IFT2125 automne 2016 - Devoir 2

Philippe Caron

11 octobre 2016

1 Récurrence

Soit la récurrence suivante :

$$t_n = \begin{cases} 3 & \text{si } n = 0, \\ 3 & \text{si } n = 1, \\ 17 & \text{si } n = 2, \\ at_{n-1} + bt_{n-2} + ct_{n-3} + (d + en)2^n & \text{si } n > 2, \end{cases}$$

dont la solution est :

$$t_n = (2n - 3)5^n + (6 - 2n^2)2^n \tag{1}$$

On cherche a, b, c, d, e.

Tout d'abord, en examinant la solution de la récurrence en (1), on peut d'emblée affirmer que les zéros du polynôme associé sont 2 et 5, et qu'ils sont de multiplicité 3 et 2 respectivement. Ce qui nous donne un polynôme de la forme suivante :

$$(r - 2)^3(r - 5)^2 \tag{2}$$

Ceci nous donne suffisamment d'information pour trouver les coefficiant du polynôme de gauche lorsqu'on isole les termes qui ne dépendent pas d'un t_n précédent * :

$$\begin{aligned} t_n &= at_{n-1} + bt_{n-2} + ct_{n-3} + (d + en)2^n \\ t_n - at_{n-1} - bt_{n-2} - ct_{n-3} &= (d + en)2^n \\ \Rightarrow (r^3 - ar^2 - br - c) \cdot (r - 2)^2 &\overset{(2)}{=} (r - 2)^3(r - 5)^2 \\ r^3 - ar^2 - br - c &= (r - 2)(r - 5)^2 \\ r^3 - ar^2 - br - c &= r^3 - 12r^2 + 45r - 50 \\ \Rightarrow a &= 12 \\ b &= -45 \\ c &= 50 \end{aligned}$$

Ensuite il nous suffit de résoudre un simple système 2 équations 2 inconnues en utilisants les prochaines valeurs de t_n calculées grâce à l'équation (1).

$$\begin{aligned} t_3 = 279 &= at_{n-1} + bt_{n-2} + ct_{n-3} + (d + en)2^n \\ &= 12 \cdot 17 - 45 \cdot 3 + 50 \cdot 3 + 8d + 24e \\ 60 &= 8d + 24e \\ 15 &= 2d + 6e \end{aligned} \tag{3}$$

$$\begin{aligned} t_4 = 2709 &= at_{n-1} + bt_{n-2} + ct_{n-3} + (d + en)2^n \\ &= 12 \cdot 279 - 45 \cdot 17 + 50 \cdot 3 + 16d + 64e \\ -24 &= 16d + 64e \\ -3 &= 2d + 8e \end{aligned} \tag{4}$$

∗. Sachant que l'annihilateur d'un terme de la forme $k^x P(y)$ est $(E - x)^{y+1}$

La solution au système formé des équations (3) et (4) est la suivante :

$$\begin{aligned} \Rightarrow d &= \frac{69}{2} \\ e &= -9 \end{aligned}$$

Ce qui termine la recherche des coefficients.

2 Dénominations

Soit les dénominations $1 = d_0, d_1, ..., d_k$ où $5d_{i-1} \leq d_i$. Il faut prouver que la procédure vorace trouvera toujours un résultat optimal.

Prenons un montant m, le cas dangereux est celui où il existe un multiple de petites dénominations (appelons la d_p) plus petit que le nombre de d_{p-1} qu'il faudra pour combler l'espace entre m et d_{p+1} et ou d_p et d_{p+1} sont premiers entre eux.

Construisons une monaie hypotéthique autour de ce cas. Commençons avec une pièce de 1\$ ($d_{p+1} = 100$). Nous cherchons le plus gros d_p possible, donc :

$$\max d_p \mid 5d_p \leq d_{p+1} \Rightarrow d_p = 20$$

or 20 divise 100. La plus petite pièce qui respecte les condition est donc celle de 19¢. Choisissons maintenant d_{p-1}. Afin de maximiser les chances d'échec, nous prendrons la pièce de 1¢.

Soit la dénomination suivante :

$$d_0 = 1$$

$$d_1 = 19$$

$$d_2 = 100$$

Vérifions que la dénomination ci-haute respecte les conditions initiales :

$$5 \cdot d_2 = 5 \leq d_1 = 19$$
$$5 \cdot d_1 = 95 \leq d_2 = 100$$

Échec de l'algorithme vorace. Si l'on soummet 1.14\$ à l'algorithme vorace, celui-ci retournera $1 \times 1\$ + 14 \times 1¢$, pour un total de 15 pièces. Or la solution optimale aurait été de remettre 6 pièces de 19¢(considérablement moins).

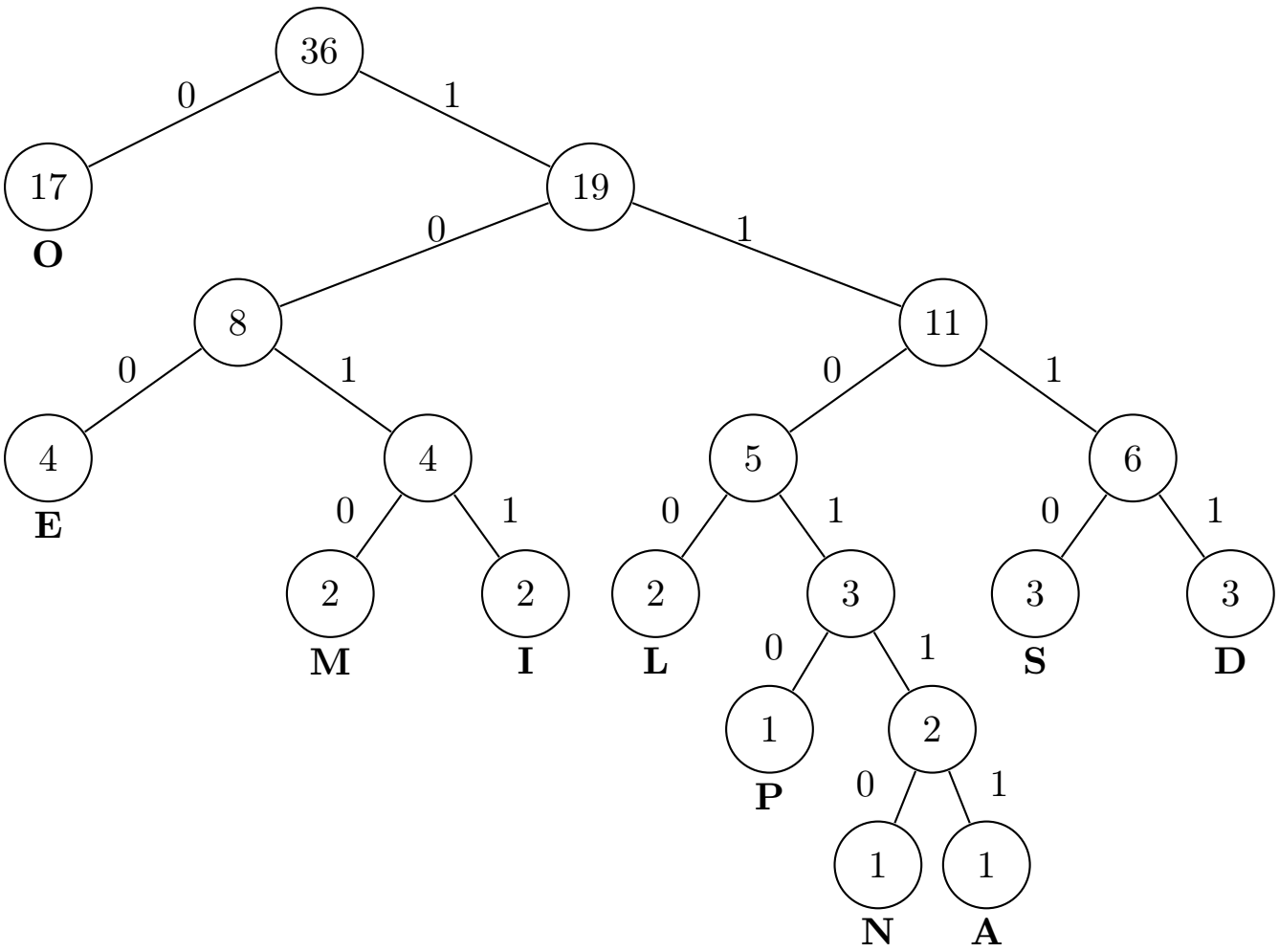
La preuve est donc faite qu'un système de dénominations qui respecte la condition initiale ne peut pas nécessairement être traité par un algorithme vorace.

3 Code de Huffman

Pour encoder la chaîne suivante :`lepessimismeooooooooodeooooooooodonald`, on fait un tableau des fréquence des lettres :

lettre	fréquence
o	17
e	4
s	3
d	3
l	2
i	2
m	2
p	1
n	1
a	1

À partir de ce tabeau on se crée un arbre :



Avec quoi on peut faire une table qui fait correspondre chaque lettre à son encodage de Huffman :

lettre	code	longueur
o	0	1
e	100	3
s	1110	4
d	1111	4
l	1100	4
i	1011	4
m	1010	4
p	11010	5
n	110110	6
a	110111	6

La longueur de Huffman du mot est donc :

$$17 \times 1 + 4 \times 3 + 3 \times 4 + 3 \times 4 + 2 \times 4 + 2 \times 4 + 2 \times 4 + 5 + 6 + 6 = 94$$

La suite la plus courte est évidemment la suite qui maximise le nombre de «o», comme par exemple la suite `anpmildseooooooooooooooooooooooooooo` (de longueur 67).

4 Algorithme

(a)

Le couple $(\{1, 2, ..., n\}, \{S \subseteq 1, 2, ..., n : \sum_{i \in S} \omega_i \leq \omega_0\})$ n'est pas un matroïde, car l'ensemble I ne respecte pas la propriété de l'échange.

Imaginons l'ensemble $A = \{i, x, y\} \subset \{1, 2, ..., n\} : \omega_x + \omega_y = \omega_i$, et l'ensemble $B = \{i, u, v, w\} \subset \{1, 2, ..., n\} : \omega_u + \omega_v + \omega_w = \omega_i$. Par définition, A et B sont éléments de I.

Preuve Selon le principe de l'échange, $[A \in I \text{ et } B \in I \text{ et } |A| < |B|] \Rightarrow (\exists a \in B \backslash A)[X \cup \{a\} \in I]$. Autrement dit, puisque A est plus court que B, il y a un élément a de B qui peut être ajouté à A de sorte que $A \cup \{a\} \in I$. Or c'est impossible puisque tout ajout non nul à A ferait augmenter le poids au dessus de ω_i. Ce qui signifie que l'échange n'est pas toujours possible et donc qu'on ne parle pas ici d'une matroïde.

Exemple $A = \{15, 7, 8\}$ et $B = \{15, 3, 4, 8\}$, peut importe l'élément de B choisi, $A \notin I$.

(b)

De **(a)** on peut déduire qu'il existe parfois plusieurs façons différentes d'arriver au même poids, et donc que la méthode vorace risque de ne pas être optimale dans tous les cas.

(c)

Voici les valeurs obtenues par l'algorithme :

tuple	Maximum de $\sum_{i=1}^{n} x_i \omega_i$ trouvé sur cet exemplaire
p1	14
p2	132
p3	158
p4	1496
p5	1799

(d)

Voici le code en python (commentaires enlevés) :

```
1   def swapsort(t):
2       l = list(t)
3       for x in range(0, len(l) - 1):
4           for y in range(1, len(l) - x):
5               if l[y - 1] > l[y]:
6                   l[y - 1], l[y] = l[y], l[y - 1]
7       return tuple(l)
8
9   def Subsetsum(p):
10      w0 = p[0]
11      wi = 0;
12      p = swapsort(p[1:])
13      p = p[::-1]
14      for x in range(len(p)):
15          while(w0 > wi + p[x]):
16              wi += p[x]
17      return wi
```