POLYTECHNIQUE
MONTRÉAL

INF8770
Technologies Multimédias

Travail pratique #1

Méthodes de codage

Matthieu Basset — 2225981

Janvier 2024

Table des matières

1 Question 1

1.1 Textes

Pour les textes on peut s'imaginer à des résultats corrects en termes de compression pour les trois premiers. Ce sont des exemples assez simples de textes utilisant un nombre limité de symboles avec des répétitions. Le texte 3 sera probablement moins bien compressé car il possède plus de syboles que les deux autres et ces derniers varient plus, il sera donc peut-être plus difficile de créer des symboles efficaces pour la compression.

Pour les deux derniers textes écrits en latin, la compression devrait aussi plutôt bien fonctionner, en effet grâce aux répétitions de mots et de lettres, l'algorithme LZW devrait pouvoir créer une représentation plus efficace du texte. Le texte 5 sera a priori mieux compressé car il est plus long et donc l'algorithme aura plus de données pour apprendre les symboles.

1.2 Images

Pour les images, on peut s'attendre à des résultats très variables.

- Pour l'image 1, la compression risque de ne pas vraiment marcher, l'image semble être un gradient continu de couleurs, il n'y aurait donc pas de répétitions de valeurs.
- Pour l'image 2, la compression devrait marcher excellement bien, en effet l'image est un fond noir uniforme, c'est donc une suite continue de symboles identiques.
- Pour l'image 3, le fond est uniforme, ce qui va être encodé efficacement en théorie, et la couleur rouge étant très présente, elle pourra probablement être aussi encodée efficacement.
- Pour l'image 4, la compression devrait être acceptable mais bien que l'image soit généralement dans des tons clairs, les variations légères de couleurs rendront probablement la compression plus difficile car les valeurs de couleur ne se répèteront pas exactement.
- Pour l'image 5, tout comme pour l'image 2 on devrait avoir une compression très efficace.

2 Question 2

On réalise l'implémentation de l'algorithme LZW et on l'applique aux données fournies.

2.1 Choix d'implémentation

Pour l'implémenation de cet algorithme certains choix ont été faits:

- L'implémentation est faite en Rust pour des raisons de performance (et de sécurité).
- Le dictionnaire est implémenté sous forme de HashMap pour une recherche rapide.
- Les données sont passées à l'algorithme de compression sous forme d'octets pour une plus grande flexibilité. Cela permet de compresser aussi bien des textes que des images. On pourrait cependant perdre la représentation UTF-8 de certains caractères, mais ce n'est pas un problème car à la décompression les octets sont simplement recopiés. Aussi, cela implique que les images sont représentées non pas par pixel mais par composante de couleur. Cela permet en principe à l'algorithme de classifier lui même des couleurs dans le dictionnaire en les ajoutant comme symbole, permettant donc une meilleure flexibilité.

2.2 Résultats

Afin d'avoir des données robustes, les exécutions de code sont menées sur respectivement 1000 et 100 exécutions pour les textes et les images. Les résultats seront présentés sous le format $(\mu \pm \sigma)$ unitée où μ est la moyenne, σ l'écart-type et l'unitée dépend du contexte (ici micro ou milli seconde).

Les facteurs de compression sont donnés en pourcentage pour une meilleure lisibilité.

Résultats pour les textes:

Texte	Temps d'exécution	Facteur de compression
Texte 1	$(6.74 \pm 0.95)\,\mu$s	11.90%
Texte 2	$(8.09 \pm 1.04)\,\mu$s	26.62%
Texte 3	$(8.64 \pm 1.64)\,\mu$s	10.78%
Texte 4	$(33.48 \pm 7.33)\,\mu$s	0.11%
Texte 5	$(179.29 \pm 15.60)\,\mu$s	26.61%

Résultats pour les images:

Image	Temps d'exécution	Facteur de compression
Image 1	(38.00 ± 3.23) ms	-33.21%
Image 2	(10.40 ± 1.32) ms	96.87%
Image 3	(113.19 ± 5.02) ms	56.03%
Image 4	(146.76 ± 8.91) ms	15.07%
Image 5	(48.93 ± 2.03) ms	98.64%

2.3 Analyse des résultats

2.3.1 Textes

On remarque que l'on a des taux de compression acceptables pour les textes, cela était attendu au vu des répétitions de caractères dans les textes. La seule surprise est le texte 4 qui n'est presque pas compressé. Cela est dû au fait que le texte, en langue naturel, ne soit pas assez long pour que l'algorithme LZW puisse "apprendre" les suites de caractères qui forment les mots du texte. On voit en comparaison que le texte 5 qui est similaire au texte 4 mais qui est simplement plus long obtient un bien meilleur taux de compression.

Pour ce qui est du temps d'exécution, on remarque que les textes simples sont effectivement sans surprise très rapides à compresser. Les deux derniers sont plus longs, mais leur complexité le justifie. En effet, leur dictionnaire sera plus complexe et donc la recherche de symboles sera plus longue. Aussi plus de nouveaux symboles pourront être ajoutés au dictionnaire. Le texte 5 est significativement plus long à compresser, ce qui est attendu car il est plus long.

2.3.2 Images

Pour les images, on remarque que les résultats sont très variables.

- L'image 1 est effectivement très mal compressée, elle est même plus lourde une fois encodée avec LZW.Ainsi on voit bien que si l'on a aucune répétition de symboles, l'algorithme pourra même être contre productif.
- Les images 2 et 5 sont extrêmement bien compressées, ce qui était attendu car elles sont composées de répétitions de symboles. Mais l'image 5 est mieux compressé ce qui est surprenant au premier abord car l'image 2 est juste une répétition de symboles identiques, mais c'est probablement là le problème, LZW n'est pas assez efficace pour compresser des données trop uniformes.
- Les images 3 et 4 sont compressées de manière acceptable, les répétitions aident la compression mais ne sont pas aussi omniprésentes que pour les images 2 et 5.

Les temps d'exéction varient par un facteu d'un peu plus de 10 entre les images 2 et 4. L'image 1 est plus petite en résolution et n'utilise pas vraiment le dictionnaire, elle est donc rapide à encoder. Les images 2 et 5 sont rapides à encoder car le travail sur le dictionnaire de symboles est encore une fois assez léger en raison des symboles fortement répétés. Les images 3 et 4 prennent plus de temps car elles sont plus complexes. On y trouve des répétitions mais elles sont moins uniformes et donc le dictionnaire est plus complexe.

2.3.3 Conclusion

Finalement, l'algorithme LZW est très efficace pour compresser des données comportant des répétitions, mais certains critères doivent être pris en compte pour être assuré d'obtenir de bons résultats. Comme vu avec le texte 1 et l'image 2, une répétition trop uniforme de caractère n'est pas nécessairement la meilleure forme de donnée. Et comme vu avec le texte 4, une donnée trop courte ne permet pas d'exploiter le plein potentiel de l'algorithme. Ainsi, on cherchera des données qui ont des redondances, mais qui sont à la fois de longueur suffisante et présentant suffisamment d'hétérogénéité pour que l'algorithme puisse apprendre des symboles efficaces.

3 Question 3

3.1 Implémentation d'une nouvelle méthode

On décide d'implémenter une nouvelle méthode de compression pour compléter le travail préliminaire afin d'augmenter l'efficacité sur les données uniformes. On implémente donc l'algorithme RLE (Run Length Encoding) qui consiste à remplacer les répétitions de symboles par un symbole suivi du nombre de répétitions.

3.1.1 Particularités de l'implémentation

On décide pour les images de travailler sur les composantes de couleur, en effet ce sont ces composantes qui sont le plus susceptibles de se répéter en grand nombre. En pratique pour le dictionnaire, on convertit les 32 bits de couleurs divisés en 4 octets en un seul entier de 32 bits pour une recherche plus rapide.

Pour cette implémentation de RLE en particulier, le nombre optimal de bit à utiliser pour le compteur est déterminé indépendamment pour chaque image par opposition à la convention usuelle d'utiliser 8 bits. Cela permet d'optimiser d'autant plus la compression.

3.2 Résultats

Les résultats sont présentés sous le même format que pour l'algorithme LZW.Le code est également exécuté 1000 et 100 fois pour les textes et les images respectivement.

Résultats pour les textes:

Texte	Temps d'exécution	Facteur de compression
Texte 1	$(2.81 \pm 0.70)\,\mu$s	47.62%
Texte 2	$(3.53 \pm 0.41)\,\mu$s	50.65%
Texte 3	$(5.55 \pm 0.60)\,\mu$s	21.57%
Texte 4	$(19.70 \pm 3.24)\,\mu$s	1.10%
Texte 5	$(125.15 \pm 11.78)\,\mu$s	2.10%

Résultats pour les images:

Image	Temps d'exécution	Facteur de compression
Image 1	(12.04 ± 0.37) ms	0.00%
Image 2	(0.38 ± 0.04) ms	99.95%
Image 3	(36.00 ± 0.74) ms	37.31%
Image 4	(72.29 ± 4.69) ms	0.02%
Image 5	(2.26 ± 0.09) ms	96.84%

3.3 Analyse des résultats

3.3.1 Textes

Pour les textes les résultats sont meilleurs pour les premiers textes, ce qui est normal car ils présentent plus de répétitions brutes plutôt que des répétitions de suites de caractères comme les textes 4 et 5 qui s'en retrouvent alors moins bien compressés.

3.3.2 Images

Les images ayant beaucoup de zones uniformes sont globalement très bien compressées là où celles n'en ayant pas vraiment sont très mal compressées.

4 Question 4

4.1 Comparaison directe des méthodes

4.1.1 Temps d'exécution

Pour ce qui est du temps d'exécution, la méthode RLE est sensiblement plus rapide que LZW comme on peut le constater dans le tableau ci-dessous (Seulement les moyenes dans les unités utilisées précedemment sont présentés, microsecondes pour les textes et millisecondes pour les images). On y voit que RLE est systématiquement plus rapide que LZW, ce qui est attendu car RLE est une méthode très simple et rapide à exécuter. Ainsi, si l'on sait que RLE compressera mieux ou presque, c'est une méthode qui sera à privilégier.

Textes	LZW	RLE	Images	LZW	RLE
Texte 1	6.74	**2.81**	Image 1	38.00	**12.04**
Texte 2	8.09	**3.53**	Image 2	10.40	**0.38**
Texte 3	8.64	**5.55**	Image 3	113.19	**36.00**
Texte 4	33.48	**19.70**	Image 4	146.76	**72.29**
Texte 5	179.29	**125.15**	Image 5	48.93	**2.26**

4.1.2 Facteur de compression

En revanche pour le facteur de compression, RLE n'est pas aussi performant que LZW, comme on peut le constater dans le tableau ci-dessous. Pour les textes, les performances sont majoritairement meilleures pour RLE, mais pour un texte qu'on pourrait considérer comme "typique" comme le texte 5, qui comporte des mots formant un texte complet en langue naturelle, les performances sont bien inférieures à LZW.

Pour les images on voit que RLE tire parti de chaque zone uniforme des images, mais cela n'est pas suffisant pour la placer au-dessus de LZW qui reste plus universel dans la majorité des cas.

Textes	LZW	RLE	Images	LZW	RLE
Texte 1	11.90%	**47.62**%	Image 1	−33.21%	**0.00**%
Texte 2	26.62%	**50.65**%	Image 2	96.87%	**99.95**%
Texte 3	10.78%	**21.57**%	Image 3	**56.03**%	37.31%
Texte 4	0.11%	**1.10**%	Image 4	**15.07**%	0.02%
Texte 5	**26.61**%	2.10%	Image 5	**98.64**%	96.84%

4.2 Autres méthodes

Utiliser la méthode RLE semblait au départ une bonne idée pour optimiser au mieux les répétitions plus uniformes qui étaient moins bien gérées par LZW, mais finalement mise à part sa vitesse d'exécution, cette méthode performe moins bien sur la pluaprt des requêtes, il faudrait donc la garder uniquement pour les données très uniformes. On aurait pu implémenter une compression entropique type Huffman, ce qui aurait certainement été plus judicieux pour avoir de meilleures performances globales.

Pour garder notre cadre actuel, on pourrait aussi essayer d'avoir un test en amont pour déterminer si les données sont uniformes ou non, et choisir la méthode de compression en conséquence. Cela pourrait être une solution pour optimiser les performances globales.