TP 14
Création et parcours de labyrinthe – Sujet

Activité préparatoire : Génération d'une grille – Déjà fait sur Capytale

TP : Génération d'un labyrinthe

L'objectif de ce TP est de générer un labyrinthe. Un labyrinthe est un graphe qui contient tous les sommets de la grille et un certain nombre d'arêtes pour les relier. Pour obtenir un labyrinthe aléatoirement on réalise un parcours de la grille obtenue dans l'activité préparatoire. Pour cela il faut visiter l'ensemble des sommets de la grille et conserver les chemins qui ont permis cette découverte. Sommets et arêtes seront stockés dans un graphe appelé labyrinthe.

Viendra ensuite le moment de résoudre ce labyrinthe : ce sera l'objectif de la dernière partie. Il faudra alors être capable de trouver le chemin qui permet d'aller du coin inférieur gauche (départ) au coin supérieur droit (arrivée) en n'empruntant que les lignes (arêtes) du labyrinthe.

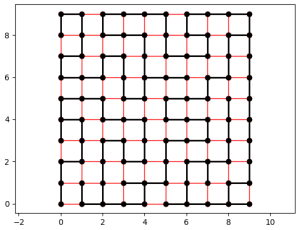


FIGURE 1 – Exemple de labyrinthe obtenu sur une grille 10*10

Ajouter une arête dans un graphe

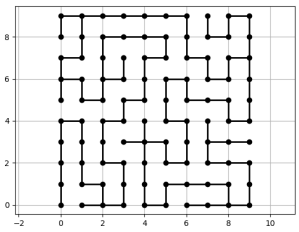
Pour générer un labyrinthe il faut construire un graphe vide puis ajouter une arête à chaque fois que l'on découvre un sommet.

Question 1 Écrire la fonction `ajouter_arete(G:dict, s1:tuple, s2:tuple) -> None` qui permet d'ajouter l'arête `([s1,s2])` au graphe `G`.

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)]}
```



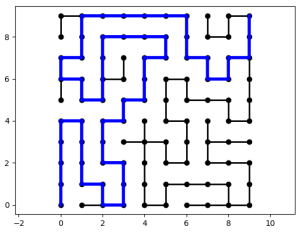


FIGURE 2 – Un labyrinthe et ce même labyrinthe résolu

Bien penser aux éléments suivants : ajouter une arête entre les sommets `s1` et `s2` consiste :

- à ajouter le sommet `s1` dans le graphe si ce sommet n'existait pas : dans ce cas il faut commencer une nouvelle liste de ses voisins avec son premier voisin `s2` ;
- à ajouter le sommet `s2` dans le graphe si ce sommet n'existait pas : dans ce cas il faut commencer une nouvelle liste de ses voisins avec son premier voisin `s1` ;
- à ajouter `s1` (respectivement `s2`) dans la listes des voisins de `s2` (resp. `s1`) dans le cas où le sommet `s2` (resp. `s1`) existait déjà.

Marquages des noeuds visités

On propose de marquer les sommets en utilisant un dictionnaire noté `visited` :

- Initialement, tous les sommets sont blancs. On dira qu'un sommet blanc n'a pas encore été découvert.
- Lorsqu'un sommet est "découvert" (autrement dit, quand on arrive pour la première fois sur ce sommet), il est colorié en gris. Le sommet reste gris tant qu'il reste des successeurs de ce sommet qui sont blancs (autrement dit, qui n'ont pas encore été découverts).
- Un sommet est colorié en noir lorsque tous ses successeurs sont gris ou noirs (autrement dit, lorsqu'ils ont tous été découverts).

Ce dictionnaire contiendra tous les sommets de la grille.

On leur associera le caractère **'W'** pour blanc (**W**hite), **'G'** pour gris et **'K'** pour noir (blac**K**).

Question 2 Créer un graphe `G` associé à une grille initiale de 5*5. L'afficher (épaisseur 1, en rouge).

Question 3 Créer un dictionnaire `visited` associé à cette grille ayant ses valeurs toutes à `'W'`.

Question 4 Créer une fonction `trace_visites` qui prend pour argument un dictionnaire `v` des sommets découverts et trace les noeuds découverts en gris ou en noir.

(*Rappel : tracé avec points noirs :* `plt.plot(x,y,'ko')` */ avec des points gris :* `plt.plot(x,y,'o',color='grey')` *)*

Question 5 Test : Après avoir modifié arbitrairement le dictionnaire `visited` pour le test (ajouter arbitrairement des noeuds en gris et en noir), le tracer sur la même figure que la grille.

Génération d'un labyrinthe par parcours en largeur

On propose de travailler par étape en modifiant successivement la fonction pour aboutir à l'agorithme complet du parcours en largeur. La fonction aura pour argument systématiquement le graphe à parcourir `G` et un noeud de départ `depart`. Vous penserez à modifier la coloration des sommets (via le dictionnaire `visited` (= dictionnaire des sommets découverts) au fur et à mesure.

Initialisation

Question 6 Ecrire une fonction `parcours_largeur_init`, qui initialise le dictionnaire des sommets découverts (tout à 'W'), crée une file ne contenant que le noeud de départ et trace l'état obtenu en utilisant votre fonction `trace_visites`.

Première étape : visite des voisins

Question 7 Modifier la fonction précédente pour obtenir la fonction `parcours_largeur_etape1` , qui explore le premier sommet (la tête de file). On entend ici par "explorer", découvrir ses voisins, les ajouter dans la file s'ils n'ont pas été déjà découverts. Tracer aussi l'état obtenu.

Algorithme complet

Question 8 Modifier la fonction précédente pour obtenir la fonction `parcours_largeur_complet` , qui continuera à explorer les voisins tant que la file n'est pas vide. A chaque étape (à chaque tour de boucle) vous pouvez tracer l'état, tracer une arête pour indiquer le chemin parcouru, et ajouter l'instruction `plt.pause(0.5)` de façon à voir progressivement le parcours du graphe.

Vous pouvez utiliser la note en bas de page pour ajouter de l'aléatoire au parcours.

Algorithme complet avec création du labyrinthe

Il est maintenant possible de créer le labyrinthe.

Question 9 Modifier la fonction précédente pour obtenir la fonction `labyrinthe_largeur` , qui construit le graphe labyrinthe `L` au fur et à mesure du parcours. Cette fonction retourne le graphe labyrinthe. Vous pouvez tracer à chaque étape le labyrinthe en trait plus épais noir.

Note : Comme vous pouvez le constater, le coté aléatoire de ce labyrinthe est discutable :). Il est possible de mélanger une liste en utilisant le module `random` : `random.shuffle(voisins)` ce qui permet de mélanger la liste de tuples voisins.

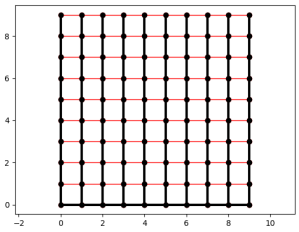


FIGURE 3 – Exemple de labyrinthe obtenu sur une grille 10*10 par parcours en largeur

Génération d'un labyrinthe par parcours en profondeur

Question 10 Modifier l'algorithme `parcours_profondeur` pour réaliser le marquage des sommets comme proposé (gris = sommet "découvert", noir = sommet dont tous les voisins ont été découverts).

Question 11 Etablir la fonction `labyrinthe_profondeur`, qui construit le graphe labyrinthe `L` par un parcours en profondeur. Vous pouvez tracer à chaque étape le labyrinthe en trait plus épais noir.