1 minute

Introduction aux listes chaînées

Définition d'une liste chaînée

Une liste chaînée est une structure de données linéaire. Contrairement à un tableau, l'adresse et l'ordre des éléments ne sont pas donnés par leur emplacement physique sur la mémoire.

Chaque élément indique l'adresse de l'élément suivant.

Il existe plusieurs versions possibles des listes chaînées, dans sa version la plus basique **chaque élément de la liste chaîné, appelé nœud (*node*) contient une valeur et une référence vers l'élément (le nœud) suivant**.

Cette structure permet d'insérer ou de supprimer des éléments à une position donnée très facilement : contrairement à un tableau il n'y a pas besoin de déplacer tous les éléments et d'effectuer de ré-allocation en mémoire. Il suffit de modifier la référence vers l'élément suivant.

En conséquence, il n'est pas possible d'avoir un accès direct (aléatoire) à un élément, i**l faut parcourir toute la liste séquentiellement jusqu'à trouver l'élément.**

Pour supprimer un élément donné, sans avoir sa référence, il faut donc parcourir la liste chaînée (complexité `O(n)`) puis le supprimer (complexité `O(1)`).

A noter que l'on dit que les opérations d'insertion et de suppression sont des complexité `O(1)` car c'est le cas si on a la référence d'un élément. En effet, c'est pour faire la distinction avec les tableaux, où même si on a la référence d'un élément ces opérations sont de complexité O(n).

Avantages des listes chaînées

L'avantage principal est de pouvoir supprimer ou ajouter un élément de la liste très facilement (complexité `O(1)`) contrairement à un tableau.

Il faut bien sûr, comme nous l'avons dit, avoir la référence de l'élément avant celui à supprimer ou avant le point d'insertion. Sinon il faut préalablement effectuer une opération de recherche de complexité `O(n)`.

Ces opérations impliquent simplement de changer la référence vers un élément.

Par comparaison dans un tableau, pour ces opérations, il va falloir, au pire, déplacer tous les éléments lors d'une ré-allocation ce qui est beaucoup plus coûteux car il faut une grande quantité de mémoire et d'écritures.

A noter que même si on peut théoriquement supprimer un élément au milieu d'un tableau sans effectuer une ré-allocation (en mettant un élément vide) cela crée ce qu'on appelle une fragmentation du tableau qui va rendre les itérations beaucoup moins performantes.

Désavantages des listes chaînées

Le désavantage principal est que l'accès aux éléments est séquentiel et non direct.

Il faut donc effectuer une recherche de complexité `O(n)` si on a pas l'adresse de l'élément.

C'est l'inconvénient par rapport aux tableaux qui permettent un accès direct (aléatoire) de complexité `O(1)`.

Il est également à noter **qu'en raison de la localité spatiale des éléments** (c'est-à-dire le fait que les éléments sont contigus en mémoire), **un tableau permet une itération bien plus rapide sur ses éléments qu'une liste chaînée.**

Nous ne détaillerons pas plus mais sachez que cela permet de mieux utiliser les registres et cache du processeur.

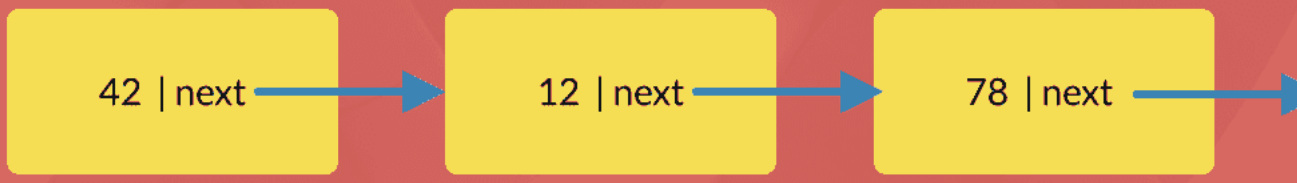
Sachez donc ce point en pratique, car même si théoriquement itérer sur les éléments d'une liste chaînée ou d'un tableau est toujours de complexité `O(n)`, en réalité l'itération sur un tableau est beaucoup plus rapide.

Un autre désavantage d'une liste chaînée par rapport à un tableau est que chaque élément contient une référence vers l'élément suivant ce qui coûte 8 octets supplémentaires par élément (une adresse mémoire sur un système 64 bits fait 8 octets). L'empreinte mémoire d'une liste chaînée est donc toujours supérieure à un tableau (elle peut être 2 ou 3 fois plus grande pour une liste chaînée d'entiers par rapport à un tableau d'entiers).

Cas d'utilisation

Nous verrons ques plusieurs structures de données sont des listes chaînées particulières (par exemple souvent les files et les piles).

Liste chaînée



	Accès	Recherche	Ajout Suppres
Début	O(1)	O(1)	O(1)
Milieu	O(n)	O(n)	O(1)
Fin	O(n)	O(n)	O(1)

Mémoire
O(n)