LANGAGES DE PROGRAMMATION
IFT 3000 NRC 54959
ÉTÉ 2024

Travail pratique 1 (individuel)

À remettre, par Intranet (pixel) avant 17h00 le vendredi 14 juin 2024

1 Énoncé

Nous vous demandons dans ce travail pratique d'implanter **le réseau d'une compagnie aérienne** en utilisant les graphes et les listes. En effet, vous aurez à faire des manipulations de base sur un graphe qui est implanté en utilisant les listes et les enregistrements.

2 Rappel

Les graphes sont des outils largement utilisés pour modéliser, analyser et vérifier des systèmes informatiques. Un graphe G est un couple G = (S,E) où S est un ensemble de sommets et E est un ensemble d'arêtes reliant deux sommets s_1 et s_2. Les arêtes peuvent être orientées (arcs). Dans ce cas, l'arc (s1, s2) est différent de l'arc (s2, s1). Dans ce TP, nous considérons seulement les graphes orientés, c'est-à-dire les graphes où les arêtes sont orientées.

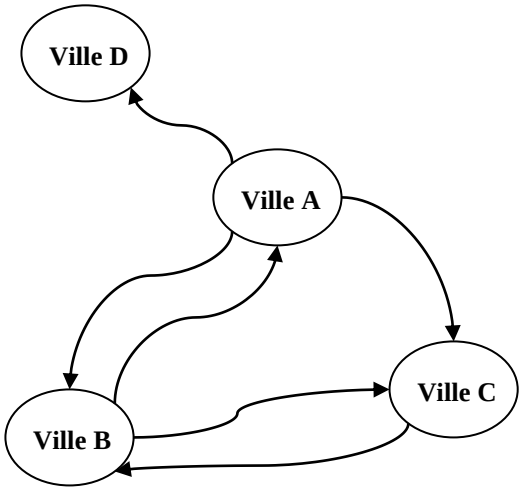
Un graphe orienté peut être représenté par un ensemble des sommets et l'ensemble des successeurs associés à chaque sommet. Si l'arc $(s_1, s_2) \in G$, alors dans le graphe G, s_2 est successeur de s_1 et s_1 est prédécesseur de s_2. Ainsi, le graphe orienté peut être défini: G = (S, succ) où succ est une fonction qui appliqué à un sommet s, donne l'ensemble des successeurs de s.

Un chemin dans un graphe est une liste de sommets $(s_1, s_2, \ldots, s_n)$ tels que chaque (s_i, s_{i+1}) est un arc dans le graphe pour $1 \leq i \leq n - 1$. Ce chemin est de longueur n. Un chemin de longueur 0, ne comporte qu'un seul sommet.

La fermeture transitive d'un graphe G est un graphe où la liste des successeurs comprend les successeurs directs et indirects. De ce fait, le calcul de la fermeture transitive permet entre autres de répondre aux questions concernant l'existence de chemins entre deux sommets d'un graphe G.

3 Réseau aérien

Supposons le réseau aérien suivant contenant un ensemble de villes (sommets du graphe) et des trajets d'avions reliant ces villes (arcs du graphe):



Ce réseau (ou graphe) est implanté en utilisant une liste d'enregistrements. Chaque enregistrement représente une ville (ou sommet). Les enregistrements contiennent deux champs: un champ représentant le nom de la ville et un champ représentant une liste de villes (liste_voisins). Ces villes (ou successeurs) peuvent être accédées par des trajets d'avions (ou arcs) à partir de la ville contenue dans le premier champ de l'enregistrement.

Dans notre exemple. Le réseau peut être représenté de la façon suivante :

```
let r =  [{nom = "Ville A"; liste_voisins = ["Ville B";"Ville C";"Ville D"]};
         {nom = "Ville B"; liste_voisins = ["Ville A"; "Ville C"]};
         {nom = "Ville C"; liste_voisins = ["Ville B"] };
         {nom = "Ville D"; liste_voisins = [] }];;
```

4 Éléments fournis

Le TP consiste à compléter le fichier tp1.ml" qui respecter les spécifications dans le fichier tp1.mli (**Il est à noter qu'il ne faut pas modifier la signature présente dans ce fichier)**.

Les fonctions fournies sont les suivantes (vous pouvez en ajouter au besoin):

A. Fonctions utilitaires de manipulation de listes:

1. **appartient e l : 'a -> 'a list -> bool** est une fonction qui prend un élément e de type polymorphe et une liste d'éléments de même type et rend un booléen qui vaut *true* si l'élément e est présent dans la liste et *false* sinon.
2. **Enlever e l : 'a -> 'a list -> 'a list** est une fonction qui prend un élément e de type polymorphe et une liste d'éléments de même type et qui rend une liste où e a été supprimé partout s'il existe.
3. **remplacer e e' l : 'a -> 'a -> 'a list -> 'a list** est une fonction qui prend deux éléments e et e' d'un même type polymorphe et une liste d'éléments également de même type et qui rend une liste où e a été remplacé partout par e'.
4. **union_liste l1 l2 : 'a list -> 'a list -> 'a list** est une fonction récursive qui prend deux listes d'éléments de même type polymorphe et rend une liste ayant comme éléments l'union des éléments des deux listes. Pour que cette fonction ait un sens pour les ensembles, elle enlève les répétitions d'éléments.

B. Fonctions de manipulation de réseau (ou graphe):

1. **initialiser_reseau: unit -> reseau** est une fonction qui permet de retourner un réseau vide.
2. **retourner_nom_ville v : ville -> nom_ville** est une fonction qui permet de retourner le nom d'une ville à partir de l'enregistrement v représentant une ville.
3. **retourner_liste_voisins1 v : ville -> nom_ville list** est une fonction qui permet de retourner la liste des successeurs (ou voisins) d'une ville à partir de l'enregistrement v représentant une ville.
4. **retourner_ville nv r : nom_ville -> reseau -> ville** est une fonction qui permet de retourner l'enregistrement correspondant à un nom de ville nv se trouvant dans le réseau r. Cette fonction retourne un message d'erreur si le réseau est vide ou si le réseau ne contient pas la ville recherchée.

5. **retourner_liste_voisins2 nv r : nom_ville -> reseau -> nom_ville list** est une fonction qui permet de retourner la liste des successeurs (ou voisins) d'une ville à partir de son nom nv et du réseau r à laquelle elle appartient. Cette fonction retourne un message d'erreur si le réseau est vide ou si le réseau ne contient pas la ville dont on cherche ses voisins.
6. **ville_existe nv r : nom_ville -> reseau -> bool** est une fonction qui retourne *true* si une ville dont le nom est nv se trouve dans un réseau r. Si la ville n'existe pas dans le réseau ou si le réseau est vide, cette fonction retourne *false*.

5 Fonctions à implémenter

Votre travail consiste à implémenter les 9 fonctions suivantes dont la spécification de typage est donnée dans le fichier tp1.mli:

1. **ajouter_ville nv r : nom_ville -> reseau -> reseau** est une fonction qui prend le nom d'une ville nv et un réseau r et qui rend le réseau mis à jour contenant la nouvelle ville. Cette fonction doit retourner un réseau inchangé si la ville est déjà dans le réseau. De ce fait, on ne rajoute pas de doublons dans le réseau. **(7 points)**.
2. **ajouter_liste_villes lv r : nom_ville list -> reseau -> reseau** est une fonction qui prend une liste de noms de villes lv et un réseau r et retourne le réseau mis à jour dans lequel toutes les villes ont été ajoutées dans ce réseau **(7 points)**.
3. **supprimer_ville nv r : nom_ville -> reseau -> reseau** est une fonction qui prend le nom d'une ville nv et un réseau r et qui rend le réseau mis à jour en supprimant la ville qui n'est plus desservie par la compagnie aérienne, ainsi que toutes les références à cette ville dans la liste des successeurs de toutes les autres villes. Cette fonction retourne un message d'erreur si le réseau est vide ou si le réseau ne contient pas la ville à supprimer **(8 points)**.
4. **ajouter_trajet (nv1,nv2): trajet -> reseau -> reseau** est une fonction qui prend un trajet entre deux villes (nv1, nv2) et un réseau r et qui rend comme résultat le réseau mis à jour dans lequel le nom de la ville nv2 a été rajoutée à la liste des successeurs de la ville dont le nom est nv1. Cette fonction retourne un message d'erreur si le réseau est vide ou si le réseau ne contient pas une des villes formant le trajet à ajouter. Cependant, elle doit retourner un réseau inchangé si le trajet existe déjà dans le réseau **(12 points)**.
5. **ajouter_liste_trajets lt r : trajet list -> reseau -> reseau** est une fonction qui prend une liste de trajets lt et un réseau r et retourne le réseau mis à jour dans lequel tous les trajets ont été ajoutés dans ce réseau **(7 points)**.
6. **supprimer_trajet (nv1,nv2) r : trajet -> reseau -> reseau** est une fonction qui prend un trajet entre deux villes (nv1, nv2) et un réseau r et qui rend comme résultat le réseau mis à jour dans

lequel le nom de la ville nv2 est supprimée de la liste des successeurs de la ville dont le nom est nv1. En fait, ceci veut dire que la compagnie aérienne n'offre plus ce trajet. Cette fonction retourne un message d'erreur si le réseau est vide ou si le réseau ne contient pas une des villes formant le trajet à supprimer **(12 points)**.

7. **trouver_chemins r : reseau -> reseau** est une fonction qui prend un réseau r et qui rend comme résultat le réseau de la fermeture transitive de r **(20 points)**. Elle permettra par exemple à une agence de voyages de connaître tous les chemins que la compagnie aérienne peut offrir à sa clientèle en utilisant des transits entre villes. Il est à noter que cette fonction retourne un message d'erreur si le réseau est vide. Nous vous donnons ici une façon de l'implanter, mais libre à vous d'opter pour une autre façon étant donné qu'il existe toujours plusieurs solutions à un même problème. Nous vous suggérons par contre d'opter pour la façon récursive suivante:

    a) Pour chaque ville du reseau r, on construit un nouveau réseau r', où on rajoute à la liste lv des successeurs de chaque ville v, les successeurs de chacun des éléments de la liste lv. Si le réseau r' ainsi obtenu est le même que r on s'arrête sinon on applique le même traitement à r' de façon récursive.
    b) Pour pouvoir réaliser ce traitement, vous aurez besoin d'un booléen pour savoir s'il y a eu des changements suite à la modification des listes de successeurs. De ce fait, vous pouvez implanter la fonction **trouver_chemins** en utilisant une fonction utilitaire **fermeture_transitive** qui a comme type : **ville list-> bool -> ville list**.
    c) Dans l'exemple du réseau donné précédemment, vous aurez normalement le retour du réseau suivant: rf = [{nom = "Ville A"; liste_voisins = ["Ville B";"Ville C";"Ville D";"Ville A"]}; {nom = "Ville B"; liste_voisins = ["Ville A"; "Ville C"; "Ville D"; "Ville B"]}; {nom = "Ville C"; liste_voisins = ["Ville B";"Ville A";"Ville D"; "Ville C"] }; {nom = "Ville D"; liste_acces = [] }.

8. **afficher_villes r : reseau -> unit** est une fonction qui prend un réseau r et affiche tous les noms de villes (sur la même ligne) appartenant à ce réseau. Cette fonction affiche le message "le réseau est vide" si effectivement le réseau est vide **(7 points).**
9. **afficher_reseau r : reseau -> unit** est une fonction qui prend un réseau r et affiche, en plus des noms des villes appartenant à ce réseau, les successeurs de chaque ville. Veuillez consulter la fin du fichier "testeur.ml" pour savoir le format de sortie de cette fonction **(8 points)**.

6 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. « tp1.mli » : définit la spécification des éléments du Tp, soit la signature des fonctions 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. « tp1.ml » : correspond à la partie « implantation » du Tp; c'est à ce niveau que vous devez programmer les différentes fonctions du Tp; notez que vous ne devez remettre que ce fichier (tp1.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 fonctions 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 « tp1.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 « NoteTp1.xlsx » contenant le barème du Tp et que vous devez remettre.

7 À remettre

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

- Le fichier « tp1.ml » complété.
- Le fichier « NoteTp1.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 : TP1-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 !