Tests de correction pour l'algorithme de Dijkstra

6 mai 2019

Lors de l'étape 3, il vous a été proposé de vérifier quelques caractéristiques du déroulement de l'algorihme (si vous ne l'avez pas fait, ces premiers tests sont rappelés ci-dessous dans les parties 1.1 et 1.2). Nous allons pousser les tests un peu plus loin pour nous intéresser aux caractéristiques des solutions obtenues. Il s'agit dans un premier temps de vérifier que les solutions sont valides (partie 1.3), puis nous nous intéresserons à l'optimalité de ces solutions, tout d'abord en supposant que l'on dispose de la connaissance des solutions optimales (partie 2.1) puis en supposant que celles-ci ne sont pas connues (partie 2.2).

Ces tests sont à reproduire pour l'algorithme en version A-star. Les tests effectués dans les parties 1.3, 2.1 et 2.2 devront être présentés dans le rapport.

1 Avez-vous codé un algorithme de Dijkstra ?

1.1 Vérification visuelle (a été demandé dans l'étape 3 du sujet)

La première vérification que l'on peut faire est une vérification visuelle :

- afficher sur la carte les sommets explorés au fur et à mesure de leur ajout dans le tas

Est-ce que l'affichage est cohérent avec le principe de la méthode ? Quelle(s) différence(s) peut-on voir lors dans les affichages lors du calcul du plus court chemin en distance et du plus court chemin en temps ?

1.2 Caractéristiques de l'algorithme (a été demandé dans l'étape 3 du sujet)

La seconde vérification consiste à tester si les étapes de calcul effectuées sont cohérentes avec le principe de l'algorithme. Quelques modifications de votre algorithme (par de simples affichages temporaires) peuvent permettre par exemple (tout n'est pas obligatoire, à vous d'adapter) :

- de vérifier que le coût des labels marqués est croissant au cours des itérations ;
- d'afficher l'évolution de la taille du tas ;
- d'afficher le nombre d'arcs du plus court chemin, le nombre d'itérations de l'algorithme.
- de vérifier que le tas reste valide au cours des itérations (cela nécessite d'ajouter une méthode `IsValide` dans la classe implémentant le tas) ;
- d'afficher le nombre de successeurs testés à chaque itération (que le successeur soit déjà marqué ou non).

Les résultats obtenus sont-il cohérents avec le principe de la méthode ?

1.3 Caractéristiques des solutions obtenues

La troisième vérification consiste à tester si chaque solution obtenue est une solution valide, c'est à dire formant un trajet existant dans le graphe. Normalement, les solutions produites ont été mémorisées sous forme de "chemin" (en utilisant la classe `Path`).

En vous inspirant des tests `JUnit` proposés pour la classe `Path`, vous allez développer des tests permettant :

1. de vérifier que le trajet obtenu est correct (en utilisant les méthodes adéquates de la classe `Path`) ;

Pour mettre en place ces tests `JUnit`, définissez plusieurs scénarios. Chaque scénario est défini par :

- une carte (tester des cartes routières et non routières)
- la nature du coût (tester en distance et en temps)
- une origine et une destination (tester plusieurs valeurs)

Les scénarios doivent être convaincants et représentatifs des situations pouvant arriver (chemin inexistant, chemin de longueur nulle, ...).

Point technique : vous aurez besoin d'un objet `ArcInspector`. Pour l'obtenir, vous pouvez invoquer la méthode `getAllFilters()` de `ArcInspectorFactory` et prendre le premier de la liste puis le troisième.

Ces tests sont à compléter pour intégrer ceux de l'algorithme en version A-star.

Note : le rapport devra contenir une explication (scénarios, résultats, analyse) des tests effectués pour la vérification des caractéristiques des deux agorithmes implémentés et de la validité des solutions obtenues.

2 Tests d'optimalité des solutions

La dernière vérification à effectuer porte sur l'optimalité des solutions obtenues.

2.1 Tests d'optimalité avec oracle

On supposera que l'algorithme de Bellman-Ford fournit une solution optimale pour le problème de plus court chemin.

En vous inspirant des tests `JUnit` réalisés à l'étape précédente, développez de nouveaux tests pour comparer les couts des solutions obtenues par l'algorithme de Dijkstra et les couts de celles obtenues par l'algorithme de Bellman-Ford (utilisé comme "oracle").
Comme précédemment, les scénarios doivent être convaincants et représentatifs des situations pouvant arriver (chemin inexistant, chemin de longueur nulle, ...).

Ces tests sont à compléter pour intégrer ceux de l'algorithme en version A-star.

Note : le rapport devra contenir une explication (scénarios, résultats, analyse) des tests d'optimalité avec oracle effectués pour les deux agorithmes implémentés.

2.2 Tests d'optimalité sans oracle

Il n'est pas toujours possible de disposer d'information sur la valeur de la solution optimale (Etes-vous d'accord avec cette affirmation ?)

Imaginez d'autres types de test permettant de vérifier l'optimalité d'une solution (ou d'avoir une forte présemption d'optimalité) et proposez des scénarios associés.

Ces tests sont à compléter pour intégrer ceux de l'algorithme en version A-star.

Note : le rapport devra contenir une explication des tests d'optimalité sans oracle envisagés pour les deux agorithmes. L'implémentation de ces tests n'est pas obligatoire (seulement conseillée).