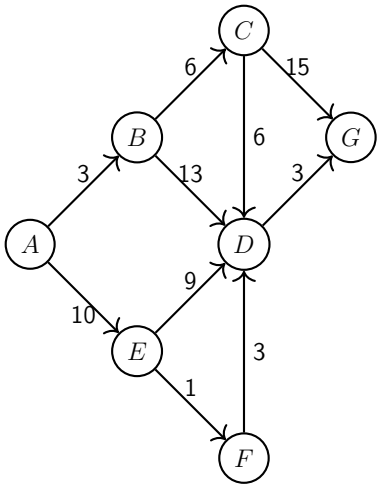
Programmation 3: TD3

Exercice 1 : Application de l'algorithme de Dijkstra.



1. Appliquez l'algorithme de Dijkstra sur le graphe orienté et pondéré ci-dessus pour déterminer la distance entre le sommet A et les autres sommets du graphe. Vous indiquerez dans le tableau ci-dessous la distance et le parent associés à chaque sommet et à chaque étape de l'algorithme.

étape-sommet	A	B	C	D	E	F	G
0							
1							
2							
3							
4							
5							
6							

2. Déduisez-en le plus court chemin entre A et G.

Exercice 2 : Implémenter une classe de graphe

Écrivez une classe *Graphe* générique permettant de représenter des graphes orientés et éventuellement pondérés. Cette classe utilisera des dictionnaires pour avoir une approche similaire à celle de la liste d'adjacence vu en cours. Cette classe sera dotée des méthodes suivantes:

1. Un constructeur sans paramètre qui crée un graphe vide (sans sommet ou arc).

2. Une méthode *ajouter_sommet(self,s)* qui ajoutera le sommet s au graphe. Le sommet s peut être un entier, un tuple, une chaine de caractère, ... Si le sommet appartient déjà au graphe, la méthode n'aura aucun effet.

3. Une méthode *ajouter_arc(self,a,p=1)* qui ajoutera l'arc a au graphe avec un poids p. L'arc est lui-même un tuple composé de deux sommets $(s1, s2)$. Si un des deux sommets n'appartient pas au graphe, il y sera ajouté. Si le poids n'est pas précisé, il sera considéré comme étant de 1. Si l'arc appartient déjà au graphe avec le même poids, la méthode n'aura aucun effet.

4. Une méthode *supprimer_arc(self,a)* qui supprime l'arc a du graphe s'il existe.

5. Une méthode *supprimer_sommet(self,s)* qui supprime le sommet s du graphe s'il existe, ainsi que tous les arcs associés.

6. Une méthode *sommets(self)* qui retourne l'ensemble de tous les sommets du graphe.

7. Une méthode *voisins_sortants(self,s)* qui retourne l'ensemble de tous les voisins sortants de s dans G.

8. Une méthode *voisins_entrants(self,s)* qui retourne l'ensemble de tous les voisins entrants de s dans G.

9. Une méthode *poids_arc(self,a)* qui retourne le poids associé à l'arc a dans G.

Exercice 3 : Programmer l'algorithme de Dijkstra.

Considérez l'algorithme en pseudo code vue en cours:

algorithme DIJKSTRA(G, s)
Entrées : $G = (S, A, w)$ un graphe pondéré et s un sommet de G
Sortie : Un dictionnaire d qui à chaque sommet associe sa distance à s. Un dictionnaire p qui à chaque sommet associe son parent.

```
1  F := S,          initialement F contient tout S
2  pour chaque sommet u ∈ S faire
3      d[u] := ∞,
4      p[u] := AUCUN,
5  d[s] := 0,
6  tant que  F ≠ ∅ faire
7      u :=EXTRAIRE_LE_MIN(F),
9      pour chaque sommet v ∈ G.voisins_sortants(u) faire
10         si d[v] > d[u] + w(u, v) alors
11             d[v] = d[u] + w(u, v),
12             p[v] = u,
13     fin pour,
14 fin tant que,
15 renvoyer d, p,
```

Implémentez cet algorithme en python dans une fonction dijkstra(G, s) qui prend en paramètre un graphe G tel que définit précédemment et un sommet source s.

Exercice 4 : Utilisation de l'algorithme de Dijkstra.

La tour d'Hanoïa est un jeu de réflexion imaginé par le mathématicien français Édouard Lucas. Ce jeu consiste à déplacer 7 disques de diamètres différents sur trois tours: une tour de « départ » sur laquelle sont originellement placés tous les disques, une tour d'« arrivée » qui à la fin du jeu doit contenir tous les disques et enfin une tour « intermédiaire ».

Le joueur doit respecter les règles suivantes :

1. on ne peut déplacer plus d'un disque à la fois ;
2. on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

On suppose que cette dernière règle est également respectée dans la configuration de départ. Le but est de minimiser le nombre de coup nécessaire pour arriver à déplacer tous les disques de la tour de « départ » à la tour d'« arrivée ».

À l'aide de la classe Graphe et de la fonction dijkstra(G, s) précédemment définies, écrivez un programme python qui représente ce problème sous la forme d'une recherche de chemin dans un graphe et calcule la stratégie optimale pour gagner le jeu en un minimum de coup.

Pour ceci posez-vous les questions suivantes:

- Comment peut-on représenter une étape quelconque du jeu? (*un sommet du graphe*)
- Comment peut-on passer d'une étape à l'autre du jeu? (*un arc du graphe*)
- Quelle représentation pour l'étape initiale du jeu? (*le sommet source dans le graphe*)
- Quelle représentation pour l'étape finale (gagnante) du jeu? (*le sommet destination du graphe*)