Rapport AAC TP1

Programmes Louis Chambenoit, Rédaction Baptiste Vallet

12 / 02 /2023

Codage de Huffman

Principe

Le codage de Huffman est un algorithme de compression de données. Le concept est de générer un arbre binaire qui associe à chaque caractère un noeud. Le chemin de ce noeud depuis la racine détermine la suite binaire représentant le caractère.

L'emplacement du noeud est défini en fonction de sa récurrence dans le texte à coder. L'arbre n'est pas parfait, les noeuds possèdent deux fils : un autre noeud et une feuille qui associe un caractère et son nombre d'apparitions.

Ensuite pour le codage du texte, il suffit de le retranscrire mais avec les nombres binaires associés à chaque caractère. La chaîne de carctère est ainsi bien plus compacte. De plus elle est facilement décodable car il n'y a besoin que de l'arbre et du texte codé.

Attention, il s'agit bien d'un codage et non d'un chiffrement. Sur un grand texte, on peut déterminer un arbre optimal pour chaque langue simplement basé sur la répartition statistique des lettres.

Structure de Données

Dans src/tree.hpp

Node

En termes de structure de données, les feuilles et les noeuds ne sont pas dissociés, il n'y a que la classe *Node*. Chaque noeud possède un caractère et son nombre d'apparitions, un fils gauche et un fils droit.

La classe ne propose que des méthodes getter,setter, des constructeurs et une surcharge de l'opérateur inférieur qui teste le nombre d'apparitions des deux noeuds.

Tree

La classe *Tree* permet de différentier un arbre d'un simple noeud bien qu'il ne soit constitué que d'un pointeur vers un noeud.

La classe ne propose que des constructeurs, un getter sur le noeud à la racine et deux méthodes d'affichage de l'arbre.

Algorithmes

Dans /src/huffman.cpp

Seuls les algorithmes pour la génération de l'arbre ont été implémentés.

huffman_init()

La fonction *huffman_init()* crée une liste de noeuds à partir d'une chaîne de caractères. C'est un moyen simple de stocker de nouveaux noeuds et de mettre à jour leur nombre d'apparition.

huffman_tree()

Ensuite, cette liste de noeuds est envoyée à la fonction *huffman_tree()*. Là, elle est transformée en *priority queue* pour pouvoir facilement accéder aux noeuds triés par le nombre d'apparition des caractères, grace à la surcharge de l'opérateur inférieur.

Tant que cette queue n'est pas vide, on prend les deux premiers noeuds, les plus faibles. Ils sont associés en fils d'un nouveau noeud, puis ce nouveau noeud est inséré dans la *priority queue* avec un nombre d'apparition valant la somme de ses deux fils. Cela correspond à la future place dans l'arborescence.

L'arbre retourné correspond au dernier noeud de la *priority queue*, donc le père de tous les autres à la racine.