LANGAGES DE PROGRAMMATION
IFT-3000
HIVER 2016

Travail pratique 2 - TRAVAIL EN ÉQUIPE (1 ou 2)
Pondération de la note finale : 14.5%

À remettre, avant minuit, le samedi 30 avril 2016

1 Énoncé

Ce travail pratique consiste à implanter différentes méthodes pour la gestion d'un réseau de transport permettant, à un utilisateur quelconque, de se renseigner facilement sur le réseau de transport en commun dans la ville de Québec. Il constitue une suite logique du travail pratique 1 ; on vous invite donc à considérer aussi l'énoncé du Tp1 qui décrit clairement les données ouvertes du RTC.

1.1 Données ouvertes

Nous proposons d'utiliser les mêmes données ouvertes du Réseau de Transport de la Capitale (RTC), utilisées dans le travail pratique 1 :

`http://www.rtcquebec.ca/Portals/0/Admin/DonneesOuvertes/googletransit.zip`

1.2 Données utilisées dans ce travail pratique

Relativement au travail pratique 1, les principales données manipulées dans ce travail pratique sont représentées sous forme d'objets, instances de certaines classes, plutôt que sous forme de valeurs de type «enregistrement» ; aussi, pour des raisons de performance, nous utilisons des tables de hachage.

Par ailleurs, outre les différentes classes définies dans le cadre de ce Tp, une nouvelle structure de données apparait afin de représenter un graphe de stations et pouvoir implanter des traitements de parcours de graphes, comme celui permettant de calculer le plus court chemin étant données une date, une heure, une position géographique de départ et une position géographique d'arrivée, et en considérant évidemment les données du RTC (différentes lignes disponibles, avec leurs différents voyages, etc.). À cette fin, nous utilisons essentiellement le module Graph.Pack.Digraph, du paquetage «ocamlgraph» (`http://ocamlgraph.lri.fr`), qui permet de manipuler des graphes orientés ; ainsi que le foncteur Graph.Path.BellmanFord qui permet de vous retourner un module (BF) comprenant un algorithme vous permettant de calculer des stations atteignables en considérant des poids sur les arêtes (le code pour la génération de ce module vous est fourni).

Au sujet de l'unicité de l'identifiant de ligne (dans le fichier «routes.txt») Dans le cadre du Tp1, nous avions fait la remarque suivante : «Veuillez, noter qu'il y a deux entrées dans le fichier pour chaque ligne d'autobus, ce qui implique qu'un numéro de ligne est associé à deux identifiants différents». Après investigation auprès du RTC, voici la réponse que nous avons obtenue :

> «4 périodes applicatives au RTC (hiver, printemps, été, automne).
> Il peut arriver que le fichier gtfs contient 2 périodes, les parcours seront alors doublés avec des id différents.
> Les horaires changent beaucoup entre les périodes.
> Si on regarde le fichier gtfs présentement disponible, on peut voir que les parcours ne sont pas doublés, car le fichier contient seulement la période printemps.»

Par conséquent, le problème n'est plus présent dans la présente version des données disponibles sur le site du RTC. Ceci dit, et pour ne prendre aucune chance, à l'instar du Tp1, nous vous invitons à utiliser la version des données que nous aurions utilisée dans le cadre du corrigé du Tp2 ; en effet, dans ce corrigé, nous avons considéré qu'il n'y a qu'un seul identifiant de ligne pour chaque ligne.

2 Éléments fournis

Le code qui vous est fourni comprend 1 répertoire et 3 fichiers :

1. «googletransit» : répertoire qui comprend la version des données du RTC que nous avons utilisée pour le corrigé ; nous vous invitons à l'utiliser ;
2. «utiles.ml» : fichier qui comprend 2 modules utiles pour la réalisation du TP :
   (a) Utiles : comprend des fonctions utiles pour réaliser les méthodes à implanter dans votre TP.
   (b) Date : comprend des fonctions de manipulation d'heures et de dates.
3. «tp2.ml» : comprend toutes les classes définies dans le cadre de ce travail, y compris évidemment la principale classe, «gestionnaireReseau», qui comprend certaines méthodes que vous devez compléter.
4. «testeur.ml» : comprend un ensemble de tests des différentes méthodes à implanter dans ce TP.

2.1 Documentation des méthodes à compléter

À l'instar du Tp1, toutes les fonctions et méthodes présentes dans le Tp2 sont documentées ; pour les méthodes à compléter, toute l'information nécessaire est disponible (description, pré-condition et post-condition).

Par ailleurs, un des objectifs du Tp2 est de vous inciter à lire vous-même le code déjà implanté en vue de compléter la partie que vous devez implanter ; ainsi, l'évaluation de l'aspect «orienté objet» du cours ne se fera pas en vous demander de concevoir vous-même les classes, mais plutôt en vous incitant à comprendre les choix de conception que nous avons établis dans le cadre de ce travail ; à vous d'imaginer des extensions éventuelles de ce type de projet, notamment par l'entremise de l'héritage et la définition de sous-classes des classes existantes dans une optique de spécialisation des fonctionnalités déjà présentes (implantées).

Pour terminer, vous remarquerez que plusieurs des fonctions que vous avez implantées dans le cadre de votre travail pratique 1 figurent, sous forme de méthodes, dans la classe «gestionnaireReseau» ; ces méthodes sont là pour vous aider à faire la transition entre l'implantation du gestionnaire en fonctionnel (modules et fonctions) et son équivalent en orienté objets.

2.2 Démarche à suivre

Voici la démarche que je vous suggère de suivre :

- Avant tout, installer le module «ocamlgraph» :

```
(dans un terminal, tapez ces 2 commandes)
opam install ocamlgraph
sudo ln -s ~/.opam/system/lib/ocamlgraph/ /usr/lib/ocaml/ocamlgraph
```

- ouvrer, à l'aide d'un éditeur (Emacs par exemple, Eclipse ou Sublime [1]), les différents fichiers .ml du TP ; lancer l'interpréteur Ocaml ;
- au niveau de l'interpréteur, charger («**#use** "tp2.ml" ; ;») le fichier «tp2.ml» ; ceci entraine aussi le chargement du fichier «utiles.ml».
  Ainsi, à cette étape, toutes les classes et modules du TP sont disponibles au niveau de l'interpréteur ; il est alors possible :
  - d'ouvrir le module «Utiles» («**open** Utiles ; ;») et de tester les fonctions qui y sont définies ;
  - de créer des instances des classes définies et de tester leurs méthodes ; notamment la classe «gestionnaireReseau» qui, via son «initializer», chargera toutes les données du RTC (celles-ci étant considérées, par défaut, stockées à cet emplacement : «/home/etudiant/workspace/tp2/googletransit/»).
- de compléter progressivement les méthodes à implanter.

Version compilée versus interprétée Notez que dans le cadre de ce travail, nous n'accepterons qu'une version interprétée de votre code («tp2.ml»). Autrement dit, avant de remettre votre code, vous devez vous assurer que le chargement («**#use** "tp2.ml" ; ;») du fichier «tp2.ml» se déroule correctement ; et idéalement, que le chargement du fichier «testeur.ml» se déroule aussi correctement et que l'appel à la fonction «test» fournit les résultats attendus.

1. Dans le Wiki qui comprend les étapes d'installation des outils utilisés dans ce cours, nous avons ajouté un lien vers une page qu'a conçue un étudiant du cours (merci !) au sujet de l'installation et de l'utilisation de l'éditeur Sublime.

3 Signature

La signature de la classe «gestionnaireReseau» est :

```
class gestionnaireReseau :
  ?rep:string ->
  unit ->
  object
    val graphe_stations : G.t
    val lignes : (string, ligne) H.t
    val noeuds_stations : (G.V.label, G.V.t) H.t
    val stations : (G.V.label, station) H.t
    val voyages : (string, voyage) H.t
    val voyages_par_date : (int, string list) H.t
    method get_graphe : G.t
    method get_noeuds : (G.V.label, G.V.t) H.t
    method ligne_passe_par_station :
      ?date:int option -> string -> G.V.label -> bool
    method lister_lignes : string list
    method lister_lignes_par_type :
      ?types:type_ligne list -> unit -> (type_ligne * string list) list
    method lister_lignes_passantes : G.V.label -> string list
    method lister_stations : G.V.label list
    method lister_stations_sur_itineraire :
      ?date:int option -> string -> (string * int list) list
    method maj_etiquette_arete :
      ?date:int -> ?heure:int -> G.V.label -> G.V.label -> unit
    method maj_toutes_aretes_reseau : ?date:int -> ?heure:int -> unit -> unit
    method nb_aretes : int
    method nb_lignes : int
    method nb_stations : int
    method paires_stations_possibles :
      ?rmax:float ->
      coordonnees ->
      coordonnees -> (G.V.label * G.V.label * float * float) list
    method plans_possibles_pour_chemin :
      ?date:int ->
      ?heure:int ->
      G.V.label list -> (string * int * int * G.V.label list) list list
    method plus_court_chemin : G.V.label -> G.V.label -> G.V.label list
    method print_stats : unit
    method prochain_arret :
      ?date:int -> ?heure:int -> direction -> string -> G.V.label -> arret
    method prochaines_lignes_entre :
      ?date:int ->
      ?heure:int -> G.V.label -> G.V.label -> (string * int * int) list
    method trouver_horaire :
      ?date:int ->
      ?heure:int -> direction -> string -> G.V.label -> string list
    method trouver_stations_environnantes :
      coordonnees -> float -> (G.V.label * float) list
    method trouver_trajet_optimal :
      ?date:int ->
      ?heure:int ->
      ?rmax:float ->
      coordonnees ->
      coordonnees ->
      (float * float * (string * int * int * G.V.label list) list) list
    method trouver_voyages_par_date : ?date:int -> unit -> string list
    method trouver_voyages_sur_la_ligne :
      ?date:int option -> string -> string list
  end
```

4 À remettre

Il faut remettre un fichier .zip contenant uniquement le fichier «tp2.ml» complété ; ne remettez pas d'autres fichiers ou répertoire comprenant les données ouvertes du RTC.

Aussi, le code doit être clair et bien indenté [2].

5 Remarques importantes

Travail en équipe : On cous encourage fortement à travailler en équipe. Aussi, notez qu'il est impératif que vous formiez votre équipe sur ENA, même si vous travaillerez finalement seul.

Plagiat : Tel que décrit dans le plan de cours, le plagiat est interdit. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toutes circonstances. Tous les cas seront référés à la direction de la Faculté.

Travail remis en retard : Tout travail remis en retard se verra pénalisé de 25% par jour de retard. Chaque journée de retard débute dès la limite de remise dépassée (dès la première minute). Un retard excédant 2 jours (48h) provoquera le rejet du travail pour la correction et la note de 0 pour ce travail. La remise doit se faire par la boite de dépôt du TP2 dans la section «Évaluation et résultats».

Barème [3] **:** Au niveau des fonctions à implanter, le barème est précisé dans le fichier "tp2.ml". Notez cependant que :
- (-10pts), si votre code ne compile pas (provoque une erreur/exception lors du chargement du fichier "tp2.ml" dans l'interpréteur, i.e. «**#use** "tp2.ml" ; ;») ;
- (-5pts), si votre code n'est pas clair et bien indenté.

Bon travail.

2. Suggestion : http://www.cs.princeton.edu/~dpw/courses/cos326-12/style.php.
3. Vous l'aurez compris, dans le cadre de ce Tp, vous pouvez utiliser et combiner les paradigmes de programmation de votre choix.