DECOOPMAN Clément
DAMIENS Lucas

Codage de Huffman

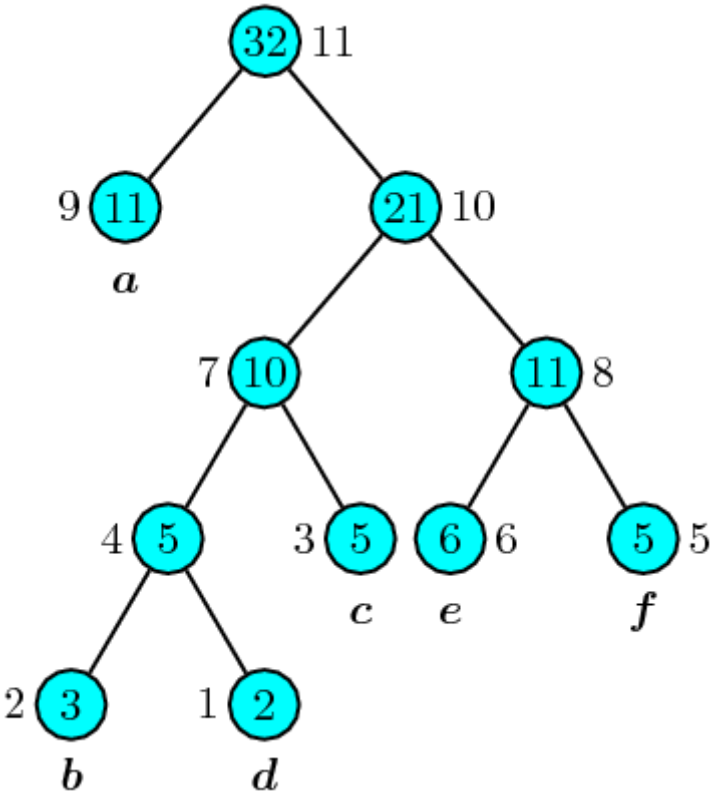

Table des matières

Introduction 2
Présentation du projet 2
Documentations 2
Analyse 3
Classes utilisées 3
Choix techniques 4

Introduction

Présentation du projet

Le projet consiste à réaliser une compression du contenu d'un fichier texte en utilisant le codage de Huffman. Le contenu du texte constituera un arbre de Huffman.
Le programme doit être utilisable en ligne de commande et accepter au moins un paramètre pour invoquer la compression et le nom du fichier de sortie, et un paramètre pour la décompression.

Documentations

- **Codage de Huffman, maps :**

http://www.cplusplus.com/

http://cermics.enpc.fr/polys/info1/main/node76.html

http://www-igm.univ-mlv.fr/~lecroq/cours/huff.pdf

http://tcharles.developpez.com/Huffman/codecpp.php

Analyse

Classes utilisées

```
class Node {
  private:
    static int NbNode;

  public:
    Node() {NbNode++;}
    ~Node() {delete [] right; delete [] left;}
    Node * left;
    Node * right;
    char nodeValue;
    int frequence;
    string code;
    inline const char& getValue() const {return nodeValue;}
    inline int getNbNode() {return NbNode;}

    friend class Tree;
};
```

```
class Tree {

  private:
    Node *root;
    void Insert(Node *);
    void Scan(Node *);
    inline Node *Root() const {return root;}
    Node *Search(const char) const;

  public:
      Tree();
    ~Tree();
    Node* Create(const char, const int);
    void Delete(const char);
    void Viewing();
    inline void NbrNodes() {cout << Node::NbNode;}
};
```

Ci-dessus les classes Node et Tree correspondant respectivement aux définitions de nos nœuds et de nos arbres.

Suivant cette implantation, un nœud possède :

- un code Huffman utile à la compression et à la décompression
- un caractère et sa fréquence
- un pointeur sur le nœud de gauche et un autre sur le nœud de droite

Et pour un arbre :

- un pointeur sur un nœud correspondant à la racine

Un arbre est donc constitué

Choix techniques

- Fréquences des caractères :

Avant toute chose, il faut pouvoir stocker les différents caractères et leur fréquence dans le texte.
Pour cela, on utilise une unordered_map afin de pouvoir stocker sous forme de « clé-valeur » et ainsi d'associer une fréquence à un caractère. De plus, le parcours des char dans le texte est facilité et plus rapide grâce à l'itérateur de la map.

- Tas de nœuds :

Une fois le stockage effectué, il suffit de créer un vecteur de références sur arbres (uniquement la racine) de taille égale au nombre de lettres stockées. Les arbres « racines » sont crées avec la fonction *Create()* qui demande en paramètre un char et sa fréquence.

Les vecteurs sont beaucoup plus performants que les simples tableaux pour l'accès à un élément.

On a choisi de stocker les références sur les racines car il est plus facile de les retrouver ainsi : il n'y a pas forcément besoin d'effectuer un parcours entier dans l'arbre pour les retrouver au moment d'encoder avec Huffman.

- Création de l'arbre et encodage :

La création de l'arbre d'Huffman s'effectue normalement selon le codage de Huffman : on rassemble par deux les racines ayant les plus faibles fréquences en seul arbre (dont le nœud racine a sa fréquence est égale à la somme de celle des deux autres racines), en mettant le plus grand des deux à droite. Pour cela, on utilise la fonction *Scan() et on* fait pointer les racines sur les nœuds de gauche ou de droite.

L'encodage peut ensuite se dérouler de manière simple : on peut éviter de parcourir tout l'arbre grâce aux références stockées. Avec la fonction *Search()* acceptant en paramètre une lettre du texte, on peut rechercher la référence du nœud qui lui correspond et ainsi affecter à son membre « code » 0 ou 1.