université angers

FACULTÉ
DES SCIENCES
Unité de formation et de recherche

Licence 3 MPCIE
Projet algorithmique des graphes

Projet - implémentation d'algorithmes efficaces

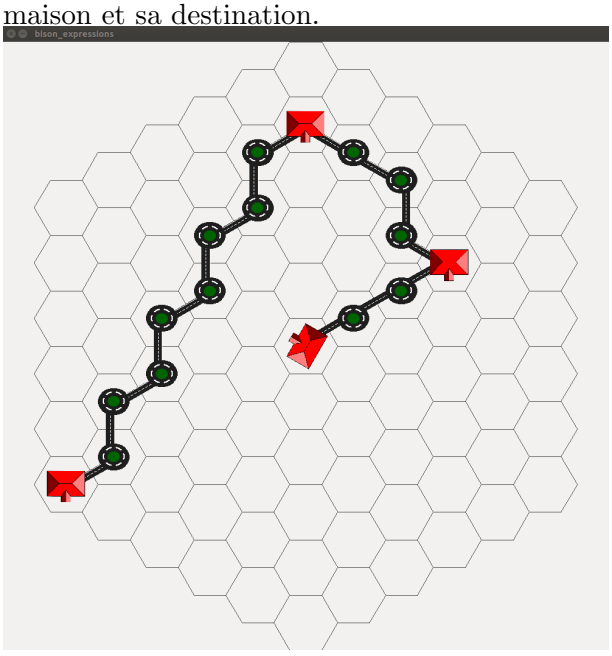
Consignes : Le projet est à effectuer en binôme ou seul et est à déposer sur moodle au plus tard le 20 Décembre 2019 à 23h59. Les sources de ce projet doivent être distinctes de celles du projet de compilation. Votre projet doit contenir une notice explicative du fonctionnement de vos algorithmes, comment compiler et exécuter. Un projet qui ne compile pas se verra retirer 4 points. Pour chaque exercice vous devrez fournir un ensemble de fichiers de tests fonctionnels permettant de vérifier que l'algorithme fonctionne correctement. Pensez à choisir des exemples pertinents permettant de comprendre les spécificités des algorithmes. Le choix des exemples entrera dans la notation.

Dans ce projet, nous souhaitons implémenter 3 algorithmes permettant de construire les réseaux d'une ville. Une ville est représentée par un graphe orienté et pondéré constitué de sommets (les maisons) et d'arcs (les routes).

Exercice 1 : Plus court chemin

Dans une ville les habitants souhaitent connaître un des chemins les plus courts pour aller d'une maison A à une maison B. Étant donné que le nombre de routes et de maisons peut être important dans une ville nous ne souhaitons pas parcourir tous les chemins possible car ce serait trop long. Contrairement aux algorithmes de Dijkstra, Bellman-Ford et Warshall-Floyd nous ne souhaitons pas obtenir forcément LE plus court mais simplement obtenir rapidement un chemin court entre ces deux maisons. Nous allons pour cela utiliser l'algorithme A* (A étoile ou A-star) qui permet d'obtenir un court chemin sans passer par tous les sommets.

Description de l'algorithme : L'algorithme A* utilise une heuristique pour fonctionner. Cette heuristique fait une estimation de la distance restante entre chaque sommet parcouru et la destination. Il est donc nécessaire de faire une estimation de la distancen, pour cela vous êtes libres de choisir le critère qui vous convient tant que celui-ci reste rapide à estimer et cohérent. Un exemple d'estimation peut être celui de la distance à vol d'oiseau entre une maison et sa destination.



Exemple :
Sur cet exemple la maison du centre est à une distance estimée de 5 de la maison au sud-ouest mais il n'y a pas de routes directes. La distance réelle est de 3 + 4 + 9 = 16. Les distances estimées pour les maisons dans l'ordre de la route sont : 5, 8, 9. L'estimation de la distance jusqu'à la destination n'est pas toujours fiable mais permettra à l'algorithme de faire un choix sur le chemin à prendre. Bien sûr l'estimation n'est pas exacte (sinon cela signifie qu'un algorithme a déjà calculé le plus court chemin pour tous les sommets). Attention une estimation trop basse ne sera pas efficace et une estimation trop haute pourra donner un chemin qui ne sera pas le plus court.

Une fois l'heuristique choisie, l'algorithme fonctionne ainsi. À chaque étape l'algorithme va choisir le chemin estimé le plus court en additionnant la distance des sommets voisins et l'estimation de la distance du prochain sommet avec la destination. Il est parfois nécessaire de revenir sur un ancien chemin si la distance estimée du

chemin courant devient plus grande que l'estimation d'un autre chemin.

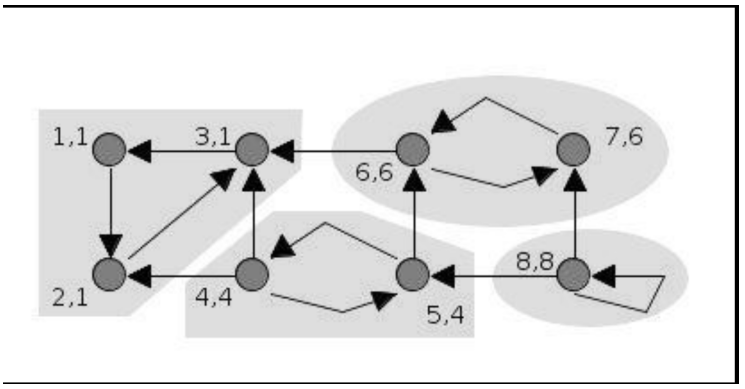
Tâches à réaliser :

- Implémentez l'algorithme A*.
- Afficher chaque étape de l'algorithme dans le terminal (choix du prochain noeud, calculs, etc...) de manière lisible.
- Comparez le temps nécessaire pour obtenir le chemin le plus court avec Dijkstra et A*. (Écrire une fonction qui effectue la comparaison pour que le correcteur puisse comparer)

Vous joindrez à votre projet un fichier appelé résultat qui comportera sur chaque exemple la comparaison entre Dijkstra et A* sous forme de tableaux.

Exercice 2 : Composantes fortement connexes

Nous souhaitons maintenant vérifier que chaque maison possède une route permettant d'atteindre n'importe quelle autre maison de la ville. De plus s'il existe une partie de la ville inateignable nous souhaitons être en mesure de raccorder les parties de la manière la plus efficace possible. Pour ça nous allons utiliser l'algorithme de Tarjan qui sert à déterminer les composantes fortement connexes d'un graphe.



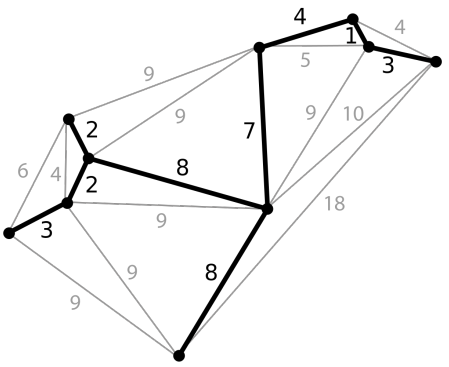
Description de l'algorithme : Tarjan est un algorithme récursif qui cherche à isoler en partition les différentes parties d'un graphe orienté dont les sommets sont connexes. L'algorithme garde une trace des sommets déjà visités afin de ne parcourir qu'une seule fois chaque sommet. Lorsqu'un sommet est visité on le numérote et l'ajoute à une pile. Tous les sommets sont parcourus en profondeur et numérotés. Lorsque l'on tombe à nouveau sur un sommet déjà visité c'est qu'il existe un cycle. On cherche alors à définir quelle est la racine (sommet de départ de la composante connexe) en recherchant le sommet du cycle avec la numérotation la plus petite. Une fois le parcours terminé on ajoute les sommets par ensemble à notre partition. S'il reste des sommets non parcourus on recommence l'opération à partir d'un des sommets non parcourus (c'est le cas lorsqu'il n'existe aucun chemin allant d'une partie du graphe à une autre).

Tâches à réaliser :

- Implémentez l'algorithme de Tarjan.
- Afficher chaque étape de l'algorithme dans le terminal (ordre des noeuds visités, passage à une autre racine, etc...) de manière lisible.
- Implémentez un algorithme qui permet de raccorder les parties de la ville entre elles de manière efficace.

Exercice 3 : Spanning tree

Pour finir le maire veut installer la fibre entre toutes les maisons de la ville en dépensant le moins possible. Il n'est possible de déposer la fibre que le long d'une route, le prix de la fibre dépendant de la longueur de câble utilisé il est nécessaire de trouver un arbre couvrant minimal entre toutes les maisons. Nous allons pour cela utiliser l'algorithme de Kruskal. Il est pour cela nécessaire de s'assurer que le graphe est connexe, nous considérons ici que le graphe n'est plus orienté (le réseau routier n'est pas le même que le réseau de câbles).



Description de l'algorithme : L'algorithme construit un arbre couvrant minimum en sélectionnant des arêtes par poids croissant. Plus précisément, l'algorithme considère toutes les arêtes du graphe par poids croissant et pour chacune d'elles, il la sélectionne si elle ne crée pas un cycle. (cf. Wikipedia)

Tâches à réaliser :

— Implémentez l'algorithme de Kruskal.

— Afficher chaque étape de l'algorithme dans le terminal (ordre des arêtes sélectionnées, sommets concernés, etc...) de manière lisible.

Exercice 4 : Bonus : K-coloration

En bonus, nous vous proposons d'implémenter une K-Coloration. Utiliser l'interface graphique proposée en compilation et un algorithme de K-Coloration pour colorer les maisons de la ville d'une couleur différente pour chaque maison adjacente.

Description de l'algorithme : Pour effectuer une K-coloration nous commençons avec K=1 puis on augmente le nombre de couleurs tant qu'il n'est pas possible d'avoir toutes les maisons adjacentes avec une couleur différente. Il n'est pas demandé ici d'avoir un algorithme efficace mais un algorithme correct.

Tâches à réaliser :

— Implémentez l'algorithme de K-Coloration.

— Utiliser l'algorithme sur un graphe créé à partir de l'interface graphique du projet de compilation.