TP n°4 sur les graphes : plus courts chemins dans des graphes pondérés

Dans tout ce TP, nous ne considérerons que des graphes pondérés avec des poids positifs. Ils peuvent être orientés ou pas.

1 Le principe de l'algorithme de Dijkstra

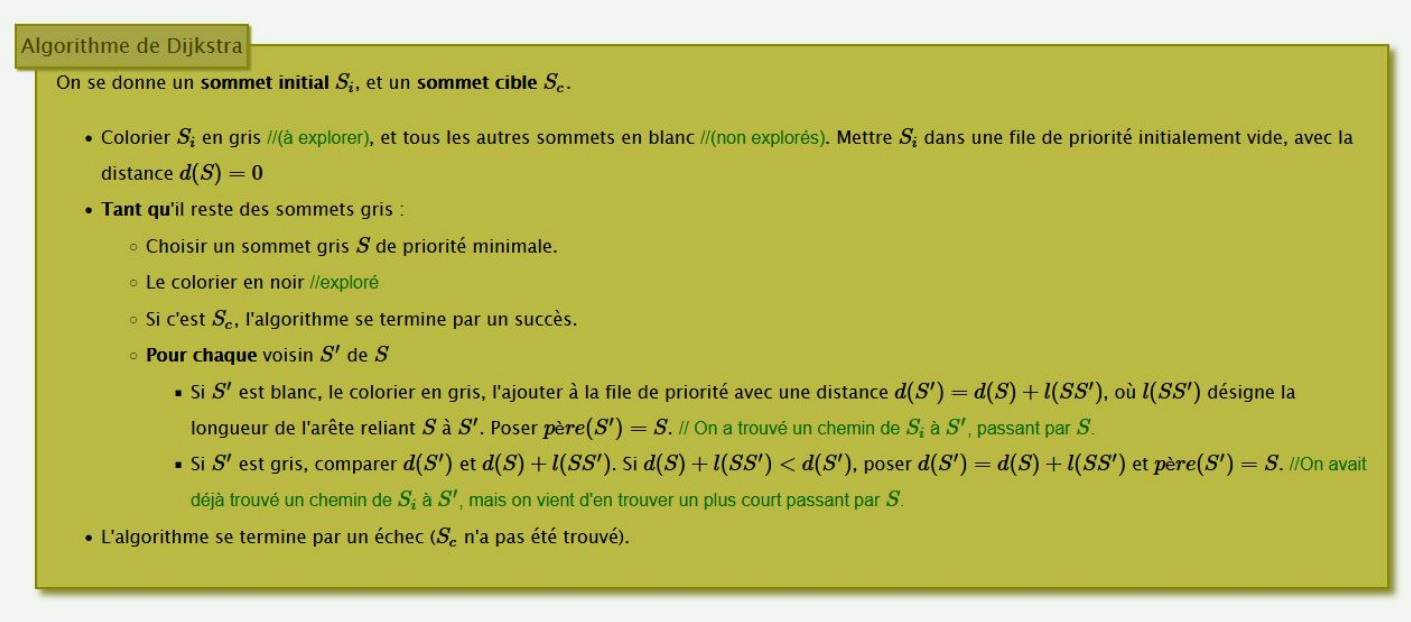

FIGURE 1 – Algorithme de Dikstra

Exercice 1 Avec un papier et un crayon, appliquer l'algorithme de Dijkstra pour les graphes ci-dessous :

1. pour le graphe 1 la source est le sommet a et la cible le sommet e (réponse : chemin $[a, b, c, e]$ de longueur 7).
2. pour le graphe 2, la source est le sommet A et la cible le sommet F (réponse : chemin $[A, D, E, F]$ de longueur 11).
3. pour le graphe 3, la source est le sommet 1 et la cible le sommet 5 (réponse : chemin $[1, 3, 6, 5]$ de longueur 20).

2 Implémentation

Exercice 2 Écrire une fonction `init(g)` qui prend en argument g un graphe défini par dictionnaire d'adjacence et qui renvoie un dictionnaire `etat` dont les clés sont les sommets du graphe et les valeurs sont toujours égales à 0. Par exemple si le graphe g a trois sommets `'A','B','C'` alors `etat` vaudra `'A':0, 'B':0, 'C':0` .

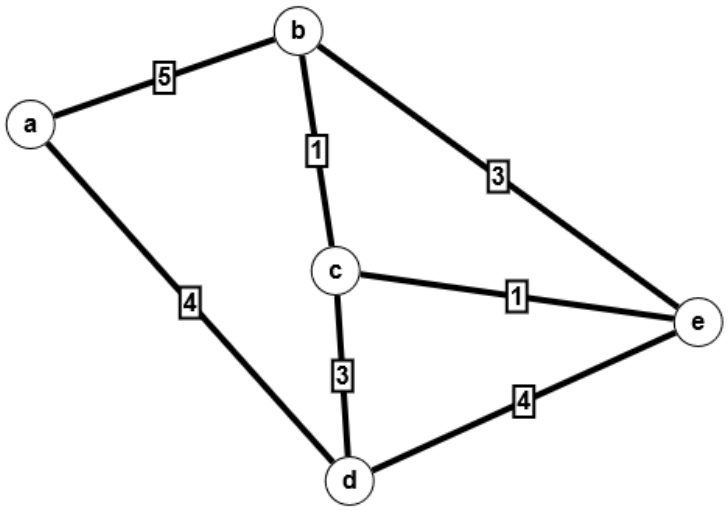

FIGURE 2 – graphe 1

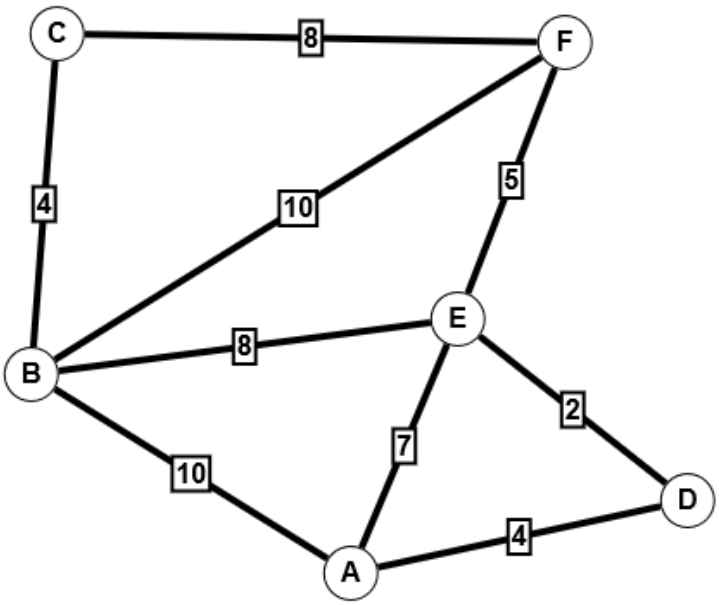

FIGURE 3 – graphe 2

Exercice 3 Écrire une fonction `extraire_min(t, dist)` qui prend en argument t une liste de sommets et *dist* un dictionnaire dont les clés sont des sommets et dont les valeurs sont des distances (représentés par des flottants). La fonction doit trouver le sommet s de t dont la distance d est minimale. Elle doit remplir deux conditions :

- supprimer ce sommet s de la liste t à l'aide de la méthode `remove`.
- renvoyer ce sommet s.

Par exemple, si $t = [4, 10, 12]$ et $dist = \{4 : 11, 10 : 3, 12 : 8, 1 : 45, 25 : 2\}$, le sommet de t

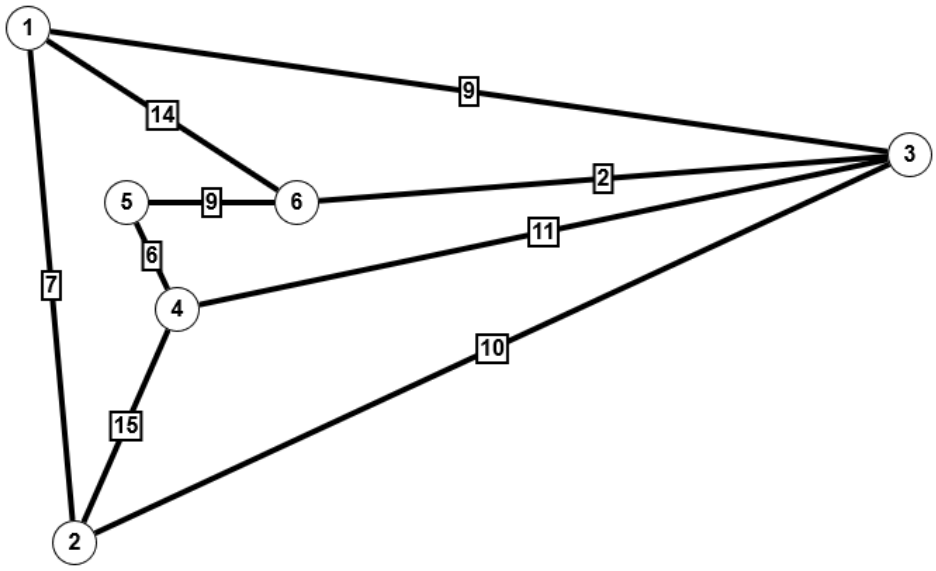

FIGURE 4 – graphe 3

de distance minimale est le 10, car sa distance associée est 3 (le sommet 25 a une distance de 2 mais il n'est pas dans t. Ainsi lorsque l'on exécute `extraire_min(t, dist)` , cela doit renvoyer 10 et après exécution la liste t est égale à $[4, 12]$. On pourra utiliser l'instruction `infinity = float('inf')` pour représenter un nombre infini.

Exercice 4 (Dijkstra, implémentation1) On se donne G un graphe pondéré à poids positifs défini par dictionnaire d'adjacence, puis `source` et `cible` deux sommets de ce graphe. On veut écrire en Python une fonction booléenne `dijkstra(G, source, cible)` qui renvoie la plus courte distance de la source à la cible, si l'on peut accéder de `source` à `cible`. Sinon, cela renvoie l'infini.

Pour cela on utilisera :

- une liste `salle_attente` dans laquelle on va stocker les éléments de la salle d'attente. Elle sera initialisée à `source`.
- la méthode `remove` qui permet d'enlever un élément quelconque d'une liste Python, par exemple si `liste = [10, 20, 45]`, `liste.remove(20)` enlève 20 de la liste.
- un dictionnaire `etat` dans lequel on va stocker les états (0 pour blanc, 1 pour gris et 2 pour noir) de chaque sommet.
- un dictionnaire `dist` des distances dont les clés sont les sommets et les valeurs sont les distances «en cours» à la source. On initialisera par `dist[source] = 0`.

Exercice 5 Tester votre fonction sur les deux graphes donnés dans le fichier élève `fichier_eleve_dijkstr`

Lorsque l'on explore le graphe, on dit qu'un sommet s est le père d'un sommet v si le sommet v est entré dans la salle d'attente en tant que voisin de s (v est donc dans la liste d'adjacence de s). On peut stocker cette relation de parenté dans un dictionnaire `parents` en écrivant `parents[v] = s`. On convient que le père du sommet `source` est `None`.

Par exemple, si l'on parcourt le graphe 1 pour accéder du sommet 0 au sommet 3, on obtient le dictionnaire `parents = {0: None, 9: 0, 2: 0, 1: 9, 3: 2, 5: 2, 6: 2}`. Le chemin entre 0 et 3 est donné par la liste $[0, 2, 3]$.

Exercice 6 (Des parents au chemin) Écrire une fonction `chemin(parents, source, cible)` qui renvoie le chemin parcouru pour aller de `source` à `cible`.

Exercice 7 Modifier votre fonction pour qu'elle renvoie aussi le plus court chemin. On utilisera donc un dictionnaire `dist` des distances dont les clés sont les sommets et les valeurs sont les distances «en cours» à la source. On initialisera par `dist[source] = 0`.