particulière, etc.

Remarque : attention à ne pas confondre le type abstrait « liste » défini ci-dessus qui est un objet mathématique avec les listes de Python. Elles sont différentes, par exemple, les listes de Python permettent d'avoir des objets de différents types.

Le langage de programmation Lisp (inventé par John McCarthy en 1958 au MIT) a été l'un des premiers langages de programmation à introduire cette notion de liste (Lisp signifie *list processing*). Il est plutôt orienté programmation fonctionnelle.

1.1.2 Implémentation

Pour implémenter ce type abstrait de données d'une manière concrète dans la mémoire RAM d'une machine la plupart des langages de programmation utilisent deux grandes familles de structures : les **tableaux** et les **listes chaînées**.

Les tableaux

Un **tableau** est une suite contiguë de cases mémoires (les adresses des cases mémoires se suivent). Le système alloue une plage d'adresses mémoires afin d'y stocker la valeur des éléments d'une liste par exemple. C'est un système simple et rapide. En effet, les adresses mémoires se suivant, il est possible de calculer l'adresse mémoire voulue à partir de l'adresse du premier élément du tableau et de l'index de l'élément voulu. On obtient ainsi un temps d'accès constant.

La taille d'un tableau est fixe : une fois que l'on a défini le nombre d'éléments que le tableau peut accueillir, il n'est pas possible modifier sa taille. Si l'on veut insérer une nouvelle donnée, on doit créer un nouveau tableau plus grand et déplacer les éléments du premier tableau vers le second tout en ajoutant la donnée au bon endroit. Il faut donc donc prévoir une grande taille de tableau, voire le surdimensionner, ce qui peut entraîner un gaspillage de mémoire.

Dans certains langages de programmation, on trouve une version « évoluée » des tableaux : les tableaux dynamiques. Ces derniers ont une taille qui peut varier. Il est donc relativement simple d'insérer des éléments dans le tableau. Ce type de tableaux permet d'implémenter facilement le type abstrait liste (de même pour les piles et les files).

Remarque : les listes de Python sont en réalité des tableaux dynamiques.

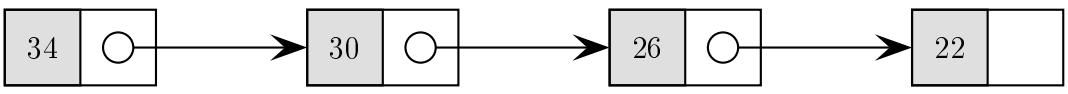
En résumé :

Avantages : l'accès et le parcours sont simples et rapides (resp. en $O(1)$ et $O(n)$ avec n taille du tableau).

Inconvénients : on est souvent amener à surdimensionner le tableau, ce qui peut entraîner un gaspillage de mémoire ; insérer ou supprimer des éléments nécessitent de recréer un tableau, on peut arriver à une complexité en temps quadratique : $O(n^2)$.

Les listes chaînées

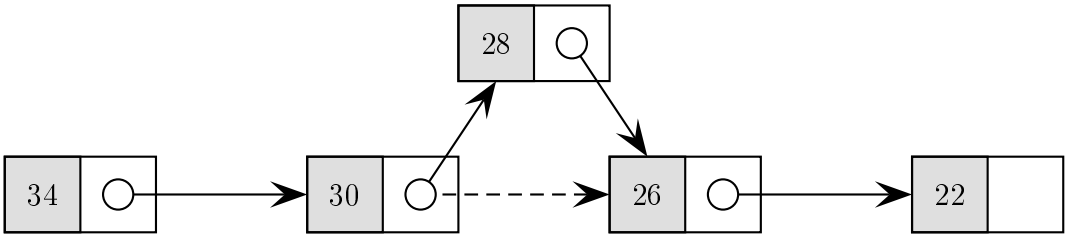
Dans une **liste chaînée**, à chaque élément de la liste on associe 2 cases mémoires : la première case contient l'élément (en gris ci-dessous) et la deuxième contient l'adresse mémoire de l'élément précédent (en blanc avec le cercle ci-dessous). Contrairement au tableau, les zones de mémoires ne sont donc pas contiguës.



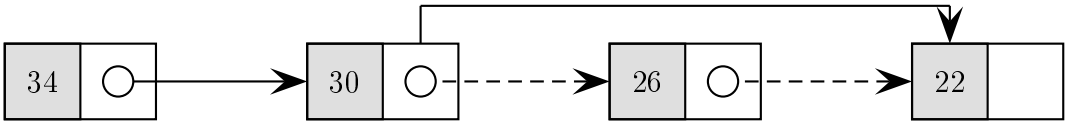
Ici, 34 est la tête de liste et 22 en est le dernier élément.

Remarque : le dernier élément de liste ne pointe vers aucune adresse.

Avec les listes chaînés, il est aisé d'ajouter ou de supprimer un élément. Il suffit de modifier les adresses des éléments précédents afin qu'elles pointent vers le nouvel élément suivant.



Insertion de 28 entre 30 et 26



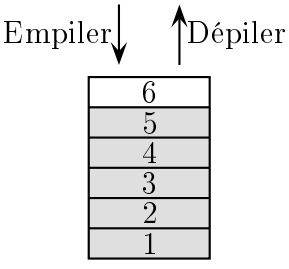
Suprresion de 26

Remarque : il existe des listes doublement chaînées dont chaque élément pointe à la fois vers son prédécesseur et son successeur à l'exception du premier et du dernier élément de la liste.

Les listes chaînées permettent une grande liberté, notamment celle d'étendre à volonté une liste. En pratique, cela peut poser problème pour des raisons de gestions de la mémoire. Les piles et les files sont des listes particulières dont les restrictions permettent d'éviter ces problèmes : l'ajout, la suppression et l'accès ne sont font que via des éléments bien précis.

1.2 Piles

Les **piles** sont des cas particuliers de listes dans lequel l'accès, l'ajout et la suppression ne peuvent se faire que par le sommet de la pile. On parle d'**empiler** pour ajouter un élément au sommet de la pile et de **dépiler** pour retirer l'élément au sommet de la pile. On est donc dans un principe du dernier arrivé, premier servi ou LIFO (*Last In, First Out* ; on n'a qu'un seul pointeur indiquant le sommet de la pile.



Les opérations possibles avec les piles sont les suivantes :

— Créer une pile vide, tester si une pile est vide.

— Afficher l'élément au sommet de la pile.

— Empiler : ajouter un élément au sommet de la pile.

— Dépiler : supprimer un élément au sommet de la pile.

Exemples d'utilisation des piles :

— Dans un navigateur web, une pile sert à mémoriser les pages web visitées ; l'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant sur le bouton « Afficher la page précédente ». La pile des précédents est dépilée au profit de la pile des suivants.

— La fonction « Annuler » (*Undo* en anglais) d'un logiciel mémorise les modifications apportées au document dans une pile.

— La récursivité (une fonction qui fait appel à elle-même) utilise également une pile.

— Les compilateurs dans les langages de programmation.

— Le parenthésage d'une expression mathématique ou informatique : lors de la lecture d'une expression, on stocke les parenthèses ouvrantes non refermées dans une pile de caractères et on dépile si on trouve une parenthèse fermante. L'expression est acceptée si la pile n'est pas vide lorsqu'on souhaite la dépiler (pas de parenthèses fermantes en trop) et si la pile est vide à la fin de la lecture de l'expression (pas de parenthèses ouvrantes en trop).

Remarque : en informatique, un dépassement de pile ou débordement de pile (en anglais, *stack overflow*) est une erreur causée par un processus qui, lors de l'écriture dans une pile, écrit à l'extérieur de l'espace alloué à la pile, écrasant ainsi des informations nécessaires au processus. Stack Overflow est aussi connu comme étant un site web proposant des questions et réponses sous la forme d'un forum d'entre-aide sur un large choix de thèmes concernant la programmation informatique.

1.3 Files

Les **files** sont des listes reposant sur le principe du premier entré, premier sorti ou FIFO (*First In, First Out*). Les nouveaux éléments ne peuvent être insérés qu'en queue de file et on ne peut accéder et supprimer que l'élément en tête de file. On parle d'**enfiler** pour ajouter un élément et de **défiler** pour en supprimer. ; On a deux pointeurs indiquant la tête et la queue de la file.

Les opérations possibles avec les files sont les suivantes :

— Créer une file vide, tester si une file est vide.

— Afficher l'élément en tête de la file.

— Enfiler : ajouter un élément à la fin de la file.

— Dépiler : supprimer l'élément en tête de la file.

Les files servent généralement à mémoriser temporairement des données ou des tâches en attente de traitement.

Exemples d'utilisation des files :

- Certains ordonnanceurs dans les systèmes d'exploitation, qui doivent accorder du temps machine à chaque tâche, sans en privilégier une plus qu'une autre, utilisent des files d'attentes.
- Les requêtes entre machines sur un réseau.
- Les serveurs d'impression, qui doivent traiter des requêtes dans l'ordre dans lequel elles arrivent, et les insère dans une file d'attente (ou une queue).
- Un algorithme de parcours en largeur utilise une file pour mémoriser les nœuds visités.
- La création de toutes sortes de mémoires tampons (*buffers*).

1.4 Exercices

1.4.1 Démarrage

Exercice 1.1. [Piles par tableau] Le but de cette exercice est de programmer une classe **Pile** reposant sur le concept du tableau et disposant de toutes les opérations de base d'une pile.

Notre tableau sera une liste Python dont on limitera artificiellement la taille à l'aide d'un compteur. La taille de la pile sera donc un attribut de l'instance à passer en entrée à sa création. Si la taille maximale de la pile est dépassée, l'utilisateur devra en être averti.

On peut donc résumer cela sous la forme du schéma UML suivant.

Pile
taille valeurs
est vide empiler dépiler afficher sommet

Programmer la classe **Pile** et tester ses méthodes.

Exercice 1.2. [Files par tableau] Le but de cette exercice est de programmer une classe **File** reposant sur le concept du tableau et disposant de toutes les opérations de base d'une file. Notre tableau sera à nouveau une liste Python.

On peut donc résumer cela sous la forme du schéma UML suivant.

File
valeurs
est vide enfiler défiler afficher tête

Programmer la classe **File** et tester ses méthodes.

1.4.2 Approfondissement

Exercice 1.3. [Piles et parenthésage] Écrire une fonction vérifiant le parenthésage d'une chaîne de caractères à l'aide d'une pile.

Exercice 1.4. [Piles par liste chaînée] Le but de cette exercice est de programmer une classe **Pile** reposant sur le concept de la liste chaînée et disposant de toutes les opérations de base d'une pile.

Une liste chaînée est soit vide, soit constituée de maillons. Une instance de notre classe **Pile** sera donc vide ou ne contiendra uniquement que le sommet de pile, i.e. le dernier maillon de la liste chaînée. Cela sera ensuite à chaque maillon de pointer vers le maillon « du dessous ».

Les méthodes pour ajouter, supprimer, afficher le sommet etc seront des méthodes de la classe **Pile**. La classe **Maillon** n'en contiendra pas à moins d'une éventuelle méthode de présentation du maillon (sous forme `__str__()`).

On peut donc résumer cela sous la forme du schéma UML suivant.

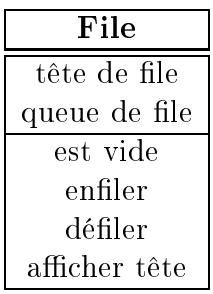
1. Quels sont les types des valeurs attendues pour les attributs « valeur » et « suivant » de la classe **Maillon**.
2. Programmer les classes **Maillon** et **Pile**. Tester leurs méthodes.

Exercice 1.5. [Files par liste chaînée] Le but de cette exercice est de programmer une classe **File** reposant sur le concept de la liste chaînée et disposant de toutes les opérations de base d'une file.

On reprendra la classe **Maillon** de l'exercice précédent. Une instance de notre classe **File** sera vide ou ne contiendra uniquement que la tête et la queue de file, i.e. le premier et le dernier maillon de la liste chaînée. Cela sera ensuite à chaque maillon de pointer vers le suivant.

Les méthodes pour ajouter, supprimer, afficher, parcourir des éléments, etc seront des méthodes de la classe **File**.

On peut donc résumer cela sous la forme du schéma UML suivant.



Programmer la classe **File** et tester ses méthodes.

Exercice 1.6. [Listes par liste chaînée] Le but de cette exercice est de programmer une classe **Liste** reposant sur le concept de la liste chaînée et disposant de toutes les opérations de base d'une liste.

On reprendra la classe **Maillon** des exercices précédents. Une instance de notre classe **Liste** sera vide ou ne contiendra uniquement que la tête de liste, i.e. le premier maillon de la liste chaînée. Cela sera ensuite à chaque maillon de pointer vers le suivant.

Les méthodes pour ajouter, supprimer, afficher, parcourir des éléments, etc seront des méthodes de la classe **Liste**.

On peut donc résumer cela sous la forme du schéma UML suivant.

Liste
tête de liste
est vide ajouter valeur supprimer valeur afficher valeur parcourir liste

Programmer la classe **Liste** et tester ses méthodes.