Structures de Données - L2, S4
TP 3 - Listes, Arborescences et Gestion de Mémoire

Un tableau dynamique est un tableau extensible, ie tant qu'il existe de l'espace mémoire on pourra toujours stocker des valeurs dans le tableau. En `Java` il existe une classe de tableau dynamique appelé Vector. Dans d'autres langages comme en `C` ou `fortran`, il n'y a pas de tableau dynamique. Une façon de faire des tableaux dynamiques dans ces derniers c'est de proposer une structure de données qui contient un tableau et un entier MAX étant la taille du tableau. A chaque fois que le programmeur veuille accéder à une case au delà de MAX, on ré-alloue un autre tableau et on fait la copie. On peut montrer qu'en moyenne on n'ait besoin de faire que $O(1)$ écritures par élément du tableau ce qui est relativement correct. Mais cela suppose que l'on puisse ré-allouer rapidement. En `C` on dispose de constructeurs pour ré-allouer avec des pointeurs, alors qu'en `fortran` on n'a pas de pointeurs. Nous allons d'abord proposer une manière de simuler des pointeurs. Ensuite, comment gérer la mémoire libre (avec existence de pointeurs ou non).

Exercice 3.1 *Manipulation de listes par tableau*
Ici nous ne disposons pas de pointeurs. Pour cela il faudra simuler les pointeurs par des indices dans des tableaux.

1. Proposer une implémentation des listes doublement chaînées par tableaux multiples. Voir dans le cours la liste des champs d'une cellule dans une liste.
2. Les mots d'une mémoire sont adressés par des entiers compris entre 0 et $M-1$ où M est la taille maximum de la mémoire. Dans la plupart des langages de programmation les objets sont stockés de façon contigüe et un pointeur dans un tel objet est l'adresse du premier mot. En vous inspirant de cette méthode, proposer une autre implémentation en utilisant un tableau unique.

Maintenant que l'on sait faire des listes sans pointeurs, il nous faudra gérer l'allocation et la libération des cellules.

Exercice 3.2 *Allocation et Libération mémoire*
Nous ne disposons toujours pas de pointeurs. Supposons que les tableaux permettant de gérer des listes aient une taille maximale MAX. Nous avons vu en cours une façon simple de gérer les cellules. On crée une liste contenant toutes les cellules libres.

1. Proposez une implémentation des deux fonctions d'allocation et de libération d'une cellule, chacune s'exécutant en temps $O(1)$ pour les deux implémentations proposées.
2. Si on souhaite que les listes soient compactes, ie occupent une liste contigüe de cellules, comment faire pour implémenter les fonctions d'allocation et de libération de cellules pour avoir à chaque fois quelque chose de compacte ?
3. Ici on suppose que l'on n'a pas une liste compacte. Proposez une fonction densifier-liste qui permet de rendre une liste compacte en temps $O(n)$ où n est la taille de la liste.

Maintenant que nous savons gérer des listes et une façon de les gérer à l'aide de tableaux, nous allons nous intéresser à la gestion des blocs mémoire, car en particulier nous ne gérons pas qu'une seule liste, mais plusieurs qui cohabitent dans la mémoire. Ce problème apparait également dans la gestion des espaces libres d'un disque dur car lors de la sauvegarde d'un fichier il faudra allouer de l'espace disque pour le stocker et donc on doit disposer des espaces libres. La lecture du fichier doit également ne dépendre que de la taille du fichier et la gestion par des différents blocs d'un fichier par une liste est clairement souhaitable. Supposons que l'on ait divisé la mémoire en blocs, chaque bloc b ayant une taille $taille(b)$ et est contigüe. A un instant t, la mémoire est divisée en blocs occupés et en blocs libres. Nous disposons d'une liste de blocs libres et le début du $(i+1)^{\text{eme}}$ bloc vient après la fin du i^{eme} bloc.

Exercice 3.3
Implémenter un algorithme qui alloue un espace contigüe de taille exactement n. L'allocation consistera à chercher un espace libre de taille $\geq n$ et à allouer une partie de cet espace et à marquer le reste comme libre.

Exercice 3.4
Implémenter un algorithme qui libère un bloc alloué.

Exercice 3.5
Vous remarquez qu'avec cette implémentation vous devez faire une recherche dans la liste des libres pour libérer un espace, et pourquoi ? Proposez une nouvelle façon de gérer les blocs libres pour éviter cette recherche qui peut coûter chère lorsque la liste des libres est longue. On peut utiliser des listes doublement chaînées et garder pour chaque bloc un drapeau signalant s'il est libre ou pas.

Exercice 3.6
Proposez maintenant une manière de manipuler des tableaux dynamiques avec des pointeurs. L'astuce c'est de doubler à chaque fois que c'est nécessaire la taille du tableau.