LANGAGES DE PROGRAMMATION
IFT 3000 NRC 54959
ÉTÉ 2024

Travail pratique 2 (individuel)

À remettre sur monportail avant 17h00 le samedi 20 juillet 2024

1 Énoncé

Nous vous demandons dans ce travail pratique d'implanter **le réseau d'une compagnie aérienne** en utilisant le paradigme orienté objet. En effet, il s'agit de reprendre le travail demandé pour le TP1, mais en ajoutant d'autres fonctionnalités.

2 Travail à faire

Le TP consiste à compléter le fichier tp2.ml" qui respecter les spécifications dans le fichier tp2.mli. **Il est à noter qu'il ne faut pas modifier la signature présente dans ce fichier**. Dans cette signature, il existe quatre classes (ville, trajet, reseau et reseau_aerien), mais votre travail consiste essentiellement à implanter les 17 méthodes de la classe **reseau_aerien** qui hérite de la classe **reseau** :

1. method ajouter_ville : string -> unit **(3 points)**
2. method ville_existe : string -> bool **(2 points)**
3. method retourner_ville : string -> ville **(3 points)**
4. method ajouter_trajet : string -> string -> float -> float -> unit **(5 points)**
5. method trajet_existe : string -> string -> bool **(3 points)**
6. method retourner_trajet : string -> string -> trajet **(3 points)**
7. method supprimer_ville : string -> unit **(4 points)**
8. method supprimer_trajet : string -> string -> unit **(5 points)**
9. method ajouter_liste_villes : string list -> unit **(3 points)**
10. method ajouter_liste_trajets : (string * string * float * float) list -> unit **(3 points)**
11. method trier_villes : unit **(4 points)**
12. method trier_trajets : unit **(4 points)**
13. method fermeture_transitive : 'a **(10 points)**

14. method trouver_successeurs : ville -> ville list -> ville list **(7 points)**
15. method chemin_existe : string -> string -> bool **(3 points)**
16. method trouver_court_chemin : string -> string -> bool -> (string list * float) **(20 points)**
17. method afficher_reseau_aerien : unit **(6 points)**

Vous connaissez normalement ce que fait la majorité de ces méthodes (consulter l'énoncé du TP1 et la documentation du TP2). Cependant, voici des explications supplémentaires surtout pour les nouvelles méthodes à implanter :

- **trier_villes : unit** est une méthode qui fait en sorte de trier en ordre croissant les villes qui sont contenues dans la liste de villes du réseau. Cette fonction retourne unit car elle fait la mise à jour du réseau. Nous vous conseillons d'utiliser la fonction **List.sort** en implantant une fonction **comparer_villes** qui prend deux villes et qui retourne 0 si les deux villes sont égales, 1 si la première ville est plus grande (en ordre alphabétique) que la deuxième et -1 sinon.
- **trier_trajets : unit** est une méthode qui fait le tri en ordre croissant des trajets. Nous vous conseillons d'utiliser également la fonction **List.sort** en implantant une fonction **comparer_trajets** qui prend deux trajets et qui retourne 0 si la concaténation des noms des villes de départ et d'arrivée du premier trajet est égale à celle du deuxième trajet. Sinon, il faut retourner 1 ou -1 comme dans la fonction **comparer_villes**.
- **fermeture_transitive :** 'a**.** Vous connaissez normalement l'utilité de cette méthode. Elle retourne un type 'a qui a le même type que l'objet courant (Regarder la classe reseau_aerien dans le fichier "tp2.mli") représentant une copie d'un reseau_aerien contenant tous les chemins possibles. On vous donne ici une méthode itérative pour l'implanter, mais libre à vous d'opter pour une façon différente comme la méthode récursive du TP1 :
    a. Créer une copie du réseau en utilisant self. Nous vous conseillons d'utiliser la fonction **Oo.copy** (< .. > as 'a) -> 'a).
    b. Utiliser trois boucles imbriquées comme ceci :

        De k = 0 à nb_villes -1 faire
            De j = 0 à nb_villes – 1 faire
                De i = 0 à nb-villes -1 faire
                    Si le trajet n'existe pas entre i et j alors
                        Si les trajets existent entre (i, k) et (k, j) alors
                            Ajouter le trajet entre i et j en faisant la somme des couts en dollars et la somme des durées du vol en heures. Il ne faut pas ajouter les chemins qui ont une ville de départ et une ville d'arrivée identiques.

    c. Retourner la copie du réseau modifié.

- **trouver_successeurs : ville -> ville list -> ville list** est une fonction qui prend une ville **v** ainsi qu'une liste de villes **lv** et retourne la liste des successeurs de **v** se trouvant dans **lv**. Vous pouvez utiliser cette méthode pour implanter **trouver_court_chemin**.
- **chemin_existe : string -> string -> bool** est une fonction qui retourne *true* si un chemin existe entre une ville de départ et une ville d'arrivée, *false* sinon. Nous vous demandons d'utiliser **fermeture_transitive** et **trajet_existe** pour implanter cette méthode. Vous pouvez également utiliser cette méthode pour implanter **trouver_court_chemin**.
- **trouver_court_chemin : string -> string -> bool -> (string list * float)** est une fonction qui prend le nom d'une ville de départ **vd**, le nom d'une ville d'arrivée **va** et un booléen (égale soit à *true* si on veut utiliser les couts en dollars, soit à *false* si on veut utiliser la durée du vol en heures pour calculer le chemin) et retourne une paire contenant une liste de chaînes de caractères représentant le plus court chemin entre **va** et **vb** et un float représentant la somme des pondérations (couts ou durées) du chemin en question. Nous vous suggérons d'ailleurs l'utilisation de l'algorithme de **Dijkstra** vu dans le cours IFT-2008 ou GLO-2100.
- **afficher_reseau_aerien : unit** est une fonction qui affiche la liste des villes ainsi que la liste des trajets et retourne unit. Voici un exemple de comment l'affichage doit se faire :

Reseau Aerien : AirCanada
Nombre des villes : 19
Liste des villes :
Auckland, Bogota, Bruxelles, Bujumbura, Chicago, Chicoutimi, Detroit, Iqaluit, La_Havane, Montreal, New_York, Ottawa, Paris, Quebec, Sherbrouke, Singapour, Tokyo, Toronto, Vancouver,
Nombre des trajets : 20
Liste des trajets :
Bruxelles -> Bujumbura : cout en dollars = 973.05 , duree du vol = 5.27
Bujumbura -> Bruxelles : cout en dollars = 937.46 , duree du vol = 7.26
Chicago -> Vancouver : cout en dollars = 1521.44 , duree du vol = 4.78
Montreal -> Bruxelles : cout en dollars = 823.96 , duree du vol = 6.87
Montreal -> Detroit : cout en dollars = 699.74 , duree du vol = 1.25
Montreal -> New_York : cout en dollars = 402.78 , duree du vol = 1.91
Montreal -> Ottawa : cout en dollars = 192.46 , duree du vol = 0.66
Montreal -> Paris : cout en dollars = 723.96 , duree du vol = 6.27
…
Toronto -> Bogota : cout en dollars = 902.15 , duree du vol = 5.42
Vancouver -> Auckland : cout en dollars = 2351.71 , duree du vol = 13.15
Vancouver -> Tokyo : cout en dollars = 1276.42 , duree du vol = 8.65
Vancouver -> Toronto : cout en dollars = 621.46 , duree du vol = 3.87

3 Démarche à suivre

Pour réaliser ce travail, vous disposez d'une page Github qui comprend les éléments suivants :

1. Un fichier README.md, affiché automatiquement lors de l'ouverture de la page web, et qui résume les étapes à suivre pour réaliser le Tp ;
2. Un dossier « lib » comprenant les fichiers du Tp :
    a. « tp2.mli » : définit la spécification des éléments du Tp, soit la signature des classes à programmer; notez que ce fichier comprend toute la documentation nécessaire à la réalisation du Tp (vous n'avez pas à modifier ce fichier) ;
    b. « tp2.ml » : correspond à la partie « implantation » du Tp; c'est à ce niveau que vous devez programmer les différentes méthodes du Tp; notez que vous ne devez remettre que ce fichier (tp2.ml), une fois complété ;
    c. « dune » : un fichier utilisé par l'outil dune (vous n'avez pas à modifier ce fichier) ;
3. Un dossier « test » comprenant un testeur pour le Tp :
    a. « testeur.ml » : comprend la liste des cas de tests définis pour les différentes méthodes à implanter dans le cadre du Tp ;
    b. « dune » : un fichier utilisé par l'outil dune (vous n'avez pas à modifier ce fichier) ;
4. Un dossier « docs » comprenant la documentation générée automatiquement à partir des commentaires qui se trouvent dans le fichiers « tp2.mli »;
5. Un dossier « .vscode » comprenant un fichier json permettant de s'assurer que vous chargerez la bonne version de l'interpréteur sous VSCode ;
6. Un ensemble d'autres fichiers de configuration ; pour votre information :
    - « .gitignore » est utilisé par Git ;
    - « .ocamlformat » est utilisé par l'outil ocamlformat et permet de formater votre code ; il comprend la version qui est censée être installée dans votre système ;
    - « dune-project » est utilisé par l'outil dune ;
    - « .ocamlinit » comprend des commandes qui seront automatiquement exécutées au lancement d'un interpréteur (utop ou ocaml) ;
    - Des fichiers d'extension « .opam » nécessaires pour la production de la documentation.
7. Un fichier « NoteTp2-IFT3000.xlsx » contenant le barème du Tp et que vous devez remettre.

4 À remettre

Vous devez rendre un fichier .zip comportant **uniquement** les fichiers suivants.

- Le fichier « tp2.ml » complété.
- Le fichier « NoteTp2-IFT3000.xlsx (un fichier Excel contenant le barème de correction. Il faut juste ajouter votre nom et matricule sur la première ligne).

Le nom du .zip doit respecter le format suivant : TP2-Matricule.zip. Nous vous rappelons qu'il est important de faire la remise par voie électronique uniquement en vous connectant à: http://monportail.ulaval.ca/ (aucun travail envoyé par courriel n'est accepté). Il est toujours possible de faire plusieurs remises pour le même travail. Vous pouvez donc remettre votre travail plus tôt afin d'être sûr qu'il a été remis en temps, et remettre par la suite une version corrigée avant l'heure limite de remise. De plus, il est de votre responsabilité de vérifier après la remise que vous nous avez envoyé les bons fichiers (**non vides et non corrompus**), sinon vous pouvez avoir un zéro.

Attention ! Il est **formellement interdit de partager ou publier** la solution de votre TP sur le Web avant ou après la remise du TP, car c'est une source évidente de plagiat et vous risquez donc d'être pénalisé. De plus, **tout travail remis en retard se verra pénalisé de -25% de la note**. Le retard débute dès la limite de remise dépassée (dès la première minute). Un retard excédant une journée provoquera le rejet du travail pour la correction et la note de 0 pour ce travail.

Il est à noter que **12 points** sont donnés pour le respect et la qualité des biens livrables ainsi que pour la structure générale du code (commentaires, indentation, compilation sans warnings, etc.). De plus, votre code doit absolument pouvoir être exécuté sans erreurs sur la machine virtuelle du cours. Si vous ne testez pas suffisamment votre travail, il risque de provoquer des erreurs à l'exécution lors de la correction. La moindre erreur d'exécution rend la correction extrêmement difficile et vous en serez donc fortement pénalisés.

IA générative : Tel que décrit dans le plan de cours, vous n'êtes pas autorisés à utiliser l'IA générative pour la réalisation de votre travail pratique. Conformément au Règlement disciplinaire à l'intention des étudiants et étudiantes de l'Université Laval, le fait d'obtenir une aide non autorisée pour réaliser une évaluation (travail pratique, en ce qui nous concerne) est considéré comme une infraction relative aux études. Dans le cadre de ce cours, l'utilisation de l'IA générative est considérée comme une aide non autorisée. Cette infraction pourrait mener à l'application des sanctions prévues au Règlement disciplinaire. Notez que nous nous gardons le droit de prendre au hasard des travaux et de poser des questions pour nous assurer que le contenu du travail en question est bien le fruit du travail de l'étudiant concerné.

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.

Bon travail !