Algorithme de Dijkstra

1. Introduction

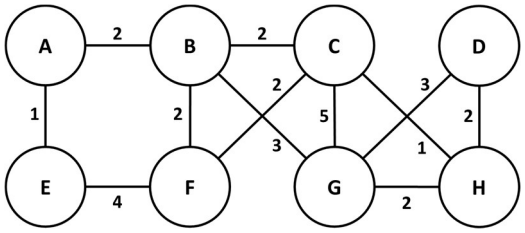
La plupart des algorithmes de recherche de chemin (ou de chaine), travaillent sur des graphes pondérés (par exemple pour rechercher la route entre un point de départ et un point d'arrivée dans un logiciel de cartographie, ou pour le chemin sur un réseau appliquant le protocole OSPF). Ces algorithmes recherchent aussi souvent les chemins les plus courts ou les plus rapides (logiciels de cartographie, routage pour les réseaux…). On peut citer l'algorithme de Dijkstra ou encore l'algorithme de Bellman-Ford qui recherchent le chemin le plus court entre un nœud de départ et un nœud d'arrivée dans un graphe pondéré.

Le mathématicien et informaticien néerlandais **Edsger Dijkstra** est à l'origine de cet algorithme.

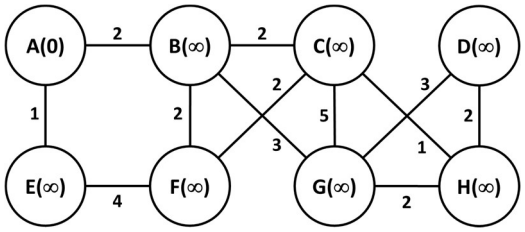
2. Principe de l'algorithme de Dijkstra

La vidéo suivante illustre le principe de l'algorithme de Dijkstra : https://youtu.be/JPeCmKFrKio

On considère le graphe pondéré suivant :

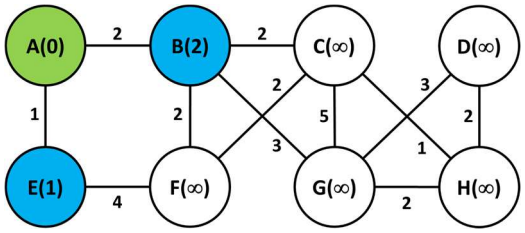


On recherche le plus court chemin entre le sommet A et chacun des autres sommets du graphe.
On commence par affecter une valeur très grande (+∞) à chacun des sommets du graphe et la valeur 0 au sommet A (ces valeurs sont les distances des chemins entre le sommet A et les autres sommets du graphe).



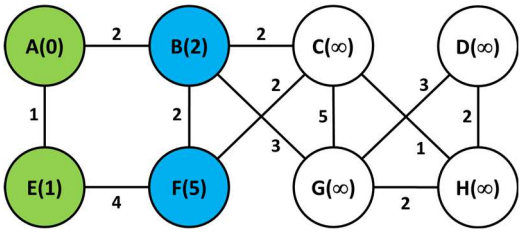
Sommets non traités : (A : 0), (B : ∞), (C : ∞), (D : ∞), (E : ∞), (F : ∞), (G : ∞), (H : ∞).

1er tour : à partir du sommet (non traité) qui a la plus petite valeur (A), on traite la distance de chacun de ses voisins (non traités). À partir de A, on peut aller à B et E avec les valeurs respectives 2 et 1.



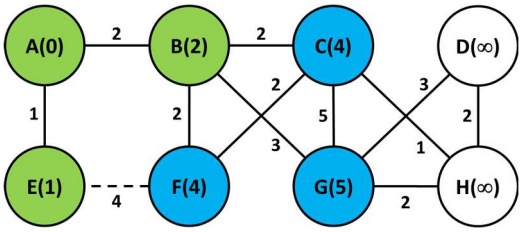
Sommets traités : (A : 0).
Sommets non traités : (B : 2), (C : ∞), (D : ∞), (E : 1), (F : ∞), (G : ∞), (H : ∞).

2ème tour : à partir du sommet (non traité) qui a la plus petite valeur (E), on traite la distance de chacun de ses voisins (non traités). À partir de E, on peut aller à F avec une distance de 4, soit un total de 5 depuis A.



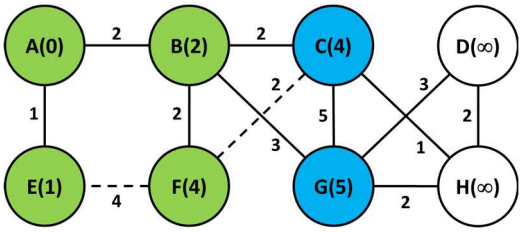
Sommets traités : (A : 0), (E : 1).
Sommets non traités : (B : 2), (C : ∞), (D : ∞), (F : 5), (G : ∞), (H : ∞).

3ème tour : à partir du sommet (non traité) qui a la plus petite valeur (B), on traite la distance de chacun de ses voisins (non traités). À partir de B, on peut aller à C, Fet G avec les distances respectives de 2, 2 et 3 donc pour des distances totales respectives de 4, 4 et 5 (pour le sommet F, la valeur calculée (4) remplace la valeur précédente (5) car elle est inférieure).



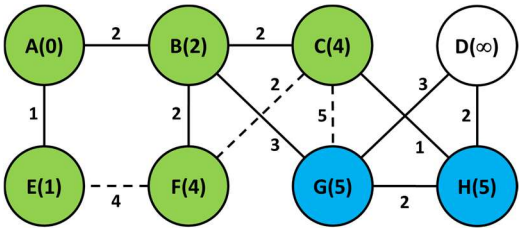
Sommets traités : (A : 0), (E : 1), (B : 2).
Sommets non traités : (C : 4), (D : ∞), (F : 4), (G : 5), (H : ∞).

4ème tour : À partir du sommet (non traité) qui a la plus petite valeur (F), on traite la distance de chacun de ses voisins (non traités). À partir de F, on peut aller à C avec une distance de 2 soit une distance totale de 6. On garde la distance minimale de C qui est de 4.



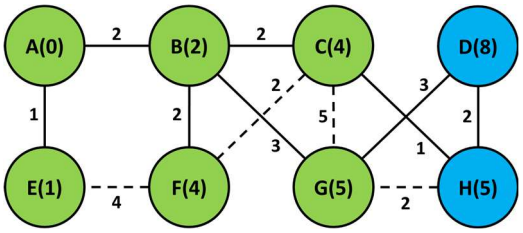
Sommets traités : (A : 0), (E : 1), (B : 2), (F : 4).
Sommets non traités : (C : 4), (D : ∞), (G : 5), (H : ∞).

5ème tour : à partir du sommet (non traité) qui a la plus petite valeur (C), on traite la distance de chacun de ses voisins (non traités). À partir de C, on peut aller à G et H avec des distances respectives de 5 et 1, soit des distances respectives de 9 et 5. On garde la distance de 5 pour G.



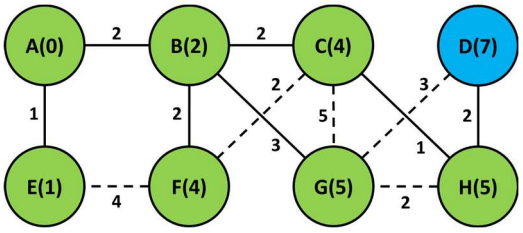
Sommets traités : (A : 0), (E : 1), (B : 2), (F : 4), (C : 4).
Sommets non traités : (D : ∞), (G : 5), (H : 5).

6ème tour : à partir du sommet (non traité) qui a la plus petite valeur (G), on traite la distance de chacun de ses voisins (non traités). À partir de G, on peut aller à D et H avec des distances respectives de 3 et 2, soit des distances respectives totales de 8 et 7. On garde la distance de 5 pour H.



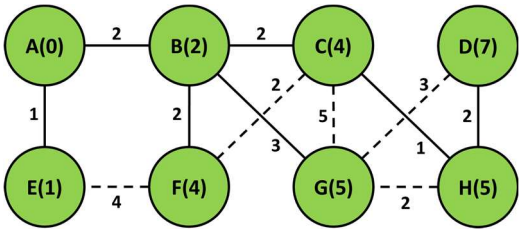
Sommets traités : (A : 0), (E : 1), (B : 2), (F : 4), (C : 4), (G : 5).
Sommets non traités : (D : 8), (H : 5).

7ème tour : À partir du sommet (non traité) qui a la plus petite valeur (H), on traite la distance de chacun de ses voisins (non traités). À partir de H, on peut aller à D avec une distance de 2, soit un total de 7. On améliore la distance de D avec la valeur de 7.



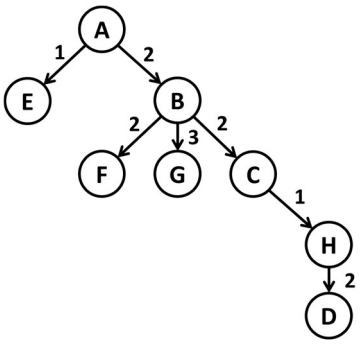
Sommets traités : (A : 0), (E : 1), (B : 2), (F : 4), (C : 4), (G : 5), (H : 5).
Sommets non traités : (D : 7).

8ème tour : à partir du sommet (non traité) qui a la plus petite valeur (D), on traite la distance de chacun de ses voisins (non traités). Ici D n'a pas de voisions non traités. Les sommets sont maintenant tous traités.



Sommets traités : (A : 0), (E : 1), (B : 2), (F : 4), (C : 4), (G : 5), (H : 5), (D : 7).

Au final, l'arborescence à partir des distances obtenues est représentée par l'arbre ci-contre.



3. Implémentation de l'algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme glouton, car à chaque étape il fait un choix optimal.

L'algorithme suivant recherche le plus court chemin entre le sommet de départ et chacun des autres sommets du graphe.

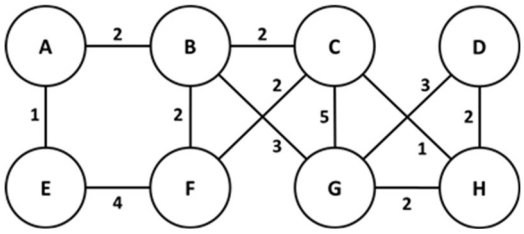
```
On crée une structure de données sommets non traités dans lequel on stocke tous les sommets
en leur attribuant la distance +∞.
Dans sommets non traités, au sommet de départ, on attribue la distance 0.
On crée une structure de données sommets traités.
Tant qu'il y a des sommets non traités faire
     Sélectionner parmi les sommets non traités, le sommet ayant la distance la plus petite
     (sommet mini)
     Pour chaque voisin du sommet mini faire
          Si le voisin ne fait pas parti des sommets traités alors
               Si la distance du sommet mini + la distance entre le voisin et le sommet mini
               est inférieure à la distance du voisin alors
                    Remplacer la distance du voisin par la distance calculée
     Placer le sommet mini dans les sommets traités
     Supprimer le sommet mini des sommets non traités
Retourner les sommets traités
```

Pour attribuer la distance +∞, importer la méthode **inf** du module **math**.

```
from math import inf
```

Le code ci-dessous permet d'implémenter le graphe ci-contre :

```
g = {'A': {'B': 2, 'E': 1},
     'B': {'A': 2, 'C': 2, 'F': 2, 'G': 3},
     'C': {'B': 2, 'F': 2, 'G': 5, 'H': 1},
     'D': {'G': 3, 'H': 2},
     'E': {'A': 1, 'F': 4},
     'F': {'B': 2, 'C': 2, 'E': 4},
     'G': {'B': 3, 'C': 5, 'D': 3, 'H': 1},
     'H': {'C': 1, 'D': 2, 'G': 1}}
```



La fonction **voisins(graphe, sommet)** renvoie les voisins d'un sommet.

```
def voisin(graphe, sommet):
    """renvoie les voisins d'un sommet"""
    return graphe[sommet]
```

✍ Implémenter la fonction **dijkstra(graphe, depart)** décrite ci-dessus. Pour récupérer le sommet ayant la distance la plus petite, créer une fonction **val_mini_sommets(liste_sommets)**.

4. Recherche du chemin le plus court entre deux sommets

Pour veut connaître le chemin à suivre pour la plus courte distance entre deux sommets, on ajoute à chaque étape de l'algorithme précédent une mémorisation du parent du sommet traité.

```
On crée une structure de donnée route dans lequel sera stocké les sommets traités et leur parent.
Dans route, au sommet de départ on attribue la valeur None car le départ n'a pas de parent.
On crée une structure de données sommets non traités dans lequel on stocke tous les sommets en leur attribuant la distance +∞.
Dans sommets non traités, au sommet de départ, on attribue la distance 0.
On crée une structure de données sommets traités.
Tant qu'il y a des sommets non traités faire
     Sélectionner parmi les sommets non traités, le sommet ayant la distance la plus petite (sommet mini)
     Pour chaque voisin du sommet mini faire
          Si le voisin ne fait pas parti des sommets traités alors
               Si la distance du sommet mini + la distance entre le voisin et le sommet mini est inférieure à la distance du voisin alors
                    Remplacer la distance du voisin par la distance calculée
                    Mettre dans route le voisin avec son parent sommet mini
     Placer le sommet mini dans les sommets traités
     Supprimer le sommet mini des sommets non traités
Retourner route
```

✍ Modifier la fonction **dijkstra(graphe, depart)** afin d'obtenir la fonction **dijkstra_chemin(graphe, depart)**.

Il reste maintenant à afficher le chemin entre le départ et l'arrivée du graphe.

```
Initialiser une liste solution à vide
Stocker le parcours du graphe à partir de départ avec les parents dans un dictionnaire parent
Initialiser le sommet visité à arrivée
Tant que le sommet visité à un parent faire
     solution ← sommet visité
     le sommet visité devient le parent
Fin tant que
Renvoyer la liste solution (de départ à arrivée)
```

✍ Implémenter une fonction **chemin(graphe, depart, arrivee)** prenant en paramètre un graphe, le sommet de départ et le sommet d'arrivé et retournant la route à suivre sous forme d'une liste de sommets.

```
>>> chemin(g, 'A', 'H')
['A', 'B', 'C', 'H']
```

Sources : https://isn-icn-ljm.pagesperso-orange.fr/NSI-TLE/co/section_chapitre4.html
http://math.univ-lyon1.fr/irem/Formation_ISN/formation_parcours_graphes/dijkstra/1_algorithme.html