TRAVAUX PRATIQUES XXVI

Plus courts chemins

Le but de ce TP est d'implémenter les algorithmes de Floyd-Warshall et Dijkstra sur des graphes pondérés.
Dans tout ce TP, on travaillera en OCaml.

On indice les sommets par $[\![0; n[\![$.

On pose : `type adja_graphe = (int*int) list array` . C'est une représentation de graphes par liste d'adjacence, où au lieu de faire des listes qui stockent uniquement les voisins on fait des listes qui stockent les **paires (voisin, poids de l'arc qui y mène)**.

Ainsi que : `type mat_graphe = int array array` .

1. Choisissez deux exemples de graphes pondérés du cours et traduisez-les en listes d'adjacence.
2. Écrire une fonction qui convertit une liste d'adjacence en matrice d'adjacence. On utilisera soit une `Option`, soit `Int.max_int` pour représenter l'infini. Testez-la.

A Floyd-Warshall

3. Implémentez l'algorithme de Floyd-Warshall.
4. Testez-le.

B Dijkstra

Le dossier fourni contient une implémentation de files de priorité (sans modification de priorité). La documentation est dans le `.mli` . Un `Makefile` est fourni.

On précise que les fichiers `prio.ml(i)` définissent un module `Prio`. Autrement dit, si l'on veut utiliser la fonction `f` de `Prio`, il faut utiliser `Prio.f` .

À l'aide de ce module, utilisez l'algorithme de Dijkstra pour calculer les plus courts chemins depuis un sommet donné dans un graphe.

Comme d'habitude, commencez par essayer de recoder les fonctions par vous-mêmes (n'hésitez pas à réfléchir au brouillon), et ne consultez le cours que si nécessaire pour vous rafraîchir la mémoire.

Conseil : Soyez attentifs à ce que vous manipulez ! Distinguez bien les sommets de leurs priorités : les deux seront représentés par des entiers…

5. Écrire une fonction `distances_depuis` prenant en argument un graphe donné par tableau de listes d'adjacence et un sommet d'origine `s` et qui renvoie le tableau des distances de `s` à tous les sommets du graphe.
6. Adaptez votre fonction pour qu'elle renvoie également un tableau `pred` permettant de reconstituer les plus courts chemins de `s` aux autres sommets du graphe ; c'est à dire que pour tout `v`, `pref[v]` doit contenir le prédecesseur de `v` dans l'arbre de parcours.
   Ce n'est pas si facile que ça. Vous avez grosso modo deux façons de procéder :
   - Ne pas insérer dans la file de priorité uniquement des sommets, mais des paires (sommet, sommet qui l'a inséré avec cette priorité). Ainsi, lorsqu'un sommet sort de la file, on récupère également
   - À chaque fois que vous insérez dans la file de priorité, si la nouvelle priorité est meilleure que l'ancienne, mettez à jour `pred`.
     Cela demande donc d'avoir mémorisé la meilleure priorité actuelle.
7. Testez votre fonction !

C Bonus

8. Jour 17 AOC 2023.