TD5: Graphe

PC0, L2 Informatique UBO

21 octobre 2022

But : Algorithme sur les graphes

1 Graphe

1.1 Graphe avec des noeuds pondérés

Reprendre la liste des noeuds pondérés définie dans le TP4 (cette liste est notée $listeNP$).

1. A partir de cette liste ($listeNP$), définir le graphe associé à la figure 1. Ce graphe est une instance de **Digraph**, cette classe définie en cours et ne sera pas modifiée.
2. Pour définir une méthode d'affichage spécifique, notamment pour faire apparaitre les poids associés aux noeuds, on définit une sous-classe de *Digraph* que l'on note **DigraphWeight** qui redéfinit l'affichage (méthode `__str__` à redéfinir). Le graphe doit être alors une instance de **DigraphWeight**.

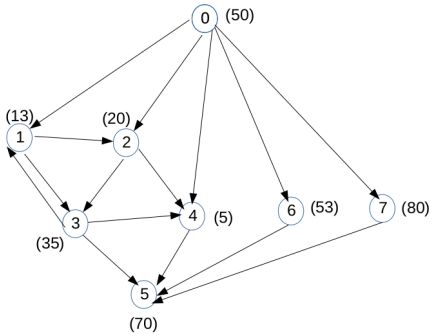

FIGURE 1 – Graphe avec des noeuds pondérés du TP4

Affichage du graphe correspondant
(0,50)->(1,13)
(0,50)->(2,20)
(0,50)->(4,5)
(0,50)->(6,53)
(0,50)->(7,80)
(1,13)->(2,20)
(1,13)->(3,35)
(2,20)->(3,35)
(2,20)->(4,5)
(3,35)->(4,5)
(3,35)->(5,70)
(3,35)->(1,13)
(4,5)->(5,70)
(6,53)->(5,70)
(7,80)->(5,70)

3. Utiliser les méthodes BFS et DFS du cours pour parcourir ce graphe et trouver le plus court chemin (en nombre de noeuds visités) pour aller du noeud 0 au noeud 5.
4. Définir une nouvelle méthode permettant de trouver tous les chemins partant du noeud 0 pour aller au noeud 5 (vous pouvez vous inspirer des méthodes BFS et DFS)

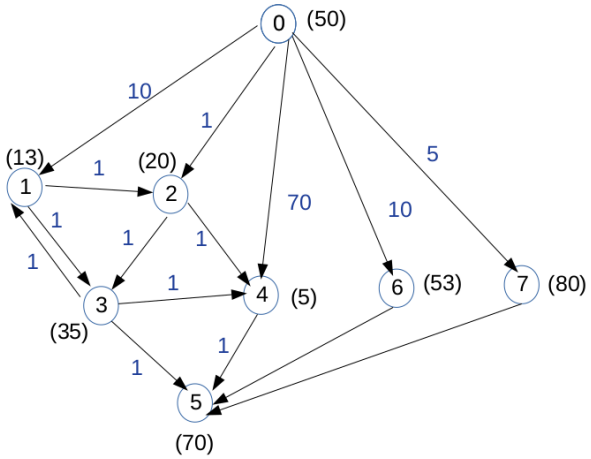

FIGURE 2 – Graphe avec des noeuds et des arcs pondérés

1.2 Graphe avec des noeuds et des arcs pondérés

On redéfinit le graphe avec des arcs pondérés comme proposé dans la Figure 2.

1. Définir la classe **WeightEdge** comme sous-classe de la classe **Edge** dans laquelle on définira les méthodes suivantes
   (a) méthode de création `__init__` où les poids des arcs sont mis par défaut à la valeur 1
   (b) accesseur pour lire la valeur du poids
   (c) méthode d'affichage
2. Rajouter les méthodes suivantes dans la classe **DigraphWeight**
   (a) *addWeightEdge* qui rajoute un arc pondéré au graphe. Le dictionnaire définissant la liste d'adjacences inclut des couples du type $(noeud, poidsArc)$
   (b) *sumArc* une méthode qui calcule la somme des poids des arcs allant d'un noeud père vers ses noeuds fils.
   (c) *poids* qui trouve le poids d'un arc dans graphe. Cette méthode passe en paramètre les deux noeuds et retourne la valeur du poids de l'arc qui les lie dans le graphe.

Définir les fonctions suivantes sur un chemin du graphe :

1. $poidsArcpath(path)$: fonction qui calcule les poids cumulés des arcs associés au chemin
2. $poidspath(path)$: fonction qui calcule les poids cumulés des arcs et des noeuds associés au chemin

A partir de l'ensemble des chemins possibles pour aller du noeud 0 au noeud 5, trouver celui qui a le poids (arc + noeud) le plus petit.