ALGORITHMIQUE ET STRUCTURES DE DONNEES
-
COMPTE RENDU DE PROJET

Stenzel Cackowski & César Dumas

Introduction:

Ce document présente le compte rendu de projet d'algorithmique et structures de données par Stenzel Cackowski et César Dumas. Le projet a pour objet l'implémentation de l'algorithme de Bentley-Ottmannn en langage Python. Cet algorithme permet de déterminer les différents points d'intersection dans un ensemble de segments par une méthode de balayage du plan. L'implémentation de l'algorithme s'effectue à l'aide des structures de données fournies par les enseignants ainsi que deux modules recommandés : *heapq* et *sortedcontainers.* Ont aussi été fournis un jeu de fichiers pour tester notre algorithme.

Contenu du rendu :

- Un fichier *bo.py*. Il est composé d'une fonction **`bentley_ottmann`** contenant l'algorithme en lui-même et une fonction **`main`** permettant d'appeler la fonction **`bentley_ottmann`** sur les fichiers passer en arguments lors de l'exécution de *bo.py*.
- Un dossier *geo* contenant les divers modules importés lors de l'exécution de *bo.py* (*segment.py*, *point.py*, etc.)
- Un fichier *naif.py*. Il contient l'algorithme naïf de recherche d'intersections dans un ensemble de segments
- Un fichier *perf.py*. Il permet de déterminer les temps d'exécution de *bo.py* et *naif.py* dans l'optique de comparer les performances.
- Un dossier tests contenant les divers fichiers sur lesquels est appliqué l'algorithme.
- Un fichier *CompteRendu.pdf*

Méthodologie

Étant en binôme pour ce projet, nous avons décidé de travailler sur un répertoire .git afin de mieux gérer nos versions et de travailler à distance. Nous n'avons pas adopté de répartition précise des tâches, nous avons fait en sorte de travailler en même temps sur le projet afin que chacun garde un point de vue global sur les modifications et les différentes versions du projet. Ainsi nous avons procédé en 2 phases principales. La première fut l'implémentation de l'algorithme à proprement parler puis la deuxième a consisté en une phase de tests, de corrections et d'ajustements qui nous a pris la majorité du temps alloué au projet.

La fonction **`bentley_ottmann`** est implémentée comme suit :
Initialisation des variables :

- **`adjuster, segments = load_segments(filename)`**
- **`evenements = []`** : Tas des événements que l'on initialise à l'aide de segments puis de **`heapify`**. Nous avons fait en sorte que le premier point d'un segment (**`segment.enpoints[0]`**) soit le point le plus bas (en ordonnée). Ainsi nous avons

choisis d'implémenter nos événements ainsi:

```
evenement = [segment.endpoints[0][1], segment.endpoints[0][0], segment, STATUT]
```

où **STATUT** prend la valeur de **'D'** (pour début), **'F'** (pour fin) ou **'I'** (pour intersection. Les segments horizontaux sont traités dans un cas à part.

- **y_cour = None** : Désigne l'événement en cours de traitement. Il s'obtiendra ainsi :

```
y_cour = evenements.heapop()
```

- **cache_inters = {}** : Ce dictionnaire nous permet de stocker les segments déjà comparés pour ne pas effectuer deux fois la comparaison
- **actifs = SortedList()** : Désigne notre liste de segment vivants
- **pt_inter = {} ;intersections = []** : Résultats renvoyés par le programme *bo.py*
- Fonction **intersection** : Cette fonction permet de calculer une intersection, de vérifier que celle-ci n'est pas déjà présente ou ne constitue pas une extrémité de l'un des segments
- Fonction **indice.** Cette fonction est en théorie équivalente à la méthode index de la classe **SortedList**, elle a été implémentée pour des raisons détaillées ci-après

Méthode clef dans *geo/segment.py* :

Cette méthode implémente le calcul de la clef d'un segment tel qu'il a été décrit par les enseignants. Elle constitue une relation de comparaison entre les segments.

Déroulement du projet – Gestion des difficultés rencontrées

A la fin de la première phase, nous avions choisi de définir la fonction clef dans notre fonction **bentley_ottmann** et de définir actifs ainsi :

```
actifs = SortedListWithKey([], key = clef).
```

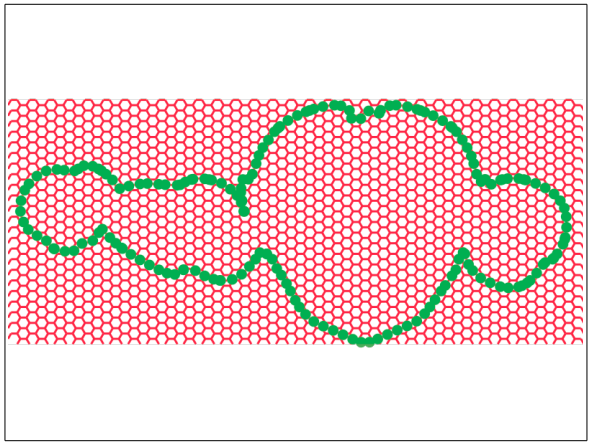
Nous avons dû abandonner cette option sous les conseils de l'enseignant qui nous a confirmé que l'argument key attendait une clef statique. Ainsi la liste **actifs** ne se mettait pas à jour lorsqu'on évoluait dans le plan ce qui provoquait évidemment des erreurs lors de la phase de test. En regardant de plus près le module **sortedcontainers**, nous avons effectivement remarqué que les méthodes de la classe qui nous renvoyait des erreurs (**index**, **remove**) travaillent non pas sur les éléments en eux-même mais sur les clefs calculées lors de l'ajout des éléments. Ainsi, nous avons décidé de surcharger la méthode **__lt__** de la classe **Segment** et d'implémenter une méthode clef dans cette même classe. La définition d'un attribut statique (**y_cour**) dans la classe segment nous a permis de résoudre les problèmes de mise à jour antérieurement évoqués. Toutefois, des problèmes quant à l'utilisation des méthodes de la classe **SortedList** ont subsisté, voilà pourquoi il nous a été nécessaire de reprogrammer notre propre fonction indice. Ainsi la méthode **remove** pouvait être remplacée par :

```
del actifs[indice(segment_a_supprimer)].
```

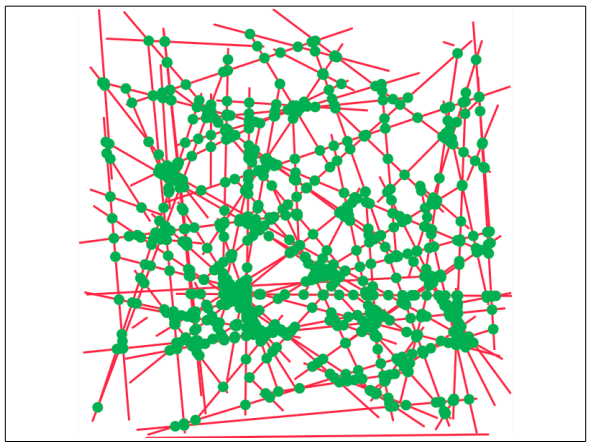
Les résultats

À la suite de la longue phase de débogage, nous avons des résultats satisfaisants à proposer. Notre programme compile sans erreur sur tous les fichiers de test proposés et arrive à déterminer l'intégralité des points d'intersection pour la majorité d'entre eux. Toutefois certains résultats sont erronés : Dans quelques rares cas, la liste des points d'intersection retournée par le programme est incomplète et nous n'avons, à ce jour, pas réussit à déterminer d'où proviennent ces omissions.
Voici quelques exemples de résultats où aucun point n'est omis :

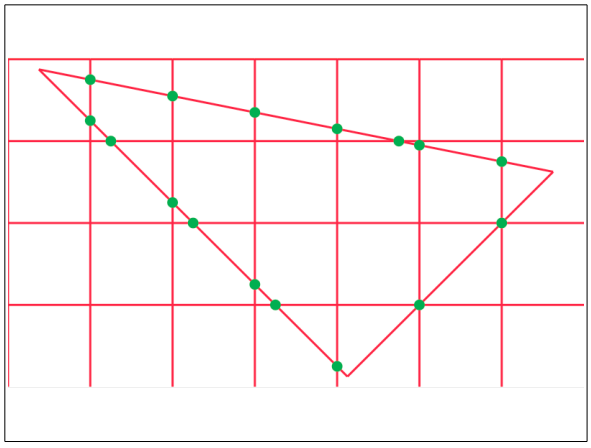
```
$./bo.py tests/carnifex_h_0.5.bo tests/random_100.bo
tests/triangle_0.8.bo tests/triangle_b_0.5.bo
```



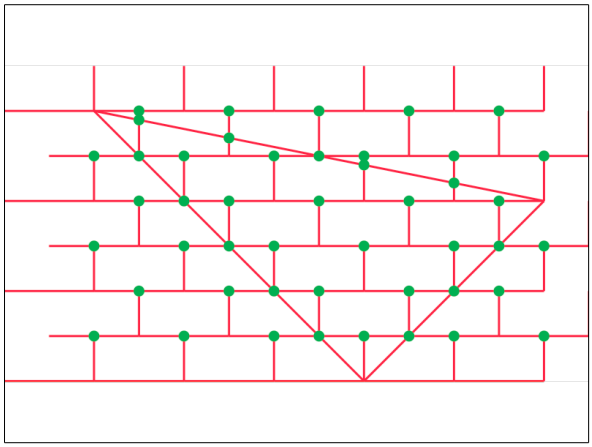
→ 183 points d'intersection , déterminés en 4,7 s



→ 821 points d'intersection, déterminés en 0.50 s



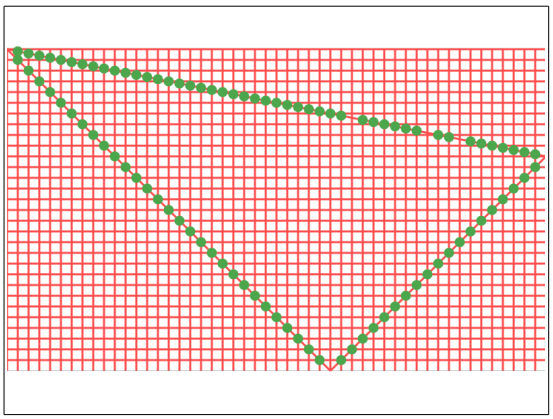
→ 16 points d'intersection, déterminés en 0.15 s



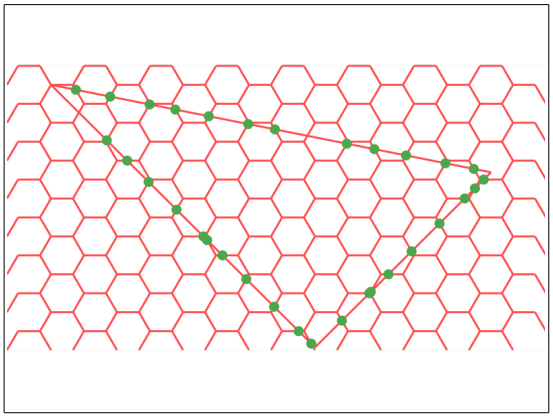
→ 46 points d'intersection, déterminés en 0.06 s

Voici deux résultats où quelques points sont omis :

```
$./bo.py tests/triangle_0.1.bo  tests/triangle_h_0.5.bo
```



94 points d'intersection, déterminés en 3,2 s
→ 3 points omis

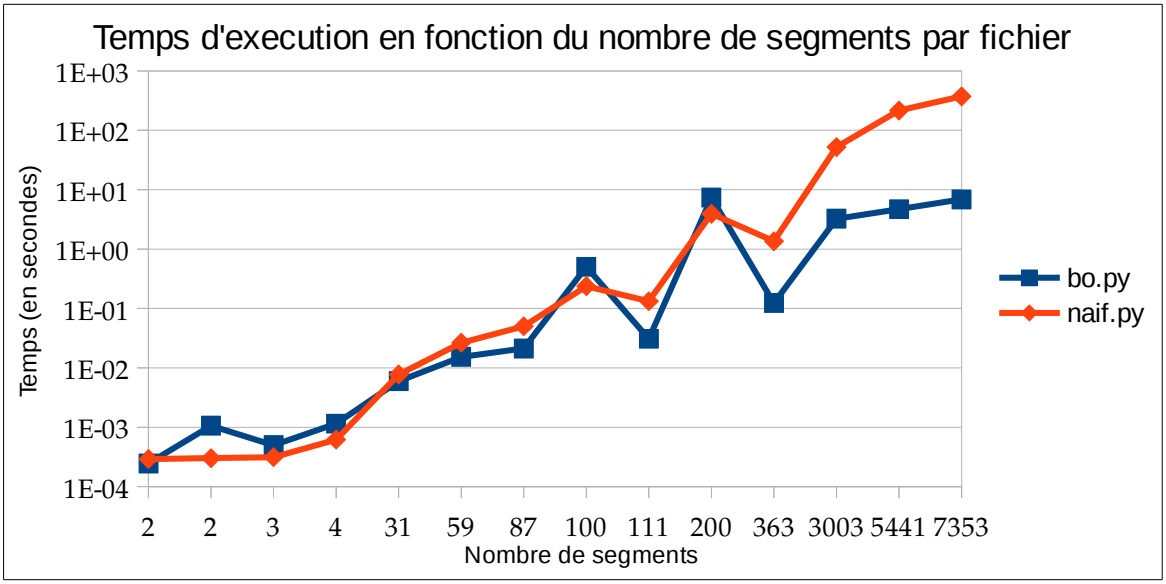


32 points d'intersection, déterminés en 0,02 s
→ 2 points omis

Nous avons implémenté l'algorithme naïf (dans *naif.py*) afin de comparer les différents temps de calculs (temps donné en secondes):

	Avec *bo.py*	Avec *naif.py*
tests/flat_simple.bo	0,0002	0,0003
tests/simple.bo	0,0011	0,0003
tests/simple_three.bo	0,0005	0,0003
tests/fin.bo	0,0011	0,0006
tests/triangle_b_1.0.bo	0,0060	0,0078
tests/triangle_0.8.bo	0,0153	0,0265
tests/triangle_b_0.5.bo	0,0212	0,0503
tests/random_100.bo	0,5042	0,2378
tests/triangle_h_1.0.bo	0,0309	0,1326
tests/random_200.bo	7,3766	3,9346
tests/triangle_h_0.5.bo	0,1240	1,3561
tests/triangle_0.1.bo	3,2554	52,2872
tests/carnifex_h_0.5.bo	4,7011	215,6780
tests/triangle_h_0.1.bo	6,8600	372,0018

, ce qui donne lieu au graphe suivant :



On remarque que l'algorithme de Bentley-Ottmann commence à être bien plus performant à partir de 200 segments. On remarquera aussi qu'une disposition aléatoire de segments est bien plus coûteuse dans le cas de l'algorithme de Bentley-Ottmann qu'une répartition régulière telle qu'elles sont proposées dans le dossier *tests.* Par exemple, entre le fichier *triangle_h_0.5.bo* (composé de 363 segments) et le fichier *random_200.bo* (composé de 200 segments, soit presque deux fois moins que le premier), le calcul est presque cent fois plus rapide pour le premier fichier. Pour ce qui est du plus gros fichier (*triangle_h_0.1.bo*), l'algorithme de Bentley-Ottmann est environ 80 fois plus rapide que l'algorithme naïf.

Conclusion - Les apports du projet

Ce projet nous a en premier lieu permis d'apprendre à travailler sur le même programme à plusieurs, ce qui n'est pas une tâche aisée lorsque le programme devient relativement gros. Ainsi nous avons pu progresser sur l'utilisation de git qui s'est révélé être un outil plus qu'utile dans notre cas, malgré le temps qu'il a fallu pour bien maîtriser son utilisation. De plus, le projet nous a posé de véritables difficultés qui n'ont pas été résolues facilement. Nous nous accordons à dire que nous pensions pouvoir effectuer ce projet sans difficultés majeures a priori, mais de nombreuses surprises étaient au rendez-vous. Ainsi notre patience et persévérance ont été mise à l'épreuve durant ce projet , ce qui formateur à terme.