Projet : extraction des contours d'une image – Version 2
Semestre pair 2022-2023

1 Introduction

Le but de ce projet est de manipuler les contours des objets présents dans une image en mettant en œuvre les principes vus en CTD et TP.

1.1 Évaluation

Le projet qui est décrit ci-après doit être réalisé de manière individuelle.
Le travail que vous allez produire sera évalué :

- à partir d'un dépôt sur Moodle que vous devrez effectuer au plus tard le dimanche 21 mai 2023 à 23h59 (n'attendez pas le dernier moment...) ; quelle que soit la raison, si le dépôt est vide le 21 mai à minuit alors vous aurez 0 ; il est donc conseillé de déposer votre projet avant, même s'il n'est pas fini ; mais la version déposée doit pouvoir être compilée sans erreur ;
- à partir d'un oral qui se déroulera :
  - le mardi 23 mai 2023 à 10h pour les groupes B11, B12, B21, B22 et B31 ;
  - le jeudi 25 mai 2023 à 15h45 pour les groupes A11 et A12.

1.2 Barème

Le projet comporte trois niveaux de difficulté croissante, numérotés de 1 à 3. La réalisation complète et exacte de chaque niveau donne la possibilité d'obtenir une certaine note maximale, comme cela est indiqué dans le tableau suivant :

Niveau	Note maximale
1	10
2	15
3	20

Pour commencer un niveau, il faut avoir complètement réalisé le niveau précédent. Par exemple, si vous n'avez pas terminé le niveau 2 et si vous réalisez une partie du niveau 3, vous serez tout de même noté sur 15. Un niveau est considéré comme réalisé quand toutes ses fonctionnalités sont programmées et que le programme correspondant au niveau s'exécute sans erreur.
Attention : si le projet que vous avez déposé sur Moodle ne peut pas être compilé sans erreur en utilisant la commande `make`, vous aurez automatiquement la note 0.

1.3 Dépôt sur Moodle

Vous déposerez sur Moodle un seul fichier archive au format ZIP qui doit être produit en suivant les directives suivantes.

1. Créez un répertoire dont le nom doit suivre le modèle :
   `NOM-PRENOM-NUMETU`
   où :
   - `NOM`, `PRENOM` et `NUMETU` doivent être remplacés, respectivement et dans cet ordre, par votre nom de famille en majuscules, votre prénom en majuscules et votre numéro d'étudiant (8 chiffres) ;

- n'apparaît aucune lettre accentuée ;
- n'apparaît aucun espace ni apostrophe, les éventuels espaces et apostrophes dans votre nom ou votre prénom devant être remplacés par des tirets `-` (signe moins) ;

2. Placez dans ce répertoire :
   - les fichiers sources des modules de votre projet, c'est-à-dire les fichiers d'extensions `.h` et `.c`, y compris `svg.h` et `svg.c` ;
   - les fichiers contenant la fonction principale (`main`) de chaque niveau (`testpolygone.c` pour le niveau 1, `contex.c` pour le niveau 2 et `regex.c` pour le niveau 3) ;
   - les fichiers de description des dépendances de nom `Makefile`n, où n est le numéro du niveau, permettant, grâce à la commande `make -f Makefile`n, de produire l'exécutable correspondant au niveau n ;
   - un fichier de texte (obtenu avec un simple éditeur de texte) de nom `niveau-`n`.txt` où vous remplacerez n par le numéro du niveau que vous avez réalisé ; ce fichier pourra être vide ou contenir d'éventuelles informations destinées à expliquer les choix que vous aurez faits.

   Vos fichiers sources doivent être indentés et contenir des commentaires permettant de bien comprendre votre travail. Aucun autre fichier (objets, exécutable, données, résultats, etc.) ne doit être présent dans le répertoire.
3. Créez une archive au format ZIP de ce répertoire ; cette archive doit porter le même nom que le répertoire auquel est ajoutée l'extension `.zip` ; pour produire cette archive, dans le terminal, vous pouvez vous placer dans le répertoire parent du répertoire contenant vos fichiers sources et utiliser la commande :
   `zip␣`NOM-PRENOM-NUMETU`.zip␣`NOM-PRENOM-NUMETU`/*`

Respectez bien toutes ces consignes car des tests et des vérifications automatiques seront effectués sur votre projet.

1.4 Oral

Au moment de l'oral, vous devrez être prêt à :
- faire fonctionner votre programme ;
- répondre aux questions de l'enseignant ;
- effectuer des tests ou des modifications demandées par l'enseignant.

2 Niveau 1 : manipulation de polygones

2.1 Module `polygone`

Il s'agit d'écrire un module[1] `polygone` qui utilise le module `svg` et qui permet de gérer un polygone dont les sommets ont des coordonnées entières. Un polygone sera représenté par la structure de données suivante :

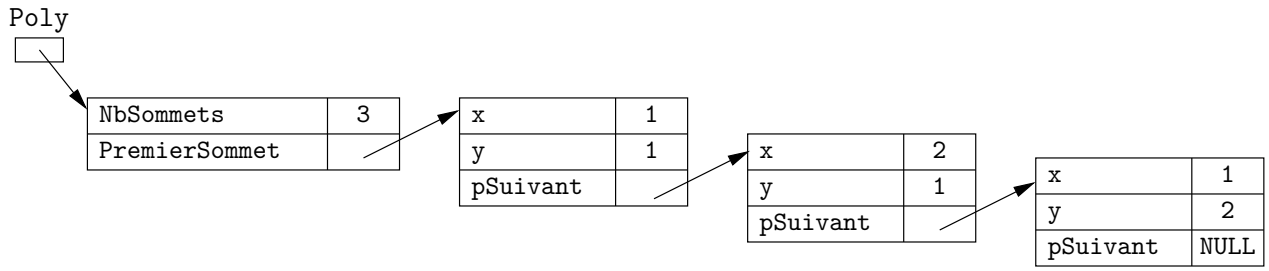
```
// Sommet d'un polygone
struct sSommet
{
  int x,y; // Coordonnées entières du sommet
  struct sSommet *pSuivant; // Adresse du sommet suivant
};
```

1. Pour chaque module que vous devez écrire, vous devez appliquer les principes vus en CTD concernant la programmation modulaire, notamment l'encapsulation des données et des traitements.

```c
// Descripteur du polygone pointant vers une liste simplement chaînée de
// sommets
struct sPolygone
{
  int NbSommets; // Nombre de sommets
  struct sSommet *PremierSommet; // Adresse du premier sommet
};

// Type permettant de manipuler un polygone
typedef struct sPolygone *tPolygone;
```

Exemple : la représentation mémoire d'un triangle dont les sommets ont les coordonnées $(1,1)$, $(2,1)$, et $(1,2)$ devra être :



Le module `polygone` devra comporter les fonctions suivantes :

- `tPolygone PolygoneCreer(void)` : crée un polygone vide ;
- `int PolygoneNbSommets(tPolygone Poly)` : retourne le nombre de sommets de `Poly` ;
- `void PolygoneAjouterSommetEnFin(int x, int y, tPolygone Poly)` : ajoute un sommet de coordonnées (`x`,`y`) à la fin de `Poly` ;
- `void PolygoneAjouterSommetEnDebut(int x, int y, tPolygone Poly)` : ajoute un sommet de coordonnées (`x`,`y`) au début de `Poly` ;
- `void PolygoneAjouterSommetEnIeme(int x, int y, int i, tPolygone Poly)` : ajoute un sommet de coordonnées (`x`,`y`) à la `i`$^{\text{e}}$ position de `Poly` (la position d'insertion est comprise entre 0 et le nombre de sommets du polygone, 0 signifiant un ajout au début et le nombre de sommets signifiant un ajout à la fin) ;
- `void PolygoneAfficher(tPolygone Poly)` : affiche à l'écran les coordonnées des sommets du polygone à `n` sommets `Poly` sous la forme :

  `n`
  `x1`␣`y1`
  `x2`␣`y2`
  `...`
  `xn`␣`yn`

  Dans le cas du triangle de l'exemple précédent, l'affichage obtenu doit être :

  `3`
  `1`␣`1`
  `2`␣`1`
  `1`␣`2`

- `void PolygoneSommetIeme(tPolygone Poly, int i, int *px, int *py)` : délivre en sortie (par l'intermédiaire de `px` et de `py`) les coordonnées du sommet qui se trouve à la `i`$^{\text{e}}$ position de `Poly` (la position est comprise entre 0 et le nombre de sommets moins un) ;

- `void PolygoneSommetSupprimerIeme(int i, tPolygone Poly)` : supprime le sommet qui se trouve à la i^e position de `Poly` (la position est comprise entre 0 et le nombre de sommets moins un) ;
- `void PolygoneLiberer(tPolygone Poly)` : libère la mémoire occupée par le polygone `Poly` ;
- `tPolygone PolygoneLiref(FILE *f)` : lit dans le fichier de texte d'identificateur `f` les coordonnées des sommets d'un polygone et retourne le polygone ; cette fonction suppose que le polygone est décrit dans le fichier selon le même format que celui de la fonction `PolygoneAfficher` ;
- `void PolygoneEcriref(tPolygone Poly, FILE *f)` : écrit dans le fichier de texte d'identificateur `f` les coordonnées des sommets du polygone `Poly` sous le même format que celui de la fonction `PolygoneAfficher`.
- `void PolygoneEcrireSvg(tPolygone Poly, tStyle *pStyle, FILE *IdFichSVG)` : écrit le polygone `Poly` dans le fichier au format SVG d'identificateur `IdFichSVG` avec le style d'adresse `pStyle` (cf. les explications dans le fichier `svg.h` ainsi que dans la section 2.2.2 ci-dessous) ; attention, dans la fonction `SvgEcrirePolygone`, le paramètre `x` correspond aux indices des colonnes et le paramètre `y` aux indices des lignes.

Remarque : dans tous les modules, il est fortement conseillé d'écrire des fonctions « auxiliaires » qui ne seront pas publiques ; elles devront donc être de classe statique.

2.2 Test du module `polygone`

2.2.1 Compilation séparée

Créez le fichier source `testpolygone.c` contenant la fonction `main` afin de tester les différentes fonctions du module `polygone`.

Effectuez la compilation séparée grâce à l'utilitaire `make`. Pour cela, écrivez un fichier de description des dépendances `Makefile1` et lancez les compilations en tapant :

```
make -f Makefile1
```

2.2.2 Visualisation avec le format SVG

En plus de l'affichage des coordonnées d'un polygone grâce à la fonction `PolygoneAfficher` et de l'écriture dans un fichier de texte grâce à la fonction `PolygoneEcriref`, vous utiliserez la fonction `PolygoneEcrireSvg` pour visualiser un polygone grâce au format SVG [2]. Cette fonction doit faire appel au module `svg` qui permet de créer des fichiers au format SVG et d'y écrire des polygones grâce à la fonction `SvgEcrirePolygone`. La documentation des fonctions fournies est donnée sous la forme de commentaires dans le fichier d'en-tête `svg.h`.

Pour afficher une image vectorielle au format SVG, utilisez le logiciel libre multiplateforme Inkscape [3] qui est installé sur les machines de TP. Certains navigateurs Web permettent d'afficher le format SVG mais la limitation du niveau de zoom ne permet pas de visualiser correctement des images de petite taille.

2.2.3 Aide à la détection de fuites de la mémoire

Afin de vérifier que votre programme n'a pas de « fuites » mémoire, c'est-à-dire gère correctement l'allocation dynamique et la libération des zones mémoires, un outil d'analyse dynamique peut vous aider. Sous Linux, vous pouvez utiliser Valgrind [4]. Par exemple, avec l'exécutable `testpolygone`, lancez

2. *Scalable Vector Graphics* : https://fr.wikipedia.org/wiki/Scalable_Vector_Graphics
3. https://fr.wikipedia.org/wiki/Inkscape
4. https://fr.wikipedia.org/wiki/Valgrind

l'analyse avec Valgrind en tapant [5] :
`valgrind ./testpolygone`
et lisez le résultat pour savoir s'il a détecté des fuites (« *leaks* »).

Sous MacOS, tapez la commande :
`leaks --atExit -- ./testpolygone`
et vérifiez le nombre de « *leaks* » à la fin de l'affichage produit.

3 Niveau 2 : manipulation de contours de régions

On suppose ici que l'on dispose d'une image au format PGM qui contient le résultat d'un étiquetage en composantes connexes (régions). Une étiquette est ici représentée par un niveau de gris. Les pixels appartenant au fond de l'image, c'est-à-dire à l'arrière-plan, ont l'étiquette (le niveau de gris) 0. Les pixels appartenant aux objets ont des étiquettes comprises entre 1 et 255. Tous les pixels qui appartiennent à la même région ont la même étiquette.

Par exemple, l'image `im2.pgm` :

```
P2
18 10
255
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   255 255 0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   255 255 255 0   0   0   0   0   0   0   0   0   0
0   0   0   255 255 255 255 255 255 0   127 127 0   0   127 127 0   0
0   0   255 255 255 255 255 255 0   0   127 127 0   0   127 127 0   0
0   255 255 255 255 255 255 255 0   0   127 127 127 127 127 127 0   0
0   255 255 255 255 255 0   0   0   0   127 127 127 127 127 127 0   0
0   0   0   255 0   0   0   0   0   0   0   127 127 127 127 0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   127 127 0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

contient deux régions : celle de droite a l'étiquette 127 et celle de gauche a l'étiquette 255.

Le contour d'une région est constitué des pixels appartenant à la région et situés sur sa bordure, c'est-à-dire ceux qui sont voisins de pixels extérieurs à la région (la notion de voisinage sort du cadre de ce projet et de l'UE).

3.1 Modules `matrice` et `image`

Récupérez les modules `matrice` et `image` que vous avez programmés lors des séances de TP.

3.2 Module `contour`

Il s'agit d'écrire un module `contour` qui utilise les modules `matrice`, `image`, `svg` et `polygone`. Il devra comporter les fonctions suivantes :

- `int ContourEtiquettes(char NomFichEtiquettes[], unsigned char Etiquettes[])` : extrait les étiquettes des régions d'une image d'étiquettes stockée dans un fichier au format PGM de nom `NomFichEtiquettes`, les stocke dans le tableau `Etiquettes` et retourne le nombre d'étiquettes trouvées (l'étiquette 0 du fond de l'image n'est pas considérée ici) ; par exemple, pour le fichier `im2.pgm` de l'exemple ci-dessus, cette fonction doit mettre les valeurs 255 et 127 dans le tableau des étiquettes et retourner 2.

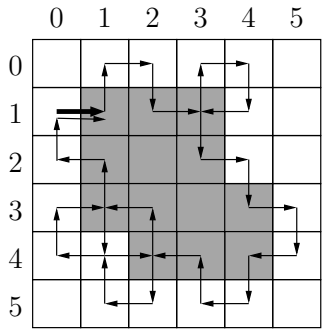
5. Si des paramètres devaient être passés à l'exécutable depuis la ligne de commande, il suffirait de les placer normalement après le nom de l'exécutable.

- `tPolygone ContourSuivi(tImage Im, unsigned char Etiquette)` : effectue, dans l'image d'étiquettes `Im`, le suivi du contour d'une région d'étiquette `Etiquette` et retourne les pixels du contour de la région sous la forme d'un polygone ; ce suivi doit être réalisé par l'algorithme de la tortue de Papert qui consiste, pour la tortue qui se déplace sur le plan de l'image [6], à :
  - commencer avec le premier pixel de la région recherchée rencontré lors d'un parcours de l'image ligne par ligne (balayage TV) ;
  - si le pixel courant est dans la région (son niveau de gris vaut l'étiquette considérée), marquer ce pixel comme faisant partie du contour et se déplacer d'un pixel vers la gauche ;
  - si le pixel courant est à l'extérieur de la région, se déplacer d'un pixel vers la droite ;
  - s'arrêter si le pixel courant est celui du départ ;
  - parcourir la liste des coordonnées des pixels trouvés et éliminer les répétitions (doublons) dues aux boucles dans le parcours.

  Considèrons, par exemple, l'image d'étiquettes `im0.pgm` suivante :

```
P2
6 6
255
0   0   0   0   0   0
0   166 166 166 0   0
0   166 166 166 0   0
0   166 166 166 166 0
0   0   166 166 166 0
0   0   0   0   0   0
```

  L'application de l'algorithme de la tortue pour l'étiquette 166 peut être illustré par le schéma suivant :



(1,1) (1,2) ~~(1,3)~~ (1,3) (2,3)
(3,4) (4,4) (4,3) ~~(4,2)~~ (4,2)
(3,2) ~~(3,1)~~ (3,1) (2,1)

  Dans la partie gauche de cette illustration, les pixels blancs, pour plus de lisibilité, correspondent à ceux qui sont à 0 (le fond de l'image) et les pixels gris sont ceux de la région d'étiquette 166. La flèche plus épaisse que les autres représente la direction de la tortue lorsqu'elle rentre pour la première fois dans la région lors du parcours initial de l'image, ligne par ligne.

  La partie droite montre les coordonnées, sous la forme (<*indice de ligne*>,<*indice de colonne*>), des pixels du contour trouvés. Les coordonnées rayées sont celles qui sont supprimées lors de la dernière étape de l'algorithme (les doublons dus à deux passages successifs sur le même pixel du contour de la région).

  Cet algorithme n'est pas parfait : son comportement n'est pas symétrique, il pose des difficultés avec les régions étroites, etc. Mais il a le mérite d'être simple.

6. La gauche et la droite ne signifient pas ici le pixel de gauche ou le pixel de droite quand on regarde l'image, mais sont relatives à la direction de déplacement de la tortue quand elle arrive sur le pixel courant.

- `void ContourExtraire(char NomFichEtiquettes[], unsigned char Etiquette[], int NbRegions, char NomFichContours[])` : localise les contours des régions, dont les `NbRegions` étiquettes sont stockées dans le tableau `Etiquette`, dans l'image d'étiquettes stockée dans le fichier au format PGM `NomFichEtiquettes` et écrit leurs coordonnées dans le fichier de texte `NomFichContours`. Le format du fichier résultat doit être le suivant :

```
NbRegions
NbSommetsRegion1
x1␣y1
x2␣y2
...
NbSommetsRegion2
x1␣y1
x2␣y2
...
```

- `void ContourEcrireSurImageSvg(char NomFichContours[], char NomFichImage[], int NbCol, int NbLig, tStyle *pStyle, char NomFichSVG[])` : écrit un ensemble de contours stocké dans le fichier `NomFichContours` sur une image stockée au format PNG dans le fichier `NomFichImage`, de largeur `NbCol` et de hauteur `NbLig`, avec le style d'adresse `pStyle`, dans le fichier au format SVG `NomFichSVG`.

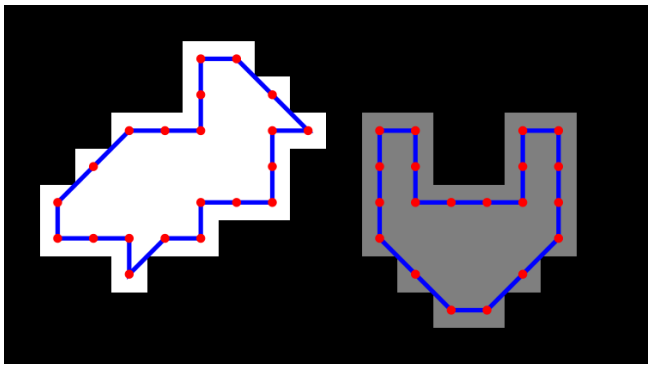
3.3 Test du module `contour`

Créez un fichier source `contex.c` contenant la fonction `main` afin de tester les différentes fonctions du module `contour`. Créez le fichier `Makefile2` destiné à produire l'exécutable `contex` en prenant en compte les dépendances liées aux différents modules utilisés (`matrice`, `image`, `svg`, `polygone`, `contour`).

Quand vous aurez mis au point les différentes fonctions du module, modifiez le fichier `contex.c` afin de produire un exécutable dont l'appel suivant (n'oubliez pas de vérifier le nombre d'arguments reçus) :

```
./contex im2.pgm im2.png res2.ct res2.svg
```

doit écrire dans le fichier de texte `res2.ct`, au format décrit pour la fonction `ContourExtraire`, les coordonnées des pixels des contours des deux régions de `im2.pgm` et, dans le fichier `res2.svg`, le résultat permettant la visualisation suivante :



4 Niveau 3 : extraction des régions d'une image

L'objectif est ici de produire, à partir d'une image de niveaux de gris, les données qui sont en entrée du niveau 2, c'est-à-dire une image au format PGM qui contient le résultat d'un étiquetage des régions (composantes connexes).

Il s'agit d'écrire un module `region`, qui utilise les modules `matrice` et `image`, et implémente deux étapes principales : la détection des objets et l'étiquetage des régions.

4.1 Détection des objets par binarisation

Pour simplifier, on suppose que l'image initiale contient des objets clairs placés sur un fond sombre. Un niveau de gris, que l'on appelle un seuil, permet de binariser l'image, c'est-à-dire de déterminer si chaque pixel appartient à un objet ou au fond. Pour cela, le module `region` doit comporter la fonction suivante :

- `tImage RegionDetection(tImage Im, unsigned char Seuil)` : retourne une nouvelle image résultat (ou `NULL` en cas de problème) dans laquelle les pixels du fond ont le niveau de gris 0 et les pixels des objets ont le niveau de gris 255. Les pixels du fond sont ceux dont le niveau de gris dans l'image initiale `Im` est strictement inférieur à `Seuil` et les autres sont ceux qui appartiennent aux objets.

4.2 Étiquetage des régions

L'objectif de l'étiquetage est, à partir d'une image dans laquelle les pixels du fond valent 0 et ceux des objets valent 255, de créer une nouvelle image d'étiquettes dans laquelle tous les pixels appartenant à la même région, c'est-à-dire à la même composante connexe, ont le même niveau de gris qui constitue l'étiquette de la région. Pour cela, vous appliquerez l'algorithme suivant :

- Initialiser l'image des étiquettes à 0 qui est l'étiquette associée aux pixels du fond.
- Parcourir l'image initiale ligne par ligne :
  - si le pixel courant appartient à un objet :
    - ▷ si le pixel courant n'a pas de voisin déjà étiqueté, créer une nouvelle étiquette et l'affecter à ce pixel ;
    - ▷ si le pixel courant a un seul voisin déjà étiqueté, lui affecter cette étiquette ;
    - ▷ si le pixel courant a plusieurs voisins déjà étiquetés avec la même étiquette, lui affecter cette étiquette ;
    - ▷ si le pixel courant a plusieurs voisins déjà étiquetés avec des étiquettes différentes, lui affecter l'une de ces étiquettes et mémoriser que ces étiquettes sont équivalentes.
- Faire un parcours de l'image des étiquettes et affecter une seule et même étiquette lorsqu'il y a des étiquettes équivalentes.

4.2.1 Choix du voisinage et hypothèses

On choisit de considérer uniquement le « 4-voisinage », c'est-à-dire que les voisins d'un pixel sont le pixel situé au dessus, le pixels situé à gauche, le pixel situé en dessous et le pixel situé à droite.

Dans l'algorithme décrit ci-dessus, pour savoir si des voisins sont déjà étiquetés, le parcours étant un balayage ligne par ligne, de la première à la dernière ligne, de la gauche vers la droite pour chaque ligne, alors il suffit d'examiner le pixel du dessus (voisin qui a été traité lors du parcours de la ligne précédente) et le pixel de gauche (voisin qui vient d'être traité).

Pour simplifier l'examen des voisins, nous supposons que tous les pixels de la première ligne et tous les pixels de la première colonne de l'image appartiennent au fond. Le parcours peut donc commencer à la deuxième ligne et à la deuxième colonne.

4.2.2 Exemple

Soit l'image des régions suivante :

0	0	0	0	0	0	0	0	0	0
0	255	255	0	0	0	255	255	0	0
0	255	255	0	0	0	255	0	0	0
0	255	255	255	255	0	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	255	255	255	0	255	255	0
0	0	0	0	0	0	0	255	0	0
0	0	0	0	0	0	0	0	0	0

Le premier parcours donne l'image des étiquettes suivante :

0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	2	2	0	0
0	1	1	0	0	0	2	0	0	0
0	1	1	1	1	0	2	0	0	0
0	0	0	0	0	3	0	0	0	0
0	0	0	4	4	3	0	5	5	0
0	0	0	0	0	0	0	5	0	0
0	0	0	0	0	0	0	0	0	0

Équivalences :
- Les étiquettes 3 et 4 sont équivalentes.

Dans cet exemple, lorsque les deux voisins sont déjà étiquetés avec des étiquettes différentes, on a choisi d'attribuer l'étiquette du voisin de dessus.

Le parcours final destiné à fusionner les étiquettes équivalentes, donne :

0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	2	2	0	0
0	1	1	0	0	0	2	0	0	0
0	1	1	1	1	0	2	0	0	0
0	0	0	0	0	3	0	0	0	0
0	0	0	3	3	3	0	5	5	0
0	0	0	0	0	0	0	5	0	0
0	0	0	0	0	0	0	0	0	0

4.2.3 Gestion des étiquettes équivalentes

La gestion des étiquettes équivalentes, y compris dans le cas du 8-voisinage, est généralement réalisé grâce à la structure de données *union-find*[7] qui permet de représenter une relation d'équivalence.

Une implémentation simple consiste à stocker les équivalences d'étiquettes dans un tableau d'étiquettes, c'est-à-dire un tableau d'entiers. Si la case numéro `i` du tableau vaut `j`, cela signifie que l'étiquette `i` est équivalente à l'étiquette `j`. Par ailleurs, chaque étiquette est équivalente à elle même.

Ajoutez au module `region` les fonctions suivantes :

- `unsigned char RegionAjoutEtiquette(unsigned char Equivalence[], int *pNbEtiquettes)` : ajoute une nouvelle étiquette dans le tableau `Equivalence` et retourne cette nouvelle étiquette ; `pNbEtiquettes` est l'adresse du nombre d'éléments dans le tableau. Par exemple, si le tableau des équivalences contient 2 éléments `{0,1}`, alors l'appel à cette fonction retourne `2` et le tableau contient maintenant 3 éléments : `{0,1,2}`.
- `unsigned char RegionEtiquetteReference(unsigned char Etiquette, unsigned char Equivalence[])` : recherche, dans le

7. https://fr.wikipedia.org/wiki/Union-find

tableau d'équivalences `Equivalence`, et retourne l'étiquette de référence associée à l'étiquette `Etiquette`. Une étiquette est une étiquette de référence si `Equivalence[i] == i`. Par exemple, si l'étiquette 1 est équivalente à l'étiquette 0, si l'étiquette 2 est équivalente à l'étiquette 1 et si l'étiquette 4 est équivalente à l'étiquette 1, le tableau d'équivalences vaut :

0	0
1	0
2	1
3	3
4	1

L'étiquette de référence de l'étiquette 4 est 0 car :

- Est-ce que `Equivalence[4] == 4` ? Non.
- Alors on examine `Equivalence[Equivalence[4]]` qui vaut `Equivalence[1]`.
- Est-ce que `Equivalence[1] == 1` ? Non.
- Alors on examine `Equivalence[Equivalence[1]]` qui vaut `Equivalence[0]`.
- Est-ce que `Equivalence[0] == 0` ? Oui.
- Alors `0` est l'étiquette de référence.

De même, l'étiquette de référence des étiquettes 4,2,1 et 0, est l'étiquette 0.

La recherche de l'étiquette de référence d'une étiquette `e` peut se résumer ainsi :
Tant que `Equivalence[e] != e` *Faire*
`e` ← `Equivalence[e]`

- `void RegionEtiquettesEquivalentes(unsigned char Etiquette1, unsigned char Etiquette2, unsigned char Equivalence[])` : prend en compte, dans le tableau `Equivalence`, l'équivalence entre les étiquettes `Etiquette1` et `Etiquette2`. Pour cela, il faut :
  - Rechercher `Ref1`, l'étiquette de référence de l'étiquette `Etiquette1`.
  - Rechercher `Ref2`, l'étiquette de référence de l'étiquette `Etiquette2`.
  - Indiquer que l'étiquette `Ref2` est équivalente à l'étiquette `Ref1`.

  Avec l'exemple précédent, pour indiquer que l'étiquette 3 est équivalente à l'étiquette 4 :
  - L'étiquette de référence de l'étiquette `3` est l'étiquette `3`.
  - L'étiquette de référence de l'étiquette `4` est l'étiquette `0`.
  - L'étiquette `0` est équivalente à l'étiquette `3` : `Equivalence[0]` ← `3`. Le tableau mis à jour vaut :

0	3
1	0
2	1
3	3
4	1

    Maintenant, l'étiquette de référence de l'étiquette 4 est l'étiquette 3.

  Pour tester vos fonctions, vous pouvez :
  - Partir d'un tableau d'équivalences vide.
  - Ajouter les treize étiquettes $0, \cdots, 12$ avec la fonction `RegionAjoutEtiquette`.
  - Ajouter, avec la fonction `RegionEtiquettesEquivalentes`, les équivalences suivantes : $(1,2)$, $(4,1)$, $(10,11)$, $(10,4)$, $(7,8)$ et $(6,7)$.

- Vérifier que la fonction `RegionEtiquetteReference` retourne bien les étiquettes de référence des étiquettes 0, $\cdots$,12 qui sont les étiquettes : 0, 10, 10, 3, 10, 5, 6, 6, 6, 9, 10, 10, 12.

- `tImage RegionEtiquetage(tImage ImRegions)` : effectue l'étiquetage des régions de l'image des régions `ImRegions` dans laquelle les pixels du fond valent 0 et les pixels des objets valent 255. La première ligne et la première colonne de `ImRegions` sont supposées faire partie du fond. Cette fonction doit retourner une nouvelle image résultat contenant des étiquettes : 0 pour le fond et des entiers compris entre 1 et 255 pour les régions correspondant aux différents objets ou `NULL` en cas de problème.

4.3 Test du module `region`

Créez un fichier source `regex.c` contenant la fonction `main` afin de tester les différentes fonctions du module `region`. Créez le fichier `Makefile3` destiné à produire l'exécutable `regex` en prenant en compte les dépendances liées aux différents modules utilisés (`matrice`, `image`).

Quand vous aurez mis au point les différentes fonctions du module, modifiez le fichier `regex.c` afin de produire un exécutable dont l'appel suivant (n'oubliez pas de vérifier le nombre d'arguments reçus) :

```
./regex postit.pgm 175 postit-reg.pgm postit-et.pgm
```

doit écrire dans le fichier `postit-reg.pgm` l'image binarisée et dans `postit-et.pgm` l'image avec les régions étiquetées.

Vous pouvez ensuite détecter les contours en utilisant :

```
./contex postit-et.pgm postit.png postit.ct postit.svg
```

Si vous souhaitez, dans le fichier SVG, superposer les contours et l'image des régions, vous pouvez la convertir au format PNG, en utilisant par exemple ImageMagick et la commande :

```
convert postit-reg.pgm postit-reg.png
```

puis :

```
./contex postit-et.pgm postit-reg.png postit.ct postit-reg.svg
```

Vous pouvez en faire de même avec l'image `pieces.pgm` et le seuil de binarisation 85.