1 Problématique

Les chemins d'un labyrinthe peuvent être représentés par un graphe. Dans le cas d'un labyrinthe sans boucle ni case inaccessible ce graphe est un arbre.

Comment retrouver la sortie d'un labyrinthe à l'aide d'un arbre ?

2 Visualisation du problème

Pour représenter le labyrinthe (figure 1) en mémoire on peut utiliser un graphe.

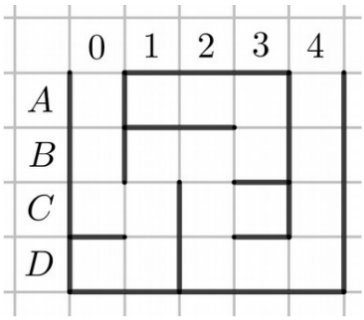

FIGURE 1 – Labyrinthe

Activité 1 :

1. Télécharger et extraire *arbre-labyrinthe.zip* sur le site https://cviroulaud.github.io .
2. Créer un fichier *labyrinthe.py*.
3. Construire le graphe représentatif du labyrinthe (figure 1).
4. À l'aide d'une méthode de la classe *Graphe* Donner le chemin du départ (A0) à la sortie (A4).
5. À l'aide de la bibliothèque *networkx*, représenter l'arbre des chemins du labyrinthe.

3 Labyrinthe aléatoire

3.1 Méthodes utiles

Activité 2 :

1. Ouvrir la fichier *labyrinthe-maxi.py* . Il contient la classe *Labyrinthe*.
2. Compléter la méthode **creer_sommets(self) → None** qui crée les sommets du graphe.
3. Compléter la méthode **get_voisins(self, en_cours : object)->list :** qui renvoie une liste des nœuds voisins de *en_cours*. Cette méthode utilise le tuple *delta* qui permet de repérer chaque voisin de *en_cours*.

3.2 Construire le labyrinthe

Le principe de construction utilise un parcours en profondeur du graphe. Les figures 2 à 7 décrivent l'algorithme.

Christophe Viroulaud

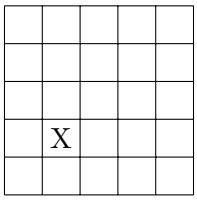

FIGURE 2 – Choisir une cellule, l'empiler

On itère sur la pile tant qu'elle n'est pas vide.

Première itération

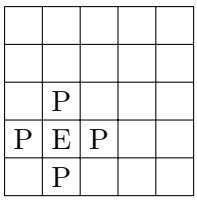

FIGURE 3 – Dépiler

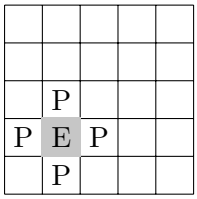

FIGURE 4 – Marquer le nœud *visité* et empiler tous les voisins

À ce stade aucun voisin n'est encore dans le labyrinthe. Aucune arête n'est encore créée. La première itération est terminée.

Deuxième itération

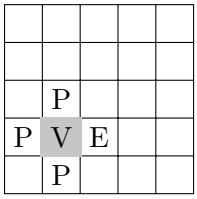

FIGURE 5 – Dépiler

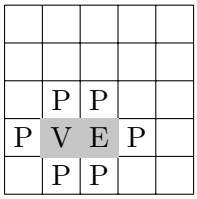

FIGURE 6 – Le marquer *visité* et empiler tous les voisins

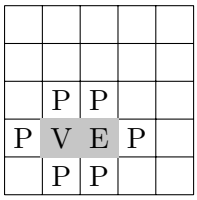

FIGURE 7 – Créer une arête entre le labyrinthe et le nœud en cours.

Christophe Viroulaud

Pour créer cette arête il faut vérifier si un des voisins du nœud en cours est déjà dans le labyrinthe.

Activité 3 : Compléter la méthode **creer_dfs(self) → None** .

3.3 Sortir du labyrinthe

Activité 4 :

1. Exécuter la méthode *affiche_chemins*. Il faut voir le labyrinthe comme un arbre de racine (0,0).
2. Compléter la méthode **chemin_sortie(self) → list** qui renvoie le parcours entre l'entrée en haut à gauche du labyrinthe et la sortie en bas à droite.

Christophe Viroulaud