Travaux Pratiques d'algorithmique n°7

Cours d'Informatique de Deuxième Année

—DEUG Sciences S3-MI—

Listes chaînées triées

Le but de ce TP est de manipuler les listes chaînées triées. Nous considérons des listes chaînées d'entiers. Les fonctions devront être écrites **récursivement**.

Dans tout ce qui suit, nous appelons simplement liste une liste chaînée d'entiers implémentée par :

```
typedef struct _cellule
{
   int valeur;                     /* donnee stockee : un entier.        */
   struct _cellule * suivant;      /* pointeur sur la cellule suivante. */
} Cellule;                         /* definition d'un nouveau type.     */
typedef Cellule * Liste;           /* definition d'un nouveau type.     */
```

▸ **Exercice 1.** *Écrire les fonctions suivantes :*

- `Liste Insere( Liste, int );` *qui insère un élément dans une liste triée ;*
- `Cellule * Recherche( Liste, int );` *qui recherche un élément dans une liste triée.*

▸ **Exercice 2.** *Écrire les fonctions suivantes :*

- `void AfficheOrdre( Liste );` *qui affiche dans l'ordre de chaînage une liste triée ;*
- `void AfficheInverse( Liste );` *qui affiche dans l'ordre inverse de chaînage une liste triée ;*
- `Creation( Liste * );` *qui crée une liste triée à partir d'entiers lus sur l'entrée standard ;*

Ré-écrire ces fonctions afin qu'elles lisent et écrivent dans un fichier ; le nom du fichier sera passé en paramètre aux fonctions (si ce nom vaut `NULL` *la lecture se fera sur* `stdin` *et l'écriture sur* `stdout` *selon le cas).*

▸ **Exercice 3.** *Écrire une fonction* `void LibererListe(Liste * );` *qui libère la place mémoire prise par une liste.*

▸ **Exercice 4.** *Écrire une fonction* `int Suppression( Liste*, int );` *qui supprime la première occurrence de l'entier dans la liste.*

▸ **Exercice 5.**

- *Écrire une fonction* `Construct` *prenant en argument un entier* `n` *et qui retourne la liste des entiers compris entre 2 et* `n` *(dans l'ordre croissant) ;*
- *écrire une fonction* `Elimine` *prenant en argument une liste d'entiers* `l` *et un entier* `a` *et qui supprime dans* `l` *tous les multiples de* `a` *;*

On veut maintenant implémenter sur des listes chaînées l'algorithme du crible d'Eratosthène. Cet algorithme permet de calculer les nombres premiers inférieurs à une borne fixée `n`*. On part de l'ensemble de tous les entiers inférieurs à* `n`*. On supprime tous les multiples de 2 (sauf 2 qui est premier). Le plus petit nombre > 2 qui reste est alors 3 qui est premier. On supprime alors tous ses multiples (sauf lui même). On recommence la même opération avec le plus petit nombre > 3 restant et ceci jusqu'à ce que la liste ne contienne que des nombres premiers.*

- *Déduire des questions précédentes une fonction* `Crible` *prenant en argument un entier* `n` *et qui retourne la liste des entiers premiers inférieurs à* `n`*.*

▸ **Exercice 6.**

Écrire une fonction qui supprime tous les éléments égaux dans une liste triée. Par exemple, la liste `(1 1 1 3 4 4 5 5 5)` *sera transformée en* `(1 3 4 5)`.

Écrire une fonction qui, étant données deux listes triées, renvoie 0 si elles contiennent les mêmes éléments aux mêmes positions, 1 sinon. Par exemple, la fonction appliquée aux listes `(1,2,2,3,3,3)` *et* `(1,2,2,3,3,3)` *retournera 1, et la fonction appliquée aux listes* `(1,2,2,3,3,3)` *et* `(1,2,3)` *retournera 0.*

Écrire une fonction qui, étant données deux listes triées, renvoie 0 si elles contiennent les mêmes éléments, 1 sinon. Par exemple, la fonction appliquée aux listes `(1,2,2,3,3,3)` *et* `(1,2,2,3,3,3)` *retournera 1, et la fonction appliquée aux listes* `(1,2,2,3,3,3)` *et* `(1,2,3)` *retournera 1. Vous travaillerez sans construire de liste temporaire.*

Calculer la complexité des deux dernières fonctions.