Codage de Shannon, Fano et Huffmann

travaux-pratiques d'algorithmique n. 5

19 novembre 2003

Objectifs. Implanter les algorithmes de Huffman, et de Fano-Shannon. Manipulation des arborescences binaires.
Lectures.

Algorithmes en Langage C, R. Sedgewick.
Algorithm and complexity, S. H. Wilf.
The Art of Computer Programming, D. Knuth.
A Mathematical Theory of Communication, C. Shannon.

Durée: 2 fois 3 heures.

Les codes préfixes binaires sont utilisés pour coder les suites de symbôles sans ambiguité. Du point de vue de la compression, un codage optimal des lettres de l'alphabet A composant un texte doit minimiser la quantité :

$$\sum_{a \in A} \lambda(a) p(a),$$

où $p(a)$ est la probabilité de la lettre a dans le texte et $\lambda(a)$ la longueur du mot utilisé pour coder la lettre a. Nous savons que les notions d'arbre binaire et de code préfixe binaire sont équivalentes : se donner un codage préfixe binaire de l'alphabet A est équivalent à se donner un arbre binaire dont les feuilles sont étiquetées par les lettres de A.

```
typedef struct tarbre{
      int car;
    float prob;
    struct tarbre *fg, *fd;
} NOEUD, *ARBRE;
```

Dans la suite, nous utilisons quelques primitives sur les arbres. Une procédure d'allocation avec initialisation `ALLOC( c, f )`. Une procédure `GREFFE(x, y)` qui construit un arbre père pour les arbres x et y. Une procédure de parcours `PREFIXE( z )` permet d'imprimer l'information représentée par l'arbre binaire z.

```
SF( x, i, j)
DONNEES
   x : tableau de reels;
 i,j : indice
 g,d : arbre;
DEBUT
   SI ( j > i ) ALORS
      k := COUPE( x, i, j);
      g := SF( x, i  , k);
      d := SF( y, k+1, j);
      RETOURNER GREFFE( g, d);
   FSI
   RETOURNER ALLOC( i, x[i] );
FIN
```

```
SHANNON-FANO( x )
DONNEES
   x : tableau de reels;
VARIABLE
   p : tableau d'indice;
   a : arbre;
DEBUT
   p  := TRIER( t );
   a  := SF( t, 1, n);
   PARCOURS( x, p);
FIN
```

FIG. 1 – Algorithme de Fano-Shannon

Méthode descendante : Fano-Shannon

La procédure de Fano construit un arbre binaire de manière descendante. Les lettres sont ordonnées par probabilité croissante. L'alphabet est partitionné en deux. Une partie est codée sur la partie gauche, la seconde sur la partie droite.

1. La procédure `coupe(x, i, j)` détermine une position k qui minimise l'écart entre les sommes $\sum_{u=i}^{k} x_u$ et $\sum_{v=k+1}^{j} x_v$.
2. La procédure `trier(x)` renvoie la permutation utilisée par la procédure de tri.
3. Un parcours de l'arbre permet d'identifier l'arbre construit.

Méthode ascendante : codage Huffman

L'algorithme de Huffman construit un arbre binaire en logeant les lettres les moins fréquentes sur les feuilles les plus profondes. Il s'agit une méthode ascendante.

1. La procédure `TRIER( f )` trie les arbres par probabilité décroissante.
2. La procédure `REPLACER( f, n)` insère le nouvel arbre en bonne position.
3. Un parcours de l'arbre permet d'identifier l'arbre construit.

Performances

1. Écrire une fonction pour calculer le coût d'un arbre binaire.
2. Mettre en évidence une distribution des lettres pour laquelle la procédure de Huffman est strictement meilleure que celui de Fano.

```
HUFFMAN( x )
DONNEES
   x : tableau de reels;
VARIABLE
   f : tableau d'arbres;
DEBUT
   n := TAILLE( x );
   POUR i DANS [1..n]
     f[i] = ALLOC( i, x[i] );
   FIP
   TRIER( f );
   TANTQUE ( n > 1 )
       f[n-1] := GREFFER( f[n-1], f[n]);
       REPLACER(f, n );
       DEC( n );
   FTQ
   RETOURNER F[1];
FIN
```

Fig. 2 – Algorithme de Huffman

3. Faire varier les distributions de lettres. Comparer le coût des arbres binaires obtenus avec l'entropie de la distribution :

$$H = -\sum_{a \in A} p(a) \log p(a).$$

4. Commentez vos résultats numériques.