Le fonctionnement et la Structure de notre base de données est la suivante :

Table : custommer, cuisinier et utilisateurs

Puisqu'un cuisinier peut aussi être client nous avons choisi de les regrouper dans une table utilisateurs et d'ajouter des Héritages.

Nous savons **qu'il y à une relation 1,1-1,1 entre les tables utilisateurs et custommer** et idem pour cuisinier, **mais pour des raisons de clarité** du MCD nous avons choisi de **ne pas mettre de relations réflexive** de la table utilisateur vers elle-même comme vu en cours.

Les tables custommer et cuisinier héritent donc de tous les attributs de la table utilisateur, elles ont uniquement un attribut, **id** chacune pour identifier chaque client et cuisinier.

Table avis et Commande:

La table avis à un liens avec la table commande puisqu'un avis porte sur une commande. Elle a tous les attributs nécessaires à un avis. Le client et le cuisinier font un retour et le client donne un nombre d'étoile aux plats reçu.

La table commande est centrale dans notre base de donnée. Le client passe une commande préparer par le cuisinier. Cette commande comporte des lignes.

table ligne de commande et livraison :

Chaque ligne de commande à 1 seule livraison et chaque livraisons à au moins 1 ligne de commande.

Les livraisons ont un id, une date et heure de livraison enregistré au moment de la livraisons

Tables plat, pays et ingrédients :

Chaque plat contient n ingrédient et chaque ingrédient est contenu dans n plats.

De plus dans l'association contient nous avons ajouter une colonne quantité, comme cela chaque plat à une quantité précise pour chaque ingrédients de sa recette.

La table Pays sert à éviter de dupliquer des données récurrentes, puisque les pays sont connus à l'avance.

Les plats ont une nationalité/origine et les ingrédients une provenance d'importation/fabrication , d'où leurs liens avec la table pays.

L'attribut prix_Kg sera utile plus tard pour calculer le cout en matière première de chaque plats et ainsi fixé le prix de vente du plat.

Partie c Sharp :

Nous avons divisé notre code en 4 classes différentes : la classe Lien, la classe noeud la classe Graphe et la classe affichagegraph.

- La classe nœud : elle représente un sommet et ses différentes connexions.
- La classe Lien : la classe Lien modélise une arête reliant deux sommets différents
- La classe Graphe : Cette classe permet de gérer l'ensemble du graphe avec ses différentes structures de données et algorithmes la composant.
- La classe affichagegraph : Cette classe va permettre la modélisation et la conception visuelle du graphe créer.

Graph

Le graphe va être représenté de deux façons différentes :

D'abord sous forme de listesAdjacentes : Une structure sous forme de tableau de listes, où chaque sommet est relié à une liste de ses voisins directs. Cette représentation va être celle que nous utiliserons principalement notamment dans l'exploration du graphique car elle est très efficace dans l'exploration des voisins de chaque sommets ainsi que l'ajout de nouvelles connexions entre eux (les liens).

On représente également le graphique sous forme de MatriceAdjacentes : Une matrice carrée où une valeur 1 indique une connexion entre deux sommets et 0 une absence de lien. Ainsi cette représentation permet un accès rapide et simplifié aux relations entre différents sommets.

Nous avons également déclaré une variable Nombredesommets du graphe qui permettra de construire le graphe en fonction du nombre de sommets demandé

Pour ce qui est de l'instanciation du graphique , nous l'avons fait ici à partir d'un fichier texte, où chaque ligne contient une relation entre deux sommets.

Ainsi nous avons créé la méthode ChargerDepuisFichier(string chemin) qui lit ces données et construit le graphe sous les deux formes mentionnées.

Parcours du Graphe :

Nous devions implémenter deux algorithmes différents afin d'explorer le graphe :

D'abord le Parcours en largeur : ce parcours va partir d'un sommet (le int départ représente le premier sommet exploré) et va explorer le graphe par niveaux c'est-à-dire en visitant tous les voisins avant de passer au niveau suivant.

- Il est implémenté via une file (Queue) qui assure une exploration ordonnée du graphique.

Il y’a ensuite le parcours en profondeur :

Le parcours en profondeur va explorer le graphe aussi loin que possible dans une direction avant de revenir en arrière , pour cela on utilise une méthode récursive : Pour cela nous utilisons la listeAdjacente représentant le graphique et on commence par le sommet de départ ; une fois visité on le marque comme visité et on continue à explorer chaque sommets jusqu’à ce qu’ils soient tous visités et si il y’a un voisin d’un sommet qui n’est pas visité on appelle récursivement la fonction parcoursenprofondeur afin qu’il soit visité aussi.

Ainsi nous avons passé en paramètre de cette fonction un int départ qui sera le sommet de départ et le premier à être visité et un bool[]visite qui affichera true afin de marquer un sommet comme visité et ainsi passer au suivant.

On utilisera cette méthode notamment pour déterminer la connexité de notre graphique

Connexité

Nous avions comme condition la connexité du graphique : C’est-à-dire l’accès à n’importe quel sommet à partir de n’importe quel point de départ :

Pour cela nous avons implémenté 2 méthodes :

La méthode public bool EstConnexe() :

Cette méthode va utiliser le parcours en profondeur afin de visiter tout les sommets du graphique , si l’ensemble des sommets est visité la méthode retourne true et le graphique est bien connexe, cependant si tout les sommets ne peuvent pas être visités la méthode revient false.

Ainsi si le graphique généré initialement n’est pas connexe on le rend connexe par la méthode public void RendreConnexe() afin d’ajouter des liens et relier les composantes déconnectées, garantissant ainsi un graphe connexe après traitement.

Pour cela on utilise en privé La méthode ParcoursprofondeurCollect utilise une version itérative du parcours en profondeur pour explorer une composante connexe d'un graphe.

Cette méthode est utilisée dans RendreConnexe() pour repérer toutes les composantes **connexes** du graphe.

Détection de cycles

Afin de vérifier si le graphe contient des circuit (cycles)

Un cycle est une séquence de sommets connectés qui ramène au sommet de départ sans repasser par une arête déjà visitée. Ainsi ici on utilise la méthode bool Contientcycle() et ContientCycleParcourslongueur comme algorithmes de détections de cycles.

Pour cela : On effectue un Parcours en profondeur, en enregistrant le parent du sommet courant.

Si on rencontre un voisin déjà visité qui n’est pas le sommet parent, cela signifie qu'on est revenu en arrière par un autre chemin, ce qui indique la présence d'un cycle.

Si le voisin est simplement le parent, cela signifie qu’on fait juste un aller-retour normal, donc ce n’est pas un cycle.

Si un sommet déjà visité est différent du parent immédiat, alors un cycle est détecté.

On arrête la recherche dès qu’un cycle est trouvé et on l’affiche.

Charger depuis le fichier

La méthode ChargerDepuisFichier va ainsi prendre en compte la connexité et la présence de cycles afin de créer à partir du fichier donné un graph.

Visualisation Graphe :

Enfin nous avons ajouté une classe visualisation graphique du graphe avec la bibliothèque visuel Système.Drawing. Cette visualisation permet de voir :

Les sommets, représentés sous forme de cercles.

Les liens (arêtes), tracés entre les sommets connectés.

Les propriétés du graphe , comme la connexité et la présence de cycles, affichées sous forme d’annotations.

Main :

Le fichier qui nous était fourni était soc-karate.mtx

Dans un premier temps nous récupérons le chemin de ce fichier afin de créer des chemins relatifs à la construction et la visualisation du graph :

Ici, AppDomain.CurrentDomain.BaseDirectory permet de récupérer le chemin du projet, et on construit un chemin relatif pour trouver le fichier soc-karate.mtx.

String CheminFichier : le chemin permettant l’accès aux donnés du fichier karaté

String CheminImage : le chemin où l’image finale sera générée

On créer un graphe g ayant pour taille (34) signifiant que le graphe contiendra 34 sommets

On réalise les deux parcours demandés (malheureusement nous n’avons pas réussi à faire fonctionner le parcours en largeur) a

On utilise la méthode ChargerDepuisFichier avec comme chemin : CheminFichier afin que le graphe g contienne les informations relatives au fichier karaté mais qu’il soit également connexe et contienne des circuits.

Enfin on utilise les outils de visualisation afin de créer l'image de ce graph.

Visualisation du Graphe

Pour la visualisation du graph j'ai d'abord demandé à chat gpt de représenter : chaque sommet par des cercles, représenter les arêtes par des lignes et de positionner chaque sommet de manières lisibles et pas les uns sur les autres et ce à partir d'un chemin pour l'image que j'ai nommé cheminImage.

Pour cela nous avons également demandé de créer ce graphe en fonction de la classe graphe que nous avions fait afin que l'image soit en accord avec la classe graphe

Ainsi nous avons donc implémenté une méthode en public : la méthode DessinerGraphe qui va donc créer l'image du graphe avec des dimensions fixés pour la largeur et la hauteur de l'image , Bitmap la fonction qui va créer l'image à partir de ses dimensions.

Ensuite nous avons demandé à chat gpt de coloriser ce graphe et qu'il soit claire et esthétique

Nous avons ensuite demandé une méthode afin que l'image s'ouvre automatiquement , il a ainsi proposé de passer la méthode DessinerGraphe en privé ainsi qu'avec une méthode OuvrirImage qui permet l'ouverture automatique de l'image une fois celle-ci générée, pour harmoniser tout cela la méthode qu'on utilise dans le main déclarée en public est DessineretAfficherGraphe qui exécute DessinerGraphe et OuvrirImage.

Nous avons ensuite demandé de marquer sur l'image la connexité du graphique ainsi que la présence de cycles.

Pour cela chat gpt à ajouter à la méthode DessinerGraphe un texte marquant l'état de la connexité du graphe en rouge

Et un texte marquant la présence de cycle en vert (pour les couleurs la fonction brush est utilisée)