Algorithmique TD6

Magistère Informatique 1ère Année

04 Novembre 2010

Exercice 1 *Codage de Huffman, Algorithmes Gloutons*

Soit Σ un alphabet fini et de cardinal supérieur ou égal à deux. On appelle mot de code toute suite finie de 0 et de 1, soit $\mathcal{C}$ l'ensemble des mots de code. On appelle codage binaire une application injective α de l'alphabet Σ dans $\mathcal{C}$. En utilisant l'opération concaténation, α s'étend de manière naturelle en :

$$\alpha : \Sigma^* \to \mathcal{C}$$

On dit qu'un codage est de longueur fixe quand toutes les lettres sont codées par un mot de code de même longueur. Un codage est dit préfixe si aucune lettre n'est codée par un mot de code qui est préfixe du codage d'une autre lettre.

1) Montrer que pour un codage de longueur fixe et un codage préfixe, α est injective sur Σ^*.

2) Montrer qu'on peut représenter un codage préfixe par un arbre binaire dont les feuilles sont les lettres de l'alphabet.

On considère un mot w. A chaque codage préfixe de w, représenté par un arbre T, est associé un coût défini par :

$$C_w(T) = \sum_{a \in \Sigma} f(a) l_T(a)$$

avec $f(a)$ le nombre d'occurences de a dans w et $l_T(a)$ la taille du mot de code associé à a par T. Un codage préfixe est dit optimal si il minimise la fonction C_w

3) Montrer qu'à un codage préfixe optimal correspond un arbre binaire où tout nœud interne a deux fils.

4) Montrer qu'il existe un codage préfixe optimal pour lequel les deux lettres dont le nombre d'occurences est le plus faible sont sœurs dans l'arbre.

Etant donnés x et y les deux lettres dont le nombre d'occurences est le plus faible dans w, on considère l'alphabet $\Sigma' = \Sigma - \{x, y\} + z$ où z est une nouvelle lettre à laquelle on associe $f(z) = f(x) + f(y)$.

5) Soit T' l'arbre d'un codage optimal pour Σ', montrer que l'arbre T obtenu à partir de T' en remplaçant la feuille associée à z par un nœud interne ayant x et y comme feuilles représente un codage optimal pour Σ.

6) En déduire un algorithme recherchant un codage optimal et donner sa complexité.

Exercise 2 *Partitions*

On définit une variante du type abstrait `partition` comme suit :

- Donnée : une partition P de $\{1,\ldots,N\}$, chaque classe $X \in P$ étant désignée par un entier de $\{1,\ldots,M\}$, *son nom*.
  Attention : le nom d'une classe n'est pas nécessairement un élément de la classe et deux classes distinctes ne peuvent avoir le même nom.
- Opérations :

$$\begin{aligned}
\texttt{créer} &: \text{Int} \times \text{Int} \to \texttt{Partition} \\
\texttt{find} &: \texttt{Partition} \times \text{Int} \to \text{Int} \\
\texttt{union} &: \texttt{Partition} \times \text{Int} \times \text{Int} \times \text{Int} \to \texttt{Partition} \\
\texttt{renommer} &: \texttt{Partition} \times \text{Int} \times \text{Int} \to \texttt{Partition}
\end{aligned}$$

- `Partition(N,M) P` crée une partition P composée des singletons de $\{1,\ldots,N\}$ et dont l'ensemble de noms est $\{1,\ldots,M\}$.
  Il faut $N \leq M$ et initialement, le nom du singleton $\{i\}$ est i.
- `P.find(i)` retourne le nom de la classe de P qui contient i.
- `P.union(x,y,z)` fusionne les classes de noms x et y en une classe de nom z.
  Ne fait rien s'il n'y a pas de classe de nom x ou pas de classe de nom y ou s'il y a déjà une classe de nom z et que $z \neq x$ et $z \neq y$.
- `P.renommer(x,y)` change le nom de la classe x en y.
  Ne fait rien s'il n'y a pas de classe de nom x ou s'il y a déjà une classe de nom y.

[5] **a)** Décrire une structure de données concrète pour implémenter très efficacement le type abstrait `partition` et donner les algorithmes pour les 4 opérations sur ce type. Les opérations `union` et `renommer` devront s'exécuter en temps constant. L'opération `créer` devra être en temps $\mathcal{O}(N+M)$. Le coût d'une suite d'opérations comportant $n-1$ `unions` et m `find` sera en $\mathcal{O}(n+m\alpha(n))$ où $\alpha(n)$ est l'inverse de la fonction d'Ackerman vue en cours.
Remarque : on peut bien sûr utiliser les théorèmes du cours.

On souhaite maintenant résoudre le problème des minima hors-ligne. La donnée est une suite d'entiers 2 à 2 distincts et d'instructions `extraire-min`, notées E dans la suite. Par exemple : $\sigma =$ 5, 4, 8, 2, E, 7, E, 1, 6, E, E, 3.
Chaque instruction E doit afficher le plus petit entier qui précède et qui n'a pas déjà été affiché. Pour l'exemple ci-dessus on affichera 2, 4, 1, 5.
On cherche à résoudre ce problème avec un algorithme *hors-ligne*, i.e., on peut lire *toute* la donnée σ *avant* de commencer à afficher le résultat.

[5] **b)** Donner un algorithme pour résoudre le problème des minima hors-ligne lorsque les entiers de la suite σ forment une permutation de $\{1,\ldots,N\}$ et que σ contient au plus N instructions `extraire-min` (E). L'algorithme devra s'exécuter en temps $\mathcal{O}(n\alpha(n))$.
Indication : on pourra commencer par créer une partition dont la partie de nom $i \geq 1$ contient les entiers qui se trouvent entre le $(i-1)$-ème E et le i-ème E. Avec l'exemple ci-dessus, nous aurons les parties $X_1 = \{5,4,8,2\}$, $X_2 = \{7\}$, $X_3 = \{1,6\}$, $X_4 = \emptyset$, et $X_5 = \{3\}$.

Une question supplémentaire pour ceux qui s'ennuieraient.

[5] **c)** Montrer que la complexité d'une suite de n unions par taille *suivie* d'une suite de m find avec compression des chemins est au pire en $\mathcal{O}(n+m)$.
Remarque : il est important pour cette question de faire *toutes* les unions *avant* les find.