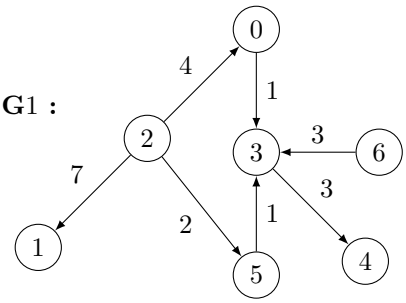
TP 13 - Graphes pondérés

Dans ce TP, on aborde l'implémentation des graphes pondérés et l'algorithme de Dijsktra.

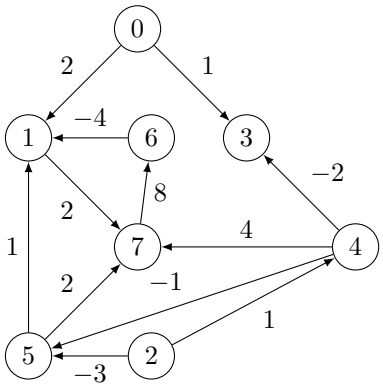
1 Graphes pondérés

On considère les graphes ci-dessous :

G1 :



G2 :



Pour pourvoir travailler sur de tels graphes, il faut déjà être capables de les implémenter sous forme de listes d'adjacence ou de matrice d'adjacence.

Question 1. Définir des variables représentant les deux graphes précédents sous forme de listes d'adjacence et de matrices d'adjacence.

Comme le graphe est pondéré, on peut étendre la notion de poids, i.e. la fonction ω, aux boucles comme étant 0 (ex : le poids de l'arc $(0, 0)$ est 0) et aux arcs inexistants comme étant $+\infty$.

Question 2. Dans les graphes précédents, donner les poids des arcs $(3, 4)$, $(4, 3)$ et $(3, 3)$.

Question 3. Ecrire une fonction `poidsArc` qui prend en entrées un graphe `G` représenté par listes d'adjacence et deux sommets `s` et `t` et qui renvoie le poids de l'arc `(s,t)` dans le graphe `G`.

Question 4. Même question avec un graphe `G` représenté par matrice d'adjacence.

On peut également étendre la notion de poids aux chemins comme étant la somme des poids des arcs composant le chemin.

Question 5. Dans les graphes précédents, donner les poids des chemins $2, 5, 3, 4$ et $2, 5, 1$.

Question 6. Ecrire une fonction `poidsChemin` qui prend en entrées un graphe `G` représenté par listes d'adjacence et un chemin `c`, de longueur $p \geq 1$, sous la forme d'une liste de sommets et qui renvoie le poids du chemin `c` dans le graphe `G`.

Question 7. Même question avec un graphe `G` représenté par matrice d'adjacence.

Une des premières question, qui découle de cette définition de poids, est la question du calcul du poids minimal d'un chemin menant d'un sommet `s` à un sommet `t`. Autrement dit, le calcul de la distance entre un coule de sommets `(s,t)`.

Question 8. Donner la distance de 2 à 4. Justifier.

2 Algorithme de Dijkstra

Dans le cas de graphes non pondérés, on a vu que la distance entre deux sommets s'obtenait facilement à l'aide d'un parcours en largeur. Dans le cas d'un graphe pondéré par des poids positifs, elle s'obtient à partir d'un algorithme généralisant le parcours en largeur : l'algorithme de Dijkstra.

À partir d'un sommet source $s0 \in S$ l'algorithme de Dijkstra va progressivement remplir une liste `dist` de longueur l'ordre du graphe de sorte qu'à la fin de l'algorithme la case d'indice `t` de `dist` soit égal à $\delta(s0, t)$, le poids du plus court chemin entre `s0` et `t`.

Cet algorithme tient à jour un ensemble `dejavu` destiné à représenter les sommets dont on a déterminé dans le tableau `dist` le poids du chemin minimal à partir de `s0`. Pour les sommets n'appartenant pas à `dejavu`, le

tableau `dist` contiendra le poids du plus court chemin ne passant que par des sommets appartenant à `dejavu`. Ainsi `dejavu` contiendra les sommets dont on a déjà parcouru les voisins.

À chaque itération, on choisit le sommet de `atraiter` n'appartenant pas encore à `dejavu` dont la valeur associée dans le tableau `dist` est minimale pour le transférer dans `dejavu`, et on modifie le tableau `dist` en conséquence. Ainsi, chaque sommet va progressivement être transféré dans `dejavu`.

On donne le pseudo-code de l'algorithme de Dijkstra.

```
Fonction Dijkstra(g,s0) :
    dist ← +∞ pour tout s in S
    dist[s0] ← 0
    atraiter ← {s0} #Contient les sommets qu'il reste à parcourir.
    dejavu ← ∅ #Contient les sommets dont on a calculé la distance à s0.
    Tant que atraiter est non vide faire :
        Retirer t de atraiter tel que dist[t] = min{dist[s], s  dans atraiter}
        Pour u voisin de t faire :
            Si u n'est ni dans atraiter ni dans dejavu faire :
                Ajouter u à atraiter
            dist[u] ← min (dist[u], dist[t]+ω(t,u))
        Ajouter t à dejavu #On a fini le parcours des voisins de t.
    Renvoie dist
```

Question 9. Appliquer l'algorithme de Dijkstra à la main sur le graphe 1 et le sommet 2.

Comme on n'a besoin d'aller chercher l'élément de distance minimale dans `atraiter`, on ne peut ni utiliser une file ni utiliser une pile. On décide d'implémenter `atraiter` et `dejavu` comme des listes des booléens telles que la case `i` contienne `True` si le sommet `i` est dans `atraiter` ou dans `dejavu`.

Question 10. Ecrire une fonction `estVide` qui prend en entrée une liste `l` de booléens et renvoie `True` si la liste ne contient que des `False` et `False` sinon.

Question 11. Ecrire une fonction `minimum` qui prend en entrées une liste `l` de booléens et une liste `d`, et qui renvoie l'indice du minimum de `d` restreinte aux indices de valeurs `True` dans`l`.

Par exemple, `minimum([True,False,False,True],[3,-2,1,2])` renvoie `3` qui est l'indice du minimum de `[3,-2,1,2]` quand on se restreint aux cases d'indices `0` et `3` (car `True` dans la première liste).

Question 12. Ecrire le code de l'algorithme de Dijkstra pour un graphe pondéré de poids positifs représenté sous forme de listes d'adjacence.