Partie 0 lire_donnees.c :

1. fonction de lecture de fichier

La fonction lire_tsplib() lit et analyse un fichier au format TSPLIB (.tsp) pour créer une instance du problème du voyageur de commerce (TSP), en extrayant les métadonnées (nom, commentaire, dimension, type de distance) et les coordonnées des nœuds, résolvant ainsi le problème de chargement et de validation des données d'entrée pour les algorithmes TSP.

Fonctionnement : elle vérifie l'extension du fichier pour s'assurer qu'il s'agit d'un .tsp, ouvre le fichier en lecture, et parse ligne par ligne les sections d'en-tête (NAME, COMMENT, DIMENSION, EDGE_WEIGHT_TYPE) en utilisant sscanf pour extraire les valeurs. Une fois la section NODE_COORD_SECTION atteinte, lit les coordonnées des nœuds (numéro, x, y) jusqu'à EOF ou la dimension atteinte, tout en gérant les erreurs de format ou de nombre de nœuds. Alloue dynamiquement la mémoire pour l'instance et les nœuds, et valide le type de distance supporté (EUC_2D, GEO, ATT) pour éviter les problèmes de compatibilité. Résout les erreurs potentielles comme les fichiers corrompus ou les allocations mémoire insuffisantes en affichant des messages d'erreur et en libérant les ressources partiellement allouées.

2. fonctions de distance

La fonction *distance_euclienne()* calcule la différence des coordonnées x et y, applique le théorème de Pythagore (sqrt(xd² + yd²)), puis arrondit à l'entier le plus proche pour résoudre le problème de distance en évitant les flottants imprécis dans les calculs TSP.

La fonction distance_geographique() convertit les coordonnées en radians, utilise la formule de la distance orthodromique (grand cercle) avec le rayon terrestre (6378.388 km), et applique acos pour calculer l'angle, résolvant ainsi les distances réelles sur une surface sphérique pour des problèmes géographiques.

La fonction distance_euclidienne_att() calcule la distance euclidienne, la divise par 10, puis arrondit à l'entier supérieur si nécessaire, résolvant le problème d'asymétrie dans ATT en forçant des coûts non-euclidiens pour des tournées plus réalistes.

NB :
les distances sont représentées en entiers (int) conformément à la spécification TSPLIB, qui définit les longueurs d'arêtes comme des valeurs entières sur 32 bits. Les coordonnées des villes restent en flottants, mais les distances sont arrondies.

3. Fonction calcul d'une tournée

La fonction longueur_tournee() calcule la longueur totale d'une tournée donnée en sommant les distances entre villes consécutives, fermant la boucle au départ. Elle prend en paramètre une Instance TSP avec les dimensions et nœuds , une tournée avec le parcours des villes et un pointeur vers la fonction de distance à utiliser. Elle parcourt le parcours de la tournée,

calcule la distance entre chaque paire consécutive (et la dernière vers la première), et accumule pour résoudre l'évaluation de la qualité d'une solution TSP.

4. Fonctions de demi-matrice

La fonction *creer_matrice_demi_distances()* crée une matrice triangulaire inférieure contenant toutes les distances entre les villes d'une instance du TSP, en utilisant une fonction de distance passée en paramètre. Cela permet d'éviter de recalculer les distances à chaque fois et de gérer différents types de métriques (EUC_2D, GEO, ATT) de manière générique.

La fonction recuperer_distance() vérifie si i == j (retourne 0), sinon accède à la matrice en utilisant l'ordre i > j pour la triangularité, résolvant l'accès efficace sans duplication de données.

La fonction *longueur_tour_cano_matrice()* calcule la longueur de la tournée canonique (0 → 1 → ... → n-1 → 0) en utilisant la matrice, fournissant une référence de base. Elle parcourt les indices consécutifs et ferme la boucle, utilisant recuperer_distance pour sommer, résolvant l'évaluation rapide d'une tournée de référence.

Partie 1 force_brute.c

1. Next_permutation :

La fonction *next_permutation()* génère la permutation suivante d'un tableau d'entiers dans l'ordre lexicographique. Elle est utilisée pour itérer sur toutes les permutations possibles sans duplication, en modifiant le tableau en place. Elle prend en paramètre un pointeur vers le tableau d'entiers à permuter (de taille n) et la taille du tableau (nombre d'éléments).

Fonctionnement : Recherche la plus grande position i où tab[i] < tab[i+1], puis trouve le plus petit élément tab[j] à droite de i qui est supérieur à tab[i], les échange, et inverse la sous-séquence de i+1 à n-1. Cela produit la permutation suivante dans l'ordre lexicographique croissant.

Elle est essentielle pour l'algorithme de force brute, où elle permet de tester toutes les tournées possibles en générant systématiquement les permutations des indices des villes.

2. force_brute (Version Matrice) :

La fonction *force_brute ()* implémente l'algorithme de force brute pour résoudre le problème du voyageur de commerce (TSP) en testant toutes les permutations possibles des villes, en utilisant une matrice de distances précalculée pour optimiser les calculs. Cette version prend en paramètre un pointeur vers l'instance TSP contenant les données des villes et une matrice carrée des distances entre toutes les paires de villes.

Elle génère toutes les permutations des indices des villes à partir de l'ordre canonique, calcule la longueur de chaque tournée en consultant la matrice, et retient la meilleure. Utilise next_permutation pour itérer sur les permutations.

Dans le cas où on a arrêté le programme avec contrôle C et qu'on a choisi d'arrêter le programme en choisissant q, la fonction ne retourne pas de tournée.

3. Fonction force_brute (Version Pointeur de Fonction) :

La fonction *force_brute()* implémente l'algorithme de force brute pour résoudre le TSP en testant toutes les permutations possibles des villes, en utilisant un pointeur de fonction pour calculer les distances à la volée. Cette version prend en paramètre un pointeur vers l'instance TSP contenant les données des villes. Et pointeur de fonction vers la fonction de calcul de distance (e.g., distance_euclidienne, distance_geographique).

Elle génère toutes les permutations des indices des villes à partir de l'ordre canonique, calcule la longueur de chaque tournée en appelant directement la fonction de distance sur les nœuds, et retient la meilleure. Utilise next_permutation pour itérer sur les permutations.

Dans le cas où on a arrêté le programme avec contrôle C et qu'on a choisi d'arrêter le programme en choisissant q, la fonction ne retourne pas de tournée.

Main0

Ce main constitue la partie 0 du projet (lecture et test de la tournée canonique).
Il commence par vérifier les arguments passés en ligne de commande : il attend -f nom_fichier -c.
Ensuite, il lit le fichier TSPLIB indiqué (via lire_tsplib) et construit une structure instance_t contenant les villes, leurs coordonnées et le type de distance (EUC_2D, GEO, ATT).
À partir de cette instance, il choisit dynamiquement la fonction de distance adéquate avec choix_distance().
Puis, il crée la demi-matrice des distances entre les villes grâce à creer_matrice().
Deux longueurs de la tournée canonique sont ensuite calculées :

Avec la demi-matrice (longueur_tour_cano_matrice()),
Avec la fonction de distance directe (longueur_tournee()).

Le programme affiche alors un résumé sous forme de tableau :
Instance ; Méthode ; Temps CPU ; Longueur ; Tour,
suivi de la liste des villes dans l'ordre canonique [1, 2, 3, …, n], puis des longueurs obtenues par les deux méthodes.
Enfin, il libère la mémoire allouée pour la matrice et l'instance avant de se terminer proprement.

Main1:

Ce main lit une instance TSPLIB (fichier .tsp) contenant les coordonnées des villes.
Il sélectionne automatiquement la fonction de distance appropriée (EUCL_2D, GEO ou ATT).Ensuite, il construit la demi-matrice des distances entre les villes.
Il affiche le nom de l'instance, la méthode utilisée, et la longueur de la tournée canonique calculée.
Puis, il exécute deux algorithmes de force brute :

force_brute() → version utilisant la matrice de distances,
force_brute2() → version utilisant la fonction de distance directement.

Ces deux fonctions testent toutes les permutations possibles de villes et renvoient la tournée optimale (plus courte).

Dans ce fichier on a également calculer le temps que les deux versions prennent pour s'exécuter (calculer la meilleure tournée) mais comme il n'est pas demandé dans les fichiers de test python il faudra les décommenter et commenter durant les phases de test.

Enfin, le programme libère toute la mémoire allouée (matrice et instance) avant de se terminer proprement.