Maxime MOTTIER, Hilaire PERREE, Antoine ROUCAU, TD Q

Rapport projet C# Transconnect

Le projet consiste à concevoir une application en langage C# permettant de centraliser la gestion des salariés, clients, véhicules et commandes, tout en intégrant un module avancé d'optimisation des trajets de livraison. Grâce à la modélisation d'un graphe représentant les distances entre les villes, l'application permettra de calculer les chemins les plus courts à l'aide des algorithmes Dijkstra, Bellman-Ford et Floyd-Warshall, et de visualiser dynamiquement les trajets sur une interface graphique. Ce système vise à offrir à la direction une vision claire de la structure hiérarchique de l'entreprise, un historique des livraisons, une gestion efficace des ressources humaines et une planification intelligente des livraisons, le tout à travers une interface conviviale et interactive.

Analyse Comparative des Algorithmes de Recherche du Plus Court Chemin

I. **Algorithme de Dijkstra** :

```
public class Dijkstra
{
    4 references
    public static TimeSpan TempsExecution { get; private set; }
    2 references
    public static long UtilisationMemoire { get; private set; }
    10 references
    public static List<Gra.Noeud> TrouverCheminLePlusCourt(Gra.Graphe graphe, Gra.Noeud source, Gra.Noeud destination)
    {
        GC.Collect();
        GC.WaitForPendingFinalizers();
        GC.Collect();

        long memoireAvant = GC.GetTotalMemory(true);


        Stopwatch chrono = Stopwatch.StartNew();

        // Dictionnaires pour stocker les distances et les prédecesseurs
        Dictionary<Gra.Noeud, double> distances = new Dictionary<Gra.Noeud, double>();
        Dictionary<Gra.Noeud, Gra.Noeud> predecesseurs = new Dictionary<Gra.Noeud, Gra.Noeud>();


        // Initialiser toutes les distances à l'infini et les prédecesseurs à null
        foreach (var noeud in graphe.Noeuds)...

        // La distance de la source est 0
        distances[source] = 0;

        // Liste des noeuds non visités
        var nonVisites = new List<Gra.Noeud>(graphe.Noeuds);

        while (nonVisites.Count > 0)...

        // Si la destination n'a pas été atteinte
        return null;
    }
```

Principe de fonctionnement :

Algorithme efficace pour les graphes à poids positifs. Il utilise une approche gloutonne pour étendre les chemins les plus courts depuis la source.

Complexité :

- Temps :
  - O((V + E) log V) avec un tas binaire
  - $O(V^2)$ avec une matrice d'adjacence
- Espace : O(V)

Avantages :

- Très rapide pour les graphes à poids positifs
- Bien adapté aux grands graphes peu denses
- Fonctionne avec des structures optimisées (tas, etc.)

Inconvénients :

- Ne fonctionne pas avec des poids négatifs
- Plus complexe à implémenter que Bellman-Ford

II. **Algorithme Bellmanfort** :

```
9     public class BellmanFord
10    {
          5 references
11        private Gra.Graphe _graphe;
          13 references
12        private Dictionary<Gra.Noeud, double> _distances;
          5 references
13        private Dictionary<Gra.Noeud, Gra.Noeud> _predecesseurs;
          3 references
14        public static TimeSpan TempsExecution { get; private set; }
          2 references
15        public static long UtilisationMemoire { get; private set; }
16
          3 references
17        public BellmanFord(Gra.Graphe graphe) ···
23
          3 references
24        public bool CalculerPlusCourtsChemins(Gra.Noeud source) ···
66
          2 references
67        private bool Relaxer(Gra.Lien lien, bool verifierSeulement = false) ···
93
          3 references
94        public List<Gra.Noeud> RecupererChemin(Gra.Noeud destination) ···
107
          0 references
108       public double GetDistance(Gra.Noeud noeud) ···
112   }
113   }
```

Principe de fonctionnement :

L'algorithme résout le problème du plus court chemin à partir d'une source unique. Contrairement à Dijkstra, il supporte les arêtes à poids négatifs.

Complexité :

- Temps : O(V × E)
- Espace : O(V)

Avantages :

- Gère les poids négatifs
- Détection des cycles négatifs
- Algorithme simple à implémenter

Inconvénients :

- Plus lent que Dijkstra ou Floyd-Warshall pour les grands graphes sans poids négatifs

III. Algorithme FloydWarshall

```
public class FloydWarshall
{
    4 references
    private Gra.Graphe _graphe;
    10 references
    private Dictionary<Gra.Noeud, Dictionary<Gra.Noeud, double>> _distances;
    9 references
    private Dictionary<Gra.Noeud, Dictionary<Gra.Noeud, Gra.Noeud>> _predecesseurs;
    3 references
    public static TimeSpan TempsExecution { get; private set; }
    2 references
    public static long UtilisationMemoire { get; private set; }

    2 references
    public FloydWarshall(Gra.Graphe graphe) ···

    1 reference
    private void Initialiser() ···

    2 references
    public void CalculerPlusCourtsChemins() ···

    2 references
    public List<Gra.Noeud> RecupererChemin(Gra.Noeud source, Gra.Noeud destination) ···

    0 references
    public double GetDistance(Gra.Noeud source, Gra.Noeud destination) ···
}
}
```

Fonctionnement :

Algorithme de programmation dynamique qui calcule les plus courts chemins entre toutes les paires de sommets.

Complexité :

- Temps : $O(V^3)$
- Espace : $O(V^2)$

Avantages :

- Donne les plus courts chemins pour toutes les paires
- Gère les poids négatifs (mais pas les cycles négatifs)
- Implémentation simple avec une matrice

Inconvénients :

- Très coûteux en temps et en mémoire pour de grands graphes
- Peu adapté aux grands graphes clairsemés

Innovation :

1. Interface utilisateur en Windows Forms / WPF :

L'application sera développée en Windows Forms ou WPF, offrant une interface ergonomique et conviviale qui centralise l'ensemble des fonctionnalités de gestion de l'entreprise. Grâce à un menu principal structuré, l'utilisateur pourra accéder facilement aux modules de gestion des salariés, des clients, des commandes, ainsi qu'au suivi logistique.

2. Persistance automatisée des données au format CSV :

Une sauvegarde automatique en fichiers CSV est mise en place pour assurer la persistance des données (salariés, clients, commandes, véhicules, livraisons, etc.). Ce choix combine la portabilité, la facilité de lecture humaine, et une intégration aisée avec d'autres outils bureautiques comme Excel.
À chaque modification (ajout, suppression ou mise à jour), les données sont automatiquement sauvegardées, garantissant une protection contre la perte d'informations. Cette solution évite le besoin d'une base de données complexe, tout en assurant une structure de données fiable, simple et exportable.

3. Graphiques statistiques et visualisation des revenus :

Le projet intègre un module de visualisation graphique des statistiques, notamment les revenus mensuels, via des courbes ou histogrammes dynamiques. Ces visualisations permettent à la direction d'avoir une vue claire sur : les tendances de chiffre d'affaires, les périodes de forte activité, les performances clients et les volumes de commandes.

4. Fonction supplémentaire dans les services :

Trié client par date de naissance,

Affichage du chiffre d'affaires global de l'entreprise,

Possibilité d'exporter les statistiques en csv.