Correction TD 4 - Algorithmes gloutons

Exercice 1 - Couverture par intervalles

début
  Trier dans l'ordre croissant les x_i ;
  $X \leftarrow \{x_1, \ldots, x_n\}$;
  $I \leftarrow \emptyset$;
  tant que $X \neq \emptyset$ **faire**
    $x_k \leftarrow \min(X)$;
    $I \leftarrow I \cup \{[x_k, x_k+1]\}$;
    $X \leftarrow X \backslash [x_k, x_k+1]$
  Retourner I ;
fin

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

Théorème : Cet algorithme est optimal.

Méthode 1 : avec des matroïdes. Cette méthode ne fontionne 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 les élements. On montre que le premier est bien contenu dans l'ensemble fournit 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 $I_{opt} = \{[a_1, a_1+1], \ldots, [a_p, a_p+1]\}$ avec $a_1 < \cdots < a_p$ et le recouvrement donné par notre algorithme $I_{glou} = \{[g_1, g_1+1], \ldots, [g_m, g_m+1]\}$ avec $g_1 < \cdots < g_m$. Montrons que $m = p$.
- On pose $I = \left(I_{opt} \backslash \{[a_1, a_1+1]\}\right) \cup \{[g_1, g_1+1]\}$.
  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 $I_{opt} \backslash \{[a_1, a_1+1]\}$ est un recouvrement de $X \backslash \{[g_1, g_1+1]\}$.
- Par hypothèse de récurrence, sur $X \backslash [g_1, g_1+1]$ une solution optimale est donnée par glouton : $\{[g_2, g_2+1], \ldots, [g_m, g_m+1]\}$ car $|X| \backslash [g_1, g_1+1] \leq n-1$
- Comme $I_{opt} \backslash \{[a_1, a_1+1]\}$ est un recouvrement de $X \backslash [g_1, g_1+1]$ avec $p-1$ intervalles, on a :
  $$\left.\begin{array}{r} p-1 \geq m-1 \Rightarrow p \geq m \\ \left.\begin{array}{l} I_{opt} \text{ optimal sur } X \\ I_{glou} \text{ recouvrement de } X \end{array}\right\} \Rightarrow m \geq p \end{array}\right\} \Rightarrow m = p$$
  Puisque p est le nombre d'intervalles que renvoie un algorithme optimal, et que $m = p$, alors, notre glouton est optimal.

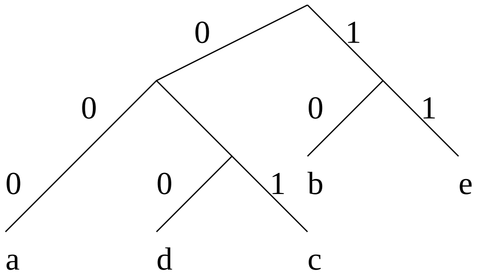
Exercice 2 - Codage de Huffman

1 – 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 x ou v est préfixe de u, donc $u = v$ pour un codage préfixe.

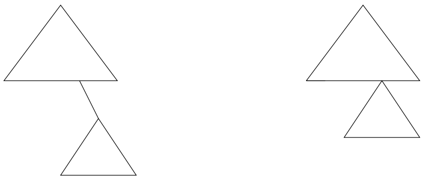
2 – On peut représenter le codage préfixe par un arbre : on associe 0 à chaque branche gauche et 1 à chaque branche droite. Pour chaque caractère, on lit le code à partir de la racine. Ainsi, l'arbre représenté dans la figure TAB.1 correspond au codage des lettres a, b, c, d et e.

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).

a	000
b	10
c	011
d	010
e	11



TAB. 1 – Codage de a,b,c,d et e



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

- Un arbre à 2 feuilles a 1 nœud.
- 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 (par hypothèse de récurrence), et l'arbre total a donc $i-1+j-1+1 = i+j-1$ nœuds, la propriété est donc vraie.

3 – 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.

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

$$\begin{aligned} B(T) - B(T'') &= f(a)l_T(a) + f(b)l_T(b) + f(x)l_T(x) + f(y)l_T(y) \\ &\quad -f(a)l_{T''}(a) - f(b)l_{T''}(b) - f(x)l_{T''}(x) - f(y)l_{T''}(y) \end{aligned}$$

Comme on a les égalités suivantes :

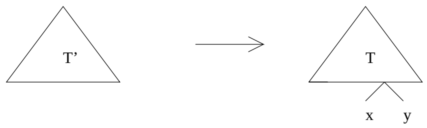
$$\begin{aligned} l_{T''}(a) &= l_T(x) \\ l_{T''}(b) &= l_T(y) \\ l_{T''}(x) &= l_T(a) \\ l_{T''}(y) &= l_T(b) \end{aligned}$$

On en déduit l'égalité :

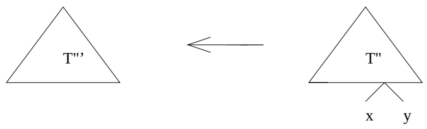
$$B(T) - B(T'') = [f(a) - f(x)][l_T(a) - l_T(x)] + [f(b) - f(y)][l_T(b) - l_T(y)]$$

$f(a) \geq f(x)$ 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 Σ', 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 Σ.



Soit T'' un arbre optimal pour Σ. 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 $B(T''') = B(T'') - f(x) - f(y)$.
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.

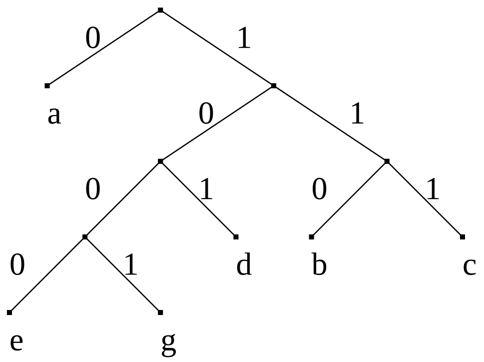
4 – On en déduit l'algorithme suivant pour le codage de Huffman :

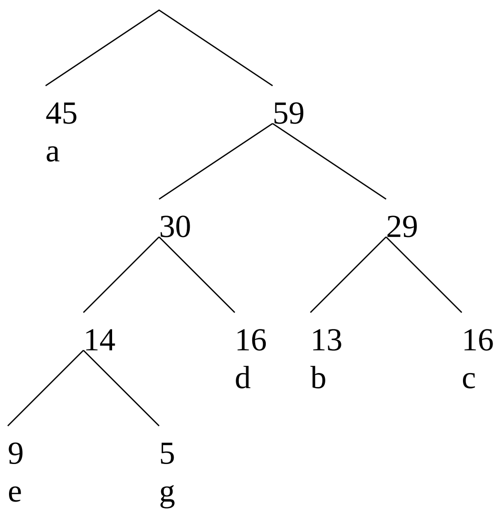
$Huffman(\Sigma, f)$
début
  $F \leftarrow$ tas_binaire(Σ, f) ;
  $n \leftarrow |\Sigma|$;
  pour i **de** *1* **à** $n-1$ **faire**
    $z \leftarrow$allouer_noeud() ;
    $x \leftarrow$extraire_min(F) ;
    $y \leftarrow$extraire_min(F) ;
    $z(gauche) \leftarrow x$;
    $z(droite) \leftarrow y$;
    $f(z) \leftarrow f(x) + f(y)$;
    Insérer$(F, z, f(z))$;
  Retourner extraire_min(F) ;
fin

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 élement $O(\log n)$
- 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 $f(a) = 45$, $f(b) = 13$, $f(c) = 12$, $f(d) = 16$, $f(e) = 9$, $f(g) = 5$:


0
1
a
0
1
0
1
0
1
0
1
d
b
c
e
g


45
a
59
30
29
14
16
d
13
b
16
c
9
e
5
g