Commentaires sur le projet de SDD2 (Juin 2021)

Notes :

— Chaque remarque est préfixée d'un "-", d'un "+" ou d'un " ?". Les "-" représentent un problème, une remarque négative. Les "+" sont des remarques positives et les " ?" sont des questions.

G 11 : KERCKHOFS Guillaume et BRENART Thomas

Code

Remarques concernant le code source et son fonctionnement

Implémentation

Le code fonctionne-t'il correctement ? Les algorithmes sont-ils efficaces ? La façon d'implémenter ne présente-t'elle pas de risques (ex : comparaison de flottants)

+ Utilisation de la structure de T pour récupérer les segments passant par un point.
- Dans FindNewEvent, on compare directement des flottants avec égalité. Pareil dans Segment ou dans Point. Si les valeurs sont obtenues à partir de calculs pouvant causer des erreurs de précision, on risque des problèmes.
- La précision utilisée (1e-4 et 1e-2) est très faible. On peut facilement utiliser 1e-8 pour des double.
- Attention, lors des comparaisons (plus petit, plus grand, ...), si on veut utiliser des inégalités avec des flottants, il faut tenir compte de l'erreur possible. Idéalement, on peut créer des méthodes qui permettent de remplacer les opérateurs de comparaison afin de toujours comparer avec la même précision sans que le code ne soit alourdit.
- Le calcul d'intersection entre deux segments pourrait être simplifié. Tout d'abord en utilisant des équations du type $a \cdot x + b \cdot y + c = 0$ à la place de $y = m \cdot x + p$ afin de ne pas avoir un cas particulier pour les segments verticaux. Ensuite en faisant attention aux cas d'égalité. Par exemple, même si $x_1 \neq x_2$, on peut avoir qu'ils sont très proches et se retrouver avec des valeurs très grandes ou très petites suite à une division. Ce qui peut poser problème pour la précision des calculs.
- Dans getCurrentPoint, on arrondit à 2 décimales près. Faire un arrondit peut causer une perte de précision. Par exemple, 0.004 et 0.005 une fois arrondis à deux décimales donnent 0 et 0.01. On passe donc d'une erreur de 0.001 à une erreur de 0.01, soit 10 fois plus.
- Toutes les intersections ne sont pas trouvées (fichier1.txt).

Lisibilité

Qualité générale du code (duplication de code, commentaires, ...)

- La gestion des exceptions consiste à les afficher en console. Dans un programme avec interface graphique, il vaut mieux ouvrir un message d'erreur ou essayer de gérer l'erreur.

- Dans MainPanel, il y a beaucoup de nombres magiques. On évite en général d'avoir des nombres qui apparaissent sans raison dans le code. À la place, on peut soit mettre un commentaire si c'est un cas unique, soit utiliser des constantes ce qui permet de facilement les modifier si besoin et rend le code plus compréhensible.
- Les variables de la classe Algo devraient être gérées par cette classe. Par exemple, ce n'est pas la méthode loadPoint de Map qui devrait remettre les intersections trouvées à zéro. Cela permet de mieux structurer la logique.
- Dans Algo, tout est en visibilité package. Cela veut dire que n'importe quelle classe pourrait modifier ces valeurs tant qu'elle est dans le même package. On risque des conflits. Surtout avec les interfaces graphiques quoi pourraient utiliser plusieurs threads.
- Dans FindIntersections, on vérifie si Q est vide en accédant directement à la racine. Pour une structure de données, on ne devrait pas accéder à ses éléments internes. Cela permet de comprendre un code sans avoir besoin de connaître le fonctionnement interne des structures de données utilisées.
- Les noms des attributs et des méthodes ne devraient pas commencer par une majuscule.
- Pour les noms des méthodes, il serait mieux de donner un nom générique à la méthode utilisables de l'extérieur et un nom plus spécifique à celle qui reste interne (ex : startInsertion et insertion dans Q). Par ailleurs, les méthodes internes devraient être privées.
- Les structures T et Q pourraient utiliser l'héritage pour éviter la redondance des opérations comme equilibrate.
- Dans Segment, la précision devrait être une constante pour pouvoir facilement la changer au besoin et utiliser la même précision partout.
- La javadoc n'est pas complète. Il y a des méthodes publiques sans documentation.
- Il vaut mieux éviter les noms de méthode comme `insert2`. On ne sait pas vraiment quelle est la différence logique avec `insert`. Aussi, s'il s'agit d'une méthode interne qui ne doit pas être utilisée depuis l'extérieur de la classe, on devrait lui donner une visibilité `private` ou `protected`.

Rapport

Remarques concernant le rapport.

Rédaction

Qualité de la rédaction (orthographe, français, présentation, ...)

- Dans un rapport, il est bien d'expliquer l'idée générale afin qu'on puisse comprendre pourquoi vous utilisez des structures de données spécifiques. Par exemple, pourquoi on a besoin de stocker les segments dans l'ordre de gauche à droite ?
- On doit expliquer les différents concepts avant de les utiliser. Par exemple, on parle d'event point et de sweep-line avant de dire à quoi ça correspond.
- Il y a des paragraphes assez longs qui pourraient être découpés. L'explication de Q parle de la structure, de l'ordre, de la gestion des doublons et de l'implémentation. Ce sont des sujets qui devraient être mieux délimités.
- Quand on explique un algorithme, on donne son pseudo-code. Cela permet d'avoir des explications qui ne donnent pas tous les détails techniques et ainsi d'avoir une idée du fonctionnement de l'algorithme quand on lit le pseudo-code.
- Les explications sont peu claires. Ça manque de détails, d'exemples et de formalisme. Idéalement, il faudrait commencer par donner l'idée en définisant les concepts avant de parler des détails importants. On peut donner un exemple pour faciliter la compréhension.
- Lorsque vous expliquez un algorithme, il vaut mieux laisser la complexité pour la suite. Sinon, on rend l'explication inutilement plus longue.
- Dans le rapport, on peut s'éloigner un peu de l'implémentation. Par exemple, on sait qu'un nœud contient une donnée et il n'est donc pas intéressant de savoir qu'il s'agit d'un champ `data`.

Contenu

Qualité du contenu du rapport (pas d'erreur, bonne explication,respecte ce qui était demandé, ...)

+ L'obtention des segments passant par un point semble bien exploiter la structure de T.
- Pour la complexité de `removeNextEvent`, on pourrait préciser que c'est du $O(\log(n))$ avec n le nombre d'éléments dans l'arbre. C'est plus précis et on peut plus facilement comparer les complexité car elle ne dépend plus de la forme de l'arbre.
- Pourquoi faire une différence entre insert et reinsert ? Si vous insérez des segments, ils sont passés par l'event point. Même si le point est l'extrémité haute.
- Votre explication de l'insertion dans T n'est pas claire. Comment fait-on pour que la donnée dans le père du nouveau nœud soit stockée dans une feuille ? D'après vos explications, on ne fait qu'insérer le segment mais on perd alors la donnée de la feuille qui devient un nœud interne.
? Pour la suppression dans le cas où la donnée est la feuille à gauche du nœud trouvé, l'explication est très difficile à comprendre. (fin de la page 7)
? Quand vous parlez du compareTo de T, vous ne parlez plus de faire une différence entre l'insertion et la suppression pour l'ordre des segments alors que vous en parliez plus haut. Qu'en est-il ? Il serait bien d'expliquer pourquoi on a besoin d'un ordre particulier et de définir ce que représentent x et y.
? Vous parlez de la complexité des algorithmes mais pas de la complexité totale. Pensez-vous avoir atteint les objectifs ?

Interface

Ergonomie de l'interface utilisateur (facile à utiliser, ...)

- On ne peut pas voir quelles intersections sont trouvées au fur et à mesure.
- La fenêtre ne peut pas être redimensionnée.
- Le zoom est réinitialisé à chaque click sur le bouton next event.
- Le programme plante si la carte n'a pas le bon format.
- On ajoute les segments via les coordonnées. Ce n'est pas pratique car on ne peut pas voir ce qu'on fait.

Remarques

Remarques générales sur le projet (problèmes de groupes, retard, ...)

— Chez moi, le programme plante en chargeant la carte 2 :

```
[java] Exception in thread "main" java.lang.ExceptionInInitializerError
[java] Caused by: java.lang.NullPointerException: Cannot invoke
       "code.logique.Segment.compareTo(code.logique.Segment, float, float)"
       because the return value of "code.logique.T_Tree.getData()" is null
[java]  at code.logique.T_Tree.suppress2(Unknown Source)
[java]  at code.logique.T_Tree.suppress2(Unknown Source)
[java]  at code.logique.T_Tree.suppress2(Unknown Source)
[java]  at code.logique.T_Tree.suppress2(Unknown Source)
[java]  at code.logique.T_Tree.suppress2(Unknown Source)
[java]  at code.logique.T_Tree.suppress2(Unknown Source)
[java]  at code.logique.T_Tree.suppress(Unknown Source)
[java]  at code.logique.Algo.HandleEventPoint(Unknown Source)
[java]  at code.logique.Algo.FindIntersections(Unknown Source)
[java]  at code.logique.Map.Open(Unknown Source)
[java]  at code.logique.Map.chooseOpen(Unknown Source)
[java]  at code.logique.Map.<init>(Unknown Source)
[java]  at code.Interface.MyWindow.<clinit>(Unknown Source)
```