ISM MP, Informatique
Année 2025/2026

TP Informatique évalué 01 - 2h

Télécharger le fichier `TP01_AUTO.py` depuis l'arborescence `INFORMATIQUE/DS/TP01 EVALUE` sur le site de la classe puis travailler directement sur ce fichier en le complétant.
Après avoir vérifié que le script s'exécute correctement, déposer le fichier sur le site de la classe (le nom du fichier n'importe pas).

⚠ Dans tout le script, on utilisera exclusivement les variables globales `blanc`, `gris`, `noir` définies dans le ficher `TP01_AUTO.py` pour affecter des couleurs à un pixel. De même, on utilisera exclusivement les variables globales `visite`, `non_visite` pour modifier l'état visité ou non visité d'un sommet `s` dans un dictionnaire `etat` :

```
config[i][j]=gris  # le pixel (i,j) de config est grisé
etat[s]=visite     # le sommet s passe à l'état visité
```

Exercice 1

Dans ce problème, on considère une image constituée de pixels blancs et noirs. On la représente par une liste de listes de 0 et de 2 où les 0 désignent les pixels blancs et les 2 des pixels noirs. Un pixel est représenté par un tuple de ses coordonnées `(x,y)` avec `x` le numéro de ligne et `y` le numéro de colonnes.

```
>>> config
[[0 0 2 0 0 0 0]
 [0 2 2 2 0 0 0]
 [2 0 2 0 0 0 2]
 [0 0 0 0 0 2 2]
 [0 0 0 0 0 0 2]]
```

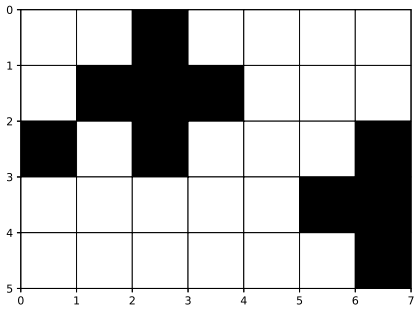

FIGURE 1 – Tâches de couleur noire

On considère que des pixels noirs appartiennent à une même tâche noire si on peut passer de l'un à l'autre par des transitions horizontales, verticales ou diagonales. Dans l'exemple de la figure 1, les pixels de coordonnées `(0, 2)`, `(1, 1)`, `(1, 2)`, `(1, 3)`, `(2, 0)`, `(2, 2)` appartiennent à une même tâche et les pixels de coordonnées `(2, 6)`, `(3, 5)`, `(3, 6)`, `(4, 6)` appartiennent à une autre tâche.

Le graphe associé à cet image est le couple (`S`, `A`) où l'ensemble des sommets `S` est la liste des pixels noirs et l'ensemble des arêtes `A` est le dictionnaire où une clé désigne un pixel noir associé à sa valeur qui est la liste, éventuellement vide, des pixels noirs vers lesquels une transition horizontale, verticale ou diagonale est possible.

1. Écrire une fonction `sommets(config)` d'argument `config` une liste de listes de 0 et 2 qui renvoie la liste des sommets du graphe associé à l'image représentée par `config`.
   Par exemple, pour la figure 1, l'appel `sommets(config)` renvoie :

```
>>> sommets(config)
[(0, 2), (1, 1), (1, 2), (1, 3), (2, 0), (2, 2), (2, 6), (3, 5), (3, 6),
 (4, 6)]
```

   Les nombres de lignes et colonnes devront être obtenus directement à partir de l'argument `config`. Pour tester la noirceur d'un pixel, on utilisera la variable globale `noir`. Par exemple pour tester, si le point `(i,j)` est noir, on écrira `config[i][j]==noir`.

2. Écrire une fonction `aretes(config)` d'argument `config` une liste de listes de 0 et 2 qui renvoie le dictionnaire des arêtes du graphe associé à l'image représentée par `config`.
   Par exemple, pour la figure 1, l'appel `aretes(config)` renvoie :

```
>>> aretes(config)
{(0, 2): [(1, 1), (1, 2), (1, 3)],
 (1, 1): [(0, 2), (1, 2), (2, 0), (2, 2)],
 (1, 2): [(0, 2), (1, 1), (1, 3), (2, 2)],
 (1, 3): [(0, 2), (1, 2), (2, 2)],
 (2, 0): [(1, 1)],
 (2, 2): [(1, 1), (1, 2), (1, 3)],
 (2, 6): [(3, 5), (3, 6)],
 (3, 5): [(2, 6), (3, 6), (4, 6)],
 (3, 6): [(2, 6), (3, 5), (4, 6)],
 (4, 6): [(3, 5), (3, 6)]}
```

   La fonction `aretes` fera appel à la fonction `sommets` (voir fichier `TP01_AUTO.py`). Les autres recommandations énoncées pour la première question valent aussi pour cette question.

Exercice 2

On reprend les notations de l'exercice 1. On utilisera un dictionnaire `etat` où les pixels noirs seront des clés et leurs valeurs désignera un état visité ou non. Un pixel noir non visité sera à l'état 0 et un pixel noir visité sera à l'état 1. On utilisera les variables globales `visite` et `non_visite` pour tester l'état d'un pixel ou le modifier. Par exemple, pour tester si un sommet `s` est considéré non visité dans le dictionnaire `etat`, on écrira `etat[s]==non_visite`.

1. Écrire une fonction `visit_comp(A,s0,comp,etat)` d'arguments `A` le dictionnaire des arêtes d'un graphe associé à une image, `s0` un sommet correspondant à un pixel noir non visité, `comp` une variable contenant une liste vide et `etat` le dictionnaire des états des sommets du graphe et qui réalise le parcours en largeur de la composante connexe du graphe depuis le sommet `s0`, autrement dit qui réalise, à l'aide d'une file (type `deque`), le parcours de la tâche noire contenant `s0`. La fonction ne renvoie rien. Les variables mutables `comp` et `etat` transmises en argument sont modifiées par la fonction. Après appel de la fonction, la variable `comp` contient tous les pixels de la tâche noire contenant `s0` et tous ses pixels sont passés à l'état visité dans la variable `etat`.

Par exemple, on reprend la configuration de l'exercice 1. On initialise une variable `etat` et une liste vide `comp` :

```
>>> etat={}
>>> for s in S: etat[s]=0

>>> comp=[]
```

L'appel de `visit_comp(A,(1,1),comp,etat)` ne renvoie rien mais modifie les variables `comp` et `etat`.

```
>>> visit_comp(A,(1,1),comp,etat)
>>> comp
[(1, 1), (0, 2), (1, 2), (2, 0), (2, 2), (1, 1), (1, 3)]
>>> etat
{(0, 2): 1, (1, 1): 1, (1, 2): 1, (1, 3): 1, (2, 0): 1,
 (2, 2): 1, (2, 6): 0, (3, 5): 0, (3, 6): 0, (4, 6): 0}
```

2. Écrire une fonction `comp_connexe(config)` d'argument `config` une liste de listes de 0 et 2 qui renvoie un dictionnaire des différentes taches de l'image représentée par `config`, c'est-à-dire les composantes connexes du graphe correspondant à `config`. Les clés du dictionnaire seront des numéros de 1 à n avec le n le nombre de tâches et pour $k \in [\![1\,;n]\!]$, la valeur associée à k sera la liste des sommets dans la k-ième tâche.

   Par exemple, pour la configuration de l'exemple 1, l'appel `comp_connexe(config)` renvoie :

```
{1: [(0, 2), (1, 1), (1, 2), (1, 3), (2, 0), (2, 2)],
 2: [(2, 6), (3, 5), (3, 6), (4, 6)]}
```

   La fonction `comp_connexe` devra faire appel aux fonctions `sommets`, `aretes` et `visit_comp`.

Exercice 3

On reprend les notations de l'exercice 1. On considère que la distance entre deux pixels est le nombre de pixels constitutifs d'un plus court chemin en le nombre de transitions autorisées, à savoir horizontales, verticales et diagonales. Par exemple, avec la configuration de l'exercice 1, le pixel `(1,1)` est à distance égale à 1 de lui-même, à distance égale à 2 des pixels `(0,2)`, `(1,2)`, `(2,0)`, `(2,2)` et à distance égale à 3 du pixel `(1,3)`. Le parcours en largeur détermine un tel plus court chemin entre deux pixels.

1. Écrire une fonction `bfs(A,s0)` d'arguments `A` le dictionnaire des arêtes d'un graphe associé à une image, `s0` un sommet correspond à un pixel noir qui réalise le parcours en largeur de la composante connexe du graphe depuis le sommet `s0` (autrement dit la tâche noire contenant `s0`) et renvoie le dictionnaire des distances entre `s0` et tous les autres sommets connectés à `s0`.

   Par exemple, avec la configuration de l'exercice 1, l'appel `bfs(A,(1,1))` renvoie :

```
>>> bfs(A,(1,1))
{(1, 1): 1, (0, 2): 2, (1, 2): 2, (2, 0): 2, (2, 2): 2, (1, 3): 3}
```

2. Écrire une fonction `diametre_comp(config,comp)` d'argument `config` une liste de listes de 0 et 2, `comp` une liste de pixels constitutifs d'une tâche noire qui renvoie le *diamètre* de cette tâche à savoir la plus grande distance entre deux pixels de cette tâche noire. Par exemple, avec la configuration de l'exercice 1, le diamètre de la tâche noire contenant le pixel `(1,1)` est de diamètre égal à 4 :

```
>>> comp=[(0, 2), (1, 1), (1, 2), (1, 3), (2, 0), (2, 2)]
>>> diametre_comp(config,comp)
4
```

   La fonction devra faire appel aux fonctions `aretes` et `bfs`.

3. Écrire une fonction `diametre(config)` d'argument `config` une liste de listes de 0 et 2 qui renvoie une liste de couples `(k,d)` avec `k` le numéro de la composante connexe attribué par la fonction `comp_connexe` de l'exercice 2 et `d` le diamètre de cette k-ième composante connexe. Par exemple, avec la configuration de l'exercice 1, on obtient :

```
>>> diametre(config)
[(1, 4), (2, 3)]
```

   La fonction `diametre` devra faire appel à `comp_connexe` et `diametre_comp`.

Exercice 4

On reprend les notations de l'exercice 1.

1. Écrire une fonction `tri(L)` performante, pas nécessairement en place, d'argument `L` une liste de couples `(k,d)` avec `k` et `d` entiers et qui renvoie cette liste triée par ordre croissant vis-à-vis de la deuxième composante comme illustré dans l'exemple :

```
>>> tri([(1, 11), (2, 9), (3, 5), (4, 26), (5, 22)])}
[(3, 5), (2, 9), (1, 11), (5, 22), (4, 26)]
```

2. On veut simuler l'effet d'un traitement contre des cellules cancéreuses : les tâches noires représentent les cellules cancéreuses et le traitement réduit d'environ 80% le nombre de ces cellules, en commençant par celles qui sont de diamètre maximal. Écrire des instructions python permettant de modifier une liste de listes `config` constituée de 0 et de 2 où les 2 désignent les pixels noirs et qui modifierait la composition de certaines tâches noires pour les griser selon la règle précédemment énoncée. Les pixels grisés passeraient à l'état 1 selon le schéma `config[i][j]=gris`.

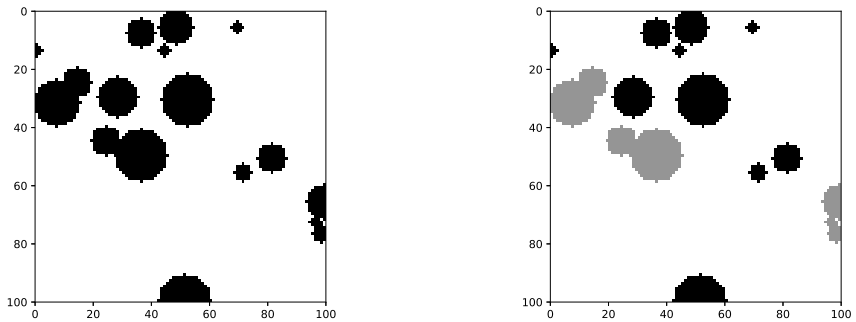

FIGURE 2 – Réduction de cellules cancéreuses