L3-INFO Algorithmique

Ifrel Rinel MAKOUNDIKA
Arthur D'HERIN
Groupe: **TD 4**

22 nov. 2024

Compte Rendu sur l'Algorithme de Huffman: APNÉE 3

1. A propos

Ce projet porte sur l'implémentation de l'algorithme de compression de Huffman. L'objectif principal est de coder un texte de manière efficace en utilisant un codage binaire dont la longueur des codes est inversement proportionnelle à la fréquence d'apparition des caractères. Nous avons été amenés à compléter deux programmes, `huff_encode.c` pour l'encodage et `huff_decode.c` pour le décodage, et à tester leur efficacité.

2. Implémentation

On implémente l'algorithme de Huffman en suivant l'algorithme donné en cours.
Il s'agit d'abord de parcourir le fichier à encoder une première fois pour créer son arbre de Huffman (à partir de l'occurrence des caractères et d'une file à priorité), on parcours ensuite le fichier une deuxième fois pour faire correspondre chaque caractère avec son code défini par l'arbre (et table).
Pour décoder, on lit d'abord l'arbre présent au début du fichier encodé, puis on lit les bits 1 à 1 pour nous déplacer dans l'arbre, retrouver les symboles au niveau des feuilles, et écrire ces symboles dans le fichier de sortie.

- **Le module `arbrebin.c` et `arbrebin.h`** a été utilisé pour la gestion de l'arbre binaire de Huffman. Nous avons utilisé les fonctions pour créer des nœuds et manipuler l'arbre lors de sa construction.
- **Le module `fap.c` et `fap.h`** a fourni une file à priorités (FAP), qui est utilisée pour gérer les nœuds de l'arbre pendant sa construction. Les nœuds sont insérés dans la FAP en fonction de leur fréquence, et les deux nœuds ayant les fréquences les plus basses sont extraits pour créer un nouveau nœud jusqu'à ce que l'arbre soit entièrement construit.
- **Le module `bfile.c` et `bfile.h`** a permis de gérer la lecture et l'écriture bit à bit des fichiers, en particulier pour l'écriture des bits codés dans le fichier compressé. La fonction `bitwrite` a été utilisée pour écrire les bits du code Huffman dans le fichier de sortie.

- **Le module `arbrebin.h`** a été utilisé pour manipuler l'arbre binaire de Huffman. La structure d'arbre est utilisée pour parcourir les nœuds en fonction des bits lus dans le fichier encodé. Chaque nœud de l'arbre contient un caractère ou un symbole (dans le cas d'un nœud feuille) et est connecté à ses enfants gauche et droit, ce qui permet de naviguer dans l'arbre lors du décodage des bits. La fonction `AfficherArbre` est utilisée pour visualiser l'arbre avant le processus de décodage.
- **Le module `bfile.h`** a été essentiel pour la gestion des fichiers bit à bit. Il permet de lire les bits du fichier compressé à l'aide de la fonction `bitread` et de gérer l'entrée et la sortie de manière efficace. Cela a permis de lire les séquences de bits encodées dans le fichier et de les décoder en parcourant l'arbre de Huffman.

3. Tests

Les tests avec `candide.txt` et `gargantua.txt` sont concluants, on retrouve le contenu original, avec `gargantua`, on passe d'un fichier de taille *264,2 KiB* à un fichier de taille *151,0 KiB*, soit une diminution d'à peu près 57%. Avec `candide`, on passe de *196,7 KiB* à *112,5Kib*, une diminution là aussi d'à peu près 57% aussi.

On a aussi expérimenté la compression sur de très grands fichiers générés aléatoirement (`tr -dc a-zA-Z < /dev/urandom | head -c "nombre_de_caractères" | tee random.txt`). On remarque que quand on cherche à compresser un fichier, quelle que soit sa taille, si les fréquences d'apparitions des différents caractères sont trop similaires, on obtient un arbre de Huffman trop équilibré et donc aucun gain suffisamment intéressant au niveau de la compression (car on ne bénéficie pas de la taille de codage réduite de caractères qui apparaissent plus souvent. Aussi, quand on compresse un fichier de petite taille, on risque d'augmenter sa taille en le "compressant"

Conclusion sur la correction et l'efficacité du programme:

Correction du programme

Le programme fonctionne correctement dans l'ensemble. Il est capable de :

- Lire un fichier texte et calculer la fréquence des caractères (via la fonction `ConstruireTableOcc`).
- Construire un arbre de Huffman à partir de la table d'occurrences en utilisant la fonction `InitHuffman` et `ConstruireArbre`.
- Générer un code Huffman pour chaque caractère dans le fichier, puis encoder le texte à l'aide de ce code.
- Sauvegarder l'arbre de Huffman et le fichier encodé dans un fichier de sortie à l'aide des fonctions `Encoder` et `EcrireArbre`.

Nous avons la garanti que algorithmes s'arrêtent parce que l'arbre de Huffman a un nombre fixe de Noeud et pareil pour le Fichier en lecture qui n'est pas infini. Autrement dit, le nombre de caractères et de noeud à traiter diminuent à chaque itération

De surcroît, les tests montrent que le programme effectue les étapes d'encodage et de décodage sans erreur. L'arbre de Huffman est bien construit (cf. image de l'arbre), et la table des codes est correctement générée pour chaque caractère et la traduction est parfaitement effectuée sans différence entre le texte original avant encodage et le fichier résultat après décodage: ***On a donc bien le résultat attendu***.

Efficacité du programme

L'efficacité de l'algorithme de Huffman a été observée à travers plusieurs tests de compression. Les fichiers contenant des structures régulières, comme des textes littéraires, ont montré des **réductions de taille significative, jusqu'à 30% de réduction dans certains cas**.

En revanche, pour des fichiers contenant peu de répétitions (par exemple, des fichiers binaires ou aléatoires), la compression n'a pas montré une grande efficacité, et parfois la taille du fichier compressé était même plus grande que celle du fichier original.

Le programme est également capable de traiter des fichiers de taille variable, et la complexité de l'algorithme reste raisonnable même pour des textes assez longs. Les étapes de construction de l'arbre et de génération des codes se déroulent en temps linéaire par rapport à la taille du texte, ce qui est satisfaisant pour les fichiers de taille courante.

Conclusion finale

Le programme d'encodage Huffman est fonctionnel et correct. Il parvient à réduire la taille des fichiers de manière significative pour des textes avec des structures répétitives. Toutefois, pour des données aléatoires ou déjà compressées, la performance peut être moins impressionnante. En résumé, l'algorithme est bien adapté pour la compression de texte en langage naturel mais moins efficace pour des fichiers binaires ou déjà optimisés.

Le programme respecte les spécifications données et a permis de mettre en pratique la théorie de l'algorithme de Huffman, tout en montrant des résultats concrets en termes de compression.

Arbre de Huffman
Pour gargantua :