Programmation fonctionnelle, projet 2017

Dates et principe

Cette page peut être mise à jour, avec informations complémentaires, précisions, *questions bonus*, etc. Pensez à y revenir souvent.

Projet à rendre pour le vendredi **12/01/2017** à **23h59**, aucun retard ne sera toléré.
Des soutenances pourront être organisées ensuite.

Lire tout le sujet (tout ? tout).

Un rendu de projet comprend :

- Un rapport précisant et justifiant vos choix (de structures, etc.), les problèmes techniques qui se posent et les solutions trouvées ; il donne en conclusion les limites de votre programme. Le rapport sera de préférence composé avec LaTeX. Le soin apporté à la grammaire et à l'orthographe est largement pris en compte.
- Un manuel d'utilisation, même minimal.
- Un code *abondamment* commenté ; la première partie des commentaires comportera systématiquement les lignes :
    1. `@requires` décrivant les pré-conditions : c'est-à-dire conditions sur les paramètres pour une bonne utilisation (**pas de typage ici**),
    2. `@ensures` décrivant la propriété vraie à la sortie de la fonction lorsque les pré-conditions sont respectées, le cas échéant avec mention des comportements en cas de succès et en cas d'échec,
    3. `@raises` énumérant les exceptions éventuellement levées (et précisant dans quel(s) cas elles le sont).

  On pourra préciser des informations additionnelles si des techniques particulières méritent d'être mentionnées.

  Le code doit enfin **compiler** sans erreur (évidemment) et sans warning.

Avez-vous lu tout le sujet ?

Protocole de dépôt

Vous devez rendre :

- votre rapport au format `pdf`
- vos fichiers de code.
- vos tests.

rassemblés dans une archive tar gzippée identifiée comme *votre_prénom_votre_nom*`.tgz`.
La commande devrait ressembler à :
```
tar cvfz randolph_carter.tgz rapport.pdf fichiers.ml autres_truc_éventuels...
```

Lisez le man et testez le contenu de votre archive. Une commande comme par exemple :
```
tar tvf randolph_carter.tgz
```
doit lister les fichiers et donner leur taille.
Une archive qui ne contient pas les fichiers demandés ne sera pas excusable. Une archive qui n'est pas au bon format ne sera pas considérée.

Procédure de dépôt
Vous devez enregistrer votre archive tar dans le dépôt dédié au cours IPF (ipf-s3-projet-2017) en vous connectant à `http://exam.ensiie.fr`. Ce dépôt sera ouvert jusqu'au 12 janvier inclus.

Contexte

Le but de ce projet est de mettre en place un planificateur de déplacements dans une base martienne..

Une base martienne est composée de modules tous suffisamment spacieux pour contenir la totalité de la colonie. Le passage entre deux modules se fera via un système de tunnels de longueurs variables et ne permettant chacun que le passage d'une personne à la fois. Bien entendu, nous désirons que le système s'adapte à tout plan de base que nous lui soumettrons.

Notre colonie se développera en trois étapes :

1. dans un premier temps, pour des problèmes de sécurité, notre colonie sera réduite à un seul individu,
2. dans un second temps, le développement aidant, de nombreux individus peupleront notre colonie mais le système aura la simple tâche de réguler leurs déplacements.
3. enfin, la capacité de calcul aidant, le système devra permettre la planification totale des déplacements.

Ne remplir que la première phase de ce travail n'est pas envisageable, répondre correctement aux deux premières phases est acceptable. Une solution, même imparfaite, à la troisième phase sera considérée comme un bon travail.

Lire les interfaces des modules Ocaml *String* et *Map* est conseillé.

Phase 1

Votre système n'aura à gérer que le déplacement d'un individu unique d'un module de la station à un autre en minimisant le temps de déplacement.

Votre programme devra prendre en entrée le plan de la base, le module de départ et celui d'arrivée, et devra afficher un chemin à suivre ainsi que le temps de parcours associé à ce chemin. Ce chemin devra être le plus rapide possible. Il y aura au plus un tunnel entre deux modules. Les tunnels peuvent être utilisés dans les deux sens.

Le format d'entrée des plans est le suivant :

- Une première ligne contenant un unique entier `n`.
- `n` lignes comprenant chacune la description d'une liaison au format
  <nom du module de départ> <nom du module d'arrivée> < durée du transit>
  La durée est exprimée sous forme d'un entier, les noms sont des chaînes de caractères.
- Une ligne comprenant deux noms de modules, un module de départ et un module d'arrivée.

Pour plus de facilité, nous vous fournirons un module de lecture de plan (schéma de la base et modules de départ et de destination). Tous les temps de parcours seront comptés en minutes (pas de fraction de minute à envisager).
Ce code est proposé sous la forme d'un module dont vous pourrez trouver le corps ici et l'interface là.

Pour cette première phase, vous devez impérativement :

1. Proposer une structure de données **pertinente** pour représenter le plan à l'intérieur de votre système (le format fourni en entrée n'est pas forcément adapté à votre problème).
2. Fournir un algorithme et un code de recherche de parcours le plus rapide possible en fonction du plan, du module de départ et du module d'arrivée.

Votre code devra écrire sur la sortie standard de votre programme le parcours trouvé au format suivant:

<temps> : <premier_module> -> ... -> <dernier_module>

Afin de faciliter la vérification de votre programme, vous **devrez** utiliser la fonction `output_sol_1` fournie dans le module `Analyse`.

Phase 2

Lors de cette phase, votre système devra gérer les déplacements de plusieurs personnes au sein de la base. Compte tenu de la faible capacité de calcul présente à ce stade sur la base, nous vous fournirons, en plus du plan, les chemins de chacun. Votre système devra retourner la *séquence des déplacements* de chacun en tenant compte des chemins à parcourir et de la contrainte d'utilisation unique de chaque tunnel en minimisant le temps de parcours global (donc le temps du dernier arrivé à destination).

Le format d'entrée des plans est le suivant :

- Une première ligne contenant un unique entier `n`
- `n` lignes comprenant chacune la description d'une liaison au format
  <nom du module de départ> <nom du module d'arrivée> < durée du transit>.
  La durée est exprimée sous forme d'un entier, les noms sont des chaînes de caractères.
- Une ligne comprenant un unique entier `m`.
- `m` lignes comprenant chacune une liste de noms de modules séparés par des `->` et correspondant au chemin d'un individu.

Votre programme devra afficher :

- `m` lignes contenant chacune une séquence de déplacement, c'est-à-dire une liste de noeuds séparés par des `-k->` où `k` est le moment où notre explorateur débute sa traversée entre les deux modules correspondants.
- Une ligne contenant un entier unique correspondant à la durée totale des déplacements.

Merci pour ce sujet...

Vous pouvez remercier (comme moi) M. Mouilleron pour toute l'aide qu'il m'a apporté dans l'élaboration de ce sujet (et pour la correction de la plupart des pheauthes).