TP 14

Parcours d'un labyrinthe

Inspirations :
Génération et résolution de labyrinthes et
`https://cpge-itc.github.io/itc1/5_graph/3_traversal/labyrinth/`

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 $i \in [\![0, n[\![$ et $j \in [\![0, p[\![$. 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 $(1, 0)$).

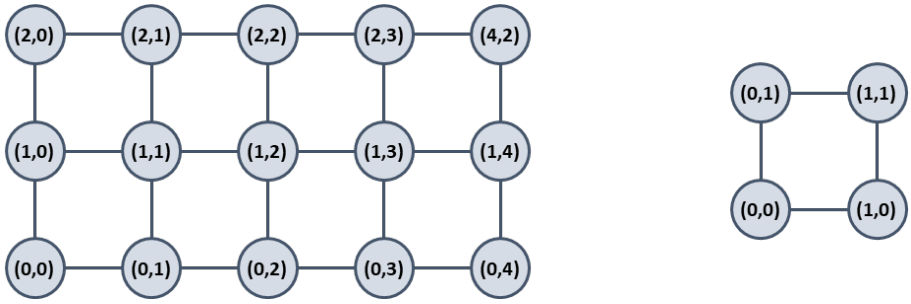

FIGURE 1 – Grille (5,3) et grille (2,2)

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.

Question 1 *Écrire la fonction* `creer_graphe(n:int, p:int) -> {}` *permettant de créer le graphe d'une grille de* `n` *colonnes et* `p` *lignes.*

▪ **Exemple** La grille 2×2 sera modélisée par le graphe suivant :

```
>>> G2 = creer_graphe(2,2)
>>> G2
       {(0, 0): [(1, 0), (0, 1)],
       (1, 0): [(1, 1), (0, 0)],
       (0, 1): [(1, 1), (0, 0)],
       (1, 1): [(0, 1), (1, 0)]}
```

▪

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.*

▪ **Exemple** Dans l'exemple qui suit, les coordonnées de sommets peuvent être dans un ordre différent.

```
>>> les_sx, les_sy = get_sommets(G2)
>>> les_sx, les_sy
```

```
        ([0, 1, 0, 1], [0, 0, 1, 1])
```

▪

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

Question 4 *Écrire la fonction* `get_aretes(G:{}) -> []` *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 où les sommets sont des tuples. Les arêtes ne devront être présentes qu'une fois.*

▪ **Exemple** Dans l'exemple qui suit, l'ordre des arêtes peut être dans un ordre différent. Pour une arête donnée, les sommets peuvent aussi être dans un ordre différent.

```
>>> get_aretes(G2)
        [((0, 0), (1, 0)), ((0, 0), (0, 1)), ((1, 0), (1, 1)), ((0, 1), (1, 1))]
```

▪

Question 5 *Écrire la fonction* `trace_aretes(G:{}) -> 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:{}) -> None` *qui permet de tracer les sommets au dessus des arêtes.*

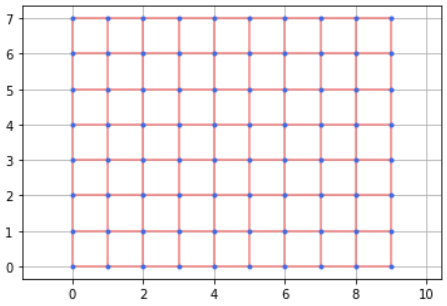

FIGURE 2 – Grille 10 colonnes 8 lignes

Génération d'un labyrinthe

Dans notre cas, résoudre le labyrinthe consiste en partir du sommet de départ `(0,0)` (sommet en bas à gauche, par exemple) 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 `G` (en largeur ou en profondeur). Le labyrinthe sera lui-même un graphe noté `L` . À chaque fois qu'on **empilera ou enfilera** un **sommet non visité**, on ajoutera une arête entre entre ce sommet et le sommet père à `L` .

Question 7 *Écrire la fonction* `ajouter_arete(G:{}, s1:tuple, s2:tuple) -> None` *qui permet d'ajouter l'arête* `([s1,s2])` *au graphe* `G` . ***Pour ajouter una arête, il est par exemple possible de traiter les cas deux suivants : on ajoute une arête si aucun des sommets appartient au graphe, ou si le le sommet d'arrivée n'existe pas dans le graphe.***

▪ **Exemple** On reprend le graphe noté `G2` précédemment.

```
>>> ajouter_arete(G2,(1,0),(2,0))
>>> G2
        {(0, 0): [(1, 0), (0, 1)],
        (1, 0): [(1, 1), (0, 0), (2, 0)],
```

```
        (0, 1): [(1, 1), (0, 0)],
        (1, 1): [(0, 1), (1, 0)],
        (2, 0): [(1, 0)]}
```

■

Question 8 *Écrire la fonction* `parcours_largeur(G:{}, depart:tuple) -> dict` *qui permet de créer un labyrinthe en largeur à partir d'un graphe* `G`. *Tracer le labyrinthe obtenu.* `s` *sera ici le point de départ du labyrinthe,* `(0,0)` *par exemple.*

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

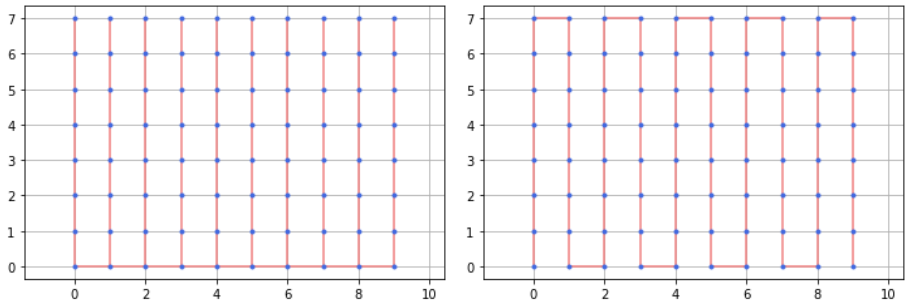

FIGURE 3 – Labyrinthes en largeur et en profondeur

Comme vous pouvez le constater, le coté aléatoire de ces labyrinthes est discutable :). Le labyrinthe que vous pourrez obtenir peut varier, notamment en fonction, de l'ordre dans lequel vous ajoutes les sommets dans le graphe dans la fonction `creer_graphe`. 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:{}, s:tuple) -> {}` *et* `labyrinthe_profondeur(G:{` *permettant de prendre cette remarque en compte. Quelle fonction permet d'obtenir un labyrinthe acceptable ?*

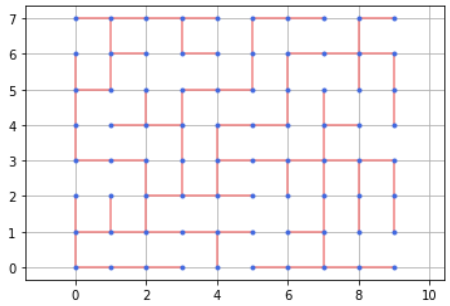

FIGURE 4 – Labyrinthe 10x8

Résolution d'un labyrinthe

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

Question 11 *Écrire la fonction* `resolution_largeur(G:{}, s:tuple) -> []` *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.*

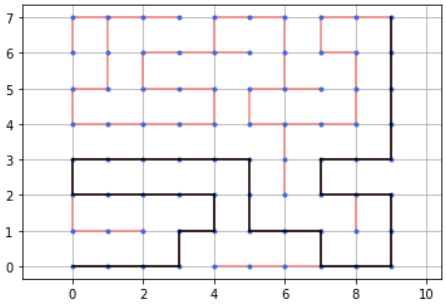

FIGURE 5 – Solution d'un labyrinthe