**1 Premier algorithme d’approximation :**

Fichier « question\_one.py »

* Classes :
  + Ville : représente une ville avec ses coordonnées et son identifiant
  + Node : représente un nœud avec son nom et ses fils(voisins)
  + Arbre : représente un arbre binaire
* Fonctions :
  + distance :
    - Input : ville\_a, ville\_b
    - Output : la distance entre les deux villes
    - Complexité : O(1)
  + read\_cities :
    - Input : fichier.txt
    - Output : dictionnaire avec les informations de chaque ville et le nombre total de villes
    - Complexité : O(taille(n))
  + get\_distance :
    - Input : ville\_a, ville\_b
    - Output : la distance entre les deux villes
    - Complexité meilleur cas : O(1)
    - Complexité pire cas : O(taille(n))
  + distances\_between\_cities :
    - Input : fichier.txt
    - Output : tableau contenant les villes avec les distances calculées entre elles, et le nb\_villes
    - Complexité meilleur cas : O(n\*n)
  + kruskal :
    - Input : villes, nb\_villes
    - Output : l’ordre de livraison à effectuer
    - Complexité meilleur cas : O(n\*n)
    - Décrit l’algorithme :
      * Je créer un tab\_pointeur dont chaque indéx représente une ville, par exemple : tab\_pointeur[1] décrit vers quelle ville pointe la ville avec l’id = 1.
      * Pour chaque élément dans villes (je prendre le max et le min entre les deux villes de chaque élément et je vérifie si j’ai déjà passé par les deux villes et le traitement à été effectué correctement :

Si la ville la plus grande pointe vers la ville la plus petit en vérifiant que la ville la plus petite parmi les deux, pointe vers une autre ville, ou bien les deux villes pointes vers la même ville.

* + - * Sinon si une des deux villes n’est pas traité (la ville pointe vers None) :
        + S’il s’agit de la plus grande ville, je vérifie si la petite ville pointe ver une autre ville, si c’est le cas, la plus grande ville pointe vers la même ville que la petite, sinon les deux villes pointent vers la petite ville.
        + S’il s’agit de la plus petite ville, je vérifie si la plus petite ville pointe vers None, si c’est le cas, elle pointera vers la même ville que la plus grande ville, sinon, je prends le min et le max des villes vers lesquelles elles pointent, ensuite toutes les villes qui pointes vers le plus grand pointeur, elles pointeront vers le pointeur le plus petit.
      * Les deux villes s’ajoutent en tant que villes voisines.
      * Je crée un arbre binaire et puis je le parcours en profondeur afin d’obtenir la distance minimale.
  + creer\_arbre :
    - Input : la liste des voisins
    - Output : arbre avec des nœuds
    - Complexité meilleur cas : O(n log n)
  + creer\_arbre :
    - Input : la liste des voisins
    - Output : arbre avec des nœuds
    - Complexité meilleur cas : O(n log n)
  + Problem\_one :
    - Input : fichier.txt
    - Output : ordre de livraison à effectuer
    - Complexité meilleur cas : O(n \* n)

**2 Approche exacte :**

Fichier « question\_two.py »

* Fonctions :
  + Check\_if\_calculated :
    - Input : ville de départ, ville d’arrivée, liste des villes intermédiaires
    - Output : si le chemin a déjà été calculé
    - Complexité : O(n log n)
  + min\_livraison\_non\_ville:
    - Input : vile de départ, ville d’arrivée
    - Output : la distance entre la ville de départ et la ville d’arrivée
    - Complexité : O(taille(n))
  + min\_livraison\_une\_ville:
    - Input : ville de départ, ville d’arrivée, une ville intermédiaire
    - Output : la distance entre la ville de départ et la ville d’arrivée en passant par la ville intermédiaire
    - Complexité : O(taille(n))
  + min\_livraison\_plusieurs\_villes:
    - Input : ville de départ, ville d’arrivée, des villes intermédiaires
    - Output : le chemin le plus court depuis la ville de départ à la ville d’arrivée en passant par toutes les villes intermédiaires
    - Complexité : factorielle
    - Décrit l’algorithme :
      * Si on a juste une seule ville intermédiaire, on vérifie si al distance en passant par cette ville a déjà été calculée, si c’est le cas, on chercher la valeur de cette distance calculée et on retourne le résultat, sinon on calcule la distance en passant par cette ville, et on l’ajoute au tableau des villes calculées et on retourne la distance
      * Si on passe par plusieurs villes, on applique la même procédure, si la distance n’a pas été calculée, on calcule la distance ente la ville d’arrivée et la ville qui la précède, on ajoute cette distance au tableau des distances calculées et puis on appelle la mémé fonction récursivement(min\_livraison\_plusieur\_villes) avec la même ville de départ et la ville qui précède la ville d’arrivée comme une ville d’arrivée, on exclue la dernière ville parmi les villes intermédiaires(puisqu’on vient de calculé sa distance par rapport à la ville d’arrivée) + la distance qu’on vient juste de calculée.
  + problem\_two :
    - Input : ville de départ, ville d’arrivée, des villes intermédiaires
    - Output : le chemin le plus court depuis la ville de départ à la ville d’arrivée en passant par toutes les villes intermédiaires, en prenant en compte toutes les combinaisons possibles en fonction de l’ordre de passages par les villes.
    - Complexité : factorielle