ITC – TD n°12-5

Dijsktra & co

I. Dijsktra naïf

On rappelle que l'algorithme de Dijsktra ressemble à l'algorithme de parcours en largeur, dans lequel on remplace la file par une file de priorité, et où on peut changer le précédent par la procédure de relâchement d'un arc.

On représentera naïvement la file de priorité comme un dictionnaire ayant pour clé les sommets u **découverts** et pour valeur la distance entre ce sommet u et le départ à un stade donné du parcours. On initialisera donc ce dictionnaire avec uniquement le sommet de départ (distance 0), on créera / mettra à jour les clés au cours du parcours, et on retirera les clés lorsqu'elles sont explorées.

1 – Écrire une fonction `extraire_min(prio: dict)` qui prend une file de priorité-dictionnaire, le parcourt pour déterminer la clé de plus petite priorité, puis retire cette clé du dictionnaire et renvoie la clé. Cette fonction devra être en $\mathcal{O}(n_S)$ dans le pire cas.

2 – Écrire une fonction `dijsktra(g: dict, dep: str)` qui prend en paramètre un graphe g et un sommet de départ dep, et qui renvoie les dictionnaires des précédents et des distances issus de l'algorithme de Dijkstra.

On souhaite à présent afficher le résultat de l'algorithme sous la forme d'un arbre d'exploration.

3 – Écrire une fonction `extraire_arbre(g: dict, prec: dict)` qui prend en paramètres le graphe et le dictionnaire des précédents, et qui créé et renvoie un graphe ne contenant que les arêtes produisant l'arbre des plus courts chemins.

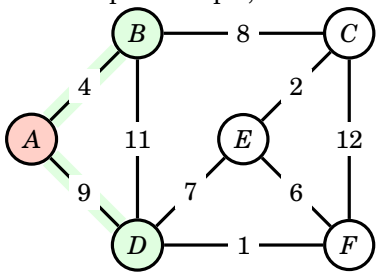
4 – Tester l'algorithme en utilisant les instructions suivantes :

```
g = gr.recuperer_graphe('g4', ponderation=True)
dep = 'h'
prec, dists = dijsktra(g, dep)
arbre_d = extraire_arbre(g, prec)
gr.afficher_graphes_multiples([g, arbre_d], labels_sommets=[None, dists])
```

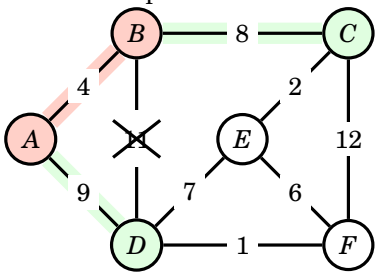
II. Problème de l'arbre couvrant minimal : algorithme de Prim

Étant donné un graphe connexe pondéré $G = (S, A)$, un arbre couvrant est un sous-graphe $G' = (S, A')$, $A' \subset A$, tel que G' soit un arbre (il ne contient donc pas de cycles) ; l'arbre couvrant minimal est celui dont la somme des poids des arêtes est le plus petit possible.

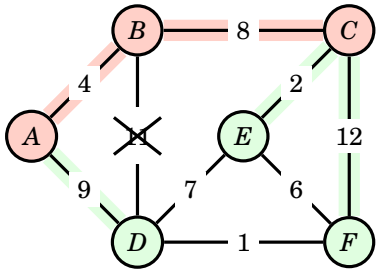
On peut créer un tel arbre couvrant minimal en suivant l'algorithme de Prim[1], qui construit l'arbre en partant d'un sommet et en ajoutant les sommets dans l'ordre des arêtes minimales : par exemple, sur l'exécution ci-dessous partant du sommet A :



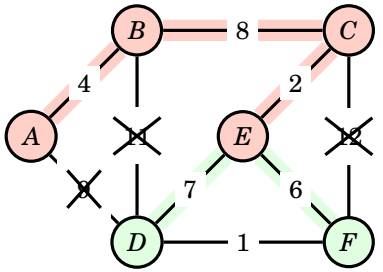
Prochains sommets : B, D



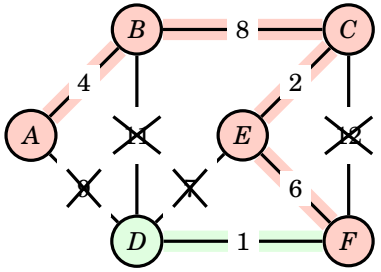
Prochains sommets : C, D



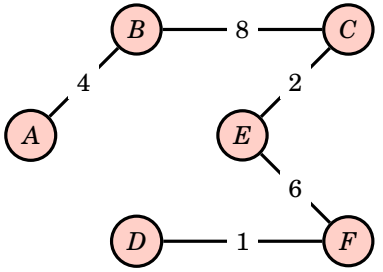
Prochains sommets : D, E, F



Prochains sommets : D, F



Prochains sommets : D



Arbre final

5 – Écrire une fonction `prim(g: dict)` qui exécute l'algorithme de Prim sur le graphe g.

6 – Comparer cet arbre à celui obtenu avec Dijkstra depuis le sommet `'h'` :

```
acm = prim(g)
gr.afficher_graphes_multiples([g, arbre_d, acm], \
   labels_sommets=[None, dists, None])
```

[1] Ou Prim-Jarnik-Dijsktra