TD4 – éléments de corrigé (algorithmes)

Exercice 1

Algorithme 0 : CreerImage2DBooleensDiagonaleNulle
Principe : Créer un tableau 2D parcourir i (lignes) et j (colonnes) lorsque i vaut j on est sur la diagonale donc mettre faux à l'indice i,j sinon mettre vrai.

```
Entrée :        - hauteur : entier
                - largeur : entier
Local :         - i : entier
                - j : entier
Sortie :        image : tableau 2D de hauteur * largeur booléens
Début
|       Soit image = tableau 2D de hauteur * largeur booléens
|       Pour i de 0 à hauteur − 1 avec un pas de +1 Faire
|       |       Pour j de 0 à largeur − 1 avec un pas de +1 Faire
|       |       |       Si i == j Alors
|       |       |       |       image[i,j] = faux
|       |       |       Sinon
|       |       |       |       image[i,j] = vrai
|       |       |       Fin Si
|       |       Fin Pour
|       Fin Pour
|       Retourner image
Fin
```

Algorithme 1 : AfficherImage2DBooleens
Principe : Parcourir les lignes (i) et les colonnes (j) avec deux boucles imbriquées et afficher ' ' si la case i,j vaut vrai et '*' sinon. Retour à la ligne après chaque boucle interne (colonnes).

```
Entrée :        - image : tableau 2D de hauteur * largeur booléens
                - hauteur : entier
                - largeur : entier
Local :         - i : entier
                - j : entier
Sortie :        void
Début
|       Pour i de 0 à hauteur − 1 avec un pas de +1 Faire
|       |       Pour j de 0 à largeur − 1 avec un pas de +1 Faire
|       |       |       Si image[i,j] == vrai Alors
|       |       |       |       AfficherCaractère('*')
|       |       |       Sinon
|       |       |       |       AfficherCaractère(' ')
|       |       |       Fin Si
|       |       Fin Pour
|       |       AfficherCaractère('\n')
|       Fin Pour
Fin
```

Algorithme 2 : AfficherNombrePixelsBlancsEtNoirsImage2DBooleens

```
Entrée :        - image : tableau 2D de hauteur * largeur booléens
                - hauteur : entier
                - largeur : entier
Local :         - i : entier
                - j : entier
                - nBlancs : entier
                - nNoirs : entier
Sortie :        void
Début
|       Soit nBlancs = 0
|       Soit nNoirs = 0
|       Pour i de 0 à hauteur − 1 avec un pas de +1 Faire
|       |       Pour j de 0 à largeur − 1 avec un pas de +1 Faire
|       |       |       Si image[i,j] == vrai Alors
|       |       |       |       nBlancs = nBlancs + 1
|       |       |       Sinon
|       |       |       |       nNoirs = nNoirs + 1
|       |       |       Fin Si
|       |       Fin Pour
|       Fin Pour
|       AfficherChaîne("Il y a {nBlancs} pixels blancs et {nNoirs} pixels noirs dans l'image.")
Fin
```

Algorithme 3 : InverserImage (ici inversion des valeurs vrai ↔ faux)

Principe : Créer une image inverse de même dimensions que l'image d'entrée. Parcourir les lignes et colonnes de l'image et écrire à la même position dans l'image inverse la négation du booléen.

```
Entrée :        - image : tableau 2D de hauteur * largeur booléens
                - hauteur : entier
                - largeur : entier
Local :         - i : entier
                - j : entier
Sortie :        inverse : tableau 2D de hauteur * largeur booléens
Début
|       Soit inverse : tableau 2D de hauteur * largeur booléens
|       Pour i de 0 à hauteur − 1 avec un pas de +1 Faire
|       |       Pour j de 0 à largeur − 1 avec un pas de +1 Faire
|       |       |       Si image[i,j] == vrai Alors
|       |       |       |       inverse[i,j] = faux
|       |       |       Sinon
|       |       |       |       inverse[i,j] = vrai
|       |       |       Fin Si
|       |       Fin Pour
|       Fin Pour
|       Retourner inverse
Fin
```

Algorithme 3 : InverserImage (ici inversion des dimensions)

```
Entrée :        - image : tableau 2D de hauteur * largeur booléens
                - hauteur : entier
                - largeur : entier
Local :         - i : entier
                - j : entier
Sortie :        inverse : tableau 2D de largeur * hauteur booléens
Début
|       Soit inverse = tableau 2D de largeur * hauteur booléens
|       Pour i de 0 à hauteur − 1 avec un pas de +1 Faire
|       |       Pour j de 0 à largeur − 1 avec un pas de +1 Faire
|       |       |       Si image[i,j] == vrai Alors
|       |       |       |       inverse[j,i] = faux
|       |       |       Sinon
|       |       |       |       inverse[j,i] = vrai
|       |       |       Fin Si
|       |       Fin Pour
|       Fin Pour
|       Retourner inverse
Fin
```

Algorithme 4 : ComparerImages2DBooleens

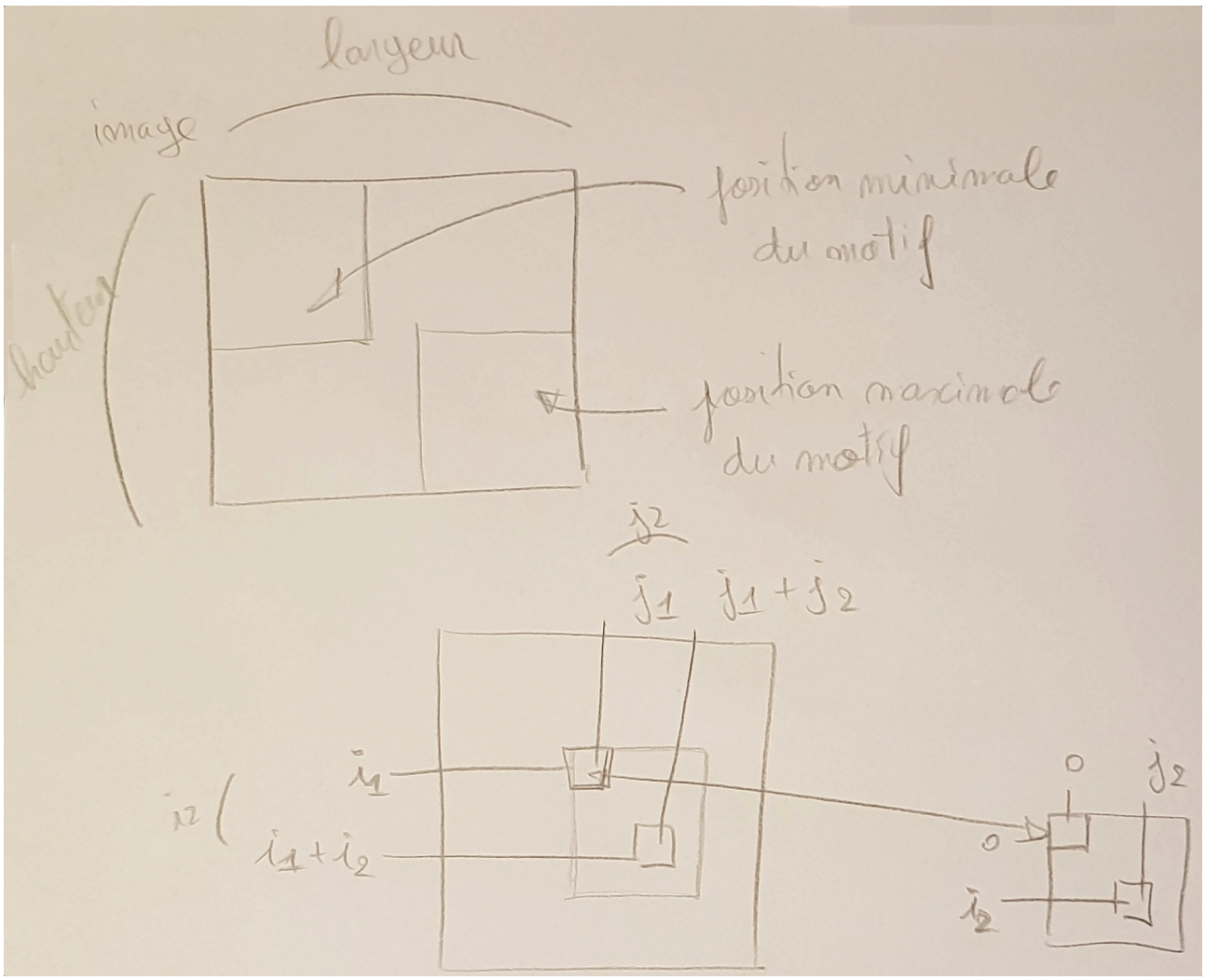
Principe : Parcourir les lignes et colonnes de l'image 1 et regarder si la case à l'indice i,j dans l'image 1 est égale à la case i,j dans l'image 2. Si une différence est trouvée, s'arrêter et afficher faux, sinon afficher vrai.

Précondition : les deux images partagent les mêmes dimensions (hauteurs et largeurs)

```
Entrée :        - image1 : tableau 2D de hauteur * largeur booléens
                - image2 : tableau 2D de hauteur * largeur booléens
                - hauteur : entier
                - largeur : entier
Local :         - i : entier
                - j : entier
                - diff : booléen
Sortie :        void
Début
|       Soit i = 0
|       Soit quitter = faux
|       Soit diff = faux
|       Tant Que i < hauteur OU quitter == faux Faire
|       |       Soit j = 0
|       |       Pour j de 0 à largeur − 1  Faire
|       |       |       Si image1[i,j] != image2[i,j] Alors
|       |       |       |       diff = diff + 1
|       |       |       Fin Si
|       |       |       j = j + 1
|       |       Fin Pour
|       |       i = i + 1
|       Fin Tant Que
|       Si diff == vrai Alors
|       |       AfficherChaîne("Les images sont différentes ! Il y a {diff} différences.")
|       Sinon
|       |       AfficherChaîne("Les images sont identiques !")
|       Fin Si
Fin
```

Algorithme 5 : RechercheMotifDansImage2DBooleens
Principe : cf mon petit schéma ci-après :)
Précondition : le motif doit être plus petit que l'image (hauteurImage ≥ hauteurMotif ET largeurImage ≥ largeurMotif)
Entrée : - image : tableau 2D de hauteurImage * largeurImage booléens
- motif : tableau 2D de hauteurMotif * largueurMotif booléens
- hauteurImage : entier
- largeurImage : entier
- hauteurMotif : entier
- largeurMotif : entier
Local : - i1 : entier
- j1 : entier
- i2 : entier
- j2 : entier
- quitter : booléen
- diff : booléen
- nMotif : entier
Sortie : void
Début

```
|       Soit nMotif = 0
|       Pour i1 de 0 à hauteurImage – hauteurMotif avec un pas de +1 Faire
|       |       Pour j1 de 0 à largeurImage – largeurMotif avec un pas de +1 Faire
|       |       |       Soit i2 = 0
|       |       |       Soit quitter = faux
|       |       |       Soit diff = faux
|       |       |       Tant Que i2 < hauteurMotif OU quitter == faux Faire
|       |       |       |       Soit j2 = 0
|       |       |       |       Tant Que j2 < largeurMotif OU quitter == faux Faire
|       |       |       |       |       Si image[i1+i2, j1+j2] != motif[i2,j2] Alors
|       |       |       |       |       |       diff = vrai
|       |       |       |       |       |       quitter = vrai
|       |       |       |       |       Fin Si
|       |       |       |       |       j2 = j2 + 1
|       |       |       |       Fin Tant Que
|       |       |       |       i1 = i1 + 1
|       |       |       Fin Tant Que
|       |       |       Si diff == faux Alors
|       |       |       |       nMotif = nMotif + 1
|       |       |       Fin Si
|       |       Fin Pour
|       Fin Pour
|       AfficherChaîne("Le motif apparaît {nMotif} fois dans l'image.")
Fin
```


largeur
image
hauteur
position minimale du motif
position maximale du motif
j_2
j_1
$j_1 + j_2$
i_1
i_2
$i_1 + i_2$
0
0
j_2
i_2