DIJKSTRA

Rapport de mise en œuvre

Algorithme des graphes

CHANTREL Romain IMR2 27/10/2019

Objectifs

L'algorithme de Dijkstra sert à résoudre le problème du plus court chemin, c'est-à-dire qu'il consiste à trouver un chemin d'un sommet à un autre de façon à ce que la somme des poids des arcs de ce chemin soit minimale.

Par exemple, un réseau routier peut être considéré comme un graphe avec des poids positifs. Les nœuds représentent des croisements de routes et chaque arc du graphe est associé à un segment de route entre deux croisements. Il est possible de modéliser des rues en sens unique avec un graphes orienté, et le poids de chaque arc peut correspondre à un taux de circulation, au temps nécessaire pour parcourir ce segment ou à la longueur de celui-ci. Ce type d'algorithme permet donc de déterminer un plus court chemin pour se rendre d'un point A à un point B sur une carte. Les applications de navigation tel que Google Maps, Waze, TomTom et bien d'autres utilisent ce genre d'algorithme afin de fournir à l'utilisateur le meilleur chemin pour aller d'un point de départ à un point d'arrivé.

Principes

L'algorithme de Dijkstra est applicable seulement si les poids des arcs sont supérieurs ou égale à zéro. Et contrairement à d'autres algorithme permettant de résoudre le problème du plus court chemin tel que l'algorithme de Bellman, l'algorithme de Dijkstra accepte l'existence de cycles dans les graphes.

L'algorithme de Dijkstra prend en entrée un graphe orienté ou non orienté et un sommet source et donne en sorti la liste des poids des plus courts chemins pour aller du sommet source à tous les autres sommets.

Il s'agit de construire progressivement un sou-graphe dans lequel sont classés les différents sommets par ordre croissant de leur distance minimale au sommet de départ, la distance correspond à la somme des poids des arcs empruntés.

Lors de notre mise en œuvre, nous intégrerons une gestion d'une table de routage. Il s'agit de sortir en plus du coût entre le sommet de départ et chaque sommet, de sortir le chemin correspondant au chemin le plus court entre le sommet de départ et chaque sommet.

Au départ, nous considérons que les distances et les prédécesseurs du sommet de départ à chaque sommet sont infinies.

Ensuite, nous définissons les distances et les prédécesseurs des sommets directement accessible depuis le sommet de départ.

Puis, au cours de chaque itération, on choisit, en dehors des sommets déjà découverts, le sommet de distance minimale, on l'ajoute aux nœuds découverts. Et on met à jour les distances et les prédécesseurs des sommets voisons de celui ajouté avec la distance la plus courte entre la distance existante et celle obtenue en ajoutant ce nouveau sommet.

L'algorithme se termine lorsqu'il n'y a plus aucun sommet à découvrir ou lorsqu'il n'y a plus aucun sommet accessible.

Choix de mises en œuvre

Pour la mise en œuvre de cet algorithme, j'ai choisi d'utiliser le langage Python. Il s'agit d'un langage de programmation interprété multiplateformes. Il permet différent type de programmation, lors de la réalisation de ce projet, j'ai choisi la programmation orienté objet.

Dans le fichier « dijkstra.py » joint à ce rapport, vous trouverez l'objet « Dijkstra » qui permet d'utiliser l'algorithme du même nom. Cet objet est composé de cinq fonctions, tout d'abord le constructeur qui permet d'initialiser l'objet, c'est-à-dire de définir la matrice, le sommet de départ, la liste des distances et la liste des prédécesseurs. Une fonction qui permet d'exécuter l'algorithme de Dijkstra définit précédemment. Une fonction qui permet, à partir de la liste des prédécesseurs du plus court chemin de chaque prédécesseur, de créer le chemin complet depuis le sommet de départ. Une fonction qui permet d'afficher le résultat sous forme de tableau dans la console. Et enfin, une fonction qui permet d'afficher le résultat sur une interface graphique, c'est-à-dire le tableau de résultat et le graphe correspondant.

Afin de représenter les matrices, j'ai utilisé la librairie « numpy » (diminutif de « Numerical Python ») qui permet d'effectuer des calculs scientifiques, notamment avec des tableaux d'entier de dimension N, et donc avec des matrices.

Pour l'interface, j'ai utilisé la librairie « matplotlib » qui permet de générer directement des graphiques à partir de Python. Au fil du temps, cette librairie est devenue puissante, compatible avec beaucoup de plateformes, et capable de générer des graphiques dans beaucoup de formats différents.

Matplotlib ne pouvant pas afficher de graphes nativement, j'ai également utilisé la librairie « networkx » qui permet d'instancier des graphes composés de nœuds et de segments. Lié à matplolib, cette librairie permet d'afficher des graphes à partir de matrices, et notamment de matrices numpy.

Tests

Pour tester l'algorithme développé, j'ai créé une liste de cinq matrices et une fonction permettant de créer des matrices aléatoirement. Dans le fichier « dijkstra_test.py » joint à ce rapport, vous trouverez une objet « Dijkstra_test » qui permet à partir d'une matrice sélectionnée (dans la liste prédéfinie, ou une créé aléatoirement) et d'un sommet de départ sélectionné, d'obtenir le résultat fournit par l'algorithme de Dijkstra. Une option permet également d'afficher le résultat sur une interface graphique au lieu de la console. A savoir que le résultat via l'interface permet d'avoir le graphe d'affiché.

Pour une utilisation plus confortable, le choix de la matrice, du nœud de départ, de l'affichage graphique ou non, et de la génération automatique de la matrice se fait en ligne de commande. La commande « `python3 dijkstra_test.py --help` » permet d'obtenir de l'aide sur la syntaxe :

```
usage: dijkstra_test.py [-h] [-m MATRICE] [-n NODE] [-d] [-r]

optional arguments:
  -h, --help            show this help message and exit
  -m MATRICE, --matrice MATRICE
                        Un entier identifiant de la matrice à utiliser
  -n NODE, --node NODE  Sommet de départ de la matrice choisie
  -d, --draw            Afficher l'interface graphique (graphe et résultat de
                        l'algorithme de Dijkstra)
  -r, --random          Générer une matrice aléatoire à utiliser
```

De plus, pour pouvoir exécuter ce programme, il faut avoir installé Python (supérieur ou égale à la version v3.6.8) et avoir installer pip (pour python 3). Ensuite, il faut installer les dépendances citées ci-dessus avec la commande : « `pip3 install -r requirements.txt` » (Voir le fichier « README.md »).