Devoir 4
CSI2520 Paradigmes de programmation
Hiver 2018
A remettre le 6 Mars (Q1+Q2) et le 2 Avril (Q3+Q4) avant 23:00 sur le Campus Virtuel
[12 points]

La Ville d'Ottawa doit maintenir et inspecter de nombreuses infrastructures. Les inspecteurs de la Ville doivent ainsi régulièrement visiter différentes installations à travers la Ville. Votre mission est donc de les aider à établir un parcours de visite. Nous recherchons ici un parcours raisonnable, c'est-à-dire un parcours pas trop long permettant de visiter tous les sites. Nous ne demandons pas de trouver le chemin optimal.

Notre solution simplifiée procèdera donc comme suit :

- Construction d'un arbre général (nombre arbitraire d'enfants par noeud) :
  - Trier les emplacements à visiter d'ouest en est.
  - L'emplacement le plus à l'ouest sera à la racine de l'arbre.
  - Chaque emplacement est ajouté à l'arbre comme enfant du nœud géographiquement le plus proche.
  - Pour ce faire, il faut donc calculer la distance (en km) entre un emplacement à ajouter et tous les nœuds se trouvant dans l'arbre courant. Utiliser la distance à vol d'oiseau, i.e. ignorer les routes.
- Construction du parcours
  - Parcourir l'arbre résultant de facon pré-ordre.
  - La liste des noeuds ainsi parcourus constitue donc le parcours recherché. Calculer la distance cumulative de la visite de noeuds en nœuds.

Utiliser le fichier JSON ci-joint afin de tester cette approche. Ce fichier contient l'emplacement de toutes les pataugeoires de la Ville. Ces emplacements sont specifiés par leur coordonnées GPS (longitude et latitude).

Votre programme doit sauvegarder la solution trouvée dans un fichier texte donnant la liste des emplacements à visiter (leur nom et la distance cumulative), telle que montré ci-dessous (les valeurs montrées sont arbitraires et ne correspondent pas à la solution cherchée)

```
Crestview 0
Bellevue Manor 3.5
…
```

Voir le devoir 1 pour obetnir la formule permettant de calculer la distance entre deux lieux specifiés par leur latitude et longitude.

Question 0. Pré-traitement du fichier d'entrée (Python ou Java)

Le fichier contient la description des installations dans un fichier JSON. Vous pouvez utiliser soit Java ou Python afin de lire ce fichier et produire un fichier texte plus simple qui sera utilisé comme entrée par vos différentes solutions. Par exemple, dans le cas du paradigme logique, vous pouvez produire un fichier contenant une base de faits d'emplacements.

Pour ce faire, vous pouvez utiliser la librairie de lecture JSON de votre choix ou écrire votre propre fonction. Simplement spécifier quelle est la librairie utilisée et fournir au besoin le module pour compilation.

Question 1. Solution orientée-objet (Java) **[3 points]**

Créer les classes nécessaires pour résoudre ce problème. Votre programme doit être une application appelée `findRoute` prenant en argument le fichier de données d'entrée.

En plus du code source, vous devez aussi remettre un diagramme de classe UML montrant toutes les classes, leur attributs, méthodes et associations. Vous ne pouvez pas utiliser de méthodes statiques (à l'exception du `main`).

Question 2. Solution logique (Prolog) **[3 points]**

Écrire un prédicat Prolog `findRoute/1` (et tous les autres prédicats requis) retournant la solution dans une liste. Les données d'entrée sont simplement contenues dans une base de faits. Vous devez aussi définir un prédicat `saveRoute/2` avec comme premier argument la liste solution et comme second argument le nom du fichier texte à produire.

En plus du code source, vous devez aussi remettre la liste des prédicats accompagnés d'une courte description de leur fonctionnement.

Question 3. Solution fonctionnelle (Scheme) **[3 points]**

Écrire une fonction Scheme `findRoute` (et toutes les autres fonctions requises) prenant le nom d'un fichier de données comme entrée et retournant la liste solution. Vous devez aussi fournir une fonction `saveRoute` prenant en argument la solution et le nom du fichier texte à produire. A noter que pour produire votre solution vous ne pouvez pas utiliser les fonctions Scheme dont le nom se termine par un `!`

```
(findRoute "pools.txt")
```

- `'(("Crestview" 0) ("Bellevue Manor" 3.5) … )`

```
(saveRoute (findRoute "pools.txt") "solution.txt")
```

- `#t`

En plus du code source, vous devez aussi remettre la liste des fonctions créées et une courte description de celles-ci.

Question 4. Solution concurrente (Go) **[3 points]**

Écrire un programmea Go contenant une fonction `findRoute` prenant en paramètre le nom du fichier d'entrée. Votre solution doit utiliser la programmation concurrente en faisant usage d'un certain nombre de Go routines. Vous devez aussi fournir la function `saveRoute` prenant en argument la solution trouvée et le nom du fichier texte à produire.

```
func findRoute( filename string, num int) route []Edge
func saveRoute( route []Edge, filename string) bool
```

En plus du code source, vous devez aussi expliquer quelle est la stratégie utilisée dans le design de votre solution concurrente.

Remise de votre travail

Soumettre un fichier zip sur le campus virtuel. Ce répertoire compressé doit inclure les fichiers suivants:

- `Preprocessing.py ou .java`
- `FindRoute.jar` (incluant le code source)
- `findroute.pl`
- `findroute.scm`
- `findroute.go`

- Documentation
  - Un document pdf incluant la description de vos solutions
  - Tous les fichiers textes utilisés en entréee
  - Tous les fichiers texte produits par vos solutions, e.g. solution.pl.txt, solution.scm.txt etc.