RAPPORT SAE APPROCHE ALGORYTHMIQUE

I/Représentation d'un graphe

Les classe que nous avons écrite sont les classes Arc, GrapheListe, Nœud. Nous avons aussi ecrit une classe de test nommé TestGraphe.

II/Point fixe

Nous avons pour cette parti ecrit les classe BellmanFord et MainBellmanFord.. la première classe est testé dans la classe TestBellmanFord.

Question 13 :

L'algorithme est :

```
fonction Bellman-Ford(G , d)
    //init.
    pour n dans G faire
           L[n] := +∞
           parent[n] := null
            L[d] = 0
    fpour
    //boucle
    continuer := vrai
    tant que continuer faire :
      continuer := faux
      pour chaque arc (x,y,poids) de G faire:
           si L[x] > L[y] + poids alors
              L[x] := L[y] + poids
              parent[y]:= x
              continuer := vrai
           fsi
      fpour
    ftant
    retourner L
```

III/Dijkstra

Les classes programmer dans cette parti sont Dijkstra et MainDijkstra. La première est elle aussi tester dans une classe de test (TestDijkstra)

IV/Validation

Dans cette partie, nous avons modifier a plusieurs reprise la classe Main afin de récupérer les donnée nécessaire a l'expérimentation

Question 21 :

L'un recalcule à chaque itération, pour chaque sommet (Bellman) alors que l'autre calcule juste pour les sommet dont on a une supposition (Dijkstra)

Question 22 :

D'apres les observation, l'algorithme de Dijkstra serait plus rapide que celui de Bellman Ford

Question 23 :

L'algorithme de Dijkstra semble plus efficace pour tous les graphe d'apres notre MainDijkstra, le temps de résolution est tout le temps plus court pour l'algorithme de Dijkstra

Question 26 :

D'apres les resultat, l'algorithme de Dijkstra est plus efficace

Question 27 :

Ce ratio moyen est environ de 1.1, ce ratio est constant d'apres nos observations

Question 28 :

Pour conclure, nous pouvons dire que les deux algorithmes sont aussi efficaces l'un que l'autre. Cependant, cela n'est pas logique, il doit y avoir une erreur dans notre code, plus le nombre de nœuds augmente, plus celui de Dijkstra est logiquement efficace par rapport a celui de Bellman-Ford.

V/ Labyrinthe

Pour cette partie nous avons mis en place plusieurs classes ainsi que plusieurs méthodes afin de créer un graphe à partir d'un objet "Labyrinthe". Pour sa construction nous avons 2 possibilités:

- Tout d'abord nous avons insérer dans la classe Labyrinthe une méthode genererGraphe() qui retourne directement le Graphe. Le calcul du chemin le plus court est présent dans la classe MainLaby
- De plus nous avons également une autre méthode pour la création du graphe. En effet nous avons créer une classe "GrapheLabyrinthe" qui implémente directement l'interface "Graphe" et qui contient donc le labyrinthe en attribut. Nous avons donc juste redéfini les méthode listeNoeuds() ainsi que suivant(). Le calcul du chemin le plus court est présent dans la classe MainGrapheLaby.

Pour la mise en place d'un labyrinthe de glace nous avons créé une classe "LabyrintheGlace" qui dérive de Labyrinthe dans laquelle nous avons redéfini la méthode "deplacerPerso()" afin de se déplacer sur la ligne entière.
De plus pour la création de portes et de clés nous avons ajouté dans Labyrinthe un attribut porte et clé qui sont des "Position" (classe contenant un x et un y). Ainsi le calcul du chemin le plus court se fait en 2 parties, une première du personnage a la clé, puis de la clé a la porte.
Les mains de ses questions bonus sont présents dans la classe "MainLabyBonus"

Cette partie a donc pu nous montrer que le mécanisme de graphe peut s'adapter et être utilisé dans diverses situations.

VI/Conclusion

Pour conclure, la principale chose que nous avons appris était qu'il y avait différente manière(algorithmes) de résoudre un graphe et cela nous a permit de comprendre par exemple comment GoogleMmap fonctionnait et sa logique de programmation.

Les principales difficultés rencontrer ont été le temps puisque a mon avis le temps pour réaliser cette SAE était trop court par rapport au sujet. Mais aussi un peu l'organisation : nous nous retrouvions parfois sur la même classe, sur la même méthode a se gêner plus qu'as s'aider.

Le bilan a tirer de ce travail est qu'il faut s'organiser avant de se lancer dans la programmation afin d'anticiper les potentiel difficulté et ainsi de gagner du temps, ce qui permet de moin stresser a la fin et de travailler plus correctement.