TP5 : Parcours de Graphes

Le but de ce TP est d'implanter des algorithmes de parcours de graphes vus en cours. On fournit une trame de code pour les exercices dans les fichiers `TP5Exo1Parcours.py` *et* `TP5Exo2Dijkstra.py`.

Exercice 1. *Parcours de Graphes*

Le but de cette partie est d'implémenter les algorithmes de parcours en largeur et parcours en profondeur vus en cours. Le fichier *TP5Exo1Parcours.py* contient une trame de programme.

Un graphe à n sommets a, par défaut, comme ensemble de sommets $\{0,\dots,n-1\}$ et est encodé par listes de voisins, c'est-à-dire une liste de listes `G` où `G[i]` est la liste des voisins du sommets i. Un graphe fixe G est donné et vous devez implémentez le calcul d'un parcours en largeur et d'un parcours en profondeur de G depuis le sommet 0. Il faudra afficher les tableau `pere`, `niveau` et `ordre` pour le parcours en largeur et les tableaux `pere`, `debut` et `fin` pour le parcours en profondeur. Les résultats attendus sont donnés ci-dessous.

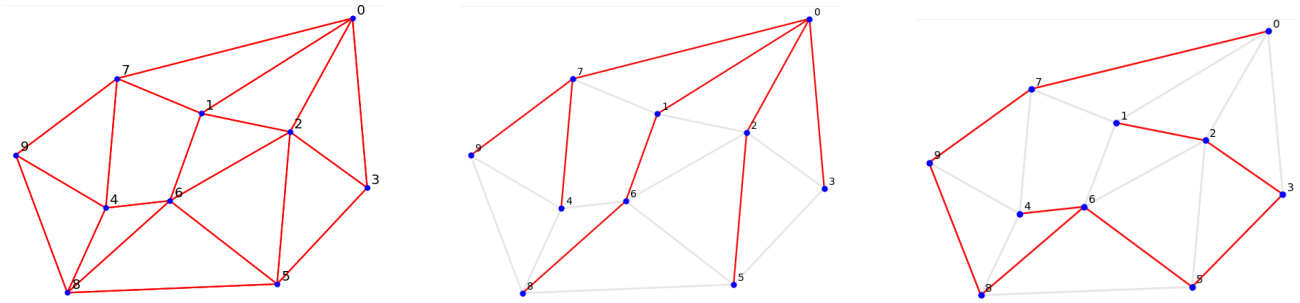

FIGURE 1 – Le graphe G fixé, un parcours en largeur de G et un parcours en profondeur de G.

```
----------------------------------------------------------------------------------------

Parcours en largeur:
Ordre: [1, 2, 3, 4, 8, 7, 6, 5, 10, 9]
Pere: [0, 0, 0, 0, 7, 2, 1, 0, 6, 7]
Niveau: [0, 1, 1, 1, 2, 2, 2, 1, 3, 2]

Parcours en profondeur:
Début: [1, 9, 8, 7, 14, 6, 5, 2, 4, 3]
Fin: [20, 10, 11, 12, 15, 13, 16, 19, 17, 18]
Pere: [0, 2, 3, 5, 6, 6, 8, 0, 9, 7]

----------------------------------------------------------------------------------------
```

FIGURE 2 – Tableaux à obtenir à l'issue des parcours.

1. Regardez les intialisations des différents tableaux et du graphe G, ainsi que les fonctions d'affichage. Noter que, pour le parcours en profondeur, l'algorithme donné en cours 'détruit' le graphe donné en instance puisqu'on dépile les sommets de chaque liste. Il est ainsi nécessaire ici d'en conserver une copie, `G1`, qui sera nécessaire pour l'affichage.

2. Implémenter le parcours en largeur. On utilisera la liste `aTraiter` comme une file ici. Penser à décommenter la boucle '`while`', et l'affichage graphique après le parcours.

   ▷ Si `L` est une liste, `L[0]` donne la première valeur, `L.pop(0)` supprime le premier élément et `L.append(v)` ajoute v à la fin de la liste.
   ▷ Noter que '`while aTraiter:`' permet de faire continuer la boucle tant que `aTraiter` est non vide.

3. Implémenter le parcours en profondeur. On utilisera la liste `aTraiter` comme une pile ici. Penser à décommenter la boucle '`while`', et l'affichage graphique après le parcours.

   ▷ Si `L` est une liste, `L[-1]` donne le dernier élément de L, `L.pop()` supprime le dernier élément et `L.append(v)` ajoute v à la fin de la liste.

4. *(Bonus)* Faire générer un graphe aléatoire sur 1000 sommets, où chaque arête à une probabilité de 1/50 d'exister. Effectuer un parcours en largeur et en profondeur depuis le sommet 0 et pour chaque parcours calculer le niveau maximum obtenu (on calculera le niveau dans le parcours en profondeur de la même façon dont il est calculé dans le parcours en largeur). Faire plusieurs essais et expliquer les résultats.

Exercice 2. *Plus courts chemins, algorithme de Dijkstra*

Le but de cette partie est d'implémenter l'algorithme de Dijkstra vu en cours. Le fichier *TP5Exo2Dijkstra.py* contient une trame de programme.
Un graphe à n sommets a, par défaut, comme ensemble de sommets $\{0,\dots,n-1\}$ et est encodé par listes de voisins. De plus, pour cet exercice, on se donne une matrice `longueur` où pour chaque arête ij existante, sa longueur est spécifiée dans $longeur[i][j]$. **Dans cette matrice, ainsi que tout l'algorithme, on prendra 10000 comme valeur correspondant à $+\infty$.**
Un graphe fixe G est donné et vous devez implémentez le calcul des plus courts chemins dans G depuis le sommet 0 par l'algorithme de Dijkstra. En plus de l'arbre de parcours, il faudra afficher les tableau `pere`, et `dist` produit par l'algorithme. Les résultats attendus sont donnés ci-dessous.

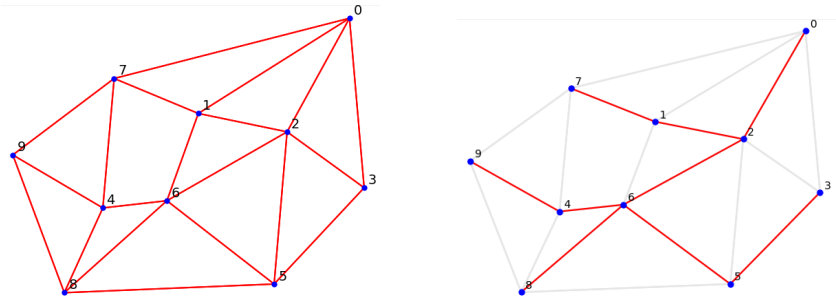

FIGURE 3 – Le graphe G fixé et l'arbre des plus courts chemins depuis le sommet 0 calculé par l'algorithme de Dijkstra.

```
-------------------------------------------------------------------------------------------

Calcul de distance avec Dijkstra:
Pere: [0, 2, 0, 5, 6, 6, 2, 1, 6, 4]
Distance: [0, 4, 3, 8, 6, 6, 5, 8, 8, 8]

-------------------------------------------------------------------------------------------
```

FIGURE 4 – Tableaux à obtenir à l'issue de l'algorithme de Dijkstra.

1. Compléter la fonction `ChoixATraiterMin` qui permet de trouver le sommet x avec $traite[x] = 1$ et une valeur $dist[x]$ minimum pour cela.
2. Compléter l'algorithme de Dijkstra. Penser à décommenter l'affichage graphique pour visualier vos résultat.
3. *(Bonus)* Gérer les sommets à traiter avec un tas afin d'améliorer la complexité de votre algorithme.

4. *(Bonus)* Générer un ensemble de 200 points au hasard dans le plan et considérer le graphe complet formé sur ces 200 points muni de la distance sur ses arêtes donnée par la distance euclidienne entre les deux extrémités de l'arête considérée.
   Calculer un arbre des plus courts chemins de racine le sommet 0.