

## TD n°4 – Algorithmes gloutons

### 1 Couverture par intervalles

**Question 1.1** Etant donné un ensemble  $\{x_1, \dots, x_n\}$  de  $n$  points sur une droite, décrire un algorithme qui détermine le plus petit ensemble d'intervalles fermés de longueur 1 qui contient tous les points donnés. Prouver la correction de votre algorithme et donner sa complexité.

**Solution :**

*début*

Trier dans l'ordre croissant les  $x_i$ ;

$X \leftarrow \{x_1, \dots, x_n\}$ ;

$I \leftarrow \emptyset$ ;

*tant que*  $X \neq \emptyset$  *faire*

$x_k \leftarrow \min(X)$ ;

$I \leftarrow I \cup \{[x_k, x_k + 1]\}$ ;

$X \leftarrow X \setminus [x_k, x_k + 1]$ ;

*fin*

*retourner*  $I$ ;

*fin*

**Algorithme 1** : Couverture par intervalles

La complexité est  $n \log n$  (tri) +  $n$  (parcours de  $X$ ), c'est-à-dire  $O(n \log n)$

**Théorème 1.** Cet algorithme est optimal

**Méthode 1 : avec des matroïdes.** Cette méthode ne fonctionne pas. Il est impossible d'exhiber un matroïde (on ne peut pas prouver la propriété d'échange)

**Méthode 2 : par récurrence sur  $|X|$ .** On ajoute un à un les éléments. On montre que le premier est bien contenu dans l'ensemble fourni par l'algorithme.

- **Initialisation :** Si on a qu'un seul point, l'algorithme renvoie un seul intervalle : c'est une solution optimale.
- Soient un recouvrement optimal  $I_{opt} = \{[a_1, a_1 + 1], \dots, [a_p, a_p + 1]\}$  avec  $a_1 < \dots < a_p$  et le recouvrement donné par notre algorithme  $I_{glou} = \{[g_1, g_1 + 1], \dots, [g_m, g_m + 1]\}$  avec  $g_1 < \dots < g_m$ . Montrons que  $m = p$ .
- On pose  $I = (I_{opt} \setminus \{[a_1, a_1 + 1]\}) \cup \{[g_1, g_1 + 1]\}$ .  
Puisque  $g_1 = x_1$ ,  $I$  est un recouvrement de  $X$  car  $a_1 \leq x_1 \Rightarrow a_1 + 1 \leq x_1 + 1 = g_1 + 1$  donc  $X \cap [a_1, a_1 + 1] \subseteq X \cap [g_1, g_1 + 1]$ ; autrement-dit  $I_{opt} \setminus \{[a_1, a_1 + 1]\}$  est un recouvrement de  $X \setminus \{[g_1, g_1 + 1]\}$ .
- Par hypothèse de récurrence, sur  $X \setminus [g_1, g_1 + 1]$  une solution optimale est donnée par glouton :  $\{[g_2, g_2 + 1], \dots, [g_m, g_m + 1]\}$  car  $|X \setminus [g_1, g_1 + 1]| \leq n - 1$

- Comme  $I_{opt} \setminus \{[a_1, a_1 + 1]\}$  est un recouvrement de  $X \setminus [g_1, g_1 + 1]$  avec  $p - 1$  intervalles, on a :

$$\left. \begin{array}{l} p - 1 \geq m - 1 \Rightarrow p \geq m \\ I_{opt} \text{ optimal sur } X \\ I_{glou} \text{ recouvrement de } X \end{array} \right\} \Rightarrow m \geq p \left. \right\} \Rightarrow m = p$$

Puisque  $p$  est le nombre d'intervalles que renvoie un algorithme optimal, et que  $m = p$ , alors, notre glouton est optimal.

□

## 2 Codage de Huffman

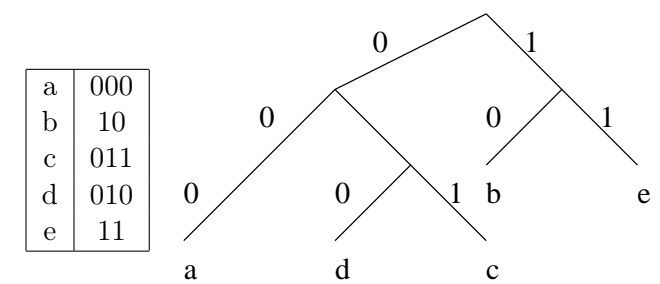
Soit  $\Sigma$  un alphabet fini de cardinal au moins deux. Un *codage binaire* est une application injective de  $\Sigma$  dans l'ensemble des suites finies de 0 et de 1 (les images des lettres de  $\Sigma$  sont appelées *mots de code*). Il s'étend de manière naturelle par concaténation en une application définie sur l'ensemble  $\Sigma^*$  des mots sur  $\Sigma$ . Un codage est dit *de longueur fixe* si toutes les lettres dans  $\Sigma$  sont codées par des mots binaires de même longueur. Un codage est dit *préfixe* si aucun mot de code n'est préfixe d'un autre mot de code.

**Question 2.1** Le décodage d'un codage de longueur fixe est unique. Montrer qu'il en est de même pour un codage préfixe.

**Solution :** Dans un codage préfixe, le décodage est unique. En effet, si  $x$  peut se décoder de deux façons, alors  $x = ua$  et  $x = vb$ , et donc  $u$  est préfixe de  $v$  ou  $v$  est préfixe de  $u$ , donc  $u = v$  pour un codage préfixe. □

**Question 2.2** Représenter un codage préfixe par un arbre binaire dont les feuilles sont les lettres de l'alphabet.

**Solution :** On peut représenter le codage préfixe par un arbre : on associe 0 à chaque branche gauche et 1 à chaque branche droite. Pour chaque caractère, on lit le code à partir de la racine. Ainsi, l'arbre représenté dans la figure 1 correspond au codage des lettres  $a, b, c, d$  et  $e$ .



TAB. 1: Codage de a,b,c,d et e

□

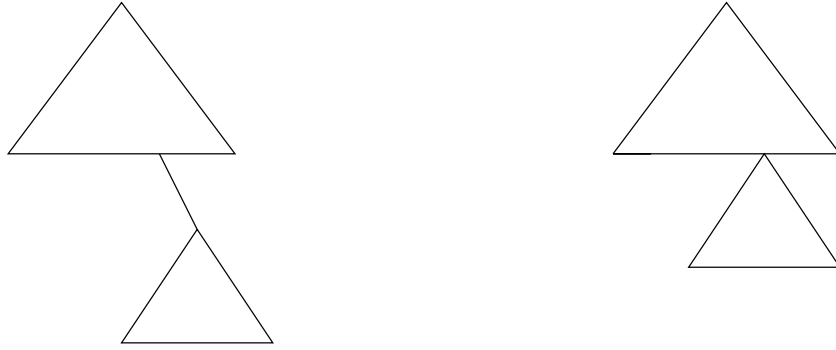


FIG. 1: Chaque codage préfixe optimal correspond à un arbre binaire dont les nœuds internes ont deux fils

**Question 2.3** On considère un texte dans lequel chaque lettre  $c$  apparaît avec une fréquence  $f(c)$  non nulle. A chaque codage préfixe de ce texte, représenté par un arbre  $T$ , est associé un coût défini par :

$$B(T) = \sum_{c \in \Sigma} f(c)l_T(c)$$

où  $l_T(c)$  est la taille du mot binaire codant  $c$ . Si  $f(c)$  est exactement le nombre d'occurrences de  $c$  dans le texte, alors  $B(T)$  est le nombre de bits du codage du texte.

Un codage préfixe représenté par un arbre  $T$  est *optimal* si, pour ce texte, il minimise la fonction  $B$ . Montrer qu'à un codage préfixe optimal correspond un arbre binaire où tout nœud interne a deux fils. Montrer qu'un tel arbre a  $|\Sigma|$  feuilles et  $|\Sigma| - 1$  nœuds internes.

**Solution :** Chaque codage préfixe optimal correspond à un arbre binaire où tout nœud interne a deux fils. En effet, s'il existe un nœud qui n'a qu'un fils, alors on diminue le  $L_T(c)$  en remontant le sous-arbre correspondant : soit la feuille correspondant à la lettre  $c$  était dans le sous-arbre, dans ce cas  $L'_T(c) = L_T(c) - 1$ , soit elle n'y était pas et  $L'_T(c) = L_T(c)$  (cf figure 1).

Un tel arbre a par construction  $|\Sigma|$  feuilles, puisqu'il code  $|\Sigma|$  lettres. Montrons par récurrence qu'il a  $|\Sigma| - 1$  nœuds internes :

- Un arbre à 2 feuilles a 1 nœud interne.
- Si l'arbre est constitué de deux sous-arbres de  $i$  et  $j$  feuilles, alors les deux sous-arbres ont  $i - 1$  et  $j - 1$  nœuds internes (par hypothèse de récurrence), et l'arbre total a donc  $i - 1 + j - 1 + 1 = i + j - 1$  nœuds internes, la propriété est donc vraie.

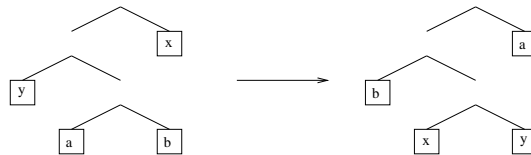
□

**Question 2.4** Montrer qu'il existe un codage préfixe optimal pour lequel les deux lettres de plus faibles fréquences sont soeurs dans l'arbre (autrement dit leurs codes sont de même longueur et ne diffèrent que par le dernier bit) - *Propriété de choix glouton*.

Etant donnés  $x$  et  $y$  les deux lettres de plus faibles fréquences dans  $\Sigma$ , on considère l'alphabet  $\Sigma' = \Sigma - \{x, y\} + \{z\}$ , où  $z$  est une nouvelle lettre à laquelle on donne la fréquence  $f(z) = f(x) + f(y)$ . Soit  $T'$  l'arbre d'un codage optimal pour  $\Sigma'$ , 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  $\Sigma$  - *Propriété de sous-structure optimale*.

**Solution :** Montrons qu'il existe un codage préfixe optimal pour lequel les deux lettres de plus

faibles fréquences sont sœurs dans l'arbre. Soit  $T$  un codage optimal. On a  $\forall T', B(T) \leq B(T')$ . Soient  $x$  et  $y$  les lettres de fréquences les plus basses, et  $a$  et  $b$  deux lettres sœurs de profondeur maximale dans  $T$ . Construisons alors un arbre  $T''$  en inversant les positions de  $x$  et de  $a$ , ainsi que celles de  $y$  et de  $b$ , voir Figure 4.



On veut montrer que  $T''$  est optimal, c'est à dire que  $B(T) = B(T'')$ .

$$B(T) - B(T'') = f(a)l_T(a) + f(b)l_T(b) + f(x)l_T(x) + f(y)l_T(y) - f(a)l_{T''}(a) - f(b)l_{T''}(b) - f(x)l_{T''}(x) - f(y)l_{T''}(y)$$

Comme on a les égalités suivantes :

$$\begin{aligned} l_{T''}(a) &= l_T(x) \\ l_{T''}(b) &= l_T(y) \\ l_{T''}(x) &= l_T(a) \\ l_{T''}(y) &= l_T(b) \end{aligned}$$

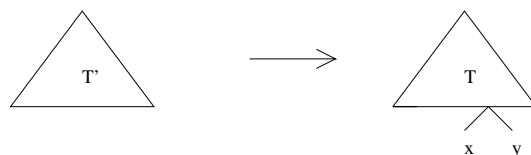
On en déduit l'égalité :

$$B(T) - B(T'') = [f(a) - f(x)][l_T(a) - l_T(x)] + [f(b) - f(y)][l_T(b) - l_T(y)]$$

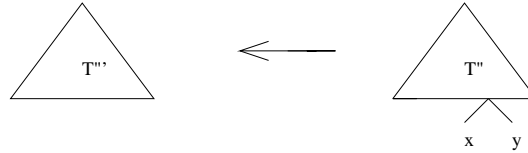
$f(a) \geq f(x)$  et  $f(b) \geq f(y)$ ,  $l_T(a) \geq l_T(x)$  et  $l_T(b) \geq l_T(y)$ , donc  $B(T) - B(T'') \geq 0$ .

Donc comme  $T$  est un codage optimal, alors  $T''$  aussi. On a donc un codage préfixe optimal dans lequel les deux lettres de plus faibles fréquences sont sœurs dans l'arbre. (Propriété de choix glouton).

On considère maintenant  $\Sigma' = \Sigma \setminus \{x, y\} + \{z\}$ , où  $z$  est une nouvelle lettre de fréquence  $f(z) = f(x) + f(y)$ . Soit  $T'$  l'arbre d'un codage optimal pour  $\Sigma'$ , on veut montrer que l'arbre  $T$  obtenu à partir de  $T'$  en remplaçant la feuille associée à  $z$  par un nœud de feuilles  $x$  et  $y$  est optimal pour  $\Sigma$ .



Soit  $T''$  un arbre optimal pour  $\Sigma$ . D'après la propriété de choix glouton, on peut supposer que  $x$  et  $y$  sont sœurs dans  $T''$ . On peut alors construire un arbre  $T'''$  en remplaçant le nœud de fils  $x$  et  $y$  par une feuille  $z$ .



On a alors  $B(T''') = B(T'') - f(x)L_T(x) - f(y)L_T(y) + (f(x) + f(y))(L_T(x) - 1) = B(T'') - f(x) - f(y)$ .

Or  $T'$  est optimal, donc  $B(T') \leq B(T''')$ . Comme  $B(T) = B(T') + f(x) + f(y)$ , on en déduit que  $B(T) \leq B(T'')$ . Donc  $T$  est optimal car  $T''$  est optimal.  $\square$

**Question 2.5** En déduire un algorithme pour trouver un codage optimal et donner sa complexité. A titre d'exemple, trouver un codage optimal pour  $\Sigma = \{a, b, c, d, e, g\}$  et  $f(a) = 45$ ,  $f(b) = 13$ ,  $f(c) = 12$ ,  $f(d) = 16$ ,  $f(e) = 9$  et  $f(g) = 5$ .

**Solution :** On en déduit l'algorithme suivant pour le codage de Huffman :

**Données :**  $\sigma, f$

;

**début**

$F \leftarrow \text{tas\_binaire}(\Sigma, f);$

$n \leftarrow |\Sigma|;$

**pour**  $i$  **de** 1 **à**  $n - 1$  **faire**

$z \leftarrow \text{allouer\_noeud}();$

$x \leftarrow \text{extraire\_min}(F);$

$y \leftarrow \text{extraire\_min}(F);$

$z(\text{gauche}) \leftarrow x;$

$z(\text{droite}) \leftarrow y;$

$f(z) \leftarrow f(x) + f(y);$

$\text{Insérer}(F, z, f(z));$

**fin**

**retourner**  $\text{extraire\_min}(F);$

**fin**

**Algorithme 2 : Huffman**

Calculons la complexité de cet algorithme :

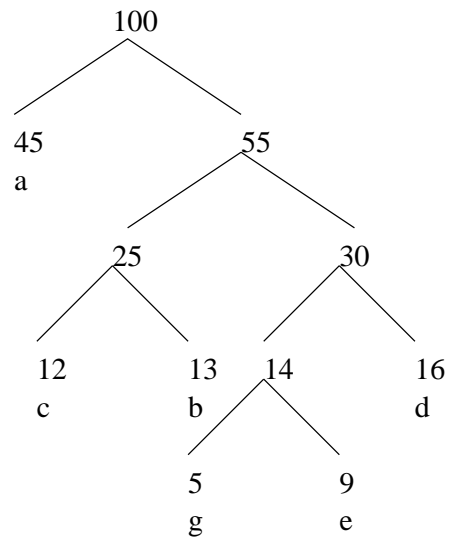
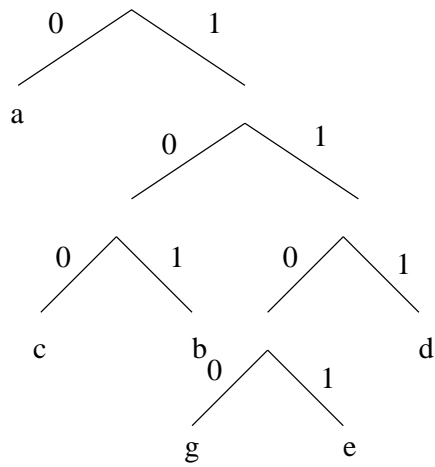
Pour un tas binaire, le coût des opérations est le suivant :

- Savoir si la file est vide  $O(1)$ .
- Insérer un élément  $O(\log n)$
- Trouver un élément de clé minimale  $O(1)$
- Extraire l'élément de clé minimale  $O(\log n)$

La complexité d'une itération de la boucle est donc en  $O(\log n)$ .

La complexité totale est donc en  $O(n \log n)$ , car il y a  $n$  itérations de la boucle, et que la construction du tas est en  $O(n \log n)$ .

Voici l'arbre que l'on obtient en appliquant cet algorithme à  $\Sigma = \{a, b, c, d, e, g\}$  avec  $f(a) = 45$ ,  $f(b) = 13$ ,  $f(c) = 12$ ,  $f(d) = 16$ ,  $f(e) = 9$ ,  $f(g) = 5$  :



□