S2-02
Exploration algorithmique d'un problème

Recherche de plus court chemin dans un graphe

Rémi Choffat - Mathieu Graff

Représentation d'un graphe

Dans cette partie, nous avons initialisé le projet en ajoutant les classes de base : `Arc`, `Arcs`, `Graphe` et `GrapheListe`. Ces classes vont nous servir de base pour le reste du sujet. Nous avons ensuite créé une classe `Main` dans laquelle nous avons créé un graphe, puis l'avons affiché afin de voir que notre code fonctionne. Enfin, nous avons écrit la classe de tests unitaires pour valider notre code.

Calcul du plus court chemin par point-fixe

Dans cette partie, nous avons réfléchi à l'algorithme du parcours de graphe selon la méthode du point fixe (voir ci-dessous). Ensuite, nous avons implémenté cet algorithme dans une classe `BellmanFord`, en utilisant la classe `Valeur`. La méthode créée retourne un objet `Valeur` contenant les distances et les parents de chaque nœud, après convergence de l'algorithme. Nous avons ensuite réalisé un `Main` qui applique cet algorithme, ainsi qu'une classe de tests `TestGraphe`, qui vérifie que toutes nos méthodes et algorithmes sont corrects. Enfin, la méthode `calculerChemin` permet de retourner une liste de nœuds correspondant au chemin le plus court entre un point de départ et un point d'arrivée donnés.

Question 8

Algorithme de la fonction `pointFixe` :

```
fonction pointFixe(Graphe g, Noeud depart) retourne v:
début
	listeNoeuds <- g.listeNoeuds()
	Pour chaque noeud de listeNoeuds
		Si noeud = depart
			v.setValeur(depart, 0)
		Sinon
			v.setValeur(noeud, Infini)
		Fin si
	Fin Pour

	list_vals <- {depart}
	list_tmp_vals <- {}

	Tant que non list_vals = list_tmp_vals Faire
		list_tmp_vals <- list_vals
		list_vals <- {}

		Pour noeud dans listeNoeuds faire
			suivants <- g.suivants(noeud)
			Pour successeur dans suivants Faire
				val <- successeur.getCout() + v.getValeur(noeud)
				Si v.getValeur(successeur.getDest()) > val Alors
					v.setValeur(successeur.getDest(), val)
					v.setParent(successeur.getDest(), noeud)
				Fin Si
				list_vals <- list_vals U {val}
			Fin Pour
		Fin Pour
	Fin tant que
fin
```

Calcul du plus court chemin par algorithme de Dijkstra

Dans cette partie, nous avons réfléchi à l'efficacité mathématique de l'algorithme de Dijkstra quant à la résolution du problème du plus court chemin. A partir de l'algorithme fourni, nous avons écrit la méthode `resoudre()` de la classe `Dijkstra` qui permet de calculer les plus courts chemins vers les autres nœuds du graphe avec l'algorithme de Dijkstra. Nous avons ensuite pu définir une interface `Algorithme` qui permet de choisir l'algorithme à utiliser selon une approche `Strategie`, pour une meilleure compréhension et maintenabilité du code. Enfin, nous avons écrit des tests unitaires ainsi qu'un programme principal, pour vérifier le bon fonctionnement de notre algorithme.

Validation et expérimentation

Question 16

Les tests sont exécutés sur les graphes fournis sur Arche.
Les tests ont été effectués avec un ordinateur ayant un processeur Intel Core i5-7300HQ.
En exécutant nos algorithmes sur ces graphes, nous avons obtenu les résultats suivants :

Graphes de 10 noeuds, exécuté 10 fois :

- Bellman-Ford :
  - Nombre d'itérations : 3
  - Nombre de calculs : 117
  - Temps de calcul moyen : 2 ms
- Dijkstra :
  - Nombre d'itérations : 10
  - Nombre de calculs : 39
  - Temps de calcul : 0 ms

Graphes de 900 noeuds, exécuté 10 fois :

- Bellman-Ford :
  - Nombre d'itérations : 3
  - Nombre de calculs : 607980
  - Temps de calcul moyen : 1036 ms
- Dijkstra :
  - Nombre d'itérations : 900
  - Nombre de calculs : 202660
  - Temps de calcul : 988 ms

Temps de calcul moyen sur tous les graphes exécutés 5 fois :

- Bellman-Ford : 151 ms
- Dijkstra : 147 ms

Question 17

Nous pouvons observer que l'algorithme de Dijkstra demande moins de calculs, et donne une réponse plus rapidement.

Question 18

L'algorithme de Dijkstra serait donc plus efficace peu importe la situation, que le graphe soit grand ou petit.

Conclusion

Lors de cette SAÉ, nous avons appris à réfléchir à des algorithmes et à leur implémentation, dans le but de répondre à un problème concret. Cela nous a permis de tester plusieurs approches pour résoudre ce problème, en nous offrant l'opportunité de comparer nos solutions expérimentalement. De plus, cela nous a donné l'occasion de développer un code optimisé, clair et fonctionnel, ainsi que de réfléchir aux différents tests à exécuter.