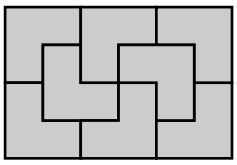
**Aix-Marseille Université**
**L1**

# UE Projet informatique

## Pavages par des triminos

Dans ce projet, on s'intéresse au problème consistant à paver un tableau rectanguaire au moyen de tuiles appelées *triminos*, consistuées de trois carrés assemblés en forme d'angle (voir figure 1a). Un exemple de tel pavage est donné en figure 1b. On cherche à savoir pour quelles tailles de rectangles un tel pavage est possible et à compter combien il y a de façons différentes de paver un rectangle donné, on cherche aussi à dessiner les pavages obtenus.

(a) Un trimino

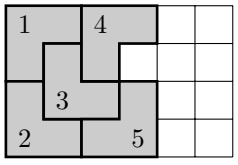


(b) Exemple de pavage d'un rectangle $6 \times 4$ par des triminos

FIGURE 1 – Triminos et pavages

## Partie I – Manipulations élémentaires

Pour traiter ce problème, on représentera en Python un tableau rectangulaire de largeur $m$ et de hauteur $n$ par une liste de $n$ listes de $m$ entiers, c'est-à-dire une liste dont les éléments correspondent aux lignes du tableau, de haut en bas. Les tuiles déjà posées dans le tableau seront numérotées en partant de 1 et dans notre représentation chaque case du tableau contiendra le numéro de la tuile qui s'y trouve, ou 0 dans le cas d'une case vide. Un exemple est donné en figure 2.




Représentation du tableau :

```
T = [ [1,1,4,4,0,0],
      [1,3,4,0,0,0],
      [2,3,3,5,0,0],
      [2,2,5,5,0,0] ]
```

Représentation de la tuile numéro 5 :

```
[(3,2), (2,3), (3,3)]
```

FIGURE 2 – Représentation des données du problème

En accord avec les conventions de Python sur les indices, on numérote les lignes en partant de 0 et du haut vers le bas et on numérote les colonnes en partant de 0 et de gauche à droite, de sorte que `T[y][x]` représente la case de coordonnées $(x, y)$, donc en colonne $x$ et en ligne $y$.

Une tuile sera représentée par la liste des coordonnées de ses cases, de haut en bas et gauche à droite. Ainsi, dans l'exemple ci-dessus, le trimino numéro 3 sera représenté par la liste `[(1,1),(1,2),(2,2)]`.

Ecrire des fonctions qui permettent de :

— produire un tableau rectangulaire vide de $m$ colonnes et $n$ lignes.

— renvoyer respectivement le nombre de colonnes et le nombre de lignes d'un tableau `T` (que l'on suppose effectivement rectangulaire de taille non nulle).

— tester si le tableau `T` a des cases vides ou pas. La fonction renverra `True` s'il n'y a pas de case vide.

— donner le trimino numéro `i` (entre 0 et 3) dont la première case est en `(x,y)`.
Au préalable vous justifierez que, pour un couple d'entiers $(x, y)$, il y a toujours 4 triminos possibles dont la première case est $(x, y)$ selon les conventions données (en autorisant des coordonnées négatives).
*Prenez soin de bien écrire les coordonnées des cases pour éviter des erreurs ultérieures. A ce stade vous*

*pouvez vous répartir le travail en deux, avec un groupe qui proposera différents exemples où la réponse est connue et qui permettront de tester la fonction.*

— ajouter au tableau `T` une tuile numérotée `i` dont les cases sont données par la liste `t`. S'il est possible d'ajouter cette tuile, `T` doit être modifié en conséquence et la fonction doit renvoyer `True`. S'il est impossible d'ajouter cette tuile `T` doit être inchangé et la fonction doit renvoyer `False`.
— renvoyer la tuile numérotée `i` du tableau `T`.
— enlever la tuile numérotée `i` du tableau `T`.

## Partie II – Recherche de pavages

Avec les fonctions définies dans la partie précédente, on va maintenant mettre en œuvre une recherche de solutions au problème du pavage. On va aborder le problème en parcourant les cases d'un tableau dans un ordre déterminé : ligne par ligne en partant du haut, et de gauche à droite sur chaque ligne.

**Question préalable :** Considérons un tableau $T$ partiellement rempli, soit $(x, y)$ la première case vide de $T$ selon les conventions ci-dessus. Montrer que dans tout pavage qui complète $T$, il y a une tuile dont la première case (selon la représentation de la partie précédente) est $(x, y)$.

Ecrire une fonction récursive `paver(T)` qui cherche à **compléter un pavage d'un tableau** `T` partiellement rempli. S'il existe une solution, la fonction devra renvoyer `True` et faire en sorte que `T` contienne une solution ; s'il n'y a pas de solution, la fonction renverra `False` en laissant `T` dans l'état initial.

*Vous pourrez avoir besoin de deux fonctions qui renvoient respectivement :*
— *l'indice de la prochaine tuile à poser dans un tableau.*
— *les coordonnées de la première case vide d'un tableau.*

Modifier la fonction `paver` pour **compter le nombre de pavages différents d'un tableau**. Faire de même en l'**énumérant toutes les solutions possibles**.

## Partie III – Dessin des pavages

Pour le dessin des pavages vous allez utiliser la librairie `matplotlib.pyplot`.

Pour commencer écrire une fonction `tracer_grille(m,n)` qui trace une grille de `m` colonnes et `n` lignes, avec le coin supérieur gauche à l'origine et des lignes espacées d'un nombre de pixels égal à `echelle`, de sorte que le coin inférieur droit est aux coordonnées `(m*echelle,-n*echelle)`.

Il existe ensuite deux stratégies pour afficher un pavage :

**Coloration de tuiles** Il s'agit de donner une couleur différente à chaque tuile. Pour cela vous commencerez par colorier une case particulière à l'aid de la fonction `fill`.

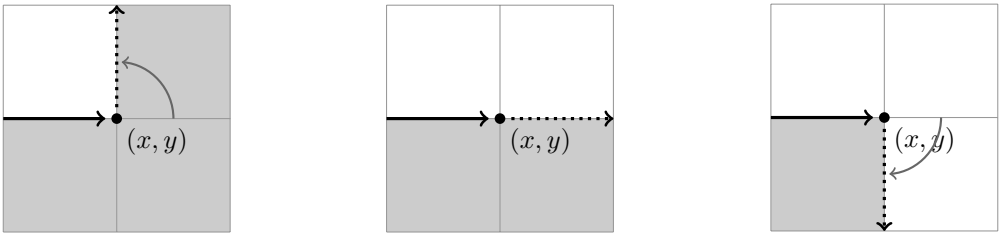
Il se posera éventuellement la question du choix des couleurs, pour éviter que des tuiles adjacentes se confondent visuellement.

**Contour de cases** Pour dessiner une tuile, on veut en dessiner un contour. Il faut donc une méthode pour parcourir le contour d'un ensemble de carrés. On va utiliser la méthode du labyrinthe : on part d'un point du bord puis on suit le bord, pas à pas, en gardant la tuile à droite.

Pour implémenter cette méthode, on pose les conventions suivantes, pour représenter des positions et des directions dans une grille, en accord avec les conventions utilisées pour représenter les tuiles et les pavages :

— le point de coordonnées entières $(x, y)$ est le point situé $x$ unités vers la droite et $y$ unités vers le bas en partant de l'origine ;
— un carré est identifié par son coin supérieur gauche, c'est-à-dire que le carré $(x, y)$ s'étend en abscisse de $x$ à $x + 1$ et en ordonnée de $y$ à $y + 1$ ;
— les quatre directions sont représentée par des vecteurs à coordonnées entières entre $-1$ et 1.

Supposons que le parcours arrive en un point $(x, y)$ avec une direction $(dx, dy)$. Par hypothèse, la tuile considérée se trouve à droite, c'est-à-dire que le carré à droite du segment correspondant au vecteur $(dx, dy)$ arrivant en $(x, y)$ est occupé par la tuile et que le carré à gauche ne l'est pas. Pour progresser dans le parcours, on pourra déterminer s'il faut continuer tout droit, tourner à gauche ou tourner à droite selon qu'il y a ou pas des carrés occupés à gauche et à droite dans la direction $(dx, dy)$, comme illustré en figure 3.




La direction dans laquelle il faut continuer est en pointillés.

FIGURE 3 – Règles de progression le long du bord d'une tuile

Montrer que si $(x, y)$ est le premier point d'une tuile $t$, alors le segment allant de $(x, y)$ à $(x + 1, y)$ est sur le bord de $t$ et a un carré de $t$ à sa droite.
Par conséquent, le premier point de $t$ permet de choisir un premier segment pour suivre le bord de $t$. Si la tuile $t$ est simplement connexe (c'est-à-dire d'un seul tenant et sans trou), suivre le bord jusqu'à revenir au point de départ permettra d'en tracer le contour.

Vous pourrez ainsi écrire une fonction qui donne la liste des coordonnées des points d'un contour. Ensuite vous utiliserez cette fonction pour afficher toutes les tuiles et vous pourrez ré-examiner également la stratégie de coloration de tuiles.

## Partie IV – Ouverture du sujet

Si vous avez terminé, vous pourrez réfléchir à quelques pistes d'exploration :

— Passer d'un domaine rectangulaire à une forme plus complexe
— Utiliser d'autres formes que les triminos