Langage C/C++

TD 3 - 4 : Création dynamique d'objets

Hubert Godfroy

27 novembre 2014

1 Tableaux

Question 1 :
Écrire une fonction prenant un paramètre n et créant un tableau de taille n (contenant des entiers).

Question 2 :
Écrire une fonction libérant la mémoire d'un tableau ainsi créé.

2 Données récursives

2.1 Listes chaînées

Question 3 :
Définir le type des listes chaînées : une liste est soit la liste vide, soit la donnée d'un entier et d'une liste chaînée.

Dans la suite on ne travaillera qu'avec des suites d'entiers.

Question 4 :
Écrire une fonction prenant une liste et lui ajoutant un élément en tête.

Question 5 :
Écrire une fonction effaçant le premier élément d'une liste.

Question 6 :
Écrire une fonction donnant la taille d'une liste.

Question 7 :
Écrire une fonction testant l'appartenance d'un entier à la liste.

Question 8 :
Écrire une fonction construisant la liste renversée d'une liste.

Question 9 :
Écrire une fonction donnant le n-ème élément d'une liste.

Question 10 :
Écrire une fonction supprimant le n-ème élément d'une liste.

Question 11 :
Écrire une fonction concaténant deux listes.

Question 12 :
Écrire une fonction triant une liste.

Question 13 :
Écrire une fonction détruisant une liste (c'est à dire libérant la mémoire de tous ses éléments).

2.2 Arbres

Question 14 :
Répondre à la question 14 de la semaine dernière (définition d'un type encodant les arbres).

Solution :

```
typedef struct _arbre {
  int val;
  struct _arbre* gauche;
  struct _arbre* droite;
} arbre;
```

À partir de maintenant, chaque nœud d'un arbre contiendra un entier.

Question 15 :
Créer une fonction construisant un arbre à partir d'un entier et de deux arbres.

Solution :

```
arbre* newArbre(arbre* g, arbre* d) {
  arbre* res = malloc(sizeof(arbre));
  res->gauche = g;
  res->droite = d;
  return res;
}
```

Question 16 :
Écrire une fonction détruisant un arbre.

Solution :

```
void deleteArbre(arbre* a) {
  if (a != NULL) {
    deleteArbre(a->gauche);
```

```
4       deleteArbre(a->droite);
5       free(a);
6     }
7   }
```

3 Codage de Huffman

3.1 Heuristique

La technique abordée ici relève de ce que l'on pourrait rapprocher des moyens mnémotechniques que l'on se donne étant enfant pour retenir ses leçons. En effet, il s'agit ici de trouver une façon de retenir l'information plus facilement, c'est-à-dire, pour un ordinateur, de diminuer la taille de l'information.

3.2 Algorithme

Pour faire cela on affecte à chaque caractère de la liste de caractères une pondération qui est sa fréquence dans la chaine à encoder. Ensuite on crée un arbre binaire de la manière suivante :

1. Les deux caractères de moindre occurence sont réunis en un arbre dont ils sont les feuilles et dont la racine est la somme de leurs deux fréquences.
2. Cet arbre devient lui-même un nouveau caractère dont la fréquence est sa racine. Il s'agit alors de le trier parmi les autres caractères.
3. On réitère le processus jusqu'à n'avoir plus qu'un arbre.

Par exemple, si l'on souhaite coder le texte en binaire :

```
01100101 01100011 01101111
01101100 01100101 00100000
01100100 01100101 01110011
00100000 01101101 01101001
01101110 01100101 01100101
00100000 01100100 01100101
00100000 01101110 01100001
01101110 01100011 01111001.
```

— On a ce tableau des fréquences :

o	l	m	i	a	y	c	d	s	n	" "	e
1	1	1	1	1	1	2	2	2	3	4	5

— puis successivement,

m	i	a	y	2 0 l	c	d	s	n	" "	e
1	1	1	1	2	2	2	2	3	4	5

a	y	2 0 l	2 m i	c	d	s	n	" "	e
1	1	2	2	2	2	2	3	4	5

— et finalement, on obtient l'arbre de Huffman de la figure 1.

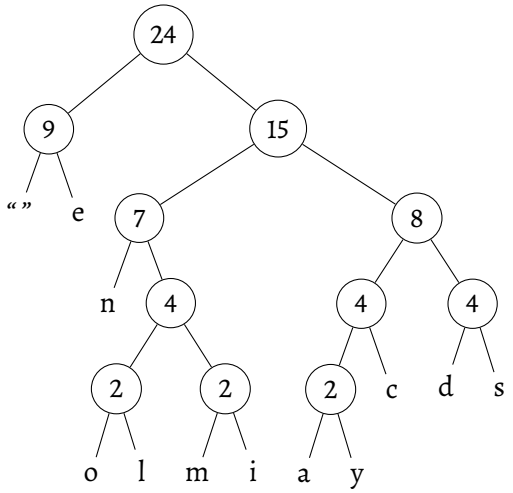

Figure 1 – Arbre de Huffman

— L'arbre ainsi créé sert au codage de chaque caractère : ce code est trouvé en parcourant l'arbre de la racine jusqu'à ses feuilles qui sont les caractères initiaux.
— On choisit une convention de parcours, par exemple, si dans le parcours, on choisit le fils droit, on ajoute 1 au code, sinon 0[1].
— À chaque caractère est ainsi associé un code binaire. On peut alors faire un tableau de conversion enregistrant ces associations que l'on écrira dans le fichier compressé pour la phase de décompression.

Finalement, le message devient

1. On notera que l'arbre obtenu est un arbre binaire, c'est à dire que chaque nœud a au plus deux fils. En effet, cette propriété est due au fait que l'on construit l'arbre de manière successive en choisissant toujours les ***deux*** caractères les moins fréquents

```
01 1101 10100 10101 01 00 1110 01
1111 00 10110 10111 100 01 1111
00 1110 01 00 100 11000 100 1101
11001.
```

Question 17 :

Programmer l'algorithme de HUFFMAN