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

NSI Tle - JB Duthoit

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.

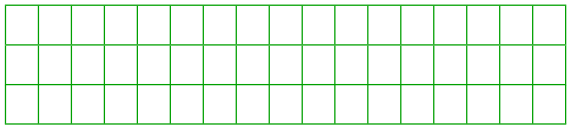
Pour implémenter les listes (ou les piles et les files), beaucoup de langages de programmation utilisent 2 structures : les tableaux et les listes chaînées.

5.2.1 Structure avec des tableaux

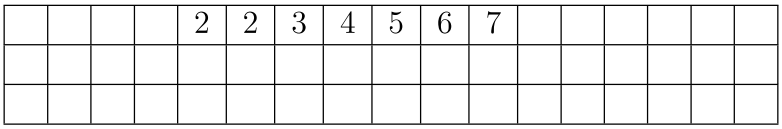
Qu'est ce qu'un tableau ?

Définition

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.



Principe

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.

Des tableaux en python ?

⚠ De tels tableau n'existent pas en python. Il est toutefois assez facile d'en faire une simulation...

Ainsi, pour simuler un tableau en python, on peut écrire :

```
tableau = [None] * 5
```

ce qui permet de créer un tableau de 5 cases, avec `None` à l'intérieur de chaque case. On peut ensuite écrire `tableau[0]=1` si on souhaite mettre 1 dans la première case par exemple.
☛ Bien sûr, cela suppose de s'interdire des méthodes telles que `append`, `pop` , qui ont pour effet de changer la taille du tableau...

5.2.2 Structures avec des 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 !

5.2.3 Structure avec des listes chaînées

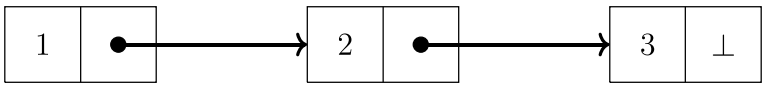
Autre structure qui permet d'implémenter des listes : les listes chaînées

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.



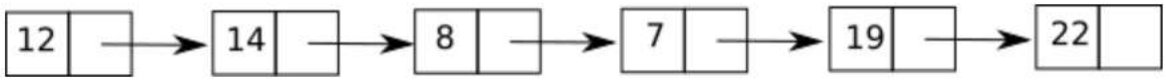
Chaque élément de la liste est matérialisé par un emplacement en mémoire contenant :

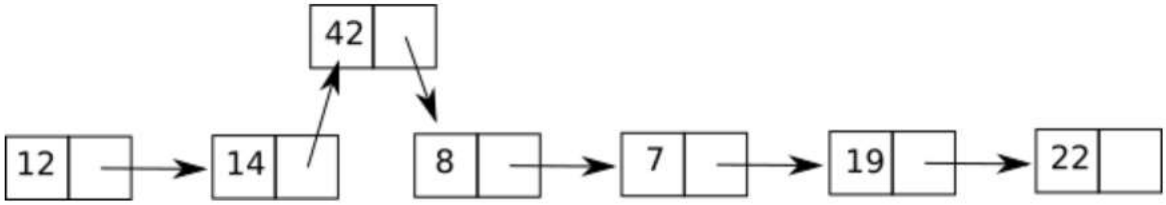
- d'une part sa valeur (dans la case de gauche)
- d'autre part l'adresse mémoire de la valeur suivante (dans la case de droite).

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.

5.2.4 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.

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).

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).

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.