Compte-rendu du miniprojet

Algorithme de dijkstra

Oscar Plaisant

L'objet `Graph`

L'objet `Graph` représente un graphe orienté. La consigne demande un graphe non-orienté, il conviendra donc de toujours utiliser la méthode `add_symetric_link`, qui ajoute un lien symétrique, et non pas la méthode `add_link` qui ajoute un lien orienté.

Cet objet possède des méthodes qui sont documentées dans le fichier. Pour accéder à cette documentation, il sera possible d'utiliser la fonction `help` dans un interptéteur python.

Cet objet possède également la méthode `dijkstra`, qui fait l'objet de ce projet. Cette méthode à été implémentée en utilisant un objet appelé "Tas", ou "Tas minimum", soit `Heap` en anglais. Le tas permet de donner une priorité à chaque noeud du graphe, afin que chaque noeud soit parcouru dans le bon ordre (en commencant par le plus proche connu, soit par le minimum, d'où "tas minimum"). Le tas implémenté contient des objets de type `Node`.

L'objet `Heap`

Comme expliqué précédemment, le tas sert ici à donner un ordre de priorité aux noeuds récupérés dans le parcours. Le tas possède principalement 2 méthodes:

la méthode `push`

Cette méthode prend en argument une valeur (ici, de type `Node`), et l'insère dans le tas.

la méthode `pop`

Cette méthode va chercher la plus petite valeur dans le tas, l'enlève du tas et la retourne. Le critère pour déterminer le minimum est l'attribut `cost`

La méthode `top`, bien que non nécessaire mais toutefoi pratique est également implémentée : comme `pop`, elle renvoie la plus petite valeur, mais elle la conserve dans le tas.

L'objet `Node`

Note: Cet objet est en fait généré par un `namedtuple`

Cet objet est une représentation d'un noeud de graphe, mais il n'est utilisé que dans la méthode `dijkstra`. Il possède simplement trois attributs : `name`, le nom du noeud, `cost`, le coût pour arriver à ce noeud, et `parent` le parent de ce noeud (dans le chemin pour y arriver depuis le noeud de départ).

Notes