Compression huffman:

Le premier choix important fut le choix de la structure de donné permettant de créer un arbre pratique . La liste était un choix tout trouvé car elle permetrait une flexibilité du nombre de donné contenu en son sein mais aussi un accés pratique à ses éléments une fois déplié en arbre . En effet , les liens de la liste si nous les faisons subsister permettent un parcour en largeur de l'arbre .

Le deuxième choix important fut le choix des donnés contenu dans la cellule (maillon de l'arbre) , pour n'exclure aucune langue nous avons opté pour un int . Il est certes plus lourd mais il fait la taille d'un mot en machine donc les operations sont facilement optimisables (à nuancer tout de même en fonction de l'architecture de la machine concerné) .De plus , il est utilisé pour stocker le nombre de feuilles des nodes , ce qui rend la profondeur casi infini et donc le nombre de caractère différents maximum de 4,3 M
(à nuancer sur la version présenté du projet car les "bitwise operator " n'étant pas sécurisés il se pourrait qu'il y est des bugs une fois la profondeur supérieure à 32)

Le troisième choix fut comment strucuturer le corps d'une cellule dans l'entête du fichier . En marquant successivement le nombre des feuilles des parents de la feuille considéré l'accés à cette dernière l'ors de la décompression est très rapide . Bien qu'ils représentent une (faible) perte du potentiel de compression ils permettent d'accéder rapidement au binaire lu l'ors de la decompression ce qui dans des fichiers avec énormément de char peut être utile .

Le code ici présenté est une version de démonstration car elle n'est pas réellement une compression mais juste une vision en "clair" du mécanisme . Ce projet devait s'éffectuer en plusieurs temps :
-le premier est la création de cette version de test ,destiné à présenter le projet (dans le cadre d'une utilisation normale et peu poussé)mais aussi débugger facilement la seconde version .Son autre but est d'être plus performante en terme de temps qu'en terme " d'espace ".Son inconvénient est qu'il ne fait que très peu cas des opérations sur les bits car les cas échéant n'arrive que très rarement . Cela dit grâce au code de la seconde version ce rare défaut peut être facilement corriger .

-le second est une réelle implémentation de la compression : en limitant au maximum la taille du fichier une fois compressé (sytème de sufixe ,de concaténation d'int ,...).La version 2 porte son intérêt sur la place occupé par le fichier compressé et sur une gestion optimal des bits de manière sécurisé .Contrairement à la version 1 le cas ou la pronfondeur venait à dépasser 32 est parfaitement gérer sans perte .

-et le troisième est un mixe des 2 avec les avantages de la version test comme faciliter l'accés l'ors de la decompression et les avantages de la version 2 : la sécurité et le gain de place .En effet , la compression de la version 1 marche car notre IDE cast implicitement les int en long int , en long long int ... si nous dépassons 32 bits . Le

fait que la profondeur dépasse 32 n'est pas vraiment prévue par le code ainsi des problèmes pourraient survenir suivant la machine employé.

Ainsi il est nécessaire de croiser les 2 versions qui sont complémentaire :
Le système de sufixe est très performant pour économiser de la place mais il est très couteux en terme performance c'est pour cela que nous comptions y intégrer l'accés rapide des feuilles l'ors de la decompression (de la version 1) qui est un peu couteux en donnés mais qui fait gagner beaucoup de temps .

Le projet final n'ayant pas abouti à temps nous n'avons qu'une compression didactique à présenter et une ébauche de la compression dans son ensemble .Mais nous sommes à même de répondre aux potentielles questions sur ces 2 versions . Leur unicité symboliser par le troisième projet n'ayant pas était commencé nous ne pouvons que spéculer sur sa future implémentation .