DEVOIR #1 :
SYSTÈME DE GESTION DE CARGAISON DE CAMIONS D'UN ENTREPÔT

OBJECTIFS

- Appliquer des structures de données simples en Java : Tableau, Liste, Pile et File.
- Résoudre un problème simple.
- Analyser la complexité temporelle d'un algorithme avec deux méthodes : asymptotique et empirique.

PROBLÉMATIQUE

Vous devez écrire un programme en Java nommé `Tp1` qui gère un processus de cargaison des camions d'un entrepôt. Un grand entrepôt possède plusieurs bâtiments d'entreposage. Le programme reçoit en entrée le nombre total de boîtes à transporter, la capacité maximale d'un camion, les positions des bâtiments d'entreposage impliqués dans la cargaison courante ainsi que le nombre de boîtes disponibles à chaque point de cargaison. Pour commencer, il faut rechercher un point possédant le plus grand nombre de boîtes. Les coordonnées de ce bâtiment deviennent la position courante du camion. Pour chaque cargaison, le programme doit afficher les positions des bâtiments d'entreposage situés à la distance la plus proche de la position du camion, et le nombre de boîtes restantes aux points de service. Si les distances sont identiques, vous devez prendre celle avec la coordonnée de bâtiment d'entreposage en latitude plus petite et finalement, si les latitudes sont pareilles, on prendra selon la plus petite longitude. On suppose qu'à partir de chaque édifice, vers le camion, le transport de toutes les boîtes disponibles est possible. Le nombre de boîtes à charger dans le camion ne doit pas dépasser la capacité maximale de celui-ci.

Les distances doivent être calculées en utilisant la formule de haversine (https://fr.wikipedia.org/wiki/Formule_de_haversine) avec le rayon de terre 6371000 mètres.

Hypothèses simplificatrices.
Le camion reste toujours à sa position spécifiée, c'est plutôt les monte-charges qui se déplacent pour aller chercher et charger les boîtes. Un monte-charge peut livrer toutes les boîtes disponibles à la fois. On maximise le chargement d'un camion seulement au début en plaçant le camion à la position du point de service possédant le maximum des boîtes. Ensuite, on cherche à minimiser la distance parcourue par les monte-charges.

Exemple :

Supposons qu'un camion doit transporter 15 boîtes de marchandise... La capacité du camion est mesurée en termes de boîtes et pour cet exemple, elle sera donc aussi de 15. Dans la liste des bâtiments de cargaison, il y a 4 points de services localisés aux coordonnées spécifiées avec le nombre de boîtes disponibles. Les données initiales et les résultats trouvés sont affichés plus bas. Pour éviter les problèmes d'encodage des caractères accentués et faciliter la procédure de correction automatique, tous les messages d'entrées et de sorties doivent être écrits en anglais et respecter à la lettre le format spécifié.

```
15 15
2 (45.515399, -73.561996) 5 (45.5092,-73.5682)
8 (45.4383,-73.8205) 4 (45.4977,-73.714)
```

```
Truck position: (45.4383,-73.8205)
Distance:0          Number of boxes:0          Position:(45.4383,-73.8205)
Distance:10611.3    Number of boxes:0          Position:(45.4977,-73.714)
Distance:21193.7    Number of boxes:2          Position:(45.5092,-73.5682)
```

STRUCTURE DU PROGRAMME

Le programme `Tp1` doit pouvoir être lancé en ligne de commande avec deux arguments en entrées : un nom de fichier de données initiales et un nom de fichier résultat.

```
java Tp1 nomfichier1.txt nomfichier2.txt
```

Le fichier `nomfichier2.txt` contient un nombre de boîtes à charger, la capacité du camion et un état d'un entrepôt, soit une liste des bâtiments de cargaison (coordonnées) avec le nombre de boîtes disponibles. Si `nomfichier1.txt` est passé en argument (`args[0]`), alors votre programme doit lire depuis le fichier `nomfichier1.txt`, faire le traitement et enregistrer le résultat dans le format spécifié dans le deuxième fichier `nomfichier2.txt` passé au programme comme deuxième argument `(args[1])`. Considérons le fichier d'entrée `camion_entrepot.txt`. Le fichier `camion_entrepot.txt` est structuré de la façon suivante. Sur la première ligne, on y retrouve un nombre de boîtes à transporter et la capacité du camion. Les autres lignes contiennent une liste des 2 champs:

- nombre de boîtes au point de service;
- point de service spécifié par les coordonnées (latitude, longitude);

Pour faciliter la lecture (le *parsing*) au moyen d'un flux de lecture des tabulations ou des espaces blancs sépareront les champs. À titre d'exemple, voici un extrait du fichier d'entrée `camion_entrepot.txt` fourni.

```
15 15
2 (45.515399, -73.561996) 5 (45.5092,-73.5682)
8 (45.4383,-73.8205) 4 (45.4977,-73.714)
```

Sortie
La recommandation de cargaison calculée par le programme doit être écrite dans un fichier avec le nom fourni comme deuxième argument au programme.

```
Truck position: (45.4383,-73.8205)
Distance:0              Number of boxes:0               Position:(45.4383,-73.8205)
```

```
Distance:10611.3        Number of boxes:0               Position:(45.4977,-73.714)
Distance:21193.7        Number of boxes:2               Position:(45.5092,-73.5682)
```

Le format de sortie: première ligne : "`Truck position:`", espace, suivi par un point de service représenté par les coordonnées; Sur chaque ligne suivante : "`Distance:`" suivi par la valeur de la distance calculée, un espace suivi par la chaine "`Number of boxes:`", la valeur des boîtes restantes à chaque point de service impliqué, un espace, la chaine "`Position:`" suivi par les coordonnées. Les espaces pourraient être remplacés par le symbole tabulation. Il faut afficher les points de services impliqués dans la cargaison. Il est très important de respecter ce format de sortie, car des scripts de correction seront utilisés. L'exécution de la commande :

```
java Tp1 camion_entrepot.txt res+.txt
```

doit produire le résultat (`res+.txt`) :

```
Truck position: (45.4383,-73.8205)
Distance:0              Number of boxes:0               Position:(45.4383,-73.8205)
Distance:10611.3        Number of boxes:0               Position:(45.4977,-73.714)
Distance:21193.7        Number of boxes:2               Position:(45.5092,-73.5682)
```

ANALYSE TEMPORELLE

Vous devez faire deux types d'analyse de complexité en temps des algorithmes : expérimental (diapositive 6 Analyse d'algorithmes) et théorique (diapositives 8-18). Les résultats de vos analyses doivent être mis dans le rapport.

REMISE

Vous devez remettre électroniquement le TP1 **au plus tard le 12 juin 2019 à 23h55**.

Remise électronique

Vous devez remettre un seul fichier archivé contenant tous vos fichiers sources via StudiUM. Le rapport avec l'analyse de la complexité temporelle peut être remis en version électronique ou papier.

Rapport

Le rapport est constituée du/d'un(e)/de:

1. **Auto-évaluation** indiquant si votre programme fonctionne correctement, partiellement ou aucunement.
2. **Analyse de la complexité temporelle (pire cas) théorique en notation grand O**
    - La complexité temporelle doit être exprimée en fonction de la taille du problème :
        - *n* indique nombre de bâtiments impliqués dans la cargaison; L'analyse asymptotique doit être faite sur un pseudo code de votre solution ou directement sur le code Java développé.
3. **Analyse de la complexité temporelle empirique.**
    - Le graphique montrant le temps d'exécution de votre solution avec les différentes valeurs de n.

ÉVALUATION

Ce travail pratique vaut 10% de la note finale.

Grille de correction

Critère	Description	Pondération
A.	**Respect des directives pour la remise**	/ 0.5
B.	**Appréciation générale** Structure du programme + Qualité du code : découpage du programme, choix des types de données; identificateurs (noms) significatifs, lisibilité du code, pertinence des commentaires; etc. Encapsulation : respect des principes de l'abstraction; cachez le maximum de la représentation des objets en rendant un maximum d'attributs privés;	/ 1
C.	**Fonctionnement correct.** Le programme produit les bonnes recommandations. Une recommandation non optimale est considérée mauvaise même si elle est très proche de l'optimale.	/ 4

	L'efficacité n'est pas directement évaluée. Cependant, l'efficacité peut être indirectement évaluée lorsqu'un programme ne parvient pas à produire des résultats dans des délais raisonnables.	
D.	**Exactitude de l'auto-évaluation.** Vous déclarez que votre programme fonctionne correctement, partiellement, aucunement**.**	/ 0.5
E.	**Analyse de l'algorithme :** théorique et empirique	/ 4
	Total :	10/ 10

Pour les cas problématiques, jusqu'à 2 points peuvent être retranchés pour la qualité de la présentation.