

## TP 13



## Parcours d'un labyrinthe

## Exercice 1 – Génération et parcours de labyrinthe

## Génération d'une grille

Soit une grille rectangulaire  $n \times p$  constituée de  $n$  colonnes et de  $p$  lignes contenant toutes les arêtes possibles. On modélise cette grille par un graphe dont l'ensemble des sommets est donné par les couples  $(i, j)$  tels que  $i \in \llbracket 0, n \rrbracket$  et  $j \in \llbracket 0, p \rrbracket$ . Les voisins d'un sommet  $(i, j)$  sont ceux situés en haut, en bas, à droite et à gauche s'ils existent (par exemple, le sommet  $(0, 0)$  a comme voisin les sommets  $(0, 1)$  et  $(1, 0)$ ).

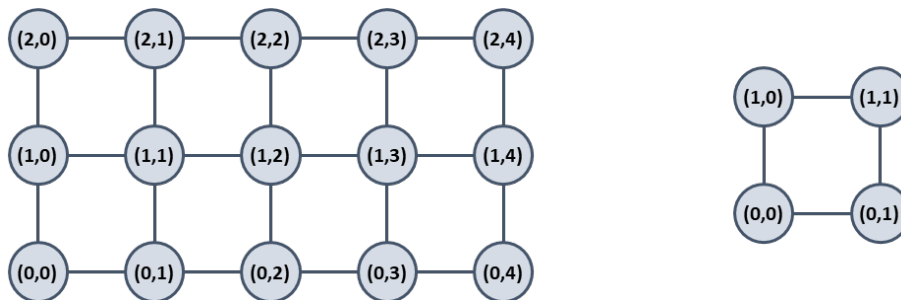


FIGURE 1 – Grille (5,3) et grille (2,2)

Le graphe est implémenté par un dictionnaire d'adjacence où les clés sont les tuples, coordonnées d'un sommet. La valeur associée est une liste des sommets voisins.

Ainsi, la grille  $2 \times 2$  sera modélisée par le graphe suivant :

$G_2 = \{(0,0):[(0,1),(1,0)], (0,1):[(0,0),(1,1)], (1,0):[(0,0),(1,1)], (1,1):[(0,1),(1,0)]\}$ .

**Question 1** Écrire la fonction `creer_graphe(n:int, p:int) -> dict` permettant de créer une grille de  $n$  colonnes et  $p$  lignes.

On souhaite afficher ce graphe en utilisant `matplotlib`. Pour cela, on va commencer par tracer chacune des arêtes puis chacun des sommets.

**Question 2** Écrire la fonction `get_sommets(G:dict) -> list, list` renvoyant deux listes `les_x` et `les_y` contenant respectivement les abscisses des sommets et les ordonnées des sommets.

**Question 3** Écrire la fonction `trace_sommets(G:dict) -> None` qui affiche les sommets en utilisant un point rouge ('r.') ou cercle rouge ('ro').

**Question 4** Écrire la fonction `get_aretes(G:dict) -> list` renvoyant la liste des arêtes du graphe sous la forme d'une liste de listes de tuples. Une arête est donc une liste de sommets ou les sommets sont des tuples. Les arêtes ne devront être présentes qu'une fois.

Par exemple : `get_aretes(G2)` peut renvoyer la liste suivante :

`[(0,0),(0,1)], [(0,0),(1,1)], [(0,1),(1,1)], [(1,0),(1,1)]` (l'ordre des arêtes et des sommets peut être complètement différent).

**Question 5** Écrire la fonction `trace_aretes(G:dict) -> None` qui affiche les arêtes en utilisant un trait bleu. Exemple : pour tracer l'arête  $[(0,2),(1,2)]$ , il faut utiliser l'instruction `plt.plot([0,1],[2,2], 'b')`.

**Question 6** Écrire la fonction `trace_graphe(G:dict) -> None` qui permet de tracer les sommets au dessus des arêtes.

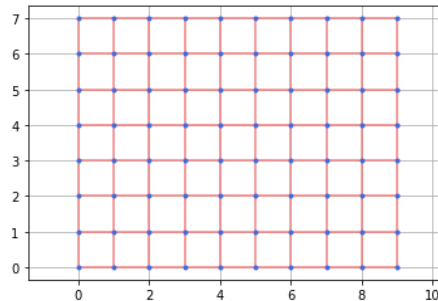


FIGURE 2 – Grille 10 colonnes 8 lignes

### Génération d'un labyrinthe

Dans notre cas, résoudre le labyrinthe consiste en partant du sommet (0,0) (sommet en bas à gauche) et à se déplacer sur les arêtes dans le but d'atteindre le sommet en haut à droite. Pour générer un labyrinthe, nous allons réaliser un parcours de la grille (en largeur ou en profondeur). Le labyrinthe sera lui-même un graphe. Lors de ce parcours, à chaque fois qu'un sommet devra être visité. Ainsi à chaque fois qu'on découvrira un sommet à visiter, on ajoutera une arête entre le sommet père et ses voisins (s'ils ne sont pas dans le labyrinthe).

**Question 7** Écrire la fonction `ajouter_arete(G:dict, s1:tuple, s2: tuple) -> None` qui permet d'ajouter l'arête  $(s1,s2)$  au graphe `G`. Il faudra vérifier que les sommets n'existent pas déjà...

**Question 8** Écrire la fonction `parcours_largeur(G:dict, s:tuple) -> dict` qui permet de créer un labyrinthe en largeur à partir d'un graphe `G`. Tracer le labyrinthe obtenu.

**Question 9** Écrire la fonction `parcours_profondeur(G:dict, s:tuple) -> dict` qui permet de créer un labyrinthe en profondeur à partir d'un graphe `G`. Tracer le labyrinthe obtenu.

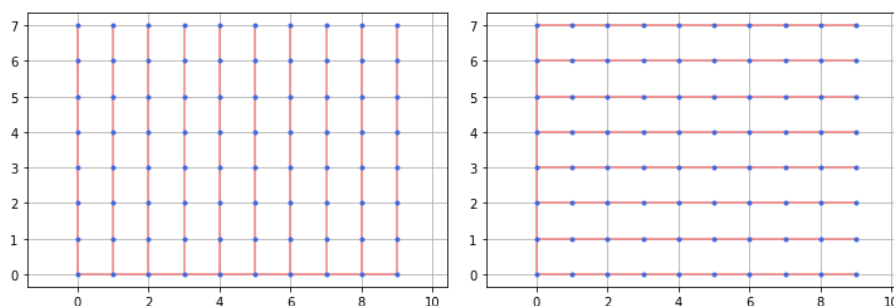


FIGURE 3 – Labyrinthes en largeur et en profondeur

Comme vous pouvez le constater, le côté aléatoire de ces labyrinthes est discutable :). Il est possible de mélanger une liste en utilisant le module `random` : `random.shuffle(voisins)` permet de mélanger la liste de tuples `voisins`.

**Question 10** Écrire les fonctions `labyrinthe_largeur(G:dict, s:tuple) -> dict` et `labyrinthe_profondeur(G:dict, s:tuple) -> dict` permettant de prendre cette remarque en compte. Quelle fonction permet d'obtenir un labyrinthe acceptable?

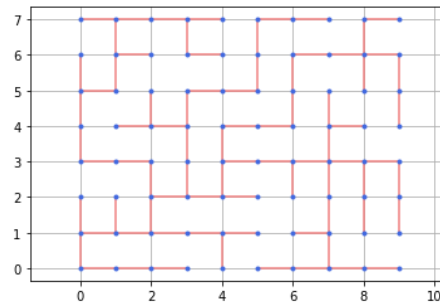


FIGURE 4 – Labyrinthe 10x8

### Résolution d'un labyrinthe

Il est possible de résoudre le labyrinthe en utilisant un parcours en longueur ou un parcours en profondeur.

**Question 11** Écrire les fonctions `resolution_largeur(G:dict, s:tuplet) -> list` qui permet de résoudre le labyrinthe en utilisant un parcours en largeur. Cette fonction renvoie la liste des sommets permettant d'atteindre le sommet en haut à droite depuis le sommet en bas à gauche.

**Question 12** Afficher en trait épais noir la solution donnée par le parcours en largeur.

**Question 13** Répondre aux mêmes questions en utilisant un parcours en profondeur.