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.

Partie 2

1. plus_proche_voisin :

La fonction plus_proche_voisin*()* implémente l'algorithme du plus proche voisin pour résoudre le problème du voyageur de commerce (TSP) en choisissant le prochain noeud de cette manière:

- Pour chaque nœud qui n'a pas encore été parcouru, on calcul la distance.
- On choisit le nœud non parcouru dont la distance avec le nœud actuel est la plus courte.
- on continue ainsi jusqu'à ce qu'on ait parcouru tous les nœuds.

La fonction récupère un pointeur sur une instance et la demi matrice associée.

Elle renvoie un pointeur sur une tournée.

2. marche_aleatoire_matrice:

La fonction marche_aleatoire_matrice() génère une tournée aléatoire pour le TSP en mélangeant complètement la liste des villes à l'aide de l'algorithme de Fisher-Yates.

Elle crée d'abord un tableau canonique des indices des nœuds, le mélange pour obtenir une permutation uniforme, puis construit la tournée correspondante en utilisant les coordonnées de l'instance.

La longueur de la tournée est ensuite calculée via la demi-matrice des distances.

La fonction reçoit une instance et sa matrice de distances, et renvoie un pointeur sur une tournée valide entièrement aléatoire.

Partie 3: genetique_aux.c (fonctions auxiliaires de l'algorithme génétique)

Ce fichier contient les blocs principaux nécessaires à un algorithme génétique standard.

1. tournament_selection()

Sélectionne des individus via un tournoi :

- on tire au hasard tournament size individus,
- on prend le meilleur (plus courte distance),
- On répète jusqu'à remplir la population sélectionnée.

But : favoriser les bons individus tout en gardant de la diversité.

2. ordered_crossover()

Implémente le croisement ordonné classique pour les permutations :

1. choisir au hasard un segment [start, end],
2. le copier dans l'enfant,
3. compléter les cases vides avec les villes du parent B dans leur ordre d'apparition,
4. garantit une permutation valide (pas de doublons, pas d'omissions).

3. random_population()

Génère une population initiale en répétant **marche_aleatoire_matrice()**.

4. swap_mutation()

Mutation simple :

- pour chaque position, avec probabilité mutation_rate,
- échanger l'élément avec un autre index aléatoire.

But : éviter la convergence prématurée.

5. compare_tournee()

Fonction de comparaison utilisée par qsort() pour trier les individus par performance croissante.

6. ind_max_tournee()

Renvoie l'indice contenant la tournée la plus longue entre deux individus.

Main

Le main regroupe toutes les méthodes de résolution de TSP jusqu'à la méthode d'algorithme génétique.
Le main respecte les balises données dans tsp.pdf pour l'appel de chaque méthode.

La comparaison de la rapidité d'exécution des deux fonctions force_brute se fait maintenant dans le fichier test_1.c .

Makefile

Pour compiler tous les fichiers du projet, il faut utiliser la commande make dans le dossier C.
Pour supprimer les fichiers .o il faut utiliser la commande make clean.
Pour pouvoir débugger avec gdb il faut utiliser la commande make DEBUG=yes .
Pour afficher tous les messages de compilation il faut utiliser la commande
make VERBOSE=1 .