Travaux Dirigés d'algorithmique n°4

Cours d'algorithmique

—Licence CFA / Troisième Année—

Structures de données, listes chaînées

Dans ce TD, nous étudions l'implantation des types File et Pile par tableaux ainsi que la manipulation des listes chaînées. Même si de nombreux langages modernes gère automatiquement la mémoire, on écrira explicitement les allocations et désallocations en évitant les fuites de mémoire. En effet, dans de nombreux cas pratiques, la gestion de la mémoire prends une part importante du temps de calcul et doit donc être prise en compte dans les calculs de complexités.

▸ Exercice 1. (Implantation d'une pile par tableau)

Une pile est une structure de donnée qui enregistre des informations selon le mode dernier entré premier sorti (LIFO : Last In First Out). On manipule une pile en utilisant les quatres opérations suivantes :

Création d'une nouvelle pile	PileVide() : Pile	
Teste si la pile est vide	EstVide(Pile) : Booleen	
Ajout d'un élément	Empiler(T, Pile)	modifie la pile
Suppression d'un élément	Depiler(Pile) : T	modifie la pile

Note : Depiler(p) est valide seulement si non EstVide(p).

1. En utilisant un tableau proposer une structure permettant de stocker une pile ;
2. Écrire les quatres fonctions précédentes ;
3. Quelle est leur complexité.

▸ Exercice 2. (Implantation d'une file par tableau)

Une file est une structure de donnée qui enregistre des informations selon le mode premier entré premier sorti (FIFO : First In First Out). On manipule une pile en utilisant les quatres opérations suivantes :

Création d'une nouvelle file	FileVide() : File	
Teste si la pile est vide	EstVide(File) : Booleen	
Ajout d'un élément	Enfiler(T, File)	modifie la file
Suppression d'un élément	Defiler(File) : T	modifile la file

Note : Defiler(p) est valide seulement si non EstVide(p).

1. Mêmes questions que pour les piles.

▸ Exercice 3. (Destruction et copie d'une liste chaînée)

1. Écrire une fonction qui prend en paramètre une liste chaînée et qui désalloue la liste.
2. Écrire une fonction qui prend en paramètre une liste chaînée et qui clone cette liste.

▸ **Exercice 4.** Écrire en constructif et en mutatif les fonctions suivantes ;

1. insertion en tête ;
2. insertion en queue ;
3. suppression en tête ;
4. suppression en queue ;
5. concaténation de deux listes.

Quel est la complexité de ces fonctions ?

▸ **Exercice 5.** Écrire en constructif et en mutatif les fonctions suivantes ;

1. insertion en position i ;
2. suppression en position i ;

▸ **Exercice 6.** Une astuce technique, pour éviter d'avoir à faire un cas particulier pour le premier élément ou pour la liste vide, est d'ajouter une fausse tête à la liste. Reprendre les deux exercices précédent en utilisant cette astuce.

▸ **Exercice 7.** Pour réduire la complexité de l'ajout en queue on peut garder un pointeur sur la dernière cellule de la liste. Dans cet exercice, on travaille en mutatif avec une fausse tête. Écrire les déclarations C correspondantes.

Écrire les fonctions suivantes en donnant leurs complexités :

1. initialisation ;
2. test à vide ;
3. insertion en tête ;
4. insertion en queue ;
5. suppression en tête ;
6. suppression en queue ;
7. concaténation ;

Une autre variante consiste à utiliser des listes circulaires où l'on pointe sur la dernière cellule qui elle même pointe sur la première.

▸ **Exercice 8.** Dans l'exercice précédent, la suppression en queue à un coût linéaire. Pour réduire ce coût, on utilise des listes doublements chaînées : chaque cellule garde un pointeur sur l'élément suivant et l'élément précédent. Reprendre l'exercice précédent en utilisant des listes doublement chaînées.