Grande Ecole de sciences, d'ingénierie, d'économie et de gestion de **CY Cergy Paris Université**

Programmation fonctionnelle

Projet : Méthodes de compression sans perte

Effectué du 12 Février au 31 Mars 2024

par

Patrice SOULIER, Quentin ROLLET, Marc-Antoine VERGNET, Valentin NOAILLES

Enseignant : *Romain DUJOL*

Contents

1 Introduction

Les algorithmes de compression, véritables énigmes mathématiques complexes, revêtent une importance cruciale dans le domaine de l'informatique. Alors que certains optent pour des méthodes de compression avec perte, particulièrement adaptées à des applications telles que la compression d'images, cette approche n'est pas toujours idéale, notamment lorsque la préservation intégrale des données est primordiale.

Dans ce rapport, nous explorerons minutieusement les mécanismes des algorithmes de compression sans perte. Ces derniers se distinguent par leur capacité à réduire la taille des données sans altérer leur contenu, une caractéristique essentielle dans de nombreux contextes. Nous aborderons les différentes phases du processus, de la conception à l'implémentation, en passant par une évaluation approfondie des méthodes existantes.

La première section de notre étude se penchera sur la conception des algorithmes, mettant en lumière la répartition des tâches et la mise en place rigoureuse des tests, deux aspects cruciaux dans le développement de solutions efficaces. Par la suite, nous plongerons dans l'implémentation, examinant en détail la méthode Run-Length Encoding (RLE), les approches statistiques, et les méthodes à dictionnaire.

La section suivante approfondira notre analyse en étudiant de près les différentes méthodes de compression, notamment le RLE, les méthodes statistiques avec une comparaison approfondie entre Huffman et Shannon-Fano, ainsi que les méthodes à dictionnaire, avec un examen détaillé des différences entre LZ78 et LZW. Nous conclurons cette section en identifiant les configurations les plus performantes au sein de chaque méthode.

Enfin, cette étude se clôturera par une conclusion éclairante, résumant les principales conclusions tirées de notre exploration approfondie des algorithmes de compression sans perte. À travers ce rapport, nous chercherons à offrir une vision complète et nuancée des différentes approches, permettant ainsi de guider les choix futurs dans le domaine de la compression des données.

2 Étude des méthodes de compression

2.1 Méthode RLE

La méthode RLE (Run-Length Encoding) est probablement la manière la plus simple pour un algorithme de compression sans perte. En effet, le principe est simple. Il suffit de compter le nombre d'occurence d'un caractère consécutifs, et d'encoder ce nombre d'occurence avec le caractère. La chaine de caractère äaaabbcbbbd̈evient alors [(a,4), (b, 2), (c, 1), (b,3)].

Cependant, cette méthode possède un assez gros défaut car elle ne peut pas compresser tout ce que l'on souhaite. Par exemple, il est très rare dans un texte qu'il y ait un grand nombre de caractères consécutifs. Lors de la compression, la sortie sera plus longue que l'entrée, ce qui n'est pas vraiment souhaitable. Par contre, il peut être assez efficace dans certains cas où il y a un grand nombre de données consécutives. Étudions ceci dans l'exemple suivant.

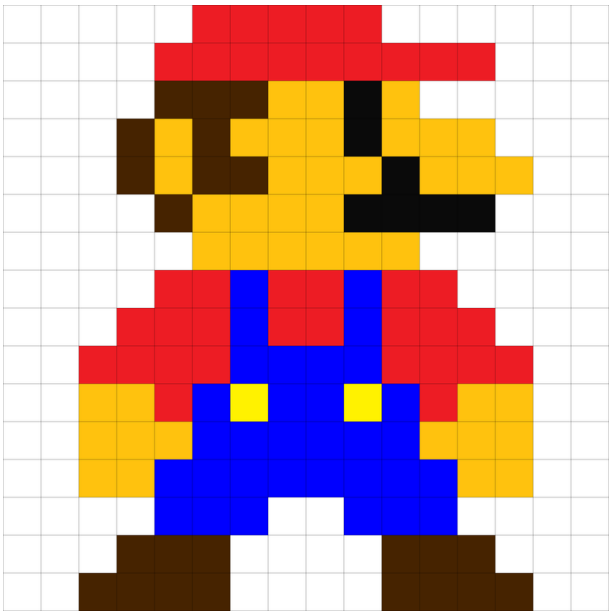

Figure 1: Mario 8-bits

Dans le cas du Mario en 8-bits, une représentation possible de l'encodage des bits pourrait être exprimée comme suit : [(5, blanc), (5, rouge), (10, blanc), ...]. Pour la première image, comprenant 256 pixels avec chaque pixel possédant 3 données (RGB), l'application de la compression RLE permet de réduire cela à seulement 75 changements de couleurs. Ainsi, la liste résultante a une longueur de 75, avec chaque élément comportant 4 données (le nombre de pixels identiques, RGB).

La première image a donc une taille de 768 bits, tandis que sa version compressée ne pèse plus que 300 bits. En conséquence, l'image compressée occupe environ 0.4 fois la

taille de son équivalent initial.

2.2 Méthodes statistiques

2.2.1 Comparaison Huffmann/Shannon-Fano

La différence entre Huffman et Shannon-Fano est l'approche du codage. L'approche de Shannon-Fano est descendante, c'est-à-dire que l'on commence par le symbole le plus fréquent et on descend jusqu'au symbole le moins fréquent. L'approche de Huffman est ascendante, c'est-à-dire que l'on commence par les symboles les moins fréquents et on remonte jusqu'au symbole le plus fréquent.

Le problème de l'approche descendante, c'est que l'on ne peut pas garantir que le code obtenu soit optimal. :En effet, lorsqu'il n'est pas possible de diviser les symboles en deux groupes de fréquences égales, on ne peut pas garantir que le code obtenu soit optimal. En revanche, l'approche ascendante garantit un code optimal.

En pratique, on décide plus souvent d'utiliser Huffman que Shannon-Fano car Huffman garantit un code optimal. Cependant, dans le cas où l'on a une liste de symboles triée par fréquence, Shannon-Fano est plus rapide à implémenter que Huffman. De plus, il est généralement plus facile de comprendre le fonctionnement de Shannon-Fano que celui de Huffman.

2.2.2 Meilleures configurations

Pour Huffman, les meilleurs configurations possibles sont celles qui permettent d'obtenir un arbre binaire parfait. C'est-à-dire que l'arbre est complet et que toutes les feuilles sont à la même profondeur. Cela permet d'obtenir un code optimal. Pour obtenir un arbre binaire parfait, il faut que le nombre de symboles soit une puissance de 2. Par exemple, la chaine de caractères "abracadabra" contient 5 symboles différents, il est donc impossible d'obtenir un arbre binaire parfait. Mais la chaine de caractères "perfect tree" contient 8 symboles différents, il est donc possible d'obtenir un arbre binaire parfait.

Dans le cas de Shannon-Fano, il faut que les symboles soient triées par fréquence pour obtenir un code optimal. Cependant, il est possible d'obtenir un code optimal sans que les symboles soient triées par fréquence. Cela dépend de la distribution des fréquences des symboles.

2.3 Méthodes à dictionnaire

2.3.1 Comparaion LZ78/LZW

La différence entre LZ78 et LZW est la manière dont les dictionnaires sont gérés. Dans LZ78, le dictionnaire est géré par un arbre de préfixes. Dans LZW, le dictionnaire est géré par une table de hachage.

En terme de performance, LZ78 est plus performant que LZW. En effet, LZ78 est plus rapide que LZW car la recherche d'un mot dans un arbre de préfixes est plus rapide que la recherche d'un mot dans une table de hachage. De plus, LZ78 est plus performant que LZW car LZ78 utilise moins de mémoire que LZW. En effet, LZ78 utilise un arbre de préfixes pour stocker les mots alors que LZW utilise une table de hachage pour stocker les mots.

2.3.2 Meilleures configurations

L'algorithme LZ78 fonctionne bien sur des données contenant des motifs répétitifs. En effet, plus il y a de motifs répétitifs, plus le dictionnaire sera rempli et plus la compression sera efficace. Par exemple, si on compresse un fichier contenant des motifs répétitifs, on obtiendra un taux de compression élevé. En revanche, si on compresse un fichier contenant des motifs aléatoires, on obtiendra un taux de compression faible. Il est aussi bénéfique d'utiliser un dictionnaire assez grand pour stocker iun grand nombre de séquences de caractères, mais il faut faire attention de ne pas utiliser un dictionnaire trop grand pour éviter de gaspiller de la mémoire.

Pour LZW, c'est un petit peu plus compliqué car la taille du dictionnaire est fixe et déterminé à l'avance. Généralement nous utilisons un dictionnaire à 256 entrées correspondant à la table ASCII, mais certaines implémentations de l'algorithmes permettent de changer la taille du dictionnaire d'entrée voire même de changer le dictionnaire de façon dynamique. On peut donc essayer de rentrer plusieurs tailles de dictionnaires différentes ou tester des stratégies différentes d'adaptation dynamique pour déterminer la configuration optimale dans chaque cas.

3 Conclusion

Pour conclure, nous avons pu étudier les différentes méthodes de compression sans perte. Nous avons pu constater que chaque méthode a ses avantages et ses inconvénients. Il est donc important de bien choisir la méthode de compression en fonction des données à compresser. Par exemple, si les données contiennent des motifs répétitifs, il est préférable d'utiliser une méthode à dictionnaire. Si les données contiennent des motifs aléatoires, il est préférable d'utiliser une méthode statistique. Il est aussi important de bien choisir les paramètres de la méthode de compression. Par exemple, pour la méthode à dictionnaire, il est important de bien choisir la taille du dictionnaire. Pour la méthode statistique, il est important de bien choisir la méthode de codage. Il est donc important de bien comprendre le fonctionnement de chaque méthode de compression sans perte pour bien choisir la méthode de compression et les paramètres de la méthode de compression.