IFT 2015 – Structures des donnée
Devoir #1

Manuela Girotti

1 Auto-évaluation

Le programme fonctionne correctement.

2 Pseudo-code

Partie 1. Lecture [1] du fichier `nomfichier1.txt`.
Création d'une classe `Building` :

```
private class Building{
        public float x_coord;
        public float y_coord;
        public int Nboxes;
        public float d;

        private static class Building{
                public double longitude;
                public double latitude;
                public int Nboxes;
                public double dist;

                public Building(double x_coord,
                        double y_coord, int N, double d){
                                longitude = x_coord;
                                latitude = y_coord;
                                Nboxes = N;
                                dist = d;
                }
        }
```

Création d'une liste chainée des instances de "Building" (utilisation de l'implementation `LinkedList` disponible sur Java).
Création des variables `max_capacity` et `tot_boxes` et leur initialization aux valeurs trouvées dans le fichier. Dans le cas où le nombre des boîtes est supérieur à la capacité maximale du camion, on notifie l'utilisateur en imprimant un message sur la console et on continue la cargaison jusqu'à quand le camion est complètement chargé.

Partie 2. Recherche du point de service principal `POS`.

```
x_POS = 0;
y_POS = 0;

max_Nboxes = 0;

for bldg in list_of_bldg
        if (max_Nboxes < bldg.Nboxes)
                max_Nboxes = bldg.Nboxes;
                x_POS = bldg.x_coord;
                y_POS = bldg.y_coord;

return x_POS, y_POS;
```

Partie 3. Calcul de la distance entre `POS` et tous les bâtiments dans l'entrepôt.

On utilise la formule de haversine [2] pour déterminer la distance entre deux points d'une sphère, à partir de leurs longitudes et latitudes.

On implémente les approximations suivantes :

— la Terre est une sphère parfaite ;
— il n'y a pas des obstacles géographiques entre les bâtiments (fleuves, collines, etc.) ;
— il n'y a pas des routes prédéterminées et le monte-charge peux se déplacer en suivant la route "à vol d'oiseau".

```
R = 6371e3;
phi_1 = latitude_1.toRadians();
phi_2 = latitude_2.toRadians();
Dphi = (latitude_2-latitude_1).toRadians();
Dlambda = (longitue_2-longitude_1).toRadians();

a = Math.sin(Dphi/2) * Math.sin(Dphi/2) +
        Math.cos(phi_1) * Math.cos(phi_2) *
        Math.sin(Dlambda/2) * Math.sin(Dlambda/2);
c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

d = R * c;
```

Partie 4. Tri de la liste de bâtiments selon leurs distances du `POS`. Avant du trier, on convertit la liste chaînée des bâtiments dans un tableau.

Afin de trier correctement les cas limites aussi (où deux ou plusieurs bâtiments peuvent avoir la même distance du point de service principal), on tri selons la procedure suivante :

(a) tri selon la latitude des bâtiments en ordre croissant ;
(b) tri selon la longitude des bâtiments en ordre croissant ;
(c) tri selon la distance des bâtiments du `POS` en ordre croissant.

On utilise l'algorithm du `merge sort` [3] trois fois. La propriété de stabilité de l'algorithme assure l'exactitude de l'ordre final.

Partie 5. Chargement du camion selon l'ordre des bâtiments, du plus proche au plus loin du `POS`.

```
for bldg in list_of_bldg
        loaded_boxes += bldg.Nboxes;

        if (loaded_boxes < min(tot_boxes, max_capacity))
                bldg.Nboxes = 0;
        else
                bldg.Nboxes = loaded_boxes - min(tot_boxes, max_capacity);
                break;
```

Partie 6 : Écriture [4] du fichier `nomfichier2.txt` selon les consignes.

3 Code en Java

Voir le fichier `GIROTTI_Tp1.java` jointe [5].

4 Analyse temporelle théorique (pire cas)

Soit N le nombre de bâtiments de l'entrepôt en question. On assume que toutes les opérations élémentaires (opérations algébriques, initialisation des variables, etc.) prennent un temps d'exécution constant ($\mathcal{O}(1)$).

Partie 1. lecture de 2 données (nombre des boîtes à transporter et capacité maximal du camion) + 3N (nombre des boîtes, longitude et latitude de chaque bâtiment) ; un avis est imprimé, le cas échéant.

$$\mathcal{O}(2+3N) = \mathcal{O}(N)$$

Partie 2. implémentation d'une recherche du maximum du nombre des boîtes sur tous les N bâtiments avec un loop.

$$\mathcal{O}(N)$$

Partie 3. calcul de la distance entre le point de service principal et tous les bâtiments dans l'entrepôt. Il prend un temps constant c pour calculer la distance pour chaque bâtiment :

$$\mathcal{O}(cN) = \mathcal{O}(N)$$

Partie 4. la conversion de la liste chaînée au tableau prend $\mathcal{O}(N)$ de temps. On trie la liste de N bâtiments avec l'algorithme `merge sort` 3 fois ($3 \cdot \mathcal{O}(N \log N)$).

$$\mathcal{O}(\max\{N, 3N\log N\}) = \mathcal{O}(N \log N)$$

Partie 5. le monte-charge visite tous les bâtiments séquentiellement pour charger le camion (le chargement prend un temps constant c) et il s'arrête quand il atteint la capacité maximale du camion ou le nombre des boîtes à transporter, le plus petit des deux. Dans le pire cas, il visite tous les N bâtiments :

$$\mathcal{O}(cN) = \mathcal{O}(N)$$

Partie 6. écriture de 2 données (position du camion) + 4N (distance entre le bâtiment et le camion, nombre des boîtes, longitude et latitude de chaque bâtiment).

$$\mathcal{O}(2+4N) = \mathcal{O}(N)$$

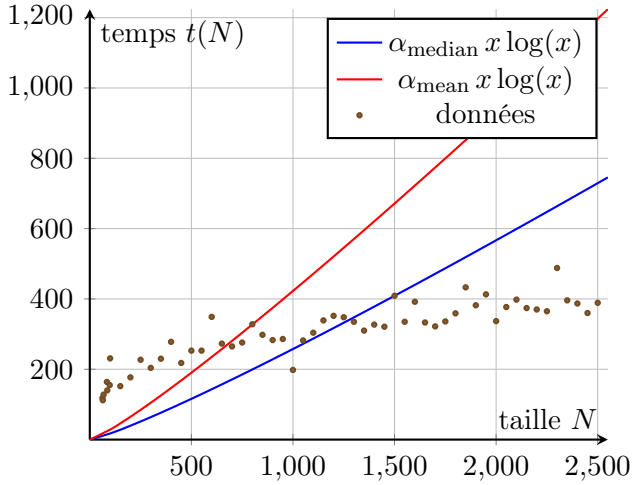
En conclusion, la complexité temporelle théorique de l'algorithme (au pire cas) est

$$\mathcal{O}(\max\{\text{complexité de la partie } j\}_{j=1,\dots,6}) = \mathcal{O}(N \log N).$$

5 Analyse temporelle expérimental

Comme dans la section précédente, soit N le nombre de bâtiments de l'entrepôt en question.

Les données sont fournies dans les fichiers `tp1Input` et `exemplaires_dev1`. La valeur $t(N)$ est obtenue avec la méthode `System.currentTimeMillis()` dans le code.



Pour l'analyse asymptotique dans le pire cas (ordre "$\mathcal{O}$"), il faudrait trouver une constante $\alpha \in \mathbb{R}_+$ telles que

$$t_{\text{empirique}}(N) \leq \alpha N \log(N)$$

(au moins pour $N \geq N_0$, pour un certain $N_0 \in \mathbb{N}$).

Les valeurs des "α" dans le graph ont été obtenues avec la méthode suivante :

$$\alpha_{\text{median}} = \text{médiane}\left\{ \frac{t(N)}{N \log N} \,\middle|\, N = 100, \dots, 2500 \right\}$$
$$\alpha_{\text{mean}} = \text{moyenne}\left\{ \frac{t(N)}{N \log N} \,\middle|\, N = 100, \dots, 2500 \right\}$$

A partir du graph en haut, on peut constater que le temps d'exécution moyen augmente plus lentement que $N \log N$ (l'ordre théorique). Une cause possible pourrait être que l'ensemble de données étudié est trop petit ou que les données sont biaisées (ou déjà partiellement triées).

Il faut aussi considérer que le résultat peut être affecté par le travail de fond de l'ordinateur pendant le traitement du code (bruit externe).

Une stratégie pour améliorer l'analyse empirique pourrait être de collecter plus de données.

Références

[1] lecture d'un fichier :
https://www.geeksforgeeks.org/different-ways-reading-text-file-java/

[2] formule de haversine : https://www.movable-type.co.uk/scripts/latlong.html

[3] `mergesort` pour un tableau :
https://algs4.cs.princeton.edu/14analysis/Mergesort.java.html

[4] écriture d'un fichier :
https://caveofprogramming.com/java/java-file-reading-and-writing-files-in-java.html

[5] plusieurs consultations pendant l'écriture du programme : https://docs.oracle.com/en/