Master 2 – Ingénierie Informatique (Parcours LP, SRI, LC)

Formats de Documents et Compression - Corrigé Exam. 17-12-2009

Notes manuscrites et sujets de TD autorisés. Livres non autorisés.
La rigueur des raisonnements, la clarté des explications, mais aussi la qualité de la présentation influera sur la note. Deux ou trois lignes de texte ***au maximum*** *pour les réponses explicatives.*

Problème 1

Partie I

Le mot `barbamaman` *a été choisi puisque d'autres petits exemples classiques tels que :* `mississippi`, `abracadabra`, `barbapapa` *ou* `barbamama`, *avaient soit les défaut de se compresser mieux que leur BTW (exemples trop petits ou trop bien faits), soit que la BWT regroupait* **toutes** *les occurrences d'une même lettre, ce qui n'est pas vrai en général (exemples fourvoyants).*

1. Simuler la méthode *Move-to-Front* (MTF) vue en cours sur la chaîne $s =$"`barbamaman`" et calculer la valeur moyenne des codes émis, en supposant que le premier élément de la liste est toujours codé par 0 et que la liste initiale est $[a, b, m, n, r]$.
   On obtient 1142231114. *Moyenne 2.*
2. A partir de la chaine s, écrire la chaîne s' obtenue en appliquant la procédure suivante :
   – Calculer tous les mots obtenus en conjuguant circulairement le mot s (les deux premiers : `nbarbamama`, `anbarbamam`, etc.). Il y a 10 de tels mots, y compris le mot s lui-même.
   – Trier ces mots en ordre lexicographique et les disposer en colonne, (cela donne une table 10x10).
   – Le mot s' est obtenu en lisant la dernière colonne de la table.
   Hup, hup hup, Barbatruc ! Et `barbamaman` *se trasforme à la volonté de la BWT en* `bmmbrnaaaa`. *Tous les étudiants n'ont pas fait ce calcul de la manière la plus économique.*
   Le mot s' est donc une permutation du mot s et la procédure appliquée s'appelle la *Transformée de Burrows-Wheeler* (BWT), c'est pourquoi on notera par la suite $s' = bwt(s)$.
3. Simuler la méthode *Move-to-Front* sur la chaîne $bwt(s)$ et calculer la valeur moyenne des codes émis pour cette chaine.
   On obtient 1201444000. *Moyenne 1,6.*
4. Expliquer cette différence entre les deux valeurs moyennes obtenues.
   La BWT crée des suites de lettres identiques et donc des codes nuls.
5. Expliquer en particulier quel est l'effet de la BWT sur un texte qui contient plusieurs occurrences d'un même facteur (comme par exemple `the` en Anglais ou `est` ou `que` en Français).
   Si il y a plusieurs "est", plusieurs conjugués commenceront par "st" (et ensuite probablement un espace, ou autre) et se termineront par "e". Ces "e" seront rapprochés par le tri en ordre lexicographique à cause des préfixes communs "st".
6. Citer une méthode autre que MTF qui est susceptible de compresser $bwt(s)$ mieux que s et expliquer pourquoi.
   RLE, puisque on crée des runs.
7. Si $longueur(s) = n$, combien de comparaisons de caractères sont nécessaires (dans le pire des cas) pour trier deux conjugués de s ?
   La bonne réponse est n (pour $s = a^n$). Les réponses $n - 1$ (pour $a^{n-1}b$) ou O(n) peuvent être considérées acceptables, mais si on a pensé à $a^{n-1}b$ pourquoi ne pas avoir pensé à a^n ?
8. Quel est donc la complexité (dans le pire des cas) attendue d'un algorithme optimal qui trie les n conjugués de s, exprimée en nombre de comparaisons de caractères ?
   Dans le pire des cas, il faut au moins $O(n \lg n)$ comparaisons entre les n conjugués pour les trier en ordre lexicographique et chacune des ces $O(n \lg n)$ comparaisons nécessite dans le pire des cas O(n) comparaisons de caractères. La bonne réponse est donc $O(n^2 \lg n)$.

9. Est-il nécessaire que l'encodeur maintient en mémoire les n conjugués de s (grosso modo, cela fait n copies du fichier pour un espace de l'ordre de $O(n^2)$) ? Est il possible de calculer la BWT en utilisant un espace $O(n)$?

   Par un tableau de n pointeurs, un sur chaque lettre du texte, et traiter le texte comme un tableau circulaire. Chaque pointeur est la première lettre d'un conjugué. On trie ces pointeurs en fonction de l'ordre lex des conjugués correspondants (p. ex. par un tri par tas en temps $O(n^2 \lg n)$, cf question 8), il est immédiat reconstruire ensuite $btw(s)$ en temps négligéable $O(n)$.

10. Quels sont les avantages si on divise le fichier en k blocs de taille n/k ? Quel est l'inconvénient ?
    C'est évident qu'en termes d'espace, on réduit la complexité d'un facteur k.
    Mais combien se sont-ils demandés comment cela affectait la complexité en temps ?
    Le tri de chacun des k blocs prendra $O(\frac{n^2}{k^2}(\lg n - \lg k))$. Soit, un réduction de k fois du temps total de calcul des bwt.
    L'inconvénient est qu'on obtiendra en principe des codes de moyennes plus élevées, puisque des runs plus longs ne seront probablement pas repérés.

11. Expliquer comment le décodeur peut reconstruire la première colonne p de la table en connaissant uniquement $bwt(s)$.

    La première colonne p de la table se construit immédiatement en triant en ordre alphabétique les symboles de $bwt(s)$: aaaabbmmnr.

    Est-il possible d'effectuer cette reconstruction en temps $O(n)$ et espace constant ?

    Ceci peut bien sûr se faire en temps $O(n)$ et espace constant en utilisant un compteur par symbole de l'alphabet (espace constant) et utiliser l'algo Insertion Counting (en temps $O(n)$), idéal pour trier valeurs appartenant à un intervalle et avec beaucoup de répétitions, comme les nôtres.

12. Expliquer pourquoi le décodeur ne peut pas reconstruire s en connaissant uniquement $bwt(s)$.

    Tous les conjugués de s ont la même bwt.

    et indiquer quelle est l'information **minimale** additionnelle dont on doit disposer à votre avis pour

Puisque le décodeur peut construire w en temps $O(n)$ et espace constant à partir de $btw(s)$, il suffit que, en plus du codage de btw(s), l'encodeur lui envoie la position de w où s commence (dans ce cas, 6), soit $O(1)$ information supplémentaire (cf. question 12.)

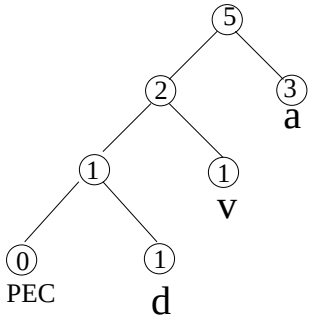
Exercice 2

Compresser le texte "`avada kedavra`" par le codage de Huffman adaptatif. En plus de fournir la chaîne compressée, vous devrez détailler toutes les étapes de construction de l'arbre de codage.

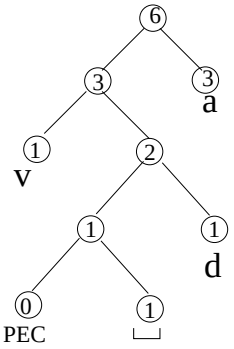
Même les fans les plus acharnés de Harry Potter auront trouvé cet exercice assez mortel, en termes d'ennui, surtout si on se perdait dans des mauvais calculs de l'arbre ensorcelé aussi malin que le Saule Cogneur.
Quels étudiants ont-ils vaincu le sortilège impardonnable de Lord Voldemort ? Lesquels sont sortis du labyrinthe dont les noeuds bougeaient aussi vite que les escaliers de Poudlard ? Qui a trouvé sur son chemin des arbres aussi crochus que la cicatrice en forme d'éclair sur le front de notre petit sorcier bien aimé ?

Sans détailler toutes les étapes on regardera juste les deux premières pièges dont le Professeur Rogue avait truffé le test et pratiquement tout le monde est tombé sur le premier ou sur le second.

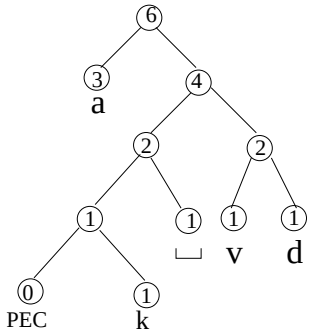
L'insertion du caractère ␣ dans l'arbre :



nécessite un seul échange entre le noeud v et son frère, ce qui donne l'arbre :



Et pourtant... Expelliarmus ! Et certains sont tombés déjà ici. L'insertion du "k" dans ce dernier arbre est un peu plus délicate. On doit d'abord échanger le frère du noeud "d" avec le noeud "v", qui est le dernier de son bloc (et non avec le noeud "d", comme certains ont fait). Ensuite, il faudra échanger encore "a" et son frère pour obtenir l'arbre :



Mis à part les problèmes avec l'arbre, venons à l'aspect du codage, quel que soit l'arbre (même faux) que vous avez obtenu.
L'erreur la plus grave commise est de penser que d'abord on construit l'arbre tout entier et après on code tout. Ceci n'est pas du tout le principe de Huffman adaptatif puisque cela nécessite deux lectures du texte, dont une est de trop. Huffman adaptatif ADAPTE son arbre au fur et à mesure qu'il lit, qu'il code et qu'il envoi (codes de chaque symbole variables dans le temps)
L'erreur la plus commune est de penser que vous pouvez d'abord mettre l'arbre à jour et après envoyer le nouveau symbole. Ce n'est pas très sympa pour le décodeur qui, lui, ne pourra jamais savoir ni comme vous avez modifié l'arbre, ni quel était le symbole. Autres erreurs sont liés au fait de pas avoir compris l'usage de PEC et la nécessité d'envoyer le symbole sous forme non codée lors de sa première occurrence. Cela n'est peut-être pas un hasard si ces aspects on été expliqués pendant le cours qui, dans le semestre, a vu la plus basse participation d'étudiants.

Supposez maintenant que chaque noeud de l'arbre est représenté par une structure (enregistrement) alors que l'arbre est représenté par un tableau de ces structures. Vous devrez choisir judicieusement les champs à inclure dans ces structures pour représenter convenablement l'arbre et pour effectuer sa construction.

On aurait voulu voir ici des noeuds d'arbres rangés dans un tableau, de manière à s'assurer que noeuds du même bloc, c'est-à-dire ayant le même poids, soient toujours dans des cases consécutives et où la relation père-fils est gérée par l'usage champs à valeur numérique indiquant père et (deux) fils de chaque noeud.

Vous devrez donc écrire le pseudo-code de la partie de l'algorithme qui met à jour le tableau à chaque étape de traitement d'un caractère de la chaîne.
Simuler votre algo sur la partie initiale "`avada`" de la chaîne.

Question pour les programmeurs purs et durs, qui, après avoir trouvé la bonne structure auraient été capables de pondre un algo convenable et grappiller quelques points. Dans les faits, personne n'est arrivé jusqu'ici.

Exercice 3

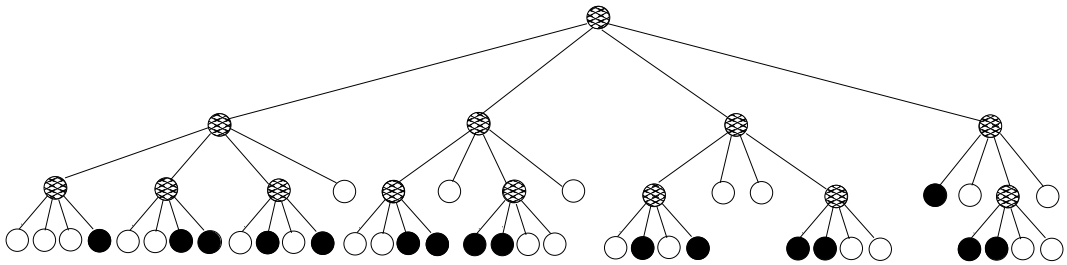
Le message suivant a été compressé par la méthode LZ78. Retrouver le message original en détaillant les étapes de construction du dictionnaire :

$$(0,s);(0,i);(0,n);(0,g);(2,n);(4,\sqcup);(0,a);(0,\sqcup);(1,o);(3,g);(8,s);(5,g);$$
$$(8,t);(0,h);(0,e);(11,o);(10,\sqcup);(1,i);(10,i);(17,a);(16,n);(4,EOF)$$

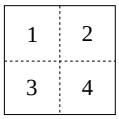
Message : "singing a song sing the song singing a song" [libre adaptation, de "Good Morning Starshine", Hair, the Musical].

Exercice 4

Le *quadtree* suivant représente une image binaire de 16 pixels par 16.



En sachant que les quadrants d'un carré sont pris en considérations dans l'ordre de la figure suivante, reconstruire l'image correspondante au quadtree.



Oh, le joli "G" pour notre nouvelle police de caractères !

Si on suppose que chaque noeud de l'arbre occupe autant de place qu'un pixel de la figure d'origine, quel est le rapport de compression de cette image quand elle est représentée par ce quadtree ?

C'est un peu bizarre que un noeud occupe autant qu'un pixel. Cependant, il y a 64 pixels et 53 noeuds, donc la bonne réponse est 64/53 ou 53/64, je ne sais jamais laquelle. Sinon, il faut multiplier cela par une constante multiplicative...