Rapport Compression Huffman Dynamique

Adrien Becchis & Stéphane Ferreira

December 16, 2013

Contents

§TOdo: organisation §GET: bon sujet

1 Modifications de l'arbre

1.1 Non échange entre Noeud et Descendant.

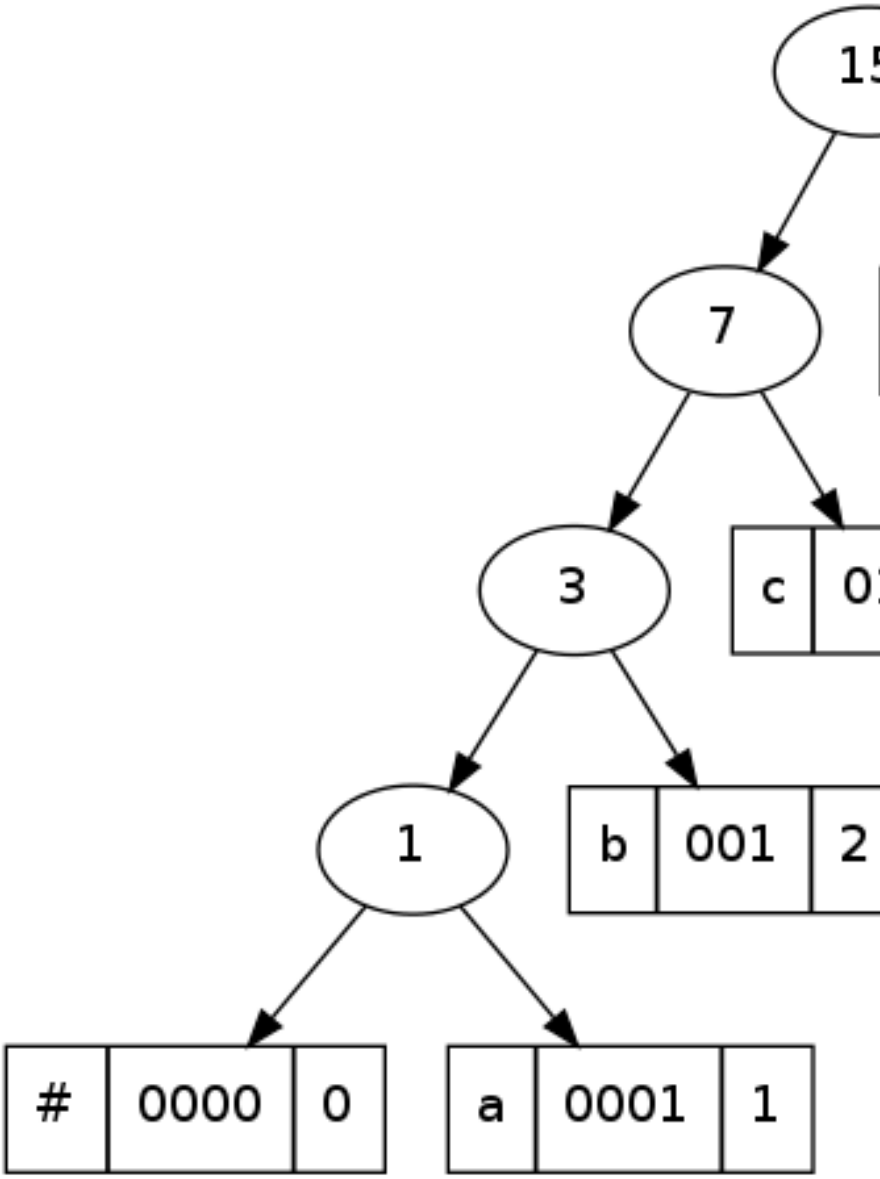
L'algorithme va déclencher un échange entre deux noeuds quand leur poids violeront la propriété d'Arbre de Huffman Adaptatif, à savoir qu'un noeud dans l'ordre GDBH doit avoir un poids supérieur ou égal à tout poids le précédent dans l'ordre. Le test `incrémentable` vérifiant si cette propriété est maintenue en alourdissant une des feuilles.

Hors il est tout simplement impossible qu'un ancêtre ai un poids inférieur à celui de ses descendants, puisque le poids d'un noeud est la somme du poids de ces deux fils (l'arbre étant complet). La seule situation où un père peut avoir le meme poids qu'un de ces fils, c'est quand l'autre fils est la feuille spéciale, au poids nul.

1.2 Nombre maximum d'échanges

Le nombre maximum d'échange possible correspond à la hauteur de l'AHA moins 1. En effet après chaque échange, on appelle la procédure de traitement sur le père du noeud que l'on vient de swapper. Donc dans le pire des cas, (si l'échange se fait avec un noeud de meme profondeur) on fait un échange par niveau à l'exception du dernier. (la racine).

La hauteur de l'arbre de huffman est dans le pire cas égale au nombre de caractère possible. Ceci arrive quand l'AHA est un peigne, par exemple quand la fréquence des lettre correspond à la suite $(2^i)_{i\in\mathbb{N}}$



Par Exemple pour le mot `abbccccddddddddd`:

Par conséquent, dans le pire cas, le nombre maximum d'échange est $n-1$ (avec n égal à la taille de l'alphabet des symboles)

2 Compression et décompression

2.1 Complexité Algorithme

2.1.1 Complexité Temporelle

Soit N le nombre de caractères, n le nombre de caractère différents, et h la hauteur de l'arbre (avec $0 \leq h \leq n-1$).

- La boucle principale de l'algorithme est répétée pour chaque caractère donc exactement N fois.
- La complexité pour retrouver la feuille correspondante à un du symbole dans l'arbre de Huffman dépend de l'implantation. Celui ci est de complexité constante si implanté avec une table de hachage, ou de complexité $O(n)$ si on a une liste des noeuds, où si on parcourt l'arbre (dont le nombre de noeuds total ne peut excéder $n + n/2$).
- la transmission du code est en temps constant si la feuille contient son propre code. (sinon il est reconstruit en $O(h)$, en remontant l'arbre)
- la procédure de modification est répétée au pire $h - 1$ fois. Cette procédure a pour complexité celle de la procédure `finBloc` seule opération de complexité non constante. (sa complexité réelle dépend des implantations,)

En agrégeant ces complexités, on a donc une complexité temporelle pire cas de $O(Nnh)$ Donc linéaire par rapport à la longueur du texte, hauteur de l'arbre , et nombre de caractère possible.

Il est cependant important de préciser qu'une **gestion optimale des entrées/sorties** (lecture et écriture de fichier) est **primordiale** pour avoir un programme efficace, les entrées sorties étant des opérations bien plus lentes que des opérations en mémoire vive/cache! (plusieurs ordre de grandeur.)

2.1.2 Complexité Spatiale

La complexité spatiale est de $O(n)$, correspondant seulement à la taille de mémoire de la structure de données utilisée pour encoder l'AHA, qui demeure $O(n)$ même si on enrichie la structure d'arbre avec une table de hachage et une liste en entre noeuds. A noter que le nombre total de caractère N n'influe nullement sur la complexité mémoire vu que l'on a affaire à des flux d'octet. Il est d'ailleurs plus qu'intéressant d'augmenter de manière négligeable la consommation mémoire en utilisant Tampons/Buffers pour diminuer le nombre d'entrées sorties chronophages.

2.2 Encodage Abracadabra

§IMG: TODO Etapes:

0. arbre vide

1. insertion de `A`, émission code #A=00000

2. insertion de `B`, émission de #B=0 00001

3. insertion de `R`, émission de #R= 00 10001. Reconstruction arbre

4. incrémentation `A`, émission A=0

5. insertion de `C`, émission #C= 100 00010. Reconstruction arbre

6. incrémentation `A`, émission A=0

7. insertion de `D`, émission #D= 1100 00011. Reconstruction

8. incrémentation `A`, émission A=0

9. incrémentation de `B`, émission B=110, reconstruction

10. incrémentation de `R`, émission de R=110

11. incrémentation de `A`, émission de 0

2.3 Algorithme de décompression

On suppose avoir les procédures correspondantes:

bits2char donne le caractère correspondant à l'octet spécifié

lireBit(F) lit un bit d'un flux (et le supprime)

lireBits(F,n) lit n bit du flux et les retournes

ecrire(F, c) écrit le caractère ans le flux de sortie.

estFeuille(H) dit si le noeud arbre huffman est une feuille

lettreFeuille(H) renvoi la lettre associé à la feuille

```
Procédure décompression(in : flux, out:flux)
   var H : Huffman, s: symbole, buff: bit[8], b : bit;
   var posCur : Huffman; // position courante arbre de huffman

   buff <-  lireBits(in,8) // en supposant mode ascii/8bits
   s <- bits2char(buff)
   modifier(H,s);
   posCur <- racine(H);
   ecrire(F,s)
```

```
   Tant que in n'est pas vide

      b <- prochainBit(in);

      Si b = 0 alors
         posCur <- noeudGauche(posCur);
      Sinon
         posCur <- noeudDroit(posCur);
      Fin Si

      Si estFeuille(posCur) alors
         Si estFeuilleSpeciale(posCur) alors
           buff <- lireBits(in,8);
           s <- bits2char(buff); ecrire(out,s);
           modifier (H,s); posCur <- racine(H);
         Sinon
            s <- lettreFeuille(posCur); ecrire(out,s)
             modifier(H,s); posCur <- racine(H);
         Fin Si
      Fin Si

   Fin tant que

FinProcedure decompression
```

Note il suffit de remplacer 8 par la taille de l'encodage utilisé pour adapter l'algorithme à la version 5 bit, ou toute autre version de son choix

3 Implantation

3.1 TODO Structures de données

3.2 TODO Résultats sur Jeux de test

Réalisée lors de la soutenance le 10 décembre.

La compression sur le fichier test10 de 90Mo a mis au mieux 4s sur une machine de l'ari.

A noter que nos bons résultats sont plus liés à une bonne gestions des flux IO, qu'à notre implémentation de l'arbre de Huffman qui pourrait être légèrement améliorée. (cf remarques sur la Complexité)

§TODO: coder, batcher..

4 TODO Dispatch

sur Sensibilité des performances aux entrées sorties.

5 Temps

Data/ExperimentalData:

filename	size (o)	tempsCompression (s)	tauxCompression (%)	tempsdecompression(s)
exp-rnd-10.txt	128203	1.079	0.38893786	1.181
exp-rnd-23.txt	173833	0.279	0.41425967	0.292
exp-rnd-06.txt	118092	0.189	0.4162094	0.209
urns.m12.n300.s1000	8953	0.047	0.44029933	0.033
exp-rnd-26.txt	99058	0.19	0.4508369	0.199
urns.m12.n4000.s10000	98956	0.181	0.44007438	0.199
exp-rnd-11.txt	127897	0.164	0.38900834	0.211
exp-rnd-33.txt	174333	0.343	0.41928953	0.342
urns.m12.n7168.s1000	7954	0.039	0.452351	0.027
urns.m12.n3000.s100	852	0.024	0.49295774	0.007

Continued on next page

filename	size (o)	tempsCompression (s)	tauxCompression (%)	tempsdecompression(s)
exp-rnd-30.txt	117407	0.223	0.442299	0.223
exp-rnd-31.txt	175356	0.266	0.41958645	0.315
exp-rnd-08.txt	203092	0.253	0.37957674	0.289
urns.m12.n4000.s100000	1088958	1.754	0.4438041	1.937
exp-rnd-16.txt	269364	0.456	0.39257288	0.388
exp-rnd-02.txt	223818	0.333	0.41591382	0.348
exp-rnd-17.txt	268922	0.323	0.3926157	0.389
exp-rnd-29.txt	117969	0.192	0.4423535	0.218
exp-rnd-15.txt	97526	0.185	0.4409901	0.229
urns.m12.n7168.s100000	988958	1.744	0.45422152	1.946
exp-rnd-22.txt	174270	0.261	0.4142308	0.277
exp-rnd-28.txt	118437	0.213	0.44234487	0.21
urns.m12.n300.s10000	98955	0.16	0.44083676	0.185
exp-rnd-04.txt	119286	0.202	0.41631037	0.19
exp-rnd-18.txt	268427	0.342	0.39262444	0.495
exp-rnd-09.txt	202736	0.276	0.37966123	0.275
exp-rnd-19.txt	117362	0.215	0.4347574	0.215
exp-rnd-24.txt	173340	0.255	0.4142206	0.283
exp-rnd-20.txt	116924	0.206	0.43485513	0.219
exp-rnd-14.txt	98027	0.171	0.44089893	0.178
urns.m13.n22000.s100000	988959	1.585	0.44870818	1.806
exp-rnd-25.txt	99528	0.259	0.45079777	0.202
exp-rnd-13.txt	98466	0.18	0.44081205	0.181
exp-rnd-01.txt	224358	0.313	0.4158889	0.373
exp-rnd-05.txt	118746	0.396	0.41625825	0.212
urns.m12.n3000.s1000	8954	0.053	0.4431539	0.024
exp-rnd-27.txt	98494	0.17	0.45064673	0.187
exp-rnd-03.txt	223158	0.329	0.4158354	0.427
exp-rnd-21.txt	116425	0.225	0.43483788	0.197
exp-rnd-32.txt	174891	0.269	0.41946125	0.309
exp-rnd-07.txt	203404	0.267	0.37953532	0.28
exp-rnd-12.txt	127547	0.18	0.38909578	0.177

Data/Test1/:

filename	size (o)	tempsCompression (s)	tauxCompression (%)	tempsdecompression(s)
upmc.eps	585794	1.134	0.17327763	1.287
puissdeux10	2047	0.008	0.2569614	0.023
test0	3000	0.062	0.6303333	0.099
puissdeux15$_rev$	65535	0.163	0.25017166	0.136
puissdeux10$_rev$	2047	0.006	0.2540303	0.005
test4	1048574	0.652	0.250031	0.873
test2	65534	0.041	0.25034332	0.051
test1	2046	0.003	0.25562072	0.004
test3	80000	1.321	0.6786125	1.185
puissdeux15	65535	0.037	0.25040054	0.072
test2rev	65534	0.163	0.2501297	0.156

Data/Test2/:

filename	size (o)	tempsCompression (s)	tauxCompression (%)	tempsdecompression(s)
Data/Test2/test10	97005568	30.931	0.13503169	43.361