Algorithmique Avancée

Implémentation de l'algorithme de compression grâce à l'Abre de Huffman Adaptatif

Roberto Medina

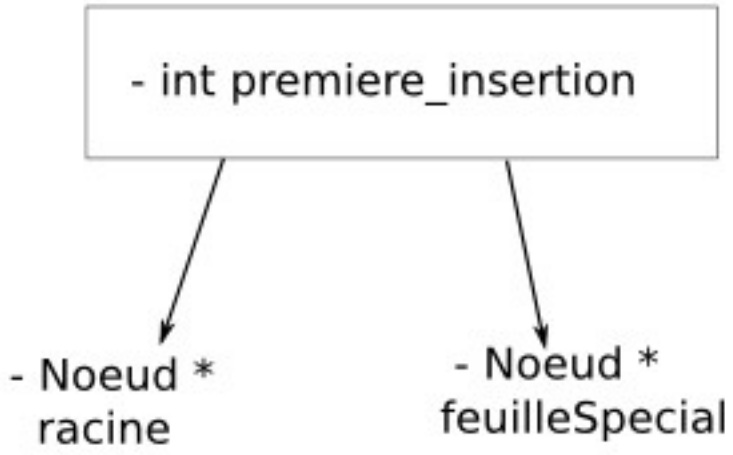
10 Décembre 2013

Caractéristiques du projet

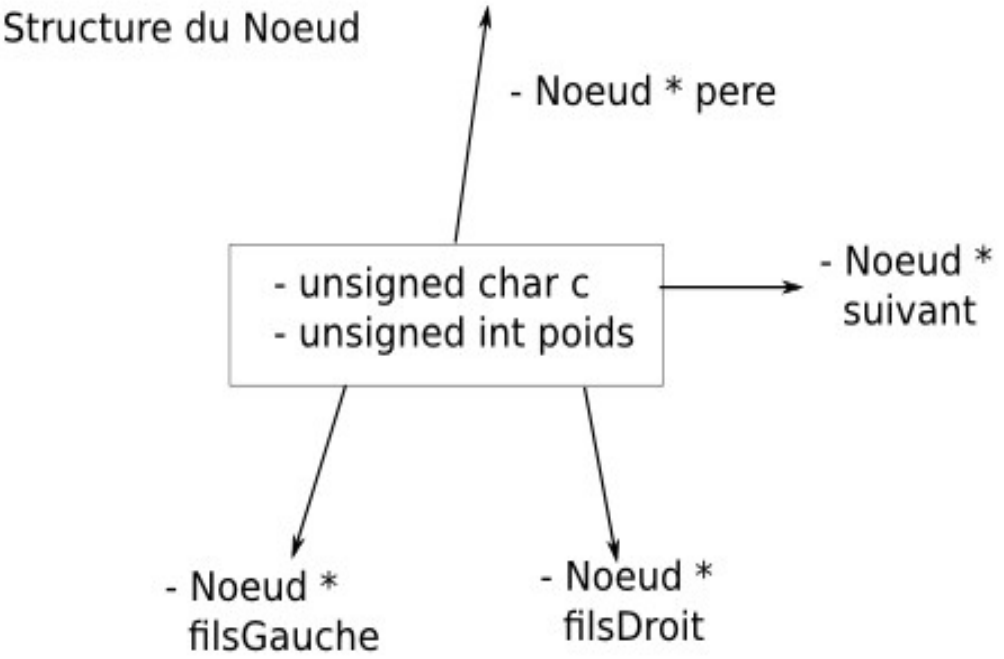
- Projet developpé en C++
  - Simplicité pour l'écriture des bits
  - Dynamique des pointeurs
  - Utilisation de structures déjà définies
- Différents modules pour les différentes structures de données
  - Module Arbre : pour la gestion de l'Arbre de Huffman.
  - Module Codage: pour l'écriture de bits sur les octets.
  - Module Compression: qui fait la compression et la decompression sur les fichiers.
  - Quelques tests de regressions qui ont été faits pour pouvoir suivre le développement du projet.

Représentation des structures

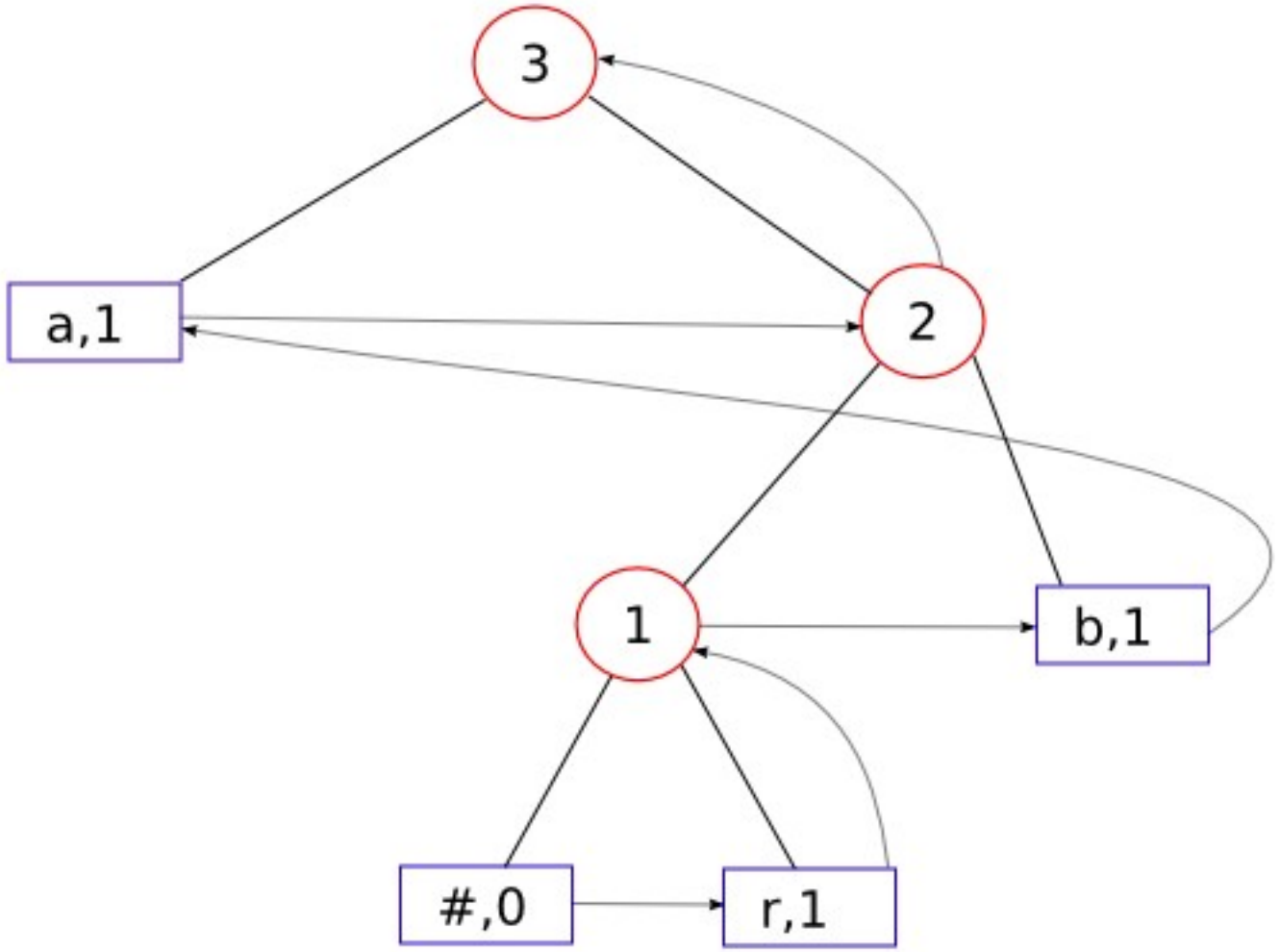
Structure de l' Arbre



Structure du Noeud



Exemple d'un arbre avec pointeurs



Complexité

- n : taille de l'alphabet
- Echanger les noeuds : O(1) c'est le changement des pointeurs
- Traitement :
  - Remplit un tableau avec le chemin jusqu'à la racine => au pire hauteur de l'arbre : *H*.
    - *H – 1* compraisons sur le tableau. Si le chemin est incrémentable on ajoute 1 sur chaque noeud => O(1)
    - Si le chemin n'est pas incrémentable on échange les noeuds mais on fait l'appel récursif sur le traitement => ???
- Modification :
  - On rajoute sur l'arbre les deux nouveaux noeuds et on met à jour les pointeurs => O(1)
  - Puis on rajoute la compléxité du traitement en fonction du cas (si incrémentable ou pas).

Compression et decompression

- Pas de parcours préalable, si c'est un gros fichier ça deviendrait très lourd. Au début on utilise le codage du caractère en ASCII.
- On compresse sur des octets et on écrit des bits.
- Trois structures qui gèrent les octets:
  - Code_Symbole : contient le code d'un symbole
  - Code_Buffer : contient les octets avec le texte comprssé
  - Decode_buffer : contient les octets avec le texte décompressé.