V1.0 Projet CY-LGORITHM

CLASSE préING1 • 2023-2024
AUTEURS Eva ANSERMIN - Romuald GRIGNON
E-MAILS eva.ansermin@cyu.fr - romuald.grignon@cyu.fr

DESCRIPTION DU PROJET

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 termes 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 d'images que nous allons utiliser sera le format PPM : du code permettant de manipuler ce format vous sera directement fourni, ainsi que des images de test.
A l'issue de ce projet, votre exécutable n'aura pas besoin d'être recompilé pour effectuer tous les traitements sur les images.
Ce projet est donc un projet de type programme exécutable qui va notamment nécessiter une bonne maîtrise :

- des arguments d'un exécutable
- des opérations binaires sur des octets
- de la prise en main des bouts de code fournis

INFORMATIONS GENERALES

Taille de l'équipe
Ce projet est un travail d'équipe. Il est autorisé de se réunir en groupe de 3 au maximum. Si le nombre total d'étudiants n'est pas un multiple de 3 et/ou si des étudiants n'arrivent pas à constituer des groupes, c'est au chargé de TD de statuer sur la formation des groupes. Pensez-donc à anticiper la constitution de vos groupes pour éviter des décisions malheureuses.
Démarrage du projet
Vous obtiendrez de plus amples informations quant aux dates précises de rendu, de soutenance, les critères d'évaluation, le contenu du livrable, …, quand le projet démarrera officiellement.
Dépôt de code
Vous devrez déposer la totalité des fichiers de votre projet sur un dépôt central Git. Il en existe plusieurs disponibles gratuitement sur des sites web comme github.com ou gitlab.com.

	Rapport du projet Un rapport écrit est requis, contenant une brève description de l'équipe et du sujet. Il décrira les différents problèmes rencontrés, les solutions apportées et les résultats. L'idée principale est de montrer comment l'équipe s'est organisée, et quel était le flux de travail appliqué pour atteindre les objectifs du cahier des charges. Le rapport du projet peut être rédigé en français. **Démonstration** Le jour de la présentation de votre projet, votre code sera exécuté sur la machine de votre chargé(e) de TD. La version utilisée sera la dernière fournie sur le dépôt Git avant la date limite de rendu. Même si vous avez une nouvelle version qui corrige des erreurs ou implémente de nouvelles fonctionnalités le jour de la démonstration, c'est bien la version du rendu qui sera utilisée. **Organisation** Votre projet complet **devrait (dans l'idéal)** être stocké sur un dépôt git (ou un outil similaire) tout au long du projet pour au moins trois raisons : éviter de perdre du travail tout au long du développement de votre application, être capable de travailler en équipe efficacement, et partager vos progrès de développement facilement avec votre chargé de projet. De plus il est **recommandé** de mettre en place un environnement de travail en équipe en utilisant divers outils pour cela (Slack, Trello, Discord, …).
CRITERES GENERAUX	• Le **but principal** du projet est de fournir une **application fonctionnelle** pour l'utilisateur. Le programme doit correspondre à la description en début de document et implémenter toutes les fonctionnalités listées. • Votre code sera généreusement **commenté**. • Tous les éléments de **votre code** (variables, fonctions, commentaires) seront écrits dans la **même langue**. Langue anglaise conseillée mais pas obligatoire. • Votre application ne doit **jamais** s'interrompre de manière intempestive (crash), ou tourner en boucle indéfiniment, quelle que soit la raison. Toutes les erreurs doivent être gérées correctement. Il est préférable de d'avoir une application stable avec moins de fonctionnalités plutôt qu'une application contenant toutes les exigences du cahier des charges mais qui plante trop souvent. Une application qui se stoppe de manière imprévue à cause d'une erreur de segmentation ou d'une exception, par exemple, sera un événement très pénalisant. • Votre application devra être **modulée** afin de ne pas avoir l'ensemble du code dans un seul et même fichier par exemple. Apportez du soin à la conception de votre projet avant de vous lancer dans le code. • Le livrable fourni à votre chargé(e) de TD sera simplement l'**URL** de votre **dépôt Git** accessible **publiquement**. Même si vous n'avez pas utilisé ce dépôt régulièrement au cours du projet, le code final sera livré dessus.
FONCTIONNALITES DU PROJET	• Votre programme doit prendre en argument le chemin du fichier de données d'entrée, le mode d'opération souhaitée (compression ou décompression), puis le chemin du fichier de sortie. • Il doit permettre 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.

- Dans le cas de la compression, l'image d'entrée sera une image au format PPM et le fichier de sortie sera au format EVA. Dans le cas de la décompression, le format du fichier d'entrée sera au format EVA, et le fichier de sortie sera au format PPM.
- Votre programme doit implémenter correctement l'algorithme de compression / décompression décrit plus bas dans ce document.
- Le programme doit pouvoir afficher le taux de compression entre l'image au format brut (sans compression) et le flux de blocs générés.
- Le programme doit également mesurer les temps de compression et de décompression et de les afficher à l'écran à la fin (attention à la mesure de temps qui ne doit pas prendre en compte les temps d'affichage des messages à l'écran mais uniquement les temps de traitement).
- Lors des phases de traitement, le programme doit afficher une barre de progression.

ALGORITHME DE COMPRESSION

- Tous les pixels des images d'entrée/de sortie doivent être lus/écrits de gauche à droite puis de haut en bas.
- Tous les pixels des images doivent comporter 3 composantes (Rouge, Verte, Bleue) encodées sur 1 octet non signé chacune (valeurs entre 0 et 255 incluses). Cet algorithme est donc prévu 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.
- Les **blocs** conformes à la définition de l'algorithme EVA sont des successions d'octets avec un formalisme précis décrit plus loin.
- Votre programme contiendra en mémoire un tableau de 64 valeurs de pixels : ce tableau sera appelé « **cache** » et contiendra certaines valeurs de pixels telles que définies plus loin dans ce document. Les 64 valeurs seront initialisées à (0,0,0).
- Votre programme contiendra en mémoire 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 le résultat ne doit pas dépasser 62 (les valeurs de bloc 0xFE et 0xFF sont réservées))

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**

EVA_BLK_INDEX							
Byte [0]							
7	6	5	4	3	2	1	0
0	0	index					
index = (3*R + 5*G + 7*B)%64							

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

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

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**. 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)

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							

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. Vous pouvez mettre 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.

ALGORITHME DE 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 pour recréer l'image d'origine.
- 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.
- Ici les blocs compressés seront lus en entrée, et les pixels de sortie seront générés à la volée.

- Chaque bloc de l'image compressée en entrée sera traité en suivant les étapes 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 discrétion (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 si le format de l'image d'entrée est correct.

RESSOURCES UTILES

Github

- www.github.com
- https://docs.github.com/en/get-started/quickstart/hello-world

Le format d'image PPM

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

Manipulation de données binaires

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