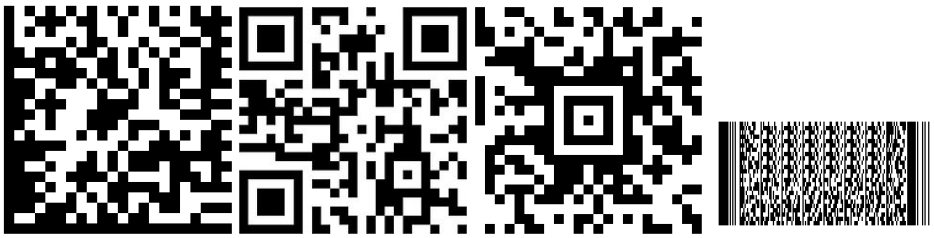
INFO0030: Projet de Programmation
Gestion de Code-Barres

B. Donnet, S. Bernard, V. Botta, S. Lens
Université de Liège

1 Contexte

On rencontre de plus en plus souvent des codes-barres carrés à deux dimensions. Vous en avez peut-être déjà vu sur des colis postaux, sur des panneaux publictaires, des cartes de visite, des certificats, des pièces électroniques, des tickets de concert, ... Ils sont même utilisés dans le domaine artistique. Ainsi, le clip vidéo de la chanson *Integral* des Pet Shop Boys[1] fait, par exemple, passer plusieurs liens via des QR Code.[2] La police de New York utilise aussi ce système pour charger directement les informations des conducteurs à partir de leur permis de conduire dans leur système TraCS.[3]

Ces code-barres sont capables de stocker une certaine quantité d'information utile à laquelle est ajoutée de l'information redondante qui permet au code-barres d'être plus robuste. En effet, même si une partie est altérée (on peut imaginer que la pluie a endommagé un ticket de concert), le lecteur de code-barres sera quand même capable de lire le message original en exploitant l'information redondante.



(a) Datamatrix (b) QR Code (c) Aztec (d) PDF-417

Figure 1: Exemples de format de code-barres

Il existe de nombreux formats utilisés en pratique. La Fig. 1 donne différents exemples de formats qu'on peut décoder avec le lecteur adéquat ou avec un smartphone, si celui-ci est équipé du logiciel adéquat.

Ces codes-barres sont très pratiques et sont de plus en plus utilisés car ils permettent de stocker beaucoup d'informations sur une petite surface tout en étant robustes aux altérations.

L'Université de Liège, découvrant le potentiel de ces tags, décide de les utiliser pour améliorer sa gestion des cartes d'étudiant. En effet, lors des examens écrits, les cartes d'étudiant sont utilisées pour vérifier à la fois que l'étudiant est bien inscrit à l'examen (en vérifiant que son matricule ULg se trouve bien dans une liste d'inscrits) et que l'étudiant est bien le propriétaire de la carte (via la photo). La première de ces vérifications peut très vite s'avérer fastidieuse, surtout lorsqu'on a affaire à plusieurs centaines d'étudiants. Dès lors, la décision a été prise d'encoder le matricule ULg d'un étudiant sous la forme d'un code-barres 2D et de placer ce code-barres dans un coin de la carte d'étudiant. La carte peut alors être scannée rapidement via un petit lecteur portable qui affiche la photo de l'étudiant ainsi qu'une indication précisant s'il est inscrit à l'examen. Une liste des étudiants ne s'étant pas présenté à l'examen peut également être générée directement au terme de l'examen.

[1] Pet Shop Boys, "Fundamental", 2006, chez Parlophone Rhino.
[2] *Quick Response Code.*
[3] *Traffic & Criminal Software.* Voir `http://criminaljustice.state.ny.us/crimnet/ojsa/initiatives/tracs.htm`

Dans ce projet, il vous est demandé de concevoir un prototype de décodeur de code-barres 2D utilisé pour encoder un matricule ULg.

2 Format du Code-Barres

2.1 Généralités

Les codes-barres utilisés pour encoder les matricules ULg sont de forme carrée et de taille 7×7. La Fig. 2 vous montre la forme d'un tel code-barres. La zone blanche est la zone qui contient le nombre entier (*zone nombre*, cfr. Sec. 2.2) et la zone grise est la zone qui contient les informations redondantes qui permettront de corriger une erreur dans les codes-barres (*zone de parité*, cfr. Sec. 2.3).

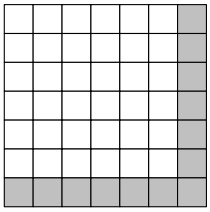


Figure 2: Format du code-barres 2D

2.2 Zone Nombre

Pour encoder un matricule ULg dans un code-barres 2D, nous allons simplement encoder sa représentation binaire inversée[4] dans la zone nombre. Pour rappel, la notation binaire classique d'un entier est sa notation en base 2, c'est-à-dire constituée de 0 et de 1 qui correspondent respectivement à "blanc" et "noir" dans nos codes-barres.

Tout nombre naturel peut s'écrire en binaire, c'est-à-dire se décomposer en somme de puissance de 2 (2, 4, 8, 16, 32, ...). Par exemple, 83 peut se décomposer en $1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ et donc se noter `1010011` en binaire.[5]

Si l'on veut généraliser la notation binaire d'un nombre N représentable dans les 36 cases de la zone nombre de notre code-barres, on peut dire que chaque case correspondra à un a_i tel que $N = \sum_{i=0}^{35} a_i \cdot 2^i$.

Si l'on considère que les cases sont numérotées de gauche à droite et de haut en bas, nous noircirons la première case si et seulement si a_0 vaut 1, la seconde si et seulement si a_1 vaut 1 et ainsi de suite. De cette manière, nous représentons le chiffre 83 dans notre code-barres comme indiqué à la Fig. 3

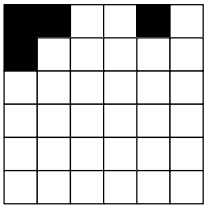


Figure 3: La zone nombre pour "83" dans notre code-barres 2D

Vu qu'un matricule ULg est un nombre de 8 chiffres, la matrice sera bien plus remplie. Par exemple, le matricule 87651234 est représenté à la Fig. 4.

[4] Inversée dans l'ordre des bits, le nombre commençant par ses bits de poids faible au lieu de ceux de poids fort.

[5] Plus de détails sur la conversion décimal/binaire se trouvent sur les slides 9 à 15 de la Partie 1 du cours INFO2009.

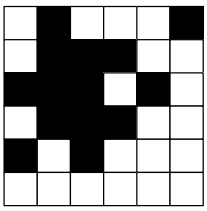


Figure 4: La zone nombre pour “87651234” dans notre code-barres 2D

2.3 Zone de Parité

Une fois les données placées, nous devons ajouter les informations redondantes qui permettront de vérifier si le code-barres a été altéré. Un *code de parité* va être utilisé pour cela.

Un code de parité se calcule en comptant le nombre de bits à 1 (noir dans notre cas). Si celui-ci est impair, le bit de parité vaut 1. Si par contre il est pair, il vaut 0. Notez que si l'on considère chaque case comme un entier valant 0 ou 1, la somme de parité correspond au reste de la division par 2 de la somme des différentes valeurs.

Dans notre code-barres 2D, un premier code de parité est calculé sur chacune des ligne et placé dans la dernière colonne. Un deuxième code de parité est calculé sur les colonnes et placé dans la dernière ligne. Enfin, la parité de la dernière ligne et de la dernière colonne (qui est nécessairement la même!) est stockée dans la case en bas à droite. La Fig. 5 reprend le précédent exemple de matricule ULg complété de ses bits de parité (ici en gris mais ils seront, bien entendu, en noir dans les codes-barres réels).

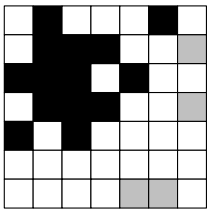


Figure 5: Le code-barre 2D complet pour le matricule ULg 87651234

3 Correction d'une Erreur

L'intérêt de la zone de parité est qu'elle permet de détecter et corriger une erreur sur un bit quelconque du code-barres pour autant qu'il n'y en ait qu'une seule. Les erreurs dans le code-barres peuvent dues à diverses circonstances: une erreur à la lecture, une altération du code-barres due à la pluie, un trait à l'encre indélibile, ...

Pour comprendre comment corriger une erreur, nous allons séparer le problème en différents scénarios.

Pour chacun des scénarios décrits ci-dessous, il faut se demander quelle est la coordonnée du point défectueux si l'on considère qu'il n'y a qu'une seule erreur (la numérotation des lignes et des colonnes se fait à partir de l'indice 0).

- la parité de la ligne 2 et de la colonne 0 est mauvaise.
- la parité de la ligne 1 et du pixel inférieur droit est mauvaise.
- la parité du pixel inférieur droit est mauvaise.

A partir de ces trois scénarios, il doit être possible de déterminer les trois grandes “familles” d'erreurs uniques possibles.

Attention, si on sait qu'il y a une seule erreur dans le code-barres, on sait qu'on pourra toujours la corriger. Maintenant, ce n'est pas parce qu'on se retrouve dans une situation où il y a un seul bit de parité ligne et un seul bit de parité de colonne qui ne sont pas valides qu'il n'y a qu'une seule erreur dans la zone nombre.

4 Génération d'Images

Le format d'image `PNM` (*Portable Anymap*) désigne un ensemble de formats de fichier graphique utilisés pour les échanges. On considère trois formats:

- `PPM` (*Portable Pixmap File Format*).
- `PGM` (*Portable Graymap File Format*).
- `PBM` (*Portable Bitmat File Format*).

Ces formats ont été définis et utilisés par le projet NetPBM.[6]
Dans le cadre de ce projet, nous allons nous intéresser uniquement au format `PBM`.

4.1 Portable Pixmap File Format

Le format `PBM` a été défini par Jeff Poskanzer dans les années 80. Il s'agit d'un format d'images bitmaps monochromes pouvant être transmises via un message électronique en format ASCII. De plus, ce format supporte tout changement dans le formatage du texte.
Un fichier `PBM` ASCII se présente comme suit:

- un nombre magique (P1)
- un caractère d'espacement (tabulation, espace, nouvelle ligne)
- la largeur de l'image (codée en caractères ASCII)
- un caractère d'espacement
- la hauteur de l'image (codée en caractères ASCII)
- un caractère d'espacement
- données ASCII de l'image:
  - l'image est codée ligne par ligne, en partant du haut
  - chaque ligne est codée de gauche à droite
  - un pixel noir est codé par un caractère '1', un pixel blanc est codé par un caractère '0'
  - les caractères d'espacement à l'intérieur de cette section sont ignorés
  - aucune ligne ne peut dépasser 70 caractères
- un '0' final

Toutes les lignes commençant par '#' sont ignorées.
Ci-dessous, un exemple de fichier `PBM` ASCII qui représente la lettre 'J':

```
P1
# Ceci est un commentaire
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
```

[6] `http://netpbm.sourceforge.net/`.

```
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

4.2 Export du Code-Barres en PBM

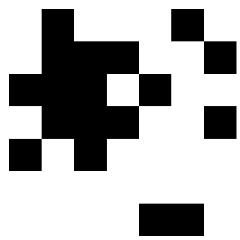


Figure 6: Image au format `PBM` pour le matricule ULg 87651234

La Sec. 4.1 a introduit le format `PBM` et on voit très bien qu'il convient parfaitement aux codes-barres représentant un matricule ULg. Ainsi, il suffirait, simplement, de stocker le code-barres dans un fichier (avec un en-tête adéquat pour respecter le format `PBM` et chaque bit du code-barres représentant un caractère ASCII de l'image) pour avoir une image `PBM` toute faite.

Cependant, stocker tel quel le code-barres risque de rendre l'image illisible. Le code-barres tel qu'envisagé à la Sec. 2 formerait une image de taille 7×7, ce qui est beaucoup trop petit.

Dès lors, au moment de stocker le code-barres dans un fichier `PBM`, il devient nécessaire d'augmenter la résolution de l'image. Pour ce faire, au lieu de stocker directement la matrice 7x7, on va stocker une matrice 70×70. Ceci signifie que chaque bit (0 ou 1) du code-barres sera stocké par un carré de 10×10 bits. Ainsi, par exemple, les deux premiers bits du code-barres correspondant au matricule ULg 87651234 (cfr. Fig. 4) seront représentés par:

```
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

Le résultat complet de l'image est donné à la Fig. 6.

5 Enoncé du Projet

Il vous est demandé d'écrire un programme permettant d'implémenter la représentation d'un matricule ULg sous la forme d'un code-barres tel que décrit à la Sec. 2. Votre programme sera capable de charger un fichier texte contenant une liste de matricules et génèrera une image (cfr. Sec. 4) du code-barres correspondant à chaque matricule. L'emplacement (et donc le nom) du fichier texte sera donné en argument de votre programme.

Le fichier texte contenant les matricules se présente de la façon suivante:

```
20100749
20111182
```

```
20103045
20105542
...
20110255
```

Le nombre de matricules dans ce fichier est indéterminé (le fichier pourrait très bien être vide).

En outre, votre programme doit implémenter les fonctions permettant de corriger une éventuelle erreur dans les codes-barres (cfr. Sec. 3).

Pour chaque matricule, une image au format `PBM` devra être générée (cfr. Sec. 4). Le nom de fichier, pour chaque image correspondra au numéro de matricule lut dans le fichier, avec l'extension `.pnm`.

Vous veillerez à ce que votre programme soit modulaire (i.e., découpé en diverses fonctions) et que chaque module soit correctement documenté (i.e., spécifications).