Roberto Medina
3360906

Algorithmique Avancée:
Arbre de Huffman Adaptatif

Table des matières

Choix d'Implémentation

Langage de programmation

Le projet a été développé en **C++** et compilé avec Clang. Ce langage me semble pertinent à cause des différentes structures de données qu'on peut créer et qu'on peut utilisé grâce à la librairie standard. C'est à dire que si j'avais besoin d'une structure de tableau dynamique j'avais pas à recoder tout un module.

De même, je voulais bénéficier des pointeurs, la dynamique des pointeurs étant quelque chose d'assez simple me permettait de faire des manipulations dans l'arbre assez facilement et très rapidement. Au lieu d'instancier des objets, j'alloue une structure de Noeud pour créer mon arbre et je le fais que quand j'en ai besoin. Sinon c'est la même structure de Noeud qui est manipulée, je fais pas de recopie et destruction ce qui serait très couteux.

Finalement le fait de devoir manipuler des bits a été un autre facteur pour choisir ce langage de programmation. Cette manipulation a été faite par un module que j'ai crée. Vers la fin du projet j'ai trouvé une bibliothèque (BitMagic) qu'aurait pu être utilisé mais le module était déjà presque terminé donc j'ai décidé de le conserver.

Le compilateur Clang me permettait d'avancer plus vite sur des erreurs grâce à l'affichage coloré. Normalement il y a pas de différences quand le projet est compilé avec GCC.

Modules du projet

Le projet se décompose en trois modules:

Le **module Arbre** qui fait la gestion de l'Arbre de Huffman, on va trouver les algorithmes de Modification, de Traitement et d'échange pour les Noeuds. Le module possède les primitives sur les arbres binaires avec les traitements spécifiques pour Huffman. Les fichiers correspondants sont: `arbre.cpp` et `arbre.hpp`

Le **module Codage**, `codage.cpp` et `codage.hpp`, gère l'écriture avec des bits sur de octets.

Le **module Compression**, utilise les deux autres modules pour faire la compression et la décompression sur un fichier. Correspond aux fichiers `compression.cpp` et `compression.hpp`

Chaque module possède des accesseurs et des mutateurs pour leur structures correspondantes.

L'idée était de créer un autre module appellé Symbole qui aurait permis de faire un Arbre de Huffman générique contenant des Symboles sur les feuilles et pas de caractères.

Pour tester au fur et à mesure les fonctions des modules il y a des tests de regression qui ont été crées. On peut les compiler avec la directive du Makefile, **huffman-regression**.

Le programme principal peut être compilé avec la directive **huffman-dynamique** ou avec un simple make.

La syntaxe du programme est la suivante:

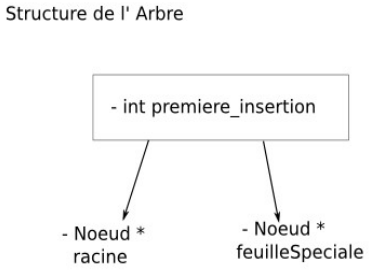
```
./huffman-dynamique -[cd] <fichier d'entrée> <fichier de sortie>
```

Structures de données

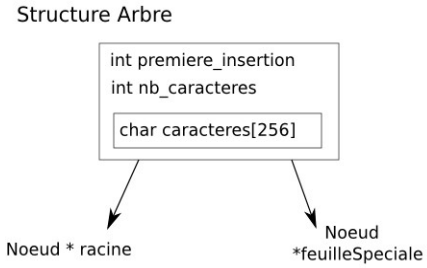
Comme expliqué au par avant, le choix du langage m'a permit de créer des structures de données plus ou moins complexes pour me permettre de gérer l'Arbre de Huffman Adaptatif.

Entre la présentation du projet et l'édition de ce rapport il y a eu quelques changements dans les structures de données. Voici quelques schémas pour mieux comprendre comment fonctionnent ces structures :

La structure d'Arbre pour la présentation :



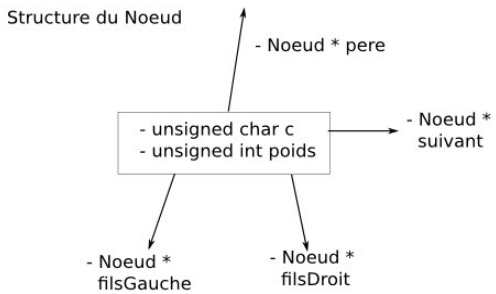
La structure d'Arbre après la présentation :



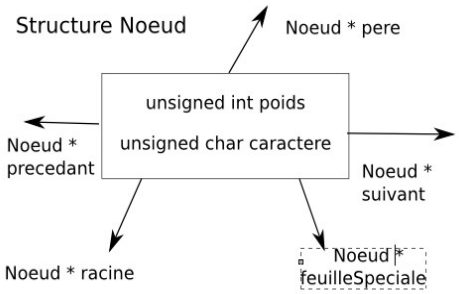
Cette nouvelle structure permet d'améliorer le test pour savoir si un caractère est présent dans l'arbre ou pas. La recherche se faisait avec deux appels recursifs avant, maintenant il faut juste parcourir le tableau de caractères pour savoir si le caractère est présent ou pas. Retrouver le Noeud ce fait toujours de manière recursive.

Les pointeurs vont sur la racine de l'arbre et sur la feuille spéciale. Le deuxième pointeur est nécessaire pour ne pas devoir faire une recherche recursive jusqu'à trouver la feuille spéciale.

La structure du Noeud pour la présentation :

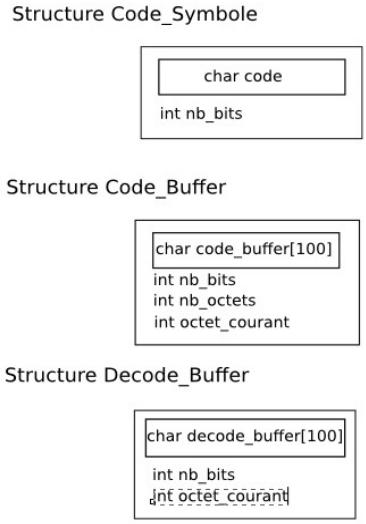


La structure du Noeud après la présentation :



Comparé à la version presentée on a un nouveau pointeur sur le prédéceseur du Noeud pour l'ordre GDBH. Ceci est utile quand on doit faire un échange avec beaucoup de décalage entre nœuds.

Pour le module qui fait le codage sur des bits d'autres structures de données ont été introduites :



Réponses aux questions

Compléxité

Déroulement des algorithmes