Elias KHODJAOUI
Alexandre COSSON

SAE 2.2 - Exploration Algorithmique Recherche de plus court chemin dans un graphe

II) Représentation d'un graphe

Dans cette partie, nous avons implémenté les classe Arc, Arcs et GrapheListe, cette dernière héritant de l'interface Graphe que nous avons également ajoutée.
Après avoir ajouté les méthodes nécessaires à l'exécution de notre code, nous avons écrit une classe TestGraphe permettant de vérifier que nos méthodes fonctionnaient bien ainsi que la classe main "Principale", reprenant le graphe fourni en exemple dans la SAE.

III) Calcul du plus court chemin par point fixe

Question 8 : Algorithme point fixe

```
algorithme
fonction pointFixe(Graphe g InOut, Noeud depart)

    taille = size(getnoeuds(g))
    noeud = depart
    arcs = getArcs(g)

    //initialisation
    L(depart) -> 0
    noeud = suivant(depart)

    pour i de 0 à taille -1 faire
        L(noeud) -> +∞
        noeud = suivant(noeud)
    fpour

    //Etapes
    pour i de 1 à taille - 1 faire
```

```
        pour j de 0 à size(arcs) - 1 faire
            u <- getDepart(arcs[j])
            a <- getArrivee(arcs[j])
            poids <- getCout(arcs[j])
            si L(u) + poids < L(a) alors
                L(a) <- L(u) + poids
                parent(a) <- u
            fsi
        fpour
    fpour

    retourne L
fin
```

Lexique:
taille : entier, nombre de noeuds dans le graphe
arcs : Arc, liste des arcs du graphe
L(X) : Valeur, valeur du noeud X
u : Noeud, noeud avec lequel on test si c'est plus avantageux de passer par celui-ci
a : Noeud, noeud de l'arc actuel dont on veut réduire la distance minimale en trouvant un autre arc
poids : décimal, poids de l'arc dont on veut voir si il est plus avantageux

Après avoir fait cet algorithme du point fixe, nous avons créé la classe BellmanFord et sa méthode statique resoudre(graphe,depart), puis avons écrit les tests nécessaires pour vérifier la bien fonction de notre algorithme. Nous avons ensuite modifié la classe principale pour envoyer les points fixes.

IV) Calcul du meilleur chemin par Dijkstra

Question 13 : Algorithme de Dijkstra

```
Algorithme
fonction resoudre(Graphe g InOut, Noeud depart)
début
        Q <- {} // utilisation d'une liste de noeuds `à traiter
        Pour chaque sommet v de G faire
                v.valeur <- Infini
                v.parent<- Indéfini
                Q <- Q U {v} // ajouter le sommet v `a la liste Q
        Fin Pour
        A.valeur <- 0
        Tant que Q est un ensemble non vide faire
                u <- un sommet de Q telle que u.valeur est minimal
                        // enlever le sommet u de la liste Q
                Q <- Q \ {u}
                Pour chaque sommet v de Q tel que l'arc (u,v) existe faire
                        d <- u.valeur + poids(u,v)
                        Si d < v.valeur
                                // le chemin est plus intéressant
                                Alors v.valeur <- d
                                        v.parent <- u
                        Fin Si
                Fin Pour
        Fin Tant que
Fin
```

Lexique :
Q : liste<Arc>, liste des noeuds à traiter
v: Arc, itération des sommets de Q
A: Arc, sommet de départ
u: Arc, valeur minimale des sommets de Q
d: reel, distance du départ jusqu'au sommet V (passe par le chemin optimal u)

Nous avons retranscrit l'algorithme précédent auquel un Lexique a été ajouté en la classe java Dijkstra qui utilise également une fonction statique resoudre(graphe,depart).
En effet, nous avons décidé, au lieu de l'implémentation d'une interface commune aux classes Dijkstra et BellmanFord, d'initialiser les méthodes résoudre des deux classes en tant que méthodes statiques.
Ainsi, pour les appeler il suffit d'écrire la ligne de code : NomClasse.resoudre(graphe,depart).
Les tests unitaires et la classe mainDijkstra ont ensuite été ajoutés.

V) Validation et expérimentation

Pour comparer les résultats, nous avons modifié la classe MainDijkstra, qui désormais renvoie le temps que met Dijkstra pour s'exécuter, puis le temps que met BellmanFord pour s'exécuter.

(Question 16 et 17)
En exécutant nos deux méthodes, nous avons pu remarquer que la méthode résoudre de Dijkstra s'exécute plus vite que celle de BellmanFord,

Données obtenues sur 5 lancements :
Dijkstra : 0.198344; 0.19305; 0.108097; 0.172481; 0.112726
BellmanFord : 0.226912; 0.218389; 0.12722; 0.190394; 0.131036

Nous estimons avec les résultats expérimentaux que l'algorithme de Dijkstra est environ 20% plus rapide que la méthode du point fixe de BellmanFord.

(Question 18)
En effet, théoriquement, nous savons que la complexité de l'algorithme de Dijkstra est plus faible que la complexité de l'algorithme de BellmanFord.

ici (graphe exemple sae) :

Dijkstra : Complexité = θ([nbarretes+nbommets]*ln(nbsommet)
= θ([7+5]*ln(5)
= θ(12*1.6)
= θ(19.2)

BellmanFord : Complexité = θ([nbarretes+nbommets]*ln(nbsommet)
Complexité = θ(7*5)
Complexité = θ(35)

Le résultat théorique serait donc une complexité presque 2 fois plus faible pour l'algorithme de Dijkstra.

Nous pouvons en tirer que bien que nous ayons correctement implémenté l'algorithme de Dijkstra, cette implémentation n'est pas optimale.

En conclusion, ce projet a été pour nous un challenge, liant les méthodes vues en cours de maths à notre notation algorithmique, et la retranscription en code java. Ces éléments nous ont été enseignés séparément, et les mettre en commun n'était pas tâche aisée.
Nous avons en effet eu du mal à reproduire les méthodes mathématiques au format algorithme, nous retardant globalement sur nos échéances. De plus, il nous apparaît que notre programme Dijkstra n'est pas aussi efficace que prévu, probablement dû à une implémentation incorrecte du code.
Malgré ceci, nous avons dû avancer dans le projet et rattraper notre retard, nous poussant à être le plus productif possible.
Nous avons appris à travailler ensemble en collaboration sur un même projet Git, chose que nous avons rarement eu l'occasion de faire, une très bonne expérience de ce point de vue et nous aidant à nous familiariser avec toutes les fonctionnalités du logiciel et son implémentation dans intelliJ.