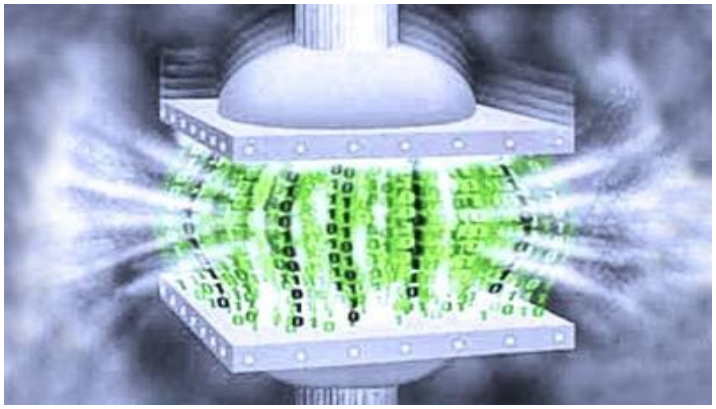

Projet Algorithme E.V.A.

CLASSE PRE-ING1 • 2021-2022

DESCRIPTION

Le but de ce projet est de réaliser un programme pouvant compresser des images en utilisant l'algorithme E.V.A. (Enhanced Value Algorithm). Cet algorithme se veut plus simple que ceux déjà existants, mais également compétitif en terme de tailles obtenues en sortie si on le compare par rapport au format PNG par exemple. Il ne tient qu'à vous de le démontrer

Pour pouvoir compresser/décompresser une image avec cet algorithme, il faudra pouvoir lire/écrire une image déjà existante avec un format lisible par un logiciel standard.

Le format que nous allons utiliser sera le format PPM : du code permettant de manipuler ce format vous sera directement fourni, ainsi que des images.

Ce projet met l'accent sur l'algorithmique, le passage de paramètres à un programme, manipulation des valeurs binaires, ainsi que l'accès à des fichiers

CAHIER DES CHARGES MINIMAL

- Réaliser un programme exécutable qui prend en paramètre plusieurs valeurs comme le nom de l'image d'entrée, l'image de sortie, ainsi que le mode de fonctionnement (compression/decompression)
- Etre capable d'ouvrir/créer/sauvegarder un fichier PPM, lire/écrire chaque pixel dedans, tout cela grâce au module de gestion des images PPM fourni. Les informations concernant ce module vous seront données au moment de votre choix de projet
- Comprendre et implémenter la spécification de l'algorithme E.V.A. fournie ci-dessous
- Afficher le taux de compression
- Afficher les durées de la compression et de la décompression (ne pas inclure d'affichage sur le terminal pour faire ces mesures au risque de les fausser)
- Montrer que l'on arrive à compresser et décompresser correctement n'importe quelles images PPM.

BONUS POSSIBLES

- Afficher une barre de progression dans le terminal
- Lister des statistiques sur les taux de compression en fonction de tailles/orientations d'images, avec différents types d'images (photos réelles, portraits, paysages, dessins abstraits, ...)
- Comparer avec le taux de compression pour les mêmes images au format PNG
- Vous pouvez implémenter d'autres améliorations de votre choix

RESSOURCES UTILES

GitHub

- Site web : https://github.com

Formats d'images PPM

- Wikipedia : https://fr.wikipedia.org/wiki/Portable_pixmap

Autres

- Données binaires : https://fr.wikipedia.org/wiki/Manipulation_de_bits

FORMAT E.V.A. COMPRESSION

➢ Tous les pixels de l'image d'entrée doivent être lus de gauche à droite et de haut en bas

➢ En sortie vous écrirez des blocs conformes à la définition de l'algorithme E.V.A. : ces blocs sont des successions d'octets avec un formalisme précis

➢ Tous les pixels doivent comporter 3 composantes (Rouge, Verte, Bleue) encodées sur 1 octet non signé chacune (valeurs entre 0 et 255 incluses). Cet algorithme est donc prévue pour une image couleur, mais il est possible de l'utiliser sur une image en niveaux de gris, les 3 composantes du pixel auront toutes la même valeur.

➢ Votre programme contiendra un tableau de 64 valeurs de pixels : ce tableau sera appelé « cache » et contiendra certaines valeurs de pixels telles que défini plus loin dans ce document. Les 64 valeurs seront initialisées à (0,0,0)

➢ Votre programme contiendra la valeur du pixel précédent, initialisée à (0,0,0)

➢ Chaque pixel de l'image d'entrée devra être traité en suivant les étapes suivantes dans l'ordre :

1. comparer au pixel précédent : si il est identique, stocker temporairement le nombre consécutif de valeurs | si il est différent, et si précédemment on a vu plusieurs pixels consécutifs identiques, il faut stocker en sortie un bloc **EVA_BLK_SAME** (le nombre de pixels identiques doit être décrémenté de 1 et ne doit pas dépasser 62 (les valeurs de bloc 0xFE et 0xFF sont reservées))

EVA_BLK_SAME							
Byte [0]							
7	6	5	4	3	2	1	0
1	1	Nb Same Pixels – 1					

2. si l'étape précédente n'est pas valide, il faut regarder si le pixel courant a déjà été vu précédemment. Pour cela il faut regarder dans les 64 pixels de cache. Pour calculer l'index où regarder dans le cache, il faut procéder au calcul indiqué ci-dessous. Si le cache à la position « index » contient la valeur du pixel courant, alors il faut stocker en sortie le bloc **EVA_BLK_INDEX**

index = (3*R + 5*G + 7*B)%64

EVA_BLK_INDEX							
Byte [0]							
7	6	5	4	3	2	1	0
0	0	index					

3. si l'étape précédente n'est pas valide, il faut maintenant regarder l'écart entre le pixel courant et le pixel précédent. Pour cela il faut calculer la différence sur chaque composante (rouge, verte, bleue) et vérifier que chacune de ces différences est dans l'intervalle [-2, +1]. Si c'est bien le cas alors il faut stocker en sortie le bloc **EVA_BLK_DIFF** dans lequel les différences de chaque composante ont un offset de +2

```
diff.r = cur_pix.r – prev_pix.r
diff.g = cur_pix.g – prev_pix.g
diff.b = cur_pix.b – prev_pix.b
```

EVA_BLK_DIFF							
Byte [0]							
7	6	5	4	3	2	1	0
0	1	diff.r + 2		diff.g + 2		diff.b + 2	

4. si l'étape précédente n'est pas valide, il faut regarder si la différence de vert est comprise dans l'intervalle [-32, +31]. Si c'est le cas, on calcule 2 nouvelles valeurs (diff.r – diff.g) et (diff.b – diff.g) qui sont les écarts de différence entre le rouge et le vert et, respectivement, le bleu et le vert. On vérifie que ces écarts sont compris dans l'intervalle [-8, +7] : si c'est bien le cas, on rajoute un bloc de 2 octets **EVA_BLK_LUMA** :

EVA_BLK_LUMA							
Byte [0]							
7	6	5	4	3	2	1	0
1	0	diff.g + 32					

EVA_BLK_LUMA							
Byte [1]							
7	6	5	4	3	2	1	0
diff.r - diff.g + 8				diff.b - diff.g + 8			

5. si l'étape précédente n'est pas valide, il n'y a plus d'autre choix que de stocker la valeur du pixel dans son ensemble. On va donc stocker en sortie un bloc de 4 octets **EVA_BLK_RGB**.

EVA_BLK_RGB							
Byte [0]							
7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	0

EVA_BLK_RGB							
Byte [1]							
7	6	5	4	3	2	1	0
red byte							

EVA_BLK_RGB							
Byte [2]							
7	6	5	4	3	2	1	0
green byte							

EVA_BLK_RGB							
Byte [3]							
7	6	5	4	3	2	1	0
blue byte							

Vous noterez la valeur 0xFE du 1er octet qui est une valeur interdite pour le bloc **EVA_BLK_SAME** (pour éviter la confusion entre les 2 blocs)

6. il vous est possible de rajouter dans votre fichier compressé des octets d'informations pour pouvoir débugger votre application. Pour cela vous utiliserez un bloc **EVA_BLK_DEBUG** qui sera suivi d'autant d'octets que vous voudrez.

EVA_BLK_DEBUG							
Byte [0]							
7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1

EVA_BLK_DEBUG							
Byte [1..*]							
7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x

L'important pour ce bloc c'est de savoir combien d'octets le suivent.Soit vous mettez un octet derrière qui indique le nombre d'octets utiles qui suivent, ou bien vous mettez systématiquement un nombre d'octets fixe. L'idée est de pouvoir détecter et décoder correctement ce bloc sans perturber le flux de l'image. Vous noterez la valeur 0xFF, valeur également interdite pour le bloc EVA_BLK_SAME.

7. A la fin du traitement d'un pixel, vous mettrez à jour la valeur du pixel précédent et vous mettrez à jour le cache avec le pixel courant (cf. étape 2 pour le calcul de l'index). Ensuite vous procéderez au pixel suivant, jusqu'à la fin de l'image.

FORMAT E.V.A. DECOMPRESSION

➢ Ici nous allons traiter tous les blocs précédemment stockés lors de la phase de compression. Ce traitement sera lui aussi très séquentiel, en détectant chaque type de bloc et en effectuant les actions associées.
➢ Le but ici est de reconstruire le flux de pixels initial

➢ Il faudra, comme à la compression, maintenir un cache des pixels (tableau de 64 valeurs) et une variable contenant la valeur du pixel précédent. Ces éléments seront initialisés comme décrits dans la phase de compression.

➢ Les différentes étapes pour la décompression sont les suivantes :

1. la première étape sera de détecter un bloc **EVA_BK_RGB** avec la valeur spécifique 0xFE. Si ce bloc est détecté, écrire en sortie un pixel avec les valeurs red/green/blue

2. si l'étape précédente n'est pas valide, on teste les 2 bits de poids fort du bloc, et on les compare avec ceux du bloc **EVA_BLK_SAME**. Si c'est le bon type de bloc, on récupère le nombre d'octets et on stocke en sortie la valeur du pixel précédent N fois. Ne pas oublier que la valeur N récupérée dans le bloc possède un offset de -1.

3. si l'étape précédente n'est pas valide, on teste les 2 bits de poids fort du bloc, et on les compare avec ceux du bloc **EVA_BLK_INDEX**. Si c'est le bon type de bloc, on va récupérer la valeur de pixel dans le tableau de cache, et on stocke en sortie cette valeur de pixel

4. si l'étape précédente n'est pas valide, on teste les 2 bits de poids fort du bloc, et on les compare à ceux du bloc **EVA_BLK_DIFF**. Si c'est le bon type de bloc on récupère les différences des 3 composantes, et on les ajoute aux composantes du pixel précédent. On stocke en sortie la valeur de pixel résultante. Attention : les valeurs stockées dans le bloc possèdent un offset de +2

5. si l'étape précédente n'est pas valide, on teste les 2 bits de poids fort du bloc et on les compare avec ceux du bloc **EVA_BLK_LUMA**. Si c'est le bon type de bloc, on récupère l'octet suivant dans le flux d'entrée. On dispose donc des 2 octets du bloc et on peut récupérer les différences des composantes par rapport au pixel précédent. Attention, les valeurs pour les composantes rouge et bleue sont relatives à la valeur de la composante verte. Attention également aux offsets : +32 pour le vert, +8 pour le rouge et bleu. Une fois le calcul effectué, il faut stocker la valeur en sortie.

6. si vous avez implémenté le bloc **EVA_BLK_DEBUG**, vous avez un cas supplémentaire à tester, mais ceci est laissé à votre description (debug, informations sur l'image, etc.)

7. si aucune des étapes précédentes n'est valide, alors on sait qu'il y a une **ERREUR** dans le flux et on peut l'indiquer explicitement pour aider au déverminage du code. Dans tous les cas, cet état ne doit jamais avoir lieu.