SAE_2.2_Graphe_2024_Cadet_Noel

- CADET Matteo
- NOEL Julien

Recherche de plus court chemin dans un graphe

Ce rapport présente les travaux réalisés dans le cadre de la SAE 2.02, axée sur l'exploration algorithmique pour la recherche du plus court chemin dans un graphe. S'appuyant sur les cours magistraux et les travaux dirigés de graphes vus dans les ressources mathématiques, cette SAE vise à appliquer des solutions algorithmiques pour résoudre ce problème.

Contexte et Objectifs

Le problème du plus court chemin consiste à déterminer le chemin d'un nœud à un autre dans un graphe orienté, de manière à minimiser la somme des coûts associés aux arcs parcourus. Ce type de problème trouve des applications pratiques variées, notamment dans la recherche de trajets optimisés par les GPS, en fonction de critères tels que la durée ou la distance.

Pour cette SAE, nous avons été amenés à :

- Représenter des graphes à partir d'une interface à implémenter.
- Développer des algorithmes pour trouver le plus court chemin, incluant l'algorithme de Bellman-Ford et celui de Dijkstra.
- Comparer les performances de ces algorithmes.
- Valider les solutions développées par des tests unitaires.

Structure du Travail

Le travail a été effectué en binôme sur une durée totale de 8 heures et les rendus attendus incluent :

- Un dépôt git structuré.
- Un code Java clair, commenté et accompagné d'une Javadoc.
- Un rapport détaillant le travail réalisé et les réponses aux questions posées.
- Des tests unitaires validant les implémentations.

Le contenu du rapport est structuré en plusieurs sections :

- Analyse des résultats
  - Tentative de vérification 1 - (échec)
  - Tentative de vérification 2 - (réussite)
  - Comparaison des algorithmes : Analyse des performances et efficacité des algorithmes implémentés.

- Conclusion : Synthèse des apprentissages, difficultés rencontrées et bilan du travail effectué.

Analyse des résultats

Les 2 méthodes permettent de trouver les mêmes résultats, comme prévu.

Maintenant, il peut être intéressant de comparer les temps d'exécution de chaque méthode.

Tentative de vérification 1 - (échec)

Nous avons voulu le vérifier grâce aux formules de calcul de la complexité, mais il s'est avéré que ce n'était pas la bonne méthode à utiliser. Voici notre raisonnement qui n'est pas à prendre en compte.

> Calculons la complexité de chaque méthode.

Nous allons prendre le graphe donné dans l'énoncé

```
A B 12
A D 87
B E 11
D B 23
D C 10
C A 19
E D 43
```

Méthode Dijkstra

```
Complexité = (arêtes + sommets) x ln(sommet)   Complexité g = (7 + 5) x ln(5) = 19,31 (environ)
```

Méthode BelmanFord

```
Complexité = arêtes x sommets   Complexité g = 7 x 5 = 35
```

Mais ce n'est pas une bonne méthode à utiliser, nous avons donc pensé à une approche plus pertinente.

Tentative de vérification 2 - (réussite)

Nous avons introduit du code dans nos Main pour calculer le temps d'exécution de nos méthodes.

Voici comment nous avons procédé :

```
        // Déclaration des variables de durées
double duree;
```

```
double duree_ms;
long date_debut = System . nanoTime () ; //début du chronomètre

//Calcul des chemins les plus courts
// - Création d'un objet Djisktra ou BelmanFord
// - Utilsation de la méthode résoudre sur l'objet Djisktra ou BelmanFord

//Arrêt du chronomètre et calcul du temps en ns et en ms + affichage
long date_fin = System . nanoTime () ;
duree = (date_fin - date_debut) ;
duree_ms = duree / 1000000;
        System.out.println("La duree de calcul du point fixe avec la méthode de Djikstra est de : " + dur
```

Suite à l'exécution de nos Mains, nous en avons conclu que les temps d'exécution n'étaient pas les mêmes à chaque fois (on étudie des temps en nanosecondes qui sont très variables dûs aux différents programmes qui tournent sur l'ordinateur), les résultats étaient très variables.

Pour palier à cela, nous avons modifié notre code pour faire en sorte d'exécuter nos méthodes 100 fois de suite puis de faire une moyenne du temps total pour obtenir des résultats cohérents.

```
        // Variables de durées
        double total = 0;
        double duree;
        double duree_ms;
        Valeur resultat_dij = null;

        Dijkstra dij = new Dijkstra();

        //Calcul des chemins les plus courts
        for (int i = 0; i < 100 ; i++){
            long date_debut = System . nanoTime () ; //début du chronomètre
            resultat_dij = dij.resoudre(graphe, "A");
            long date_fin = System . nanoTime () ;
            total += date_fin-date_debut;
        }

        //Arrêt du chronomètre et calcul du temps en ns et en ms + affichage
        duree = total/100 ;
        duree_ms = duree / 1000000;
        System.out.println("La duree de calcul du point fixe avec la méthode de Djikstra est de : " + dur
```

Nous avons adapté le même code pour l'autre Main (Belman Ford).

Suite à plusieurs exécutions, nous en avons conclu que l'algorithme de Belman Ford était plus efficace sur le graphe 1.

Malheureusement, nous n'avons pas eu assez de temps pour tester d'autres graphes.

Conclusion

En conclusion, nous avons trouvé cette SAE assez pertinente et concrète, puisqu'elle était directement liée au cours de mathématiques, qui nous présentait les méthodes de la sorte à nous rendre compte qu'il était possible de calculer les temps de réalisation d'un projet dans des organisations complexes.

Mon coéquipier et moi-même avons pu utiliser Git sans problèmes et nous avons pu nous répartir aisément les tâches.

Personnellement, c'est la SAE que j'ai préférée.