Programmation parallèle

Cycle 2020-2021

ECE – Ing5 – Systèmes Embarqués

Etienne Hamelin

ehamelin@inseec-edu.com

Alexandre Berne

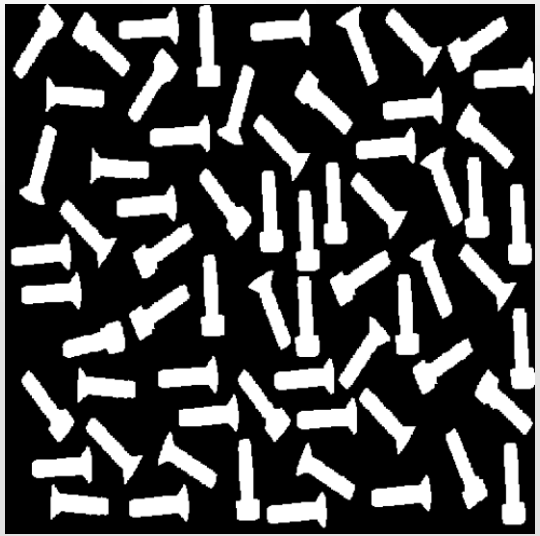
aberne@inseec-edu.com

Projet

ÉTIQUETAGE EN COMPOSANTES CONNEXES

Etiquetage en composantes connexes

- Objectif: identifier, compter, analyser les objets “blobs” dans une image



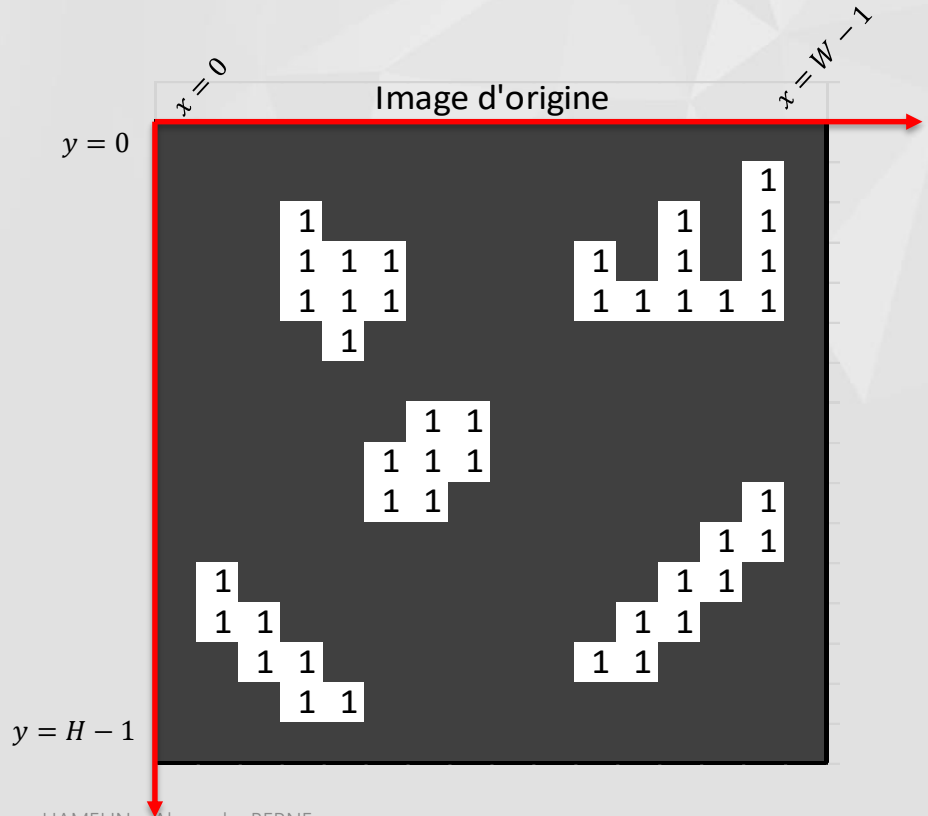
```
…
Connected component 64: bounding box (240,928),(355,999), 4476 pixels
Connected component 65: bounding box (508,942),(621,1011), 4400 pixels
…
```

Spécification

- Entrée: une image binaire
- Sortie
  - nombre de composantes connexes (« blobs »),
  - pour chacune:
    - position, taille (nombre de pixels)
  - Et, pour vérifier visuellement:
    - Une image où chaque composante connexe a une couleur distincte

Rappels: représentation d'images

- Image $W \times H$: $I_{x,y}$ = couleur du pixel (x, y)
- Pixel (0,0) : en haut à gauche
- Image noir & blanc binaire: $I_{x,y} = 0 \; ou \; 1$
- Image niveaux de gris:
  $I_{x,y} = 0$ à 255 ou 0 à 65535
- Image couleur
  $I_{x,y} = (r, g, b)$ ou $0 \leq r, g, b \leq 255$



- Librairie fournie: manipulation d'images selon plusieurs formats
- Types d'images supportés
  - IMAGE_BITMAP, IMAGE_GRAYSCALE_8, IMAGE_GRAYSCALE_16, IMAGE_RGB_888
- Lecture/écriture de fichiers NetBPM
  - Formats PBM (bitmap), PGM (grayscale), PPM (couleur)

Principales API

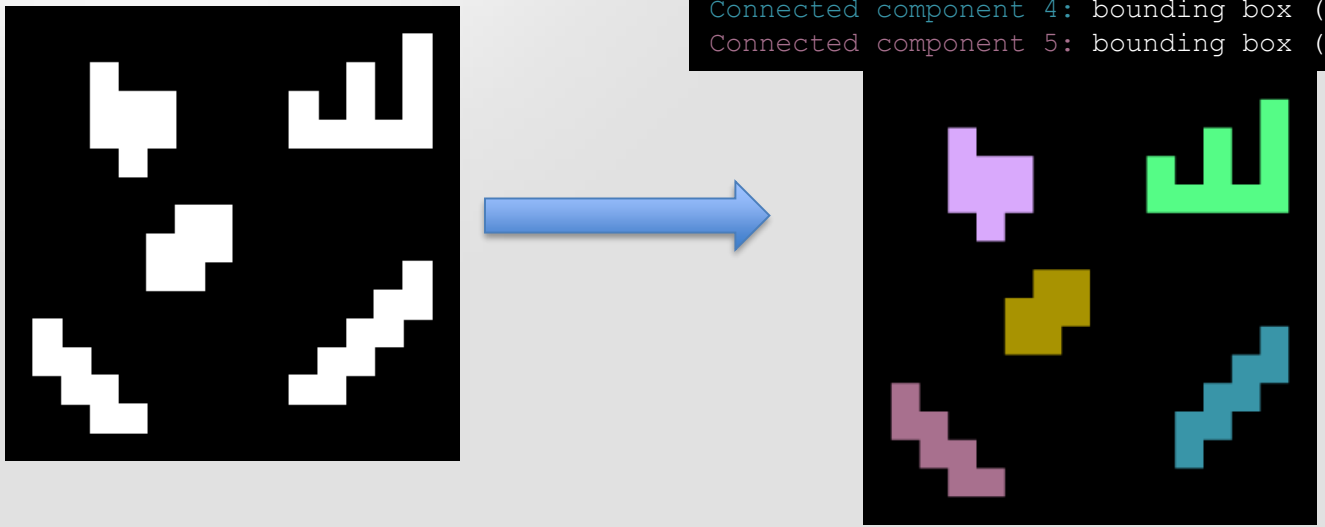
```
/* Créer une image 320x200 pixels noir & blanc */
image_t *img = image_new(320, 200, IMAGE_BITMAP);
/* lire, écrire un pixel noir & blanc */
bool b = image_bmp_getpixel(img, x, y).bit;
image_bmp_setpixel(img, x, y, (color_t){.bit = 1});

/* Créer une image couleur */
image_t *img = image_new(320, 200, IMAGE_RGB_888);
/* lire, écrire un pixel couleur */
uint8_t r = image_rgb_getpixel(img, x, y).rgb.r;
image_rgb_setpixel(img, x, y, (color_t){.rgb = {.r = 255, .g = 128, .b = 0}});

/* Paramètres utiles */
img->width, img->height,

/* Lire, écrire un fichier NetBPM */
image_t *img = image_new_open("file.ppm");
image_save_ascii(img, "file.ppm")
image_save_binary(img, "file.ppm")
```

Parenthèse: comment écrire du C « orienté objet »

```c
typedef struct object_s {
        int champ;
        int (*methode)(struct object_s *self, int param);
} object_t;

object_t *object_new() {// constructor
        object_t *self = calloc(1, sizeof(object_t));
        self->methode = methode_par_defaut;
        return self;
}

object_t *derived_object_new() {// constructor of derived class
        object_t *self = object_new(); // call default constructor
        self->methode = methode_derivee; // override method
        return self;
}

int methode_par_defaut(object_t *self, int param) {// base class method
        // do something with self
}

int methode_derivee(object_t *self, int param) {// derived class method
        // do something else
}

int main() {
        object_t *objA = object_new();
        object_t *objB = derived_object_new();
        objA->methode(objA, param);
        objB->methode(objB, param);
        object_delete(objA);
        object_delete(objN);
}
```

Algorithme de Rosenfeld

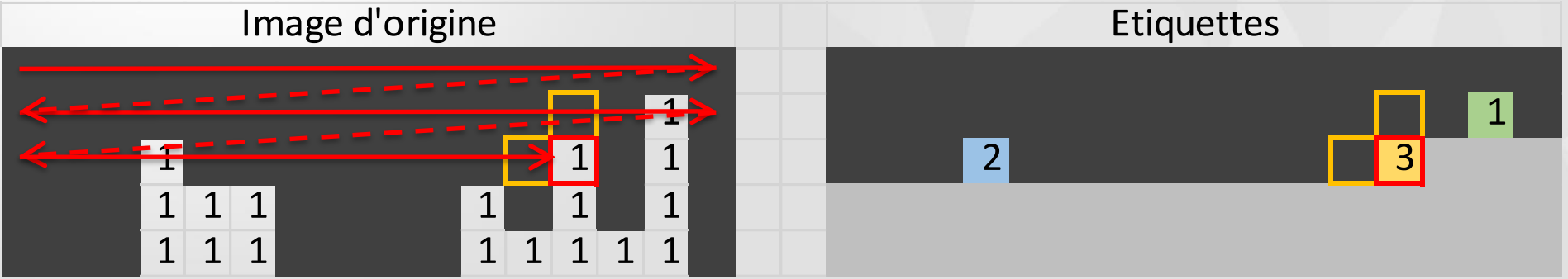
```
Connected component 1: bounding box (10,1),(14,4), 11 pixels
Connected component 2: bounding box (3,2),(5,5), 8 pixels
Connected component 3: bounding box (5,7),(7,9), 7 pixels
Connected component 4: bounding box (11,9),(14,13), 8 pixels
Connected component 5: bounding box (1,11),(4,14), 7 pixels
```



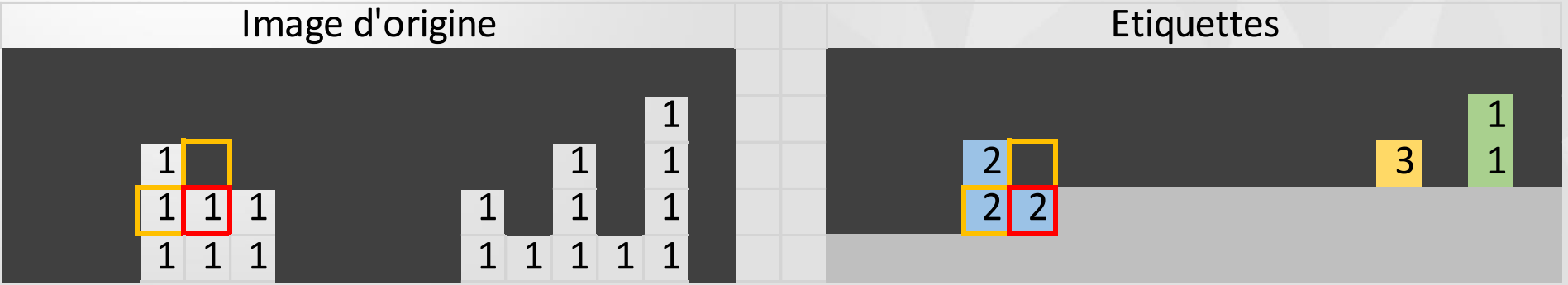
Algorithme de Rosenfeld & Pfalz (1966)

1. En une passe sur l'image, on marque les pixels avec des étiquettes temporaires
    - A ce stade, plusieurs étiquettes « équivalentes » peuvent désigner la même composante connexe
2. Par analyse des équivalences, on désigne une étiquette définitive pour chaque classe d'étiquettes (çàd chaque composante connexe)
3. On remplace les étiquettes temporaires par l'étiquette finale de la composante connexe

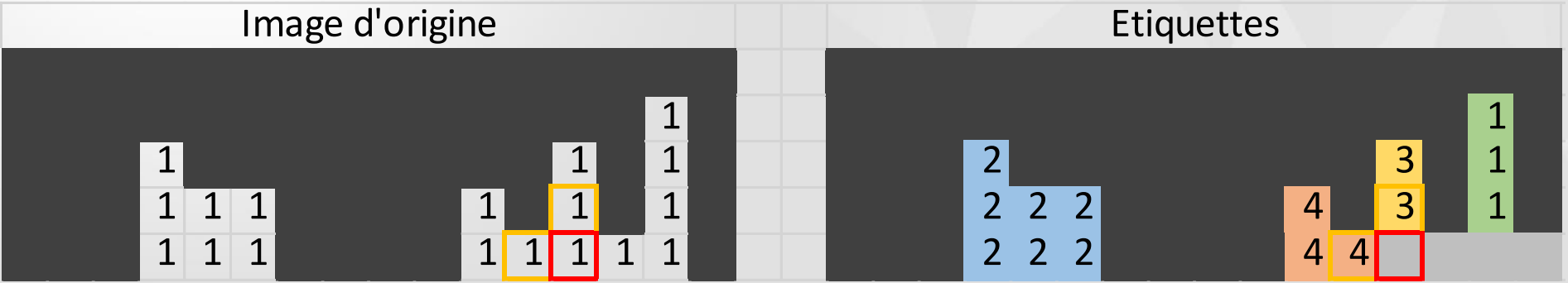
Illustration



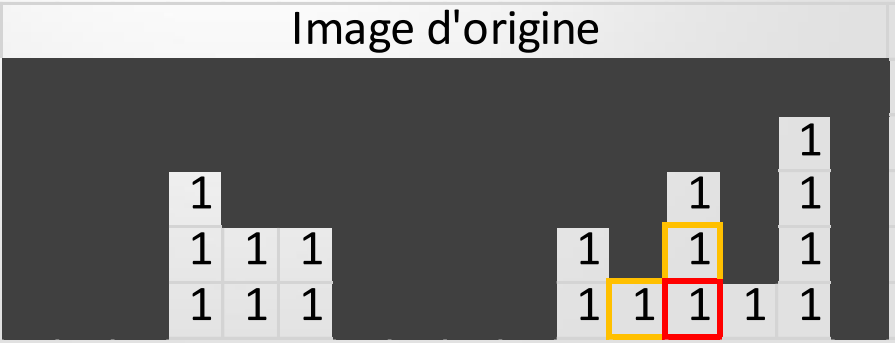
Etiquetage en composantes connexes



Etiquetage en composantes connexes



Etiquetage en composantes connexes

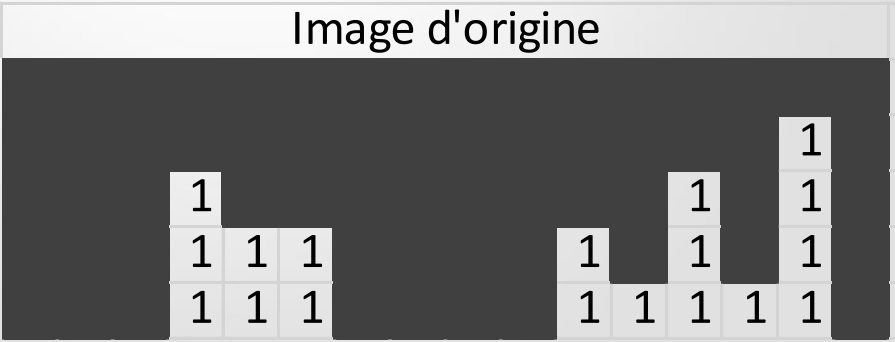


Equivalences

e	1	2	3	4
T[e]	1	2	3	3

$e_3 \sim e_4$

Etiquetage en composantes connexes

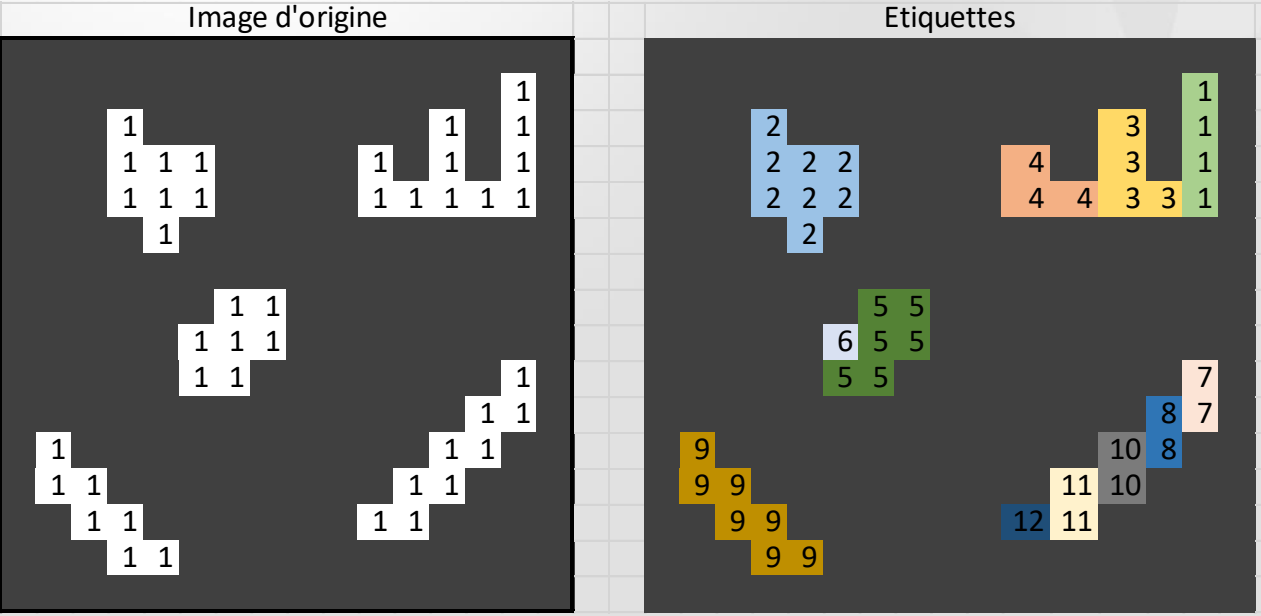


Equivalences

e	1	2	3	4
T[e]	1	2	1	3

$e_1 \sim e_3 \sim e_4$

Fin de la première étape



Equivalences												
e	1	2	3	4	5	6	7	8	9	10	11	12
T[e]	1	2	1	3	5	5	7	7	9	8	10	11

Etiquettes équivalentes

$e_1 \sim e_3 \sim e_4$

$e_5 \sim e_6$

$e_7 \sim e_8 \sim e_{10} \sim e_{11} \sim e_{12}$

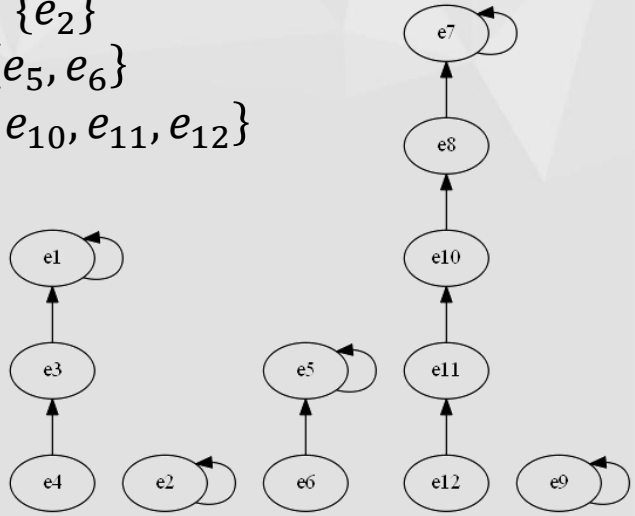
Deuxième étape: réduction des équivalences

Equivalences												
e	1	2	3	4	5	6	7	8	9	10	11	12
T[e]	1	2	1	3	5	5	7	7	9	8	10	11

Racine des classes d'équivalences												
e	1	2	3	4	5	6	7	8	9	10	11	12
T[e]	1	2	1	1	5	5	7	7	9	7	7	7
N[e]	1	2			3		4		5			

Renumérotation												
e	1	2	3	4	5	6	7	8	9	10	11	12
N[e]	1	2	1	1	3	3	4	4	5	4	4	4

$$c_1 = \{e_1, e_3, e_4\}$$
$$c_2 = \{e_2\}$$
$$c_3 = \{e_5, e_6\}$$
$$c_4 = \{e_7, e_8, e_{10}, e_{11}, e_{12}\}$$



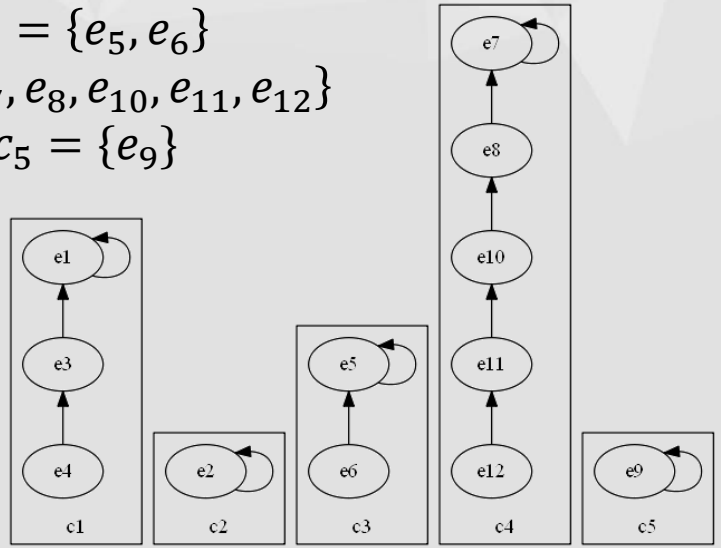
Deuxième étape: réduction des équivalences & renumérotation

Equivalences												
e	1	2	3	4	5	6	7	8	9	10	11	12
T[e]	1	2	1	3	5	5	7	7	9	8	10	11

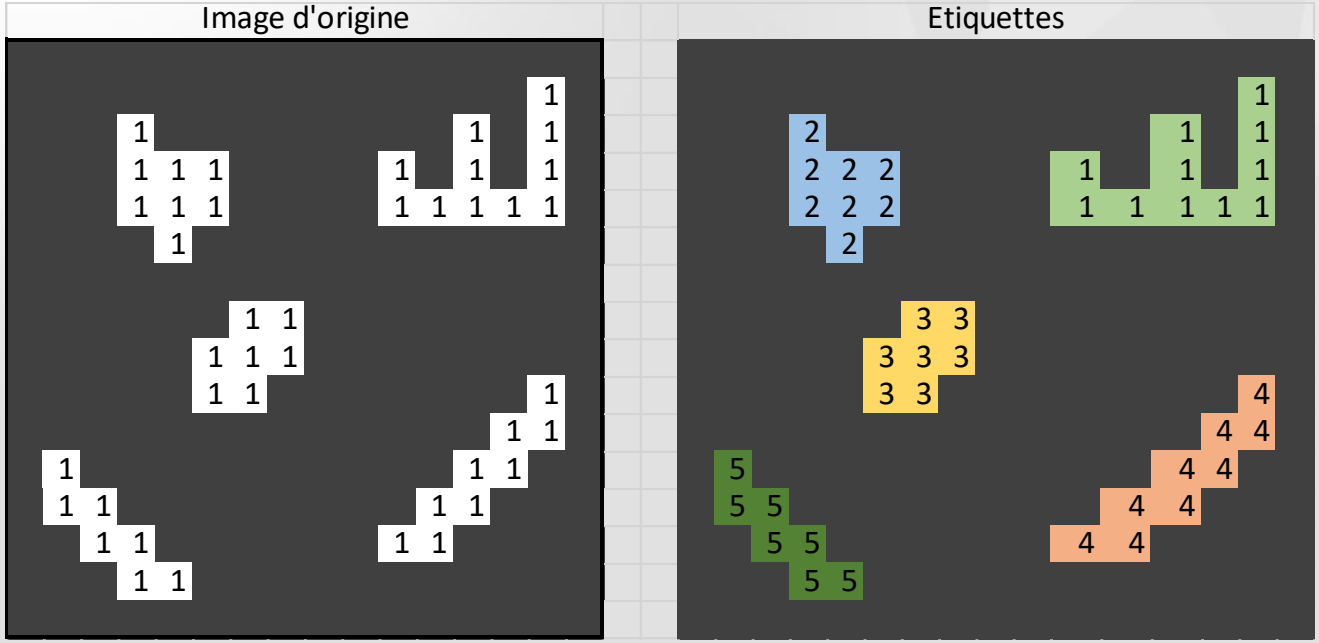
Racine des classes d'équivalences												
e	1	2	3	4	5	6	7	8	9	10	11	12
T[e]	1	2	1	1	5	5	7	7	9	7	7	7
N[e]	1	2			3		4		5			

Renumérotation												
e	1	2	3	4	5	6	7	8	9	10	11	12
N[e]	1	2	1	1	3	3	4	4	5	4	4	4

$$c_1 = \{e_1, e_3, e_4\}$$
$$c_2 = \{e_2\}$$
$$c_3 = \{e_5, e_6\}$$
$$c_4 = \{e_7, e_8, e_{10}, e_{11}, e_{12}\}$$
$$c_5 = \{e_9\}$$



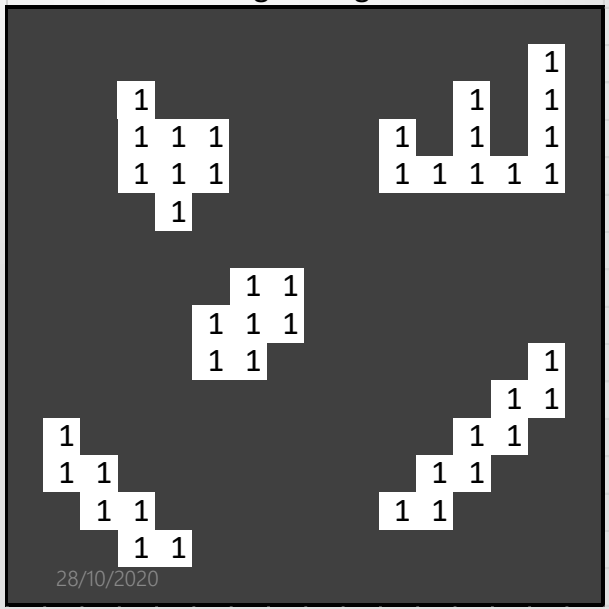
Deuxième balayage: remplacement des étiquette temporaires



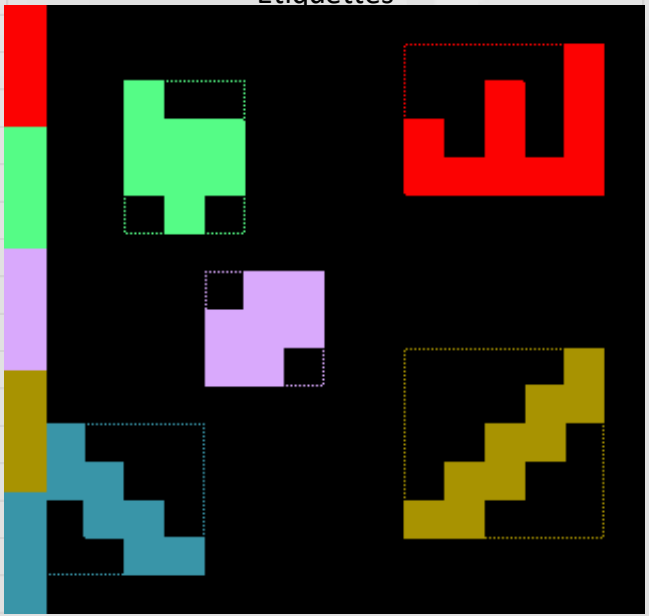
Analyses des composantes connexes

Ne font pas partie de l'algorithme de Rosenfeld, mais pratiques pour la visualisation
Pour chaque composante connexe:
- remplacer l'etiquette par une couleur unique
- calculer ses dimensions (nb de pixels, rectangle *bounding box*)

Image d'origine



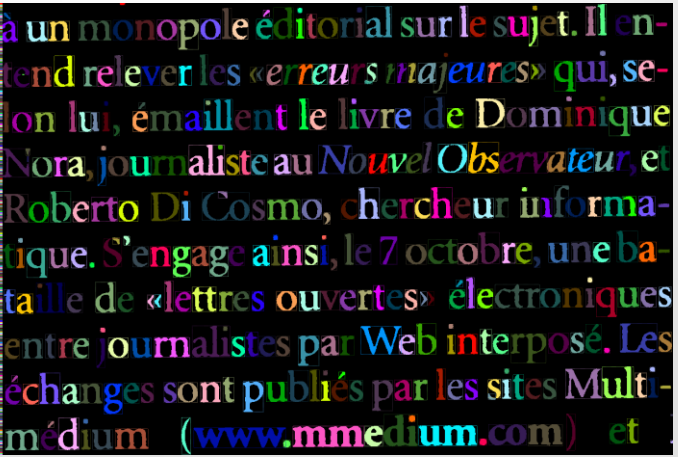
Etiquettes



Etiquetage en composantes connexes

- Pour vous familiariser avec le code fourni
  - Nettoyer
    ```
    make clean
    ```
  - Compiler
    ```
    make
    ```
  - Exécuter
    ```
    ./main img/test1.pbm
    ```
  - Visualiser les résultats
    ```
    display color.ppm
    cat classes.pgm
    ```
  - Débugger les problèmes mémoire (segfault, free, …)
    ```
    valgrind ./main img/test1.pbm
    ```

Quelques exemples



Structure du code fourni

`main() [main.c]`

`+ test_image_connected_components(filename) [main.c]`

`  + image_connected_components [image_connected_components.c]`

`    + ccl_temp_tag` 1° étape, balayage, attribution d'étiquettes temporaires

`    + ccl_reduce_equivalences` 2° étape, réduction des équivalences & renumérotation

`    + ccl_retag` 3° étape, remplacement des étiquettes temporaires par définitives

`    + ccl_analyze` analyse (des dimensions) des composantes connexes

`    + ccl_draw_colors` génération d'une image en couleurs

- Et en parallèle maintenant?

Etiquetage en composantes connexes

- Et en parallèle?
  - On gardes les étapes séquentielles
  - Etapes “lourdes” (balayages de l’image) découpées en threads
- Principal défi:
  - Etiquettes équivalentes entre plusieurs threads

Contre-exemple

Thread0

Thread1

Thread2

Thread3