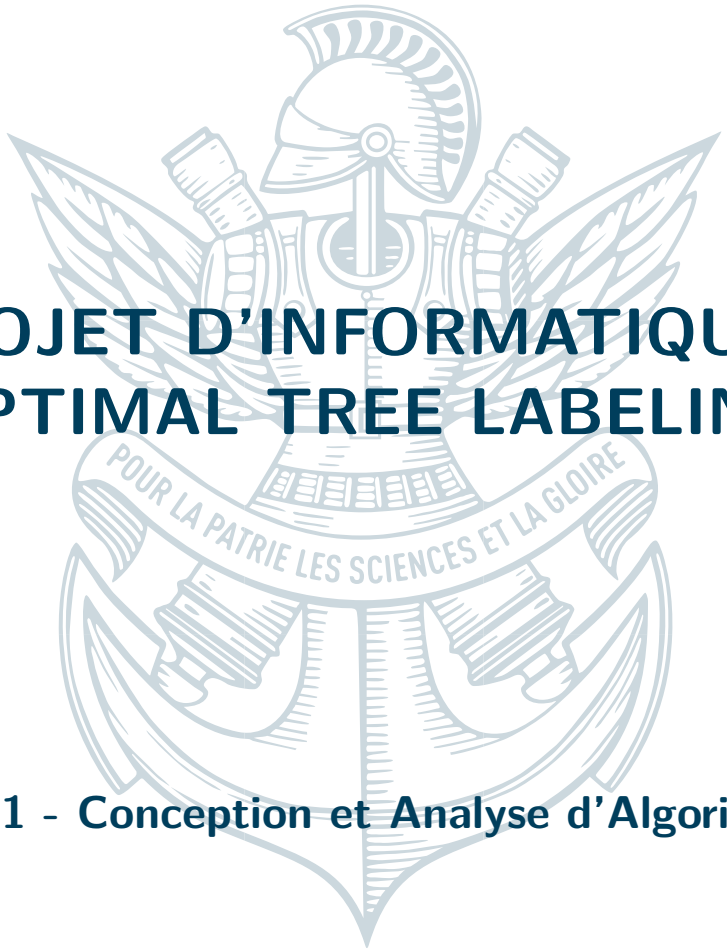

PROJET D'INFORMATIQUE - OPTIMAL TREE LABELING

INF421 - Conception et Analyse d'Algorithmes

Mercredi 6 Février 2019

Skandère SAHLI
Alban ZAMMIT

TABLE DES MATIÈRES

1 INTRODUCTION

Notre projet d'informatique porte sur la façon de légender les sommets d'un arbre afin de minimiser son poids total. Plus précisément, on considère un arbre dont seules les feuilles sont légendées par une partie de l'alphabet $\Omega = \{A, B, C, ..., Y, Z\}$. Le but de l'exercice est de légender les sommets intérieurs de l'arbre par des parties de Ω de sorte que le poids total de l'arbre soit minimal. Le poids total de l'arbre est la somme des poids des arrêtes de l'arbre, et le poids d'une arrête est la distance de *Hamming* entre les deux légendes des sommets qu'elle connecte. La Figure 1 donne un exemple de situation initiale.

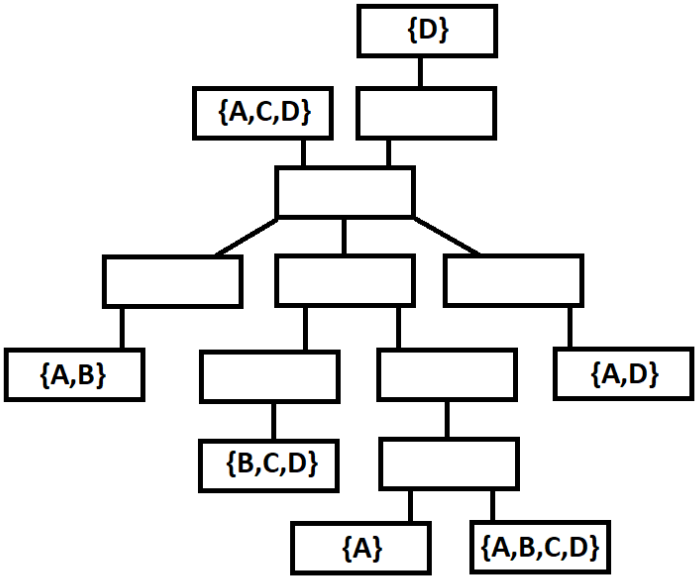

Figure 1 – Exemple de situation initiale

Pour résoudre ce problème, nous avons utilisé une méthode de **Programmation Dynamique**. Ce choix a été principalement motivé par deux raisons. D'une part, on nous demandait de ne retourner que la valeur du poids optimal de l'arbre, et non la manière de le légender pour obtenir ce poids, ce qui est caractéristique de la programmation dynamique. De plus, on s'aperçoit très vite que la méthode *greedy*, naturelle, qui consiste à remonter en partant des feuilles et à choisir à chaque fois une légende qui minimise le poids d'un arc en particulier, est mise en défaut sur des exemples très simples (cf. Figure 2)

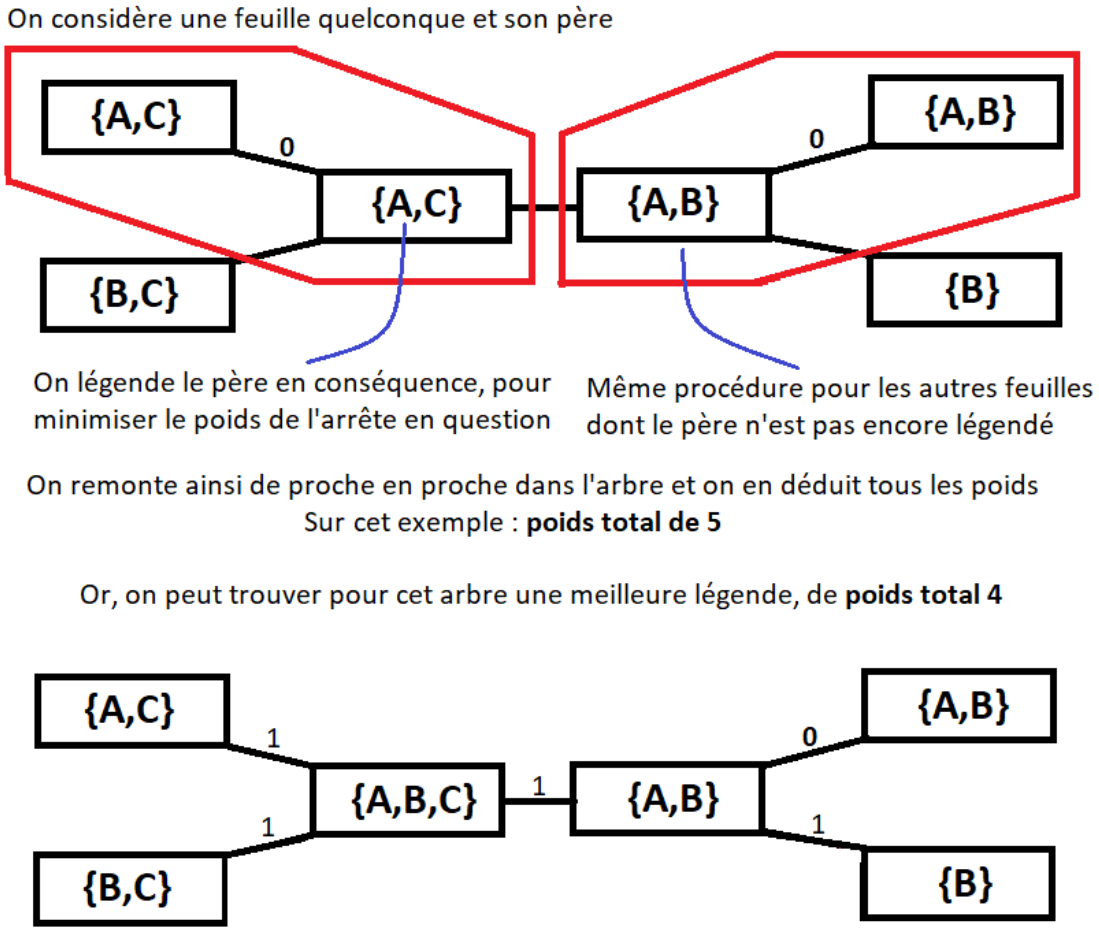

Figure 2 – Contre-exemple pour la méthode *greedy*

2 UNE SOLUTION PAR PROGRAMMATION DYNAMIQUE

2.1 Remarque préliminaire sur l'Indépendance des Lettres entre elles

La distance de Hamming, notée $H(.,.)$ dans la suite, possède une propriété très intéressante pour notre problème. En effet, si l'on a deux ensembles ω_1, $\omega_2 \subset \Omega = \{A, B, C, ..., Y, Z\}$, que l'on écrit $\omega_1 = (x_1 x_2 ... x_{|\Omega|})$ et $\omega_2 = (y_1, y_2, ..., y_{|\Omega|})$, c'est-à-dire sous la forme d'une séquence binaire, où x_k vaut 1 si la k^{eme} lettre de Ω est dans ω_1, 0 sinon, alors on a :

$$H(\omega_1, \omega_2) = \sum_{k=1}^{|\Omega|} \mathbf{1}_{x_k \neq y_k}$$

Nous pouvons donc diviser le problème en $|\Omega|$ problèmes indépendants, un pour chaque lettre de l'alphabet, et faire à la fin la somme des poids obtenus pour chaque arbre dont les feuilles sont labellisées simplement par $l \in \Omega$ ou $\emptyset$.

2.2 Idée initiale - Formule de récurrence

On dispose maintenant de $|\Omega|$ arbres (un pour chaque lettre $l \in \Omega$) à N sommets, numérotés de 1 à N. On prend ensuite n'importe quel sommet qui n'est pas une feuille, et on le définit comme racine de l'arbre. Soit alors a le numéro d'un sommet quelconque qui n'est pas une feuille. On note alors a_1, a_2, ..., a_k les numéros des k enfants directs de a. La Figure 3 résume ces notations sur un exemple.

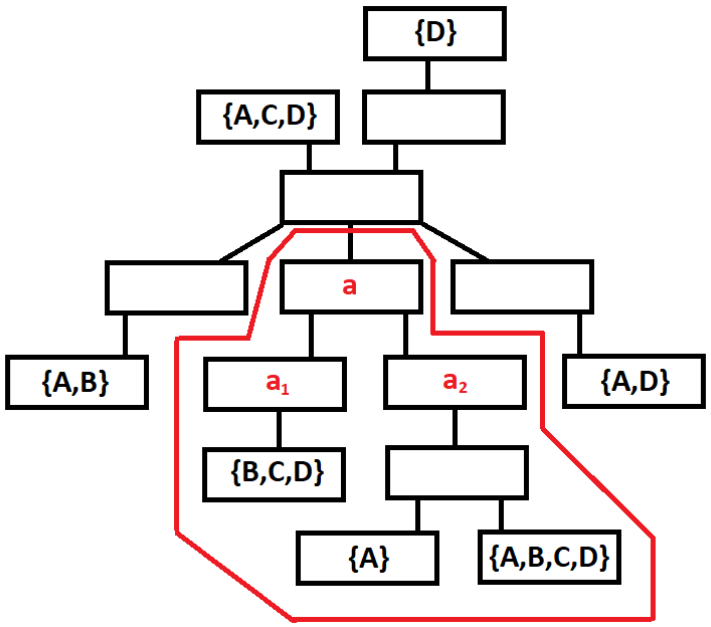

Figure 3 – Exemple d'utilisation des notations pour $N = 15$ sommets. Le sous-arbre engendré par a plus l'arrête qui le lie à son père est encadré en rouge. Les enfants de a sont a_1 et a_2

On note alors $C_a(l, 0)$ (resp. $C_a(l, 1)$) le coût minimal du sous arbre engendré par a, plus l'arc qui le lie à son père, dans l'arbre concernant la lettre l, lorsque l'on légende le père de a par $\emptyset$ (resp. l). On a alors la formule de récurrence, en notant b le booléen 1 ou 0 :

$$C_a(l, b) = \sum_{i=1}^{k} min\left(C_{a_i}(l, b); C_{a_i}(l, 1-b) + 1\right)$$

Lorsque a est une feuille, la valeur de $C_a(l, b)$ est alors très simple : $C_a(l, b) = 0$ si la légende de la feuille a est la même que b (dans ce cas, le père est la fille ont la même légende et donc l'arc a un poids nul au sens de la distance de Hamming), 1 sinon.

Cette formule est assez intuitive (on rajoute 1 à la distance si le père et le fils ne sont pas légendé de la même manière, 0 sinon) nous a permis de coder une fonction récursive `opimalLetter` qui résout le problème très efficacement.

2.3 Initialisation et Terminaison

Il s'agit simplement de prendre le premier sommet qui n'est pas une feuille, le définir comme racine de l'arbre et utiliser la formule de récurrence ci-dessus. Pour les besoins de la chose, on ajoute un père artificiel à la racine, qui a toujours la même légende que la racine, pour que la formule de récurrence reste valable.

On conserve les valeur prises par $C_a(l, b)$ dans une matrice de mémoïsation à trois dimensions, qu'on initialise avec -1 partout sauf pour les feuilles de l'arbre, pour lesquelles on met 0, pour indiquer que la valeur est déjà définie.

Dans le cas pathologique où notre arbre ne contiendrait que des feuilles (arbre à un ou deux sommets), notre algorithme reste vrai puisqu'il calcule alors simplement la distance de Hamming entre ces feuilles.

2.4 Implémentation en Java

L'arbre étant un graphe très creux, nous avons opté pour une représentation de l'arbre par des listes d'adjacences. Nous avons codé en `Java` au coeur de trois classes : une classe `OptimalTreeLabel` (un champ `noeuds`, matrice $N \times |\Omega|$ d'`int` qui indique pour l'arbre concernant la lettre l, si le sommet k la contient où non ; une table de memoïsation `Memoisation`, qui correspond à la fonction $C_a(l, b)$; une `HashMap<Integer,List<Integer>>` représentant les listes d'adjacence du graphe ; un champ `alphabet` liste de `String` qui contient les lettres de l'alphabet ; un champ `bijectionAlphabet` qui permet de réaliser en temps constant l'aller-retour en chaque lettre et son numéro dans l'alphabet), et deux classes auxiliaires aux noms assez transparents : `CoupleIntString` et `CoupleOfInt`.

La table de mémoïsation de la classe `OptimalTreeLabel` est remplie grâce à la fonction récursive `optimalLetter` qui applique la formule de récurrence pour $C_a(l, b)$. L'exécution du `main` est très rapide pour tous les tests fournis (moins d'une seconde à chaque fois).

3 ANALYSE

L'étape cinétiquement déterminante de l'algorithme est le remplissage de la table de mémoïsation. Pour chacune des $|\Omega|$ lettres de l'alphabet, on parcourt deux fois chaque sommet (selon qu'on le légende ou non pour la lettre en question) et on calcule un minimum en temps constant.

La complexité $T(N)$ de l'algorithme pour un arbre à N sommets est donc :

$$\boxed{T(N) = O(|\Omega|.N)}$$