**Kuresa Ramen – Virgile Regnier – TD K**

**Rapport – Étape 2 : Généralisation et Intégration des Modules**

**1. Introduction**

L’étape 2 du projet « Liv’In Paris » vise à généraliser la solution initiale et à intégrer les deux parties développées précédemment :

* **Partie Graphes** : Implémentation d’un modèle générique pour la représentation du réseau, avec application au plan du métro de Paris.
* **Partie Base de Données et Interface** : Intégration des modules de gestion (clients, cuisiniers, commandes) et des statistiques dans une application console en C#, avec accès à la base de données.

**2. Généralisation du Modèle de Graphe**

**2.1. Objectif**

L’objectif était de rendre les classes de graphes génériques afin de pouvoir les utiliser avec n’importe quel type de nœud. Cela a permis d’adapter la solution au plan du métro de Paris, où chaque station est identifiée par des coordonnées (longitude, latitude).

**2.2. Modifications apportées**

* **Classes Génériques** : Les classes Noeud<T> et Graphe<T> ont été créées pour permettre une instanciation flexible des données.
* **Adaptation au Métro** : Les informations des stations (longitudes, latitudes, nom de station) étaient fournies, nous avons dû rajouter des valeurs dans les colonnes « précèdent, suivant, temps d’attente et temps de changement » .
* **Algorithmes de Recherche** : Trois algorithmes ont été implémentés pour calculer le chemin le plus court entre deux stations :
  + **Dijkstra** (pour les graphes sans poids négatif)
  + **Bellman-Ford** (permet de gérer d’éventuels poids négatifs, ici il n’y en a pas)
  + **Floyd-Warshall** (calcul de tous les plus courts chemins en une passe)

**2.3. Comparaison des Algorithmes**

Chaque algorithme a été évalué selon :

* **Complexité Algorithmique**
  + Dijkstra : O((V + E) log V)
  + Bellman-Ford : O(V × E)
  + Floyd-Warshall : O(V³)
* **Rapidité d’Exécution** : Les tests sur un échantillon de données (extraction des données du plan de métro) ont montré que Dijkstra est le plus rapide pour des requêtes ponctuelles, tandis que Floyd-Warshall offre une solution complète mais au coût d’un temps de calcul plus élevé. Aussi, comme il n’y avait aucun poids négatif, nous avons priorisé l’algorithme de Dijkstra.

**3. Intégration des Modules dans l’Application Console**

**3.1. Module Commande**

Ce module permet :

* **Création/Modification de Commandes** : Une commande est créée uniquement si le client existe dans la base. Sinon, le client est ajouté.
* **Calcul du Prix** : moyenne du prix des commandes disponible via l’interface.
* **Détermination du Chemin de Livraison** : Le chemin le plus court entre les deux points est établi à l’aide des algorithmes de recherche (notamment Dijkstra).

**3.2. Module Statistiques**

Ce module fournit des bilans généraux pour améliorer la gestion et le suivi :

* **Nombre de Livraisons par Cuisinier** : Permet d’identifier les cuisiniers les plus actifs.
* **Commandes sur une Période** : Affiche les commandes effectuées sur une période donnée.
* **Moyenne des Prix des Commandes** : Permet d’évaluer la rentabilité et la compétitivité des tarifs.

**4. Intégration avec la Base de Données**

L’intégration de la base de données se base sur le schéma Entité-Association fourni. Les points clés sont :

* **Création et Peuplement** : Le script SQL a été utilisé pour créer les tables (Plat, Utilisateur, Client\_E, Client\_P, Commande, etc.) et insérer des données d’exemple.
* **Accès via C#** : Utilisation de MySql.Data.MySqlClient pour la communication entre l’application console et la base de données.
* **Transactions et Intégrité** : Les transactions sont utilisées pour garantir l’intégrité lors des opérations de création/modification (ex : ajout d’un client, création d’une commande).

**5. Tests Unitaires et Validation**

Une suite de tests unitaires a été développée pour vérifier :

* La bonne lecture et écriture des données dans la base.
* La validité des parcours calculés par les algorithmes de chemin le plus court.
* L’exactitude des statistiques calculées par le module statistique.

**6. Conclusion**

L’étape 2 du projet a permis de :

* Généraliser le modèle de graphe en le rendant générique et adaptable à divers contextes (ici, le métro de Paris).
* Intégrer les deux parties du projet en assurant la cohérence entre la gestion des commandes et le traitement des données (BDD et interface console).
* Mettre en place une solution complète et évolutive, en s’appuyant sur des algorithmes performants et une architecture orientée objet robuste.