OSM - PathFinder : Rapport de réalisation

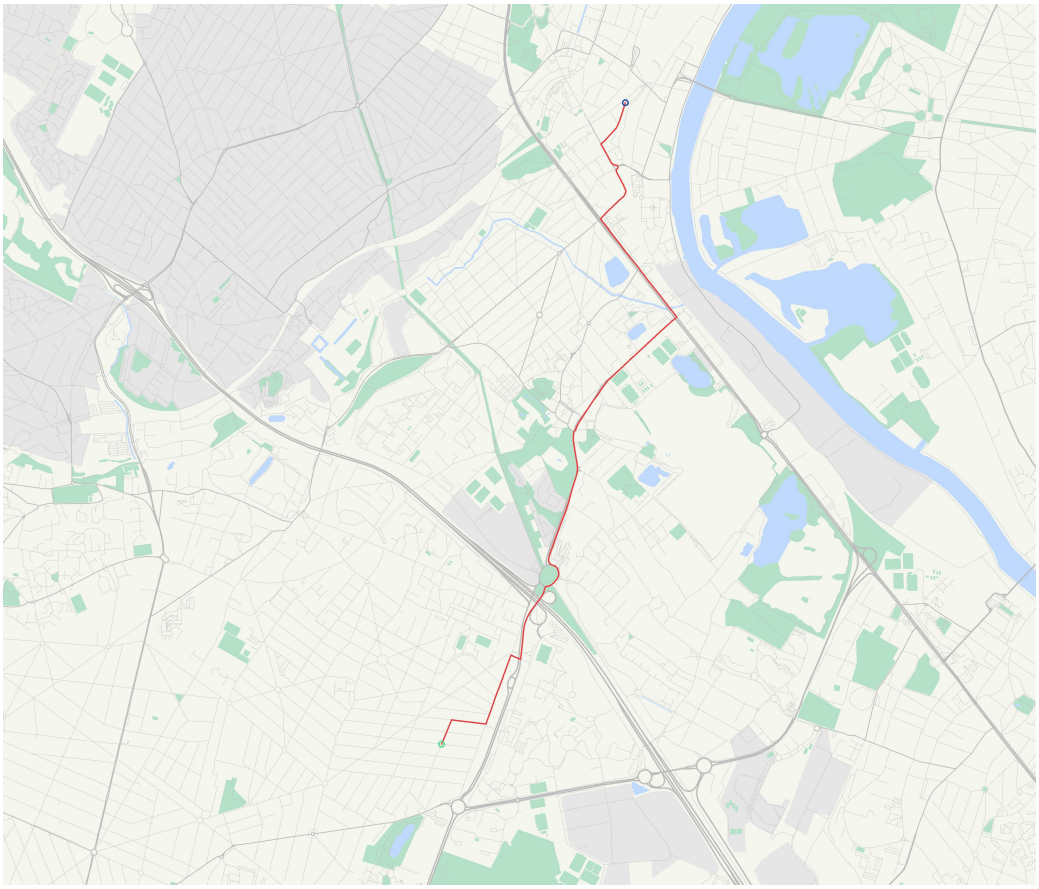

Table des matières

Choix de conception

Réflexions sur la structure de données

Au début du projet, le premier choix a été de réfléchir à la meilleure structure de données visant à contenir des données Open Street Map. Utiliser une table de hachage nous semblait être la meilleure option pour représenter le graphe du réseau routier : Il s'agit d'un excellent moyen de ne pas dupliquer de données tout en permettant d'y accéder de manière très efficace.

Pour ce qui est des itinéraires, il nous est venu l'idée de n'utiliser que les points d'intersections des routes : L'ensemble de tous les autres points, c'est à dire les points permettant de représenter la courbure des routes, deviennent inutiles pour le calcul du plus court chemin dès lors qu'on connait le coût qui sépare deux point d'intersection. Ceci s'applique aussi bien si l'on considère comme coût le temps ou bien la distance d'un carrefour à un autre. L'idée est donc la suivante : Nous adaptons notre structure de données pour retenir tous les points d'intersection et nous pré-calculons les coûts (invariables), qui les sépareront. Pour chaque route, nous enregistrons à part les liste de nœuds intermédiaires correspondants aux tronçons. Les coordonnées de ces noeuds peuvent en effet être utiles pour redessiner la carte. La structure se servira également d'autres données. Par exemple : le type de transition que l'on a entre deux carrefours (s'il est fréquentable par un piéton, un vélo ou une voiture).

Organisation modulaire du projet

Afin de faciliter le développement et la cohérence de notre projet, nous avons jugé important de bien découper notre programme en différents modules et packages. L'utilisation de foncteurs, notamment, nous a semblé primordiale pour obtenir un style de programmation qui se veut le plus généraliste. Ceci permet de maintenir possible toute évolution potentielle que pourrait subir le programme mais aussi de prévoir l'élaboration de fonctionnalités plus générales voir différentes de celles que nous avons pu concevoir.

Les foncteurs ont été principalement utilisés pour créer les modules suivants :

- *OrientedGraph :*
  Ce module permet l'implémentation d'un graphe orienté à partir de types représentant des noeuds et des transitions.
- *Astar :*
  Ce module offre la possibilité d'appliquer l'algorithme d'A* pour n'importe quel structure de données en définissant une heuristique particulière. Si l'heuristique vaut 0, l'algorithme est alors équivalent à l'algorithme de Dijkstra. Le caractère générique de ces deux derniers modules ont permis notamment de tester leur fiabilité en les testant conjointement à l'aide d'une structure de donnée beaucoup plus simple que celle imposée par le réseau routier Open Street Map.
- *ListZipper :*
  Structure vue en cours permettant de gérer un zipper de liste. Ce module a été principalement utilisé pour gérer efficacement le parcours et l'affichage d'un itinéraire.
- *NearTree :*
  Ce module permet de générer un arbre depuis une structure de donnée modulable dans le but d'implémenter l'algorithme du kd-arbre pour une recherche des voisins.

Utilisation de fonctionnalitées avancées d'Ocaml

Nous avons trouvé utile l'emploi de GADT pour utiliser des types données générales paramétrés par des type précis. Par exemple dans notre structure de carte routière, « `'a cost` » ou « `'a node` » permettant respectivement de gérer des coûts, qu'ils soient en temps ou en distance, et des nœuds d'intersection ou représentant le chemin les séparant.

Comme nous l'avons dit plus haut, nous avons utilisé des zipper de listes. Cette structure était particulièrement efficace pour représenter un itinéraire puisqu'elle permet notamment de savoir d'où l'on vient.

L'utilisation de constructeurs polymorphes nous aide pour paramétrer les fonctions d'entrée des différents foncteurs. On peut notamment s'en apercevoir dans le module fournit à Itinerary.Make.

Nous avons également utilisé des Lazy (à l'intérieur du module de NearestTree). Ces Lazy permettent de construire la structure d'arbre mutuellement récursive dont on ne connait pas l'aboutissement sans les paramètres. Ainsi, la structure peut potentiellement être infinie et l'on peut forcer la recherche du point le plus proche en lui donnant les coordonnées.

Difficultés rencontrés et Solutions apportés

La diversité des informations à l'intérieur d'Open Street Map a constitué la première et principale difficulté au cours de ce projet. Il y a un nombre extrêmement important de données contenues dans ces cartes. Toutes ces informations ne nous sont pas utiles mais, bien que la documentation soit bien fournie, savoir desquelles nous nous servirons et comment les garder en mémoire a été une véritable épreuve, jusqu'a la fin.

Le temps et la charge de travail ont aussi été une difficulté. Nous avions beaucoup d'idées pour ce projet, mais n'avons pas pu toutes les mettre au point. C'est le cas par exemple de l'algorithme du kd-arbre qui a été implémenté, mais n'ayant pas été testé, n'a pas été rajouté au programme. C'est également le cas pour la recherche d'adresses qui était prévue mais n'a pas été implémentée. Nous avions également prévue une option pour faire toutes les opérations de préprocessing à part, mais ceci n'a pas été implémenté.

Nous avons un problème que nous n'avons résolu faute de temps qui est le suivant : le calcul d'un itinéraire tenant compte en tant que premier nœud d'un type particulier de véhicule. Par exemple, il sera impossible d'obtenir un itinéraire si le point le plus proche utilisé se trouve entre deux transitions qui ne sont pas de type voiture et que l'on utilise une voiture.

Fonctionnalités du projet

Makefile

Nous fournissons ce projet avec un Makefile qui utilise ocamlbuild et automatise la compilation.

Un simple make permet de compiler les sources et de créer un exécutable « osm_pathfinder ». Pour les informations concernant la ligne de commande du programme, veuillez vous référer au fichier README fournis à la racine du projet.

Un make doc permet de générer une documentation complète du projet au format html respectant la norme ocamldoc. Cette documentation peut être consultée en ouvrant le fichier doc/html/index.html avec votre navigateur.

Fonctionnalitées du programme

Les fonctionnalités que nous avons choisi d'implémenter sont les suivantes :

- Le choix du type de transport : on peut choisir entre être piéton, vélo ou voiture.
- On peut choisir d'optimiser le trajet en temps ou en distance.
- Un Marshaling est fait à la fin de la première création de carte pour récupérer notre structure. Au second appel sur un même fichier, une vérification est faite qui récupère le fichier marshal pour construire la structure plus rapidement sans passer par une nouvelle étape de parsing.
- On peut sélectionner le point de départ et le point d'arrivé selon deux choix différent : deux coordonnées (Point de départ latitude/longitude, point d'arrivée latitude/longitude).
- Enfin on peut choisir entre effectuer un trajet via le Roadmap, ou via une image que l'on sauvegarde au format jpeg.