Rapport du projet de Recherche Opérationnelle fait par Eliel WOTOBE

Ce projet consiste à implémenter les différents algorithmes vu dans le cours de Recherche Opérationnelle

Pour finaliser ce projet, nous avons eu à utiliser les algorithmes suivants :
l'algorithme d'Edmonds-Karp, l'algorithme de Bellman-Ford et d'autres algorithmes qui sont plus des utilitaires.

Dans les lignes qui suivent, nous détaillerons les algorithmes suscités et la structure de notre graphe :

I - Structure de notre graphe :

Pour pouvoir représenter notre graphe, nous avons utilisé une classe fournie en première année par nos professeurs du cours Algorithme et Structures de données.

Quant aux algorithmes utilisés pour implémenter les tâches exigées par le projet, elles sont le fruit du travail de l'étudiant seul.

Cette graphe se présente comme suit : elle contient deux classes imbriquées : Vertex et Edge suivi de deux dictionnaires comme propriétés.
Notre graphe est représenté par une liste d'adjacences qui est plus adaptée pour les graphes de grande taille.

La structure de donnée que nous avons utilisé à cet effet est un dictionnaire qui a pour clé une instance de Vertex dont la valeur est un autre dictionnaire dont la clé est le voisin du nœud précédent et la valeur une instance de Edge contenant les informations entre les deux sommets.

II- Algorithmes

1. **Algorithme d'Edmonds-Karp**

   Cet algorithme est une spécialisation de l'algorithme de Ford-Fulkerson.
   Nous nous sommes fortement inspirés de l'algorithme proposé sur cette page web :
   https://fr.wikipedia.org/wiki/Algorithme_d%27Edmonds-Karp
   notre algorithme prend en entrée trois paramètres : le graphe initial, la source et le puits et retourne le flot maximum, le graphe modifié et la liste de flots parcourant chaque arc.

   Cet algorithme fonctionne tel quel :
   Tant qu'il existe un chemin augmentant dans le graphe résiduel,
   on va appliquer la capacité résiduelle sur ce chemin et le cumuler
   A la fin on retourne le flot maximal, le graphe modifié et la liste de flots.

2. **Algorithme de Bellman-Ford :**

   Cet algorithme prend en entrée le graphe, la source et le puits.
   Il retourne le tableau de parents des nœuds et la distance finale entre la source et le puits.
   Cet algorithme parcourt chaque nœud pour modifier la distance entre la source et chaque noeud en respectant la propriété suivante :
   pour chaque arc **(u,v) si d(v) > d(u)+c(u,v) alors d(v) = d(u) + c(u,v)**

   Cet algorithme est utilisé dans la fonction minCostMaxFlow pour pouvoir récupérer le coût minimum avec le flot maximal.

3. **Autres algorithmes :**

   execute():
   Cette fonction est celle qui exécute les algorithmes en spécifiant en entrée standard les arguments adéquats
   python Maxflow.py -m <chemin du fichier> pour la coupe minimale
   python Maxflow.py -f <chemin du fichier> pour le flot maximal
   python Maxflow.py -M <chemin du fichier> pour le coût minimum avec le flot maximal

   parseInputFile(file_input) :
   cette fonction parse le chemin du fichier représentant le graphe fourni à l'entrée standard

   detect_double_sens(E,complement_informations):
   Cette fonction détecte s'il y a deux nœuds qui ont deux arêtes entre eux, prend une de ces arêtes qu'elle subdivise en deux autres arêtes avec la création d'un nouveau nœud.
   param E: array of tuple of edges
   param complement_informations:
   return: E,complement_informations

   print_flow_list(flow_list):
   Cette fonction affiche la liste de flots sous un format donnée :
   (origine,destination)flot/capacité : liste de flots

   print_arcs_min_cut(edges, part1, part2):
   Cette fonction affiche les arcs qui forment la coupe minimale du graphe. Il prend pour entrée les arêtes du graphe, les nœuds visités(part1) et les nœuds non visités(part2) dans le graphe final.

Somme toute, implémenter ces algorithmes était un peu ardu, mais cela a permis de mieux comprendre les notions abordées
de gagner encore en compétence et de pouvoir à l'avenir expliquer les notions de flot maximum, coupe minimale.