Faculté des Sciences
Département d'Informatique

INFO-F-201 – Systèmes d'exploitation

Projet 1 de programmation système
« Quel est ce Pokémon ? » – édition processus

Grand fan des jeux Pokémon, vous avez décidé d'implémenter un système pour permettre à vos amis d'identifier le numéro de Pokédex associé à un Pokémon sur base de son image. Vous voulez cependant un système relativement robuste, au cas où l'image serait partiellement dégradée ou avec d'autres couleurs. Pour cela, vous avez décidé d'utiliser un système de code de hachage perceptif (qui vous est fourni) afin de pouvoir facilement comparer les différentes images dans votre base de données.

Comme vous souhaitez que votre système puisse s'adapter à un grand nombre d'images, vous avez décidé d'implémenter ce projet en utilisant plusieurs processus pour comparer plusieurs images de manière concurrente.

1 Détails du projet

Ce projet est à réaliser par groupe de **trois** étudiants et comporte différentes parties :

1. Un programme C (*img-dist*) qui génère le code de hachage perceptif (le code est fourni, vous ne devez rien faire pour celui-ci à part le compiler) ;
2. Un script bash (*list-file*) qui liste les fichiers d'un dossier ;
3. Un programme C ou C++ (*img-search*) qui reçoit le chemin vers une image et indique l'image la plus similaire (au choix s'il y en a plusieurs) parmi celles d'une banque d'images dont chacun des fichiers est transmis sur stdin ;
4. Un script (*launcher*) pour lancer le programme *img-search*.

Contactez Arnaud Leponce (arnaud.leponce@ulb.be) avant le 26 octobre 2023 si vous n'avez pas de groupe. Vous serez pénalisés si vous réalisez le projet seul sans en avoir discuté auparavant.

Vous êtes autorisés à utiliser des fonctionnalités de C++ telles que `vector`, `string`, `hashmap`, `new` et `delete`, les classes, etc[1].

Notez que ce premier projet servira de base pour le second[2], nous vous conseillons donc de penser à écrire du code maintenable.

2 Programme fourni « img-dist » : comparaison d'images

Pour réaliser ce projet, vous recevez le code C et le Makefile pour créer le programme appelé **img-dist**. Vous n'avez rien à coder pour ce programme-là mais vous devrez le compiler afin de pouvoir utiliser le programme généré avec le reste du projet.

Pour le compiler, allez simplement dans le dossier *img-dist/* et entrez la commande `make`. Ceci devrait vous générer le programme img-dist.

Le programme img-dist fonctionne de la manière suivante :

```
img-dist [-v] chemin_première_image chemin_deuxième_image
```

1. Ce projet évalue les compétences en programmation système, vous ne serez donc pas récompensés pour avoir implémenté une magnifique chaîne d'héritage car ce n'est pas le sujet de ce cours.
2. Nous vous fournirons une solution type.

avec :

- `-v` : activer le mode verbeux. Est optionnel et ne devrait pas être utilisé dans le projet rendu ;
- `chemin_première_image` : chemin (avec nom du fichier) vers la première image à comparer ;
- `chemin_deuxième_image` : chemin (avec nom du fichier) vers la deuxième image à comparer.

En cas de succès, ce programme retourne une valeur comprise entre 0 et 64. Plus cette valeur de retour est élevée, moins les images sont jugées similaires[3] (0 correspond donc à des images jugées très similaires et 64 à des images jugées complètement différentes).

En cas d'erreur (c.-à-d., un paramètre manquant ou un mauvais chemin d'image), une valeur > 65 est retournée et un message d'erreur est écrit sur stderr.

Exemple d'utilisation :

```
./img-dist ../img/1.bmp ../img/20.bmp
```

Si vous êtes curieux du fonctionnement du hachage perceptif, plus d'explications sont fournies en annexe. À noter que ce concept dépasse le cadre du cours. Ces informations sont uniquement là pour les curieux.

3 Programme « img-search » : recherche d'images

3.1 Objectif

Le but de ce programme est de comparer une image au format BMP[4] (.bmp) avec une série d'autres images du même format. Vous devez écrire celui-ci à partir de zéro et il devra respecter les contraintes suivantes :

- Être écrit en C ou en C++ ;
- Être compilé à l'aide d'un Makefile (que vous devez donc écrire) ;
- Prendre en paramètre l'image à comparer avec les images données sur son entrée standard (stdin) – voir Section 3.2 ;
- Utiliser le programme img-dist pour comparer les images – voir Section 3.3 ;
- Créer 2 autres processus pour mener la recherche (ceux-ci sont créés au lancement du programme et se terminent en même temps que le processus principal) – voir Section 3.4 ;
- Gérer les signaux SIGUSR1, SIGUSR2 et SIGINT – voir Section 3.5.

3.2 Récupération des chemins vers les images

Il y a deux types d'images : celles de la banque d'images et celle dont on cherche les images similaires dans la banque d'images.

L'image dont on cherche à trouver une image similaire est transmise comme premier paramètre du programme *img-search*.

Les images de la banque d'images sont transmises via l'entrée standard (stdin). Vous pouvez considérer que le chemin individuel de chaque image ne pourra pas excéder 999 `char` (sans compter le `'\0'` final en C).

3. Notez que la technique du code de hachage perceptif utilisée ici pour juger de la similarité n'est pas infaillible en pratique. Elle vise simplement à servir de première approximation mais ce sera suffisant dans le cadre de ce projet.

4. Pour des raisons de simplicité, toutes les versions d'images BMP ne sont pas supportées et seule la plus rudimentaire l'est. Si vous souhaitez ajouter une image BMP à la banque d'images, assurez-vous qu'elle soit encodée dans une version simple.

3.3 Comparaison de paires d'images

La comparaison de deux images doit être réalisée en récupérant le code de retour de *img-dist* avec comme paramètres les chemins des deux images. L'option -v ne pourra pas être utilisée dans le projet que vous rendrez (mais vous pouvez l'utiliser si ça vous aide lors du développement).

Concernant le chemin d'exécution de *img-dist*, vous devrez considérer que celui-ci se trouve dans un des dossiers référencés dans la variable $PATH du programme. Cette modification de $PATH ne devra être effective **que** pour *img-search* et ses fils (vous ne pouvez donc pas modifier le $PATH de manière visible pour les autres processus). Cette modification sera réalisée au travers du script bash *launcher* (plus de détails Section 4.2).

3.4 Répartition en processus pour la recherche concurrente

La recherche de la ou des images les plus similaires dans la banque d'images doit se dérouler de manière concurrente dans 2 processus fils de *img-search*. Ceux-ci doivent être créés une seule fois au début de *img-search* et s'exécuter jusqu'à ce que toutes les images de la banque d'images aient été comparées. Le nombre d'images comparées doit être équilibré entre les 2 processus fils.

Vous aurez besoin de faire communiquer *img-search* avec ses 2 fils. Pour cela, vous devrez utiliser un *pipe* anonyme par processus fils. Ces pipes doivent servir à transmettre les chemins des images de la banque d'images au processus fils associé. Vous devrez également utiliser de la mémoire partagée pour conserver la distance la plus faible trouvée par chaque processus.

Ces techniques de communication inter-processus ne seront normalement pas suffisantes pour assurer le bon fonctionnement du programme. C'est à vous de choisir les autres et de les justifier dans votre rapport.

Lorsqu'il n'y a plus d'images associées à la banque de données (ex. : stdin est terminé via Ctrl + D), *img-search* devra attendre la fin des comparaisons faites par ses 2 fils et afficher le nom d'une des images de la banque d'images parmi les plus similaires à celle passée en paramètre de *img-search*. Si plusieurs images ont la même similarité minimale, vous pouvez choisir celle à afficher arbitrairement.

Si l'image la plus similaire est, par exemple, l'image « *img/22.bmp*» et avait, par exemple, une distance (c.-à-d., valeur de retour de *img-dist*) de 12, votre programme terminera son exécution en écrivant sur stdout :

```
Most similar image found: 'img/22.bmp' with a distance of 12.
```

Si aucune image n'a pu être comparée correctement (ex. : aucune image n'a été donnée sur stdin ou tous les chemins entrés étaient incorrects), votre programme terminera son exécution en écrivant sur stdout :

```
No similar image found (no comparison could be performed successfully).
```

Veillez à bien conserver ce format sans quoi les tests automatiques ne fonctionneront pas.

3.5 Gestion des signaux

Le signal SIGINT devra être géré par *img-search* (et ignoré par ses 2 fils). Si ce signal est reçu par *img-search*, ses 2 fils et lui devront se terminer de manière propre (pas de crash et attente de l'arrêt de tous les processus fils après les avoir notifiés qu'ils devaient se terminer).

Le signal SIGPIPE ne doit pas générer le crash de *img-search* ni de ses fils mais devra entraîner l'arrêt

de manière propre de tous ces processus.

4 Bash

4.1 Script « list-file » pour écrire les noms des fichiers d'un dossier

Vous devez créer un script Bash appelé **list-file** qui liste le nom de chaque fichier présent dans un dossier (ignore les sous-dossiers). Le dossier dans lequel on compte le nombre de fichiers est fourni en tant que premier (et seul) paramètre. Exemple :

```
./list-file img/
```

Si le chemin du dossier n'est pas spécifié, le script se termine avec le code de retour 1 et affiche le message suivant sur la sortie standard d'erreur : « *Missing directory name.* ».

Si le chemin est donné mais ne référence pas un dossier, alors le script se termine avec le code de retour 2 et affiche le message suivant sur la sortie standard d'erreur : « *The specified path is not a directory.* ».

4.2 Script « launcher » de lancement de votre programme

Ce script Bash aura pour but de lancer *img-search* en proposant 2 modes différents : l'image de base à comparer avec celles de la banque d'images et un chemin optionnel dont l'interprétation va dépendre du mode. La forme générale de la commande est :

```
launcher [-i|--interactive|-a|--automatic] image [database_path]
```

Les paramètres doivent être interprétés en fonction du mode :

- mode interactif (option `-i` ou `--interactive`) :
  - La banque d'image sera transmise manuellement (c'est-à-dire image par image) sur stdin par l'utilisateur ;
  - `image` : correspond à l'image à comparer avec celles de la banque d'images ;
  - `database_path` : préfixe du chemin des images transmises sur stdin (vide par défaut si non spécifié). L'ajout du préfixe doit se faire dans le script Bash ;
- mode automatique (option `-a` ou `--automatic`) :
  - La banque d'image sera transmise automatiquement par le script ;
  - `image` : correspond à l'image à comparer avec celles de la banque d'images ;
  - `database_path` : dossier dont tous les fichiers (mais pas ceux des sous-dossiers) constituent la banque d'images que le script transmet lui-même à img-search. Si ce paramètre n'est pas spécifié, sa valeur est fixée à « ./img/ ».

Exemple de commande avec le mode automatique :

```
./launcher -a img/6.bmp img/
```

Exemple de commande avec le mode interactif :

```
./launcher -i img/6.bmp
```

Lors de l'exécution de *img-search*, la variable de $PATH pour ce programme (et ce uniquement pour lui et ses fils) devra être modifiée en se voyant ajouter le chemin img-dist/.

5 Ce qui est mis à votre disposition

Vous pouvez télécharger la base du projet sur l'Université Virtuelle. Ce répertoire contient :

— le code C du programme img-dist (situé dans le dossier img-dist/) ;
— une banque d'images de Pokémon dans le dossier img/ ;
— des tests automatiques utilisables en vous rendant dans le dossier test/ et en exécutant la commande `./tests` ;
— un Makefile à compléter.

6 Critères d'évaluation

Si vous écrivez votre code en C, votre projet doit compiler avec `gcc` version 9.4 (ou ultérieure) et les options ci-dessous (présentes dans le Makefile fourni) :

```
FLAGS=-std=c11 -Wall -Wextra -O2 -Wpedantic
```

Si vous écrivez votre code en C++, votre projet doit compiler avec g++ version 9.4 (ou ultérieure) et les options ci-dessous :

```
FLAGS=-std=c++17 -Wall -Wextra -O2 -Wpedantic
```

Si votre programme ne compile pas, vous recevrez une note de 0/20.

N'hésitez pas à utiliser les flags des assainisseurs lors du développement de votre projet, ceci vous aidera à détecter vos erreurs plus facilement (ceci n'est pas obligatoire et pensez à recompiler tous vos fichiers lorsque vous ajoutez ou retirez ces options) :

```
-g -fsanitize=address,undefined
```

Assurez-vous également que **tous** vos appels systèmes ont leur valeur de retour correctement traitée (à vous de **gérer correctement les cas où une erreur est survenue**). Cela fait partie de l'évaluation. Un projet dont le code fonctionne mais ne prend pas compte des erreurs pouvant survenir perdra des points.

6.1 Pondération

— Tests automatiques **/3**
— Ce projet de programmation système va principalement évaluer votre compétence à manier correctement les outils liés aux systèmes d'exploitation (processus, *pipes*, signaux, ...) dans le langage C. La pertinence des outils utilisés ainsi que la manière dont ils sont utilisés (trop, pas assez, au mauvais endroit, trop longtemps, ...) sont évalués. Assurez-vous également que les codes d'erreurs sont bien traités. **/7**
— Bash **/3**
— Ce projet doit contenir un rapport dont la longueur attendue est de deux à trois pages (max 5). Ce rapport décrira succinctement le projet, les choix d'implémentation si nécessaire, les difficultés rencontrées et les solutions originales que vous avez fournies. **/5**
  — Orthographe
  — Structure
  — Légende des figures
— Votre code sera aussi évidemment examiné en termes de clarté, de documentation, de commentaires et de structure. **/2**

7 Remise du projet

Vous devez remettre un projet par groupe contenant un fichier zip contenant

- le code source C/C++ de *img-search* ;
- les scripts Bash *launcher* et *list-file* ;
- un Makefile[5] ;
- votre rapport au format PDF avec le nom des membres du groupe et leur ULBID ;
- les éventuels tests que vous auriez écrits.

N'incluez ni la banque d'images qui vous a été fournie ni les dossiers img-dist/ & test/ et leur contenu !

Vous devez soumettre votre projet sur l'**Université Virtuelle** pour le **12 novembre 2023 23h59** au plus tard.

Retards

Tout retard sera sanctionné d'un point par tranche de 4h de retard, avec un maximum de 24h de retard, et le projet devra être soumis sur l'université virtuelle.

Questions

Le meilleur moment pour poser vos questions sera pendant les séances de travaux pratiques. Vous pouvez cependant aussi poser vos questions via courriel à arnaud.leponce@ulb.be si nécessaire.

5. Celui-ci doit uniquement servir à compiler ou tester *img-search*.

A Code de hachage perceptif

Pour rappel, le contenu de cette annexe **ne fait pas partie** de la matière évaluée lors du projet. Elle est simplement là pour les curieuses et les curieux.

Fondamentalement, l'algorithme est réalisé en 6 étapes :

1. Redimensionner l'image en 32×32 ;
2. Passer l'image en niveaux de gris ;
3. Appliquer la transformée en cosinus discrets[6] (DCT) à 2 dimensions sur l'image (donne une matrice 32×32 en résultat) ;
4. Conserver la sous-matrice de taille 8×8 du coin supérieur gauche ;
5. Calculer la valeur moyenne μ de valeurs dans la sous-matrice 8×8 en ignorant le premier élément d'indice $(1, 1)$;
6. Utiliser un entier sur 64 bits pour lequel chaque bit est associé à une valeur de la matrice distinctes des autres. Chaque bit vaut 1 quand sa valeur associée est $> \mu$ et 0 sinon.

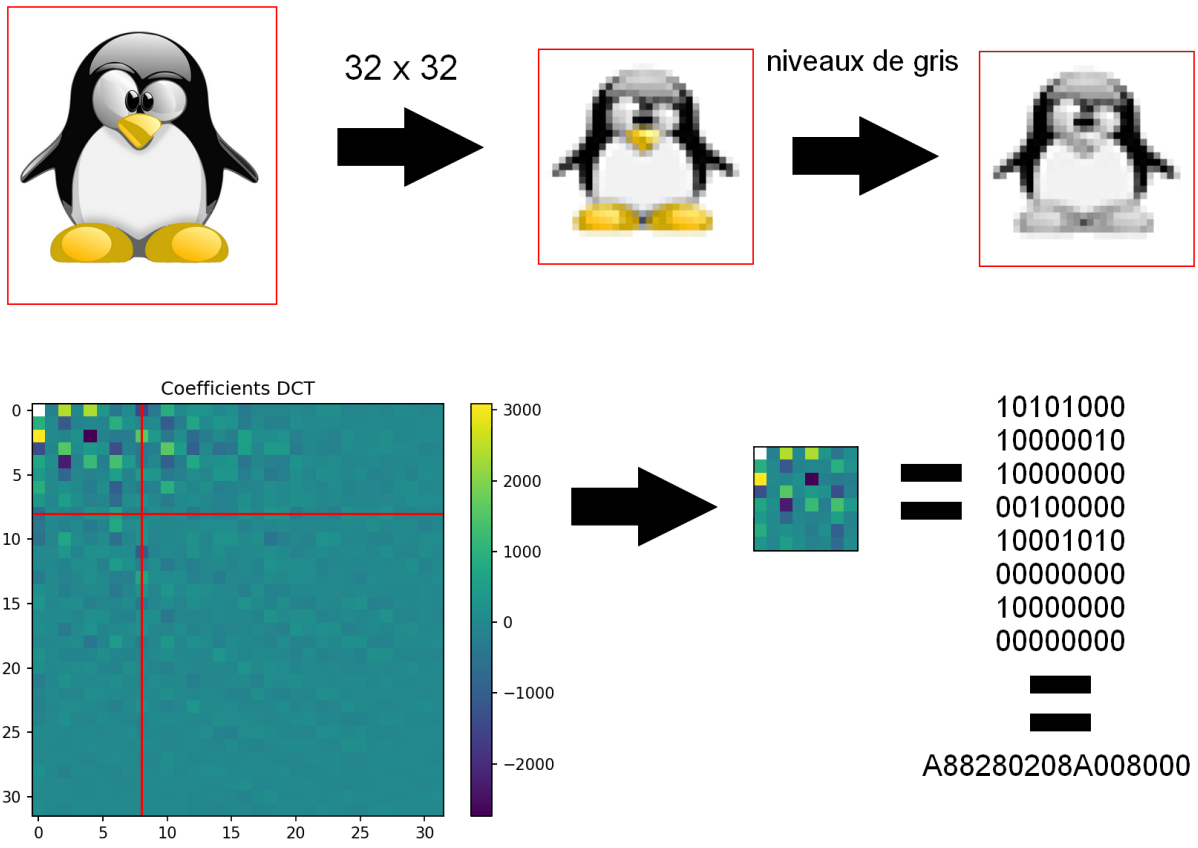

FIGURE 1 – Récapitulatif visuel des étapes du hachage perceptif en partant d'une image de Tux pour obtenir (par exemple) $0xA88280208A008000$ comme code de hachage final.

Un élément clé de cet algorithme est l'utilisation de la transformée en cosinus discrets. C'est une technique bien connue en traitement du signal (son, image, ...) et qui est souvent utilisée également dans le domaine de la compression avec perte. Par exemple, elle est utilisée par le format d'image JPEG ou dans la réduction des dimensions de variables en apprentissage automatique.

Pour une fonction $f : \mathbb{N} \to \mathbb{R} : x \mapsto f(x)$ avec $0 \leq x < N$ où $f(x)$ va représenter la composante d'indice x d'un vecteur, la formule de la DCT (de type II) à une dimension est :

6. https://fr.wikipedia.org/wiki/Transform%C3%A9e_en_cosinus_discr%C3%A8te

$$DCT_{1D}(k) = \sum_{x=0}^{N-1} f(x) \cdot cos\left(\frac{k\pi}{N} \cdot \left(x + \frac{1}{2}\right)\right)$$

En deux dimensions avec $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R} : (x, y) \mapsto f(x, y)$ pour $0 \leq x < N$ et $0 \leq y < M$ (où (x, y) représenterait l'élément $[y][x]$ d'un tableau en C par exemple), celle-ci devient [7] :

$$DCT_{2D}(u, v) = \frac{2}{\sqrt{M \cdot N}} C(u) \cdot C(v) \cdot \sum_{y=0}^{M-1} \sum_{x=0}^{N-1} f(x, y) \cdot cos\left(\frac{u\pi}{N} \cdot \left(x + \frac{1}{2}\right)\right) \cdot cos\left(\frac{v\pi}{M} \cdot \left(y + \frac{1}{2}\right)\right)$$

avec :

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{si } x = 0 \\ 1 & \text{sinon} \end{cases}$$

L'intérêt d'utiliser une DCT de dimension 2 pour les images est que cela permet de mieux capturer à la fois la relation verticale et celle horizontale entre les pixels.

Pour rappel, le calcul du code de hachage perceptif ne retient au final que les valeurs de $DCT_{2D}(u, v)$ du "coin supérieur gauche" (c.-à-d., pour $0 \leq u < 8$ et $0 \leq v < 8$). Le fait de ne conserver que ces valeurs revient à se concentrer sur les tendances générales car des petits u et v peuvent être vus comme des cosinus de fréquences plus basses. Nous ne rentrerons pas plus dans les détails ici mais vous pouvez vous-même voir sur la Figure 1 qu'au plus les valeurs se rapprochent du coin inférieur droit, au plus ces valeurs s'approchent de 0, ce qui fait que si elles sont approchées comme valant 0, moins d'informations seront perdues. C'est ce qui est fait implicitement en ne conservant que la sous-matrice 8×8.

Pour terminer cette annexe, précisons que la comparaison des codes de hachage se fait en utilisant une simple distance de Hamming est utilisée.

7. https://www.mathworks.com/help/images/ref/dct2.html