

# ESISAR : Algorithmique MAN 3<sup>ème</sup> année

## TD4 (CS-318) : Listes. Piles. Files

Ioan Marius BILASCO, 4 octobre 2004

### Listes

#### **Exercice 1**

Une liste à double entrées est une structure de données permettant de représenter une liste  $L=(a_0, \dots, a_{n-1})$  et d'effectuer en temps constant les opérations d'insertion en tête et en queue et l'extraction du premier ou du dernier élément de la liste. Donner les déclarations pour ce type de données et les algorithmes associés effectuant les opérations d'insertion en tête et en queue et d'extraction du premier et du dernier élément.

#### **Exercice 2**

Soit  $L=(a_0, \dots, a_{n-1})$ . On veut faire tourner  $L$  de  $k$  positions vers la gauche pour obtenir la liste  $L'=(a_k, \dots, a_{n-1}, a_0, \dots, a_{k-1})$ .

#### **Exercice 3**

On représente les polynômes à une variable à coefficients entiers par des listes chaînées de monômes, un monôme  $aX^e$  étant représenté par le couple d'entiers  $(a, e)$ . Les monômes d'un polynôme sont classés par degré croissant et chaque monôme a un coefficient  $a \neq 0$ . Ecrire des fonctions d'addition/soustraction, multiplication et évaluation dans  $x$  d'un polynôme.

#### **Exercice 4**

Comment peut-on réutiliser les algorithmes de l'exercice précédent pour réaliser l'addition des deux nombres entiers de taille quelconque ?

Proposer une représentation et un algorithme qui permet d'additionner deux nombres entiers de taille quelconque. Proposer une méthode adaptée de saisie des nombres entiers de taille quelconque.

## Exercice 5

Le tri rapide consiste à réorganiser une liste  $L=(a_0, \dots, a_{n-1})$  en trois listes  $L_1=(b_0, \dots, b_{p-1})$ ,  $L_2=(a_0)$  et  $L_3=(c_0, \dots, c_{q-1})$  dont la concaténation est une permutation de  $L$  et telles que tous les éléments de  $L_1$  sont inférieurs ou équivalents à  $a_0$  et tous les éléments de  $L_3$  sont supérieurs ou équivalents à  $a_0$ . Pour trier  $L$ , il reste à trier récursivement  $L_1$  et  $L_3$  et à concaténer les listes obtenues avec  $L_2$ .

Il est interdit d'allouer des nouveaux éléments ou d'échanger le contenu entre éléments. Il faut simplement modifier les liaisons entre éléments.

## Exercice 6 (codage Huffman – 1<sup>ère</sup> partie)

Le codage de Huffman est une méthode de compression de données, utilisée en particulier pour la transmission de messages par télécopie et sur minitel. Ce codage utilise le modèle suivant : dans chaque message, les symboles sont indépendants et apparaissent en toute position avec une probabilité connue indépendante de la position. Chaque symbole est codé en une suite de 0 et de 1 de telle façon que les symboles de forte probabilité auront un code court alors que les symboles de faible probabilité auront un code long.

Les codes étant de longueur variable, il est nécessaire, pour décoder un message, que le code de chaque caractère ne soit pas le préfixe du code d'un autre caractère. C'est la règle des préfixes.

L'algorithme de Huffman permet de construire un code de longueur minimale vérifiant la règle des préfixes. Pour cela, on construit un arbre binaire dont tous ces symboles sont les noeuds terminaux. Le code de chaque symbole correspond au chemin à parcourir depuis le sommet jusqu'à ce symbole avec gauche=0 et droite=1.

La première étape de codage correspond à la construction d'un algorithme qui compte les occurrences des caractères dans un texte. Proposez une représentation sous forme de liste chaînée pour les résultats. Choisissez entre une liste chaînée non-triée, triée selon l'ordre ascendant, triée selon l'ordre descendant. Intuitivement la quelle des trois représentation semble la plus efficace ? Justifiez votre réponse.

Construisez la fonction suivante :

1. Fonction `construire_huff([in] texte :Element) :ListeFrequence` qui prend une chaîne de caractères qui se finit par le caractère '\0' en argument, et retourne la liste de fréquences de chaque caractère.

## Piles

### Exercice 7

En utilisant le principe de la pile, affichez à l'inverse une chaîne de caractères saisie au clavier.

### Exercice 8

Donnez la version itérative de tri-rapide en utilisant une pile pour simuler les appels récursifs.

Version récursive :

```
Fonction Tri([in] g,d :entier ; [in/out] a : tableau d'entiers)
VAR
    k :entier ;
DEBUT
    Separer ag, ..., ad tq. ai ≤ ak si g ≤ i < k et ai > ak si k < i ≤ d
    Si g < k alors tri(g,k,a) ; finsi
    Si k < d alors tri(k,d,a) ; finsi
FIN
```

Pour simuler les appels récursifs la pile contiendra les plages (g,d) qui correspondent à chaque appel récursif. A chaque itération on dépile la plage en-tête de la pile, on réalise la séparation et on empile les plages correspondant aux appels concernant les sous-plages détectées (g,k) et (k,d). On s'arrête lorsqu'il n'y a plus de plage à traiter.

## Files & Files à priorité

### Exercice 9

On se propose d'implémenter une file d'attente à l'aide d'une liste chaînée.

a) Définir un type *File* permettant de gérer une file d'attente de chaînes de caractères.

b) Écrire les primitives de gestion d'une file d'attente suivantes :

```
Procédure      déposer_dans_file([in/out]      file :File,      [in]
element :Element) ;
```

Place l'élément dans la file. Le type *Element* est ici une chaîne de caractères qui se termine par le caractère ayant le code zéro ('Marius\0') et que l'on recopiera.

```
Fonction taille_file([in] file :File ) :entier ;
```

Retourne le nombre d'éléments dans la queue.

```
Fonction retirer_de_file([in/out] file :File ) :Element ;
```

Retire l'élément qui est depuis le plus longtemps dans la queue (affiche un message d'erreur si la queue est vide).

c) Ajouter une procédure `afficher_file([in] file :File)` et écrire un petit programme de test.

## Exercice 10

On désire maintenant introduire une notion de priorité dans la file d'attente. La priorité est codée par un nombre entier positif, 0 étant le moins prioritaire.

Lorsque l'on retire un élément de la queue, on veut obtenir le plus ancien avec la priorité la plus haute.

Lors du dépôt d'un élément on indique sa priorité.

1- Définir un nouveau type *FilePriorite* pour gérer la queue avec priorités.

2- Ecrire les fonctions de dépôt et de retrait.

```
Procédure déposer_dans_file([in/out] file :FilePriorite ; [in] elt :Element, [in] priorite :entier);
```

```
Fonction retirer_de_file([in/out] file :FilePriorite ) :Element ;
```

## Exercice 11

En utilisant une file à priorité contenant des caractères vérifiez si deux mots représentent une anagramme (ils contiennent exactement les mêmes lettres).