Chapitre 3Listes, piles et files

Table des matières

1 Type abstrait de données : les listes, les piles, les files

1.1 Les listes

1.1.1 Définition

Définition 3.1
une liste est une structure de données permettant de regrouper des données de manière à pouvoir y accéder librement

Remarque

- La liste est à la base de structures de données plus complexes comme la pile, la file, les arbres, etc.
- L'importance de la liste comme structure de données est telle qu'elle est à la base du langage de programmation Lisp (de l'anglais list processing).

1.1.2 Opérations

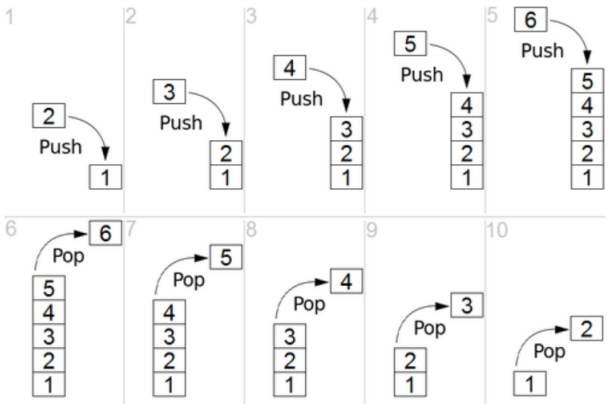
Voici quelques opérations qui peuvent être effectuées sur une liste :

- créer une liste vide (L=vide() on a créé une liste L vide)
- tester si une liste est vide (estVide(L) renvoie vrai si la liste L est vide)
- ajouter un élément en tête de liste (ajouteEnTete (x,L) avec L une liste et x l'élément à ajouter)
- supprimer la tête x d'une liste L et renvoyer cette tête x (supprEnTete(L))
- Compter le nombre d'éléments présents dans une liste (compte(L) renvoie le nombre d'éléments présents dans la liste L)

Exemple

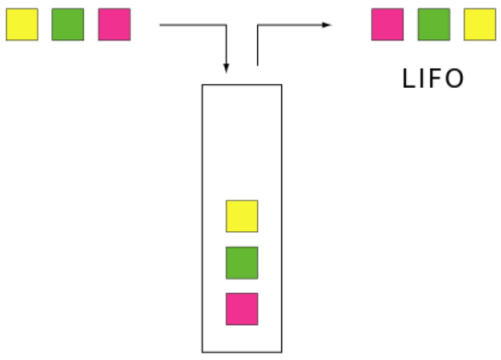
Voici une suite d'instructions ; quelle liste est représentée à la fin ?
L=vide()
ajoutEnTete(3,L)
ajoutEnTete(5,L)
ajoutEnTete(8,L)
t = supprEnTete(L)

1.2 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 et 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.



1.2.1 Opérations sur les piles

- On peut créer une pile vide. (creer_ pile)
- on peut savoir si une pile est vide (pile_ vide)
- on peut empiler un nouvel élément sur la pile (empliler)

- on peut récupérer l'élément au sommet de la pile tout en le supprimant. On dit que l'on dépile (pop) (depiler)
- on peut accéder à l'élément situé au sommet de la pile sans le supprimer de la pile (sommet)
- on peut connaitre le nombre d'éléments présents dans la pile (taille)

Exemple

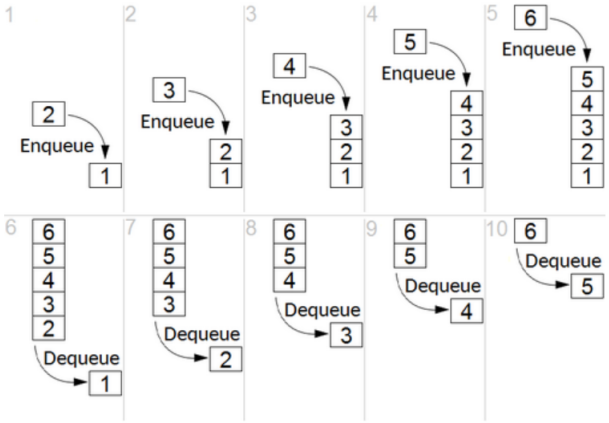
Soit une pile P composée des éléments suivants : 15, 11, 32, 47 et 61 (le sommet de la pile est 61). Quel est l'effet de l'instruction pop(P) ?

1.2.2 A quoi servent les piles ?

Les piles sont extrêmement utiles en informatique et vous les utilisez quotidiennement, parfois même sans vous en rendre compte :

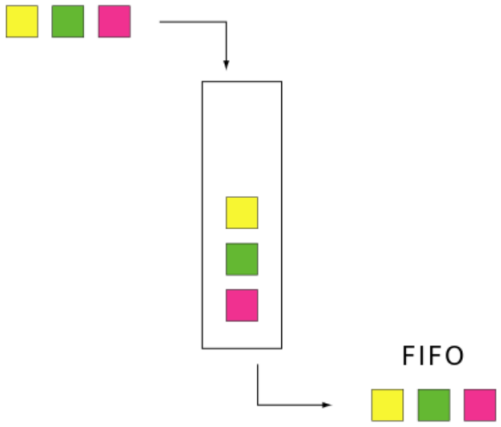
- La fonction annuler (Ctrl-Z) de votre traitement de textes par exemple est une pile : Quand vous tapez Ctrl-Z, vous annulez la dernière opération effectuée. Quand vous faites une nouvelle opération, celle-ci est mémorisée au sommet de la pile. Vous ne pouvez pas annuler l'avant dernière opération sauf à annuler la dernière.
- Le bouton retour de votre navigateur internet fonctionne également à l'aide d'une pile. Les pages web consultées lors de votre navigation sur une page sont empilées et le bouton retour permet d'accéder à la dernière page présente sur la pile.
- Certaines calculatrices fonctionnent à l'aide d'une pile pour stocker les arguments des opérations : c'est le cas de beaucoup de calculatrices de la marque HP, dont la première calculatrice scientifique ayant jamais été produite : la HP 35 de 1972.

1.3 Les files



1.3.1 Définition

Dans une file, les éléments sont placés les uns à cotés des autres comme dans une pile, à la différence que seul l'on sort les éléments du plus ancien vers le plus récent.
Cela correspond à ce qui se passe dans une file d'attente
Les files sont basées sur le principe FIFO (First In First Out : le premier qui est rentré sera le premier à sortir. Ici aussi, on retrouve souvent ce principe FIFO en informatique.



1.3.2 Opérations sur les files

Voici les opérations que l'on peut réaliser sur une file :

- on peut savoir si une file est vide (file_ vide ?)
- on peut ajouter un nouvel élément à la file (ajout)
- on peut récupérer l'élément situé en bout de file tout en le supprimant (retire)

- on peut accéder à l'élément situé en bout de file sans le supprimer de la file (premier)
- on peut connaitre le nombre d'éléments présents dans la file (taille)

Exemple
Soit une file F composée des éléments suivants : 1, 12, 24, 17, 21 et 72 (le premier élément rentré dans la file est 72 ; le dernier élément rentré dans la file est 1). Quel est l'effet de l'instruction ajout(F,25)

1.3.3 Utilisation des files

Dans le domaine informatique, on retrouve par exemple les files dans les files d'impression où le premier document envoyé à l'imprimante sera le premier document à être imprimé.

2 Les différentes structures pour implémenter les liste, les pile et les file

On va ici chercher à "traduire" ces algorithmes dans un langage compréhensible pour un ordinateur (Python, Java, C,...) : on dit alors que l'on implémente un algorithme.

L'implémentation d'un type de données dépend du langage de programmation. Il faut, quel que soit le langage utilisé, que le programmeur retrouve les fonctions qui ont été définies pour le type abstrait (pour les listes, les piles et les files cela correspond aux fonctions définies ci-dessus).
Certains types abstraits ne sont pas forcément implémentés dans un langage donné, si le programmeur veut utiliser ce type abstrait, il faudra qu'il le programme par lui-même en utilisant les "outils" fournis par son langage de programmation.

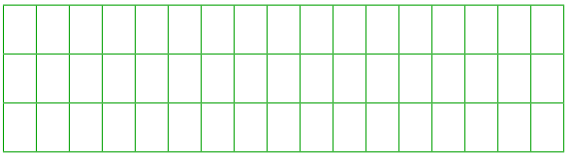
Pour implémenter les listes (ou les piles et les files), beaucoup de langages de programmation utilisent 2 structures : les tableaux et les listes chaînées.

2.1 Structure avec des tableaux

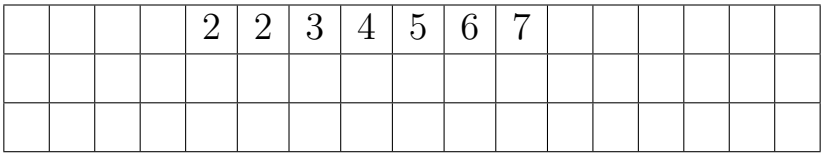
2.1.1 Définition

Définition 3.2

Un tableau est une suite contiguë de cases mémoires (les adresses des cases mémoire se suivent).



Le système réserve une plage d'adresse mémoire afin de stocker des éléments.



les éléments d'un tableau étant contigus et ordonnés en mémoire, insérer un élément dans une séquence demande de déplacer tous les éléments qui le suivent pour lui laisser une place. Si par exemple on veut insérer une valeur 4 à la première position du tableau

2	2	3	4	5	6	7

il faut d'une façon ou d'une autre construire le nouveau tableau

4	2	2	3	4	5	6	7

dans lequel la case d'indice 0 contient maintenant la valeur 4.
Cette opération est **très** couteuse, car il faut déplacer tous les éléments d'une case vers la droite.

La structure de tableau permet de stocker des séquences d'éléments mais n'est pas adaptée à toutes les opérations que l'on pourrait vouloir effectuer sur des séquences.

2.1.2 Les tableaux dynamiques

Dans certains langages de programmation(comme dans le langage Python), on trouve une version "évoluée" des tableaux : les tableaux dynamiques. Les tableaux dynamiques 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)

Les tableaux de Python permettent par exemple d'insérer ou de supprimer efficacement des éléments à la fin d'un tableau, avec les opérations append et pop. Il permettent aussi d'insérer un élément avec :

```
t.insert(0,4)
```

⚠ En python, les listes sont en réalité des tableaux dynamiques, à mi-chemin entre les listes et les tableaux !
Dans ce chapitre nous étudions une structure de données, la liste chaînée, qui d'une part apporte une meilleure solution au problème de l'insertion et de la suppression au début d'une séquence d'éléments, et d'autre part servira de base à plusieurs autres structures dans les prochains chapitres.

2.2 Structure avec des listes chaînées

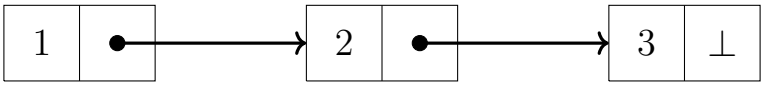
Autre structure qui permet d'implémenter des listes : les listes chaînées

2.2.1 Définition

Une *liste chaînée* sert à représenter une liste, c'est-à-dire une séquence finie de valeurs, par exemple des entiers.

Chaque élément est stocké dans un petit bloc alloué quelque part dans la mémoire, que l'on pourra appeler maillon ou cellule, et y est accompagné d'une deuxième information : l'adresse mémoire où se trouve la cellule contenant l'élément suivant de la liste.

Considérons la liste contenant trois éléments, respectivement 1, 2 et 3.

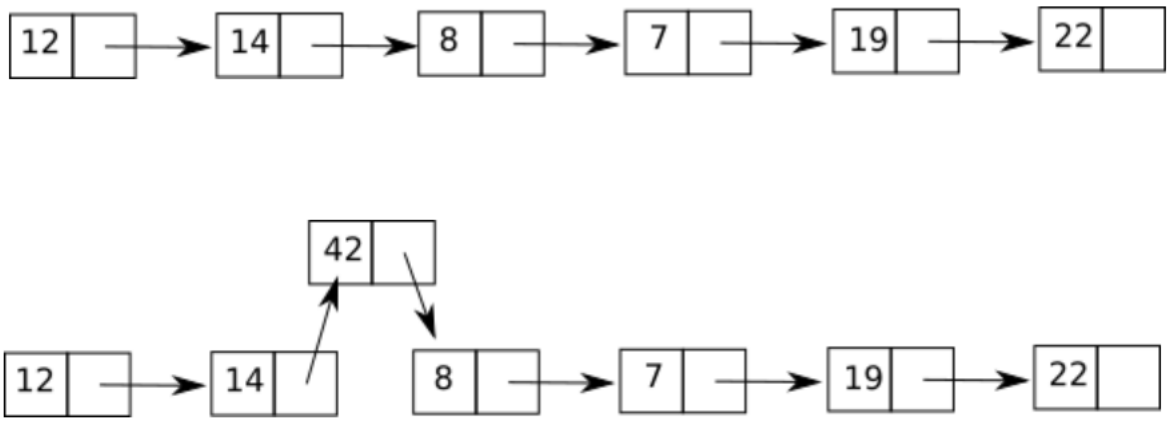


Chaque élément de la liste est matérialisé par un emplacement en mémoire contenant :

- d'une part sa valeur (dans la case de gauche)
- d'autre part l'adresse mémoire de la valeur suivante (dans la case de droite).

Remarque
Pour le dernier élément, qui ne possède pas de valeur suivante, on utilise une valeur spéciale désignée ici par le symbole ⊥ et marquant la fin de la liste

Il est assez facile d'insérer un élément lorsque l'on travaille avec une liste chaînée :
Supposons que l'on veuille insérer l'élément "42" en troisième position dans cette liste :



Histoire

Le premier interpréteur fonctionnait sur un ordinateur IBM 704 et deux instructions de cette machine devinrent les deux opérations primitives de Lisp pour décomposer les listes :

- car (contents of address register) : le premier élément de la liste.
- cdr (contents of decrement register) : le reste de la liste.
- L'opération qui consiste à fabriquer une liste à partir d'un premier élément et d'une liste est notée cons.

2.3 Complexité

Pourquoi implémenter plusieurs structures ? Après tout, on peut tout faire avec des listes Python !

Parce que l'efficacité est fondamentale. Certaines structures sont plus adaptées à certains problèmes.

2.3.1 Accéder à un élément : tableau

Pour accéder à l'élément 2 du tableau T = ['a', 'b', 'c'],
On se rend à l'adresse où débute T

On se déplace de deux positions. On lit : 'c' Le temps est constant : Accéder se fait en complexité O(1).

2.3.2 Accéder à un élément : liste

Pour accéder à l'élément 2 de la liste
L = ('a', ('b', ('c', [])))
On se rend à l'adresse où débute L On suit le lien jusque l'adresse de la queue du premier élément On suit le lien jusque l'adresse de la queue du second élément On lit la valeur de la tête : 'c' Le temps est linéaire : Accéder se fait en complexité O(n).

2.3.3 Insérer un élément

Comme on l'a vu plus haut, c'est le contraire !
Cette opération est plus rapide pour les listes que pour les tableaux.

3 Implémentation en Python des listes chainées

3.1 Implémenter une liste chainée avec les classes

Une façon d'utiliser des listes chaînées avec Python, est d'utiliser une classe :

```
class Cellule:
"""une cellule d'une liste chaînée"""
        def __init__(self, v, s):
                self.valeur = v
                self.suivante = s
```

Ainsi la liste 1,2,3 est possible avec l'instruction :

```
liste = Cellule(1, Cellule(2, Cellule(3, None)))
```

Remarque
None remplace $\perp$.

Définition 3.3
Une liste chaînée est une structure de données pour représenter une séquence finie d'éléments. Chaque élément est contenu dans une cellule, qui fournit par ailleurs un moyen d'accéder à la cellule suivante. Les opérations sur les listes chaînées se programment sous la forme de parcours qui suivent ces liaisons, en utilisant une fonction récursive ou une boucle

3.2 Implémenter avec les tableaux ou tuples

Plutôt qu'un objet de la classe Cellule, on pourrait utiliser un couple, et dans ce cas écrire (1,(2,(3,None))).

On peut aussi encore un tableau à deux éléments, et dans ce cas écrire [1,[2,[3,None]]].

Une liste chaînée peut être encore être représentée par deux tableaux, l'un (contenu) contenant des valeurs et l'autre (suivant) contenant des indices. Le chaînage sera effectué de la façon suivante : l'élément suivant contenu[k] aura suivant[k] comme indice dans le tableau contenu.

	contenu	suivant
0	'2'	1
1	'3'	-1
2	'1'	0

Exercice 3.1
Ecrire une fonction listeN(n) qui reçoit un argument entier n et qui renvoie la liste chaînée des entiers 1,2,3,...,n dans cet ordre. Si n = 0, la liste renvoyée est vide.

Exercice 3.2
Ecrire une fonction longueur(lst) qui reçoit pour argument une liste chaînées et qui retourne la longueur de la liste chaînées.

1. L'écrire avec une boucle While
2. L'écrire comme fonction récursive

Exercice 3.3
Ecrire une fonction affiche_ liste(lst) qui reçoit en argument une liste chaînée et qui affiche, en utilisant print, tous les éléments de la liste lst, séparés par des espaces, suivies d'un retour chariot.

1. L'écrire avec une boucle While
2. L'écrire comme fonction récursive

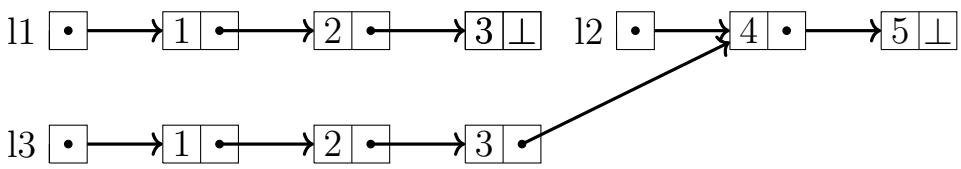
Exercice 3.4
Ecrire une fonction n_ ieme_ element(lst) de paramètre lst une liste chaînée et qui renvoie le n-ième élément de la liste

1. L'écrire de façon récursive
2. L'écrire avec une boucle While

Exercice 3.5
On souhaite maintenant mettre bout à bout deux listes. On appelle cela une concaténation. Ecrire une fonction concatener qui reçoit deux listes en arguments et revoie une troisième liste contenant la concaténation.

```
def concatenerr(l1,l2):
    ''' concatène l1 et l2 ,
    sous la forme d'une nouvelle liste
```

```
    si l1 est vide, alors on renvoie l2
    sinon la concaténation est obtenue en concaténant la tête de l1 et la contenation du reste de l
    l1=Cellule(1,Cellule(2,Ceellule(3,None)))
    l2= Cellule(4,Cellule(5,None))
    l3 = concatener(l1,l2)
    '''
```



Remarque

On voit que les cellules de l1 ont été dupliquée tandis que les cellules de l2 partagées.Les cellules l2 permettent de constituer la liste l2 et la fin de la liste l3.Une alternative consisterait à dupliquer également les cellules de l2, mais ceci n'est pas forcément nécessaire.

4 Opérations avec les listes chaînées

4.1 Longueur de la liste

```
def longueur(l):
    if lst is None:
        return 0
    else:
        return 1 + longueur(l.suivante)
```

Complexité : la complexité du calcul de la longueur est directement proportionnelle à la longueur elle-même, puisqu'on réalise un nombre constant d'opérations pour chaque cellule de la liste.
Ainsi, pour une liste l de mille cellules, longueur(l) va effectuer mille tests, mille appels récursifs et mille additions.

4.2 Renvoyer le n-ème élément d'une liste

```
def nieme_element(n, l):
    if l is None:
        raise IndexError("indice invalide")
    if n == 0:
        return l.valeur
    else:
        return nieme_element(n-1,l.suivante)
```

4.3 Concaténer deux listes

```
def concatener(l1, l2):
"""concatène les listes l1 et l2,
sous la forme d'une nouvelle liste"""
i f l1 i s None:
return l2
else:
return Cellule(l1.valeur, concatener(l1.suivante,l2))
```

5 Implémentation en Python des files et des listes

5.1 Implémenter une pile avec Python

L'implémentation des piles en python se fait facilement à l'aide des méthodes append() et pop() du type list :

ma_ pile.append(ma_ valeur) # permet d'empiler une valeur
ma_ pile.pop() # permet de dépiler une valeur
len(ma_ pile) # renvoie la longueur de ma_ pile

5.2 Implémenter une file avec Python

ma_ file.append(ma_ valeur) # permet d'enfiler une valeur
ma_ file.pop(0) # permet de défiler une valeur
len(ma_ file) # renvoie la longueur de ma_ file

5.3 Implémenter des piles et des files avec des classes

cf. les TDs vus à ce sujet dans le chapitre POO.