Midterm Assignment 3

October 12, 2021

Indicaciones generales

1. Fecha de publicación: 11 de octubre de 2021.
2. Fecha de entrega: 25 de octubre de 2020 hasta las 23:55.
3. Único medio de entrega: `https://e-aulas.urosario.edu.co`.
4. Formato de entrega: código en Python 3.
5. Importante: no use acentos ni deje espacios en los nombres de los archivos que cree.
6. La actividad **debe** realizarse **individualmente**.
7. Los grupos pueden consultar sus ideas con los profesores para recibir orientación; sin embargo, la solución y detalles del ejercicio debe realizarlos **individualmente**. Cualquier tipo de fraude o plagio es causa de anulación directa de la evaluación y correspondiente proceso disciplinario.
8. El grupo de trabajo debe indicar en su entrega de la solución a la actividad cualquier asistencia que haya recibido.
9. El grupo no debe consultar ninguna solución a la actividad que no sea la suya.
10. El grupo no debe intentar ocultar ningún código que no sea propio en la solución a la actividad (a excepción del que se encuentra en las plantillas).
11. Las entregas están sujetas a herramientas automatizadas de detección de plagio en códigos.
12. `e-aulas` se cerrará a la hora acordada para el final de la evaluación. La solución de la actividad debe ser subida antes de esta hora. El material entregado a través de `e-aulas` será calificado tal como está. Si ningún tipo de material es entregado por este medio, la nota de la evaluación será 0.0.

Introduction

So you've decided to spend your undergraduate internship at a robot motion planning company. The company is widely known for their expertise in the control of the sailing of autonomous cargo ships. These crewless vessels transport either containers or bulk cargo over navigable waters with little or no human interaction. See Figure 1. Unfortunately, the last project on autonomous cargo ships of the company has presented some problems related to the self-navigation system and help is needed getting it back up again.

Given that you're a savvy intern with wide knowledge on algorithms and data structures, your boss has assigned to you fixing the project problem(s). Solving the issues with the self-navigation system of an autonomous cargo ship requires the implementation of a brand new interface. This piece of software will be plugged into the rest of the software developed by other colleagues in the company.

Problem

Your job is then to provide a computational application that offers different options of autonomous ship sailing (motion) planning. The functionality required evaluates aspects of autonomous ship navigation within a fleet (a group of ships). These aspects include global and local properties such as nearest and farthest ships, analysis of potential collisions, and drawing maps of the fleet.

Figure 1: Schematic view of how the motion planning of autonomous cargo ships work.

The application must be implemented as **a class with a collection of methods and a test suite**, coded in `Python 3`. We will treat locations of the ships on the map as defined by an ordered pair (x, y) of coordinates and measure all distances using the standard Euclidean norm in the plane. The methods must satisfy the following specifications:

1. Nearest ships. Communication between ships has a finite range. A ship has to know to which ships send information relevant to the navigation route. Given the location of a vessel (point in the plane), find and report the s closest ships to the location using the Euclidean distance. If there is more than s closest to the location of the vessel, report only s of them, choosing them randomly.
2. Avoiding collisions. As the ships navigate, they need to take into account if there are other vessels within some distance so maneuvers can be performed and collisions can be avoided. Find and return whether there are any ships within a square of arbitrary orientation of length r and center given by the location of a ship. The orientation of the square must set by either the library or the application. Only one range search needs to be performed.
3. Leading and lagging vessels. Suppose you want to explore areas in four directions: north (N), south (S), west (W), and east (E). It would be advisable to send the ships that are either lagging or leading within the fleet to save fuel. Find and report all the ships that have locations with either maximal or minimal x- or y-coordinates. Report all the ships if there is more than one ship for each direction. It might happen that one ship can have maximal/minimal position in more that one direction.
4. Implement a method that generates a two-dimensional map with the location of each ship in the fleet. Represent ships as dots (or some other relevant symbol) and the splitting lines using different colors for x- and y-coordinates. The resulting method must generate a plot similar to that of Fig. 2. To solve the problem, it might be useful to explore the source code of the base library.

The class should be named `AutonomousFleet`. You have the freedom to choose its attributes so redundant computation is avoided as much as possible. Extra methods may be implemented if necessary. The solution to each problem must generate a plot clearly *showing and labeling* the answer; you may also print extra information to the terminal with further details. Regarding input data, randomly generate around 10–100 points in the interval $R = [-1, 1] \times [-1, 1]$.

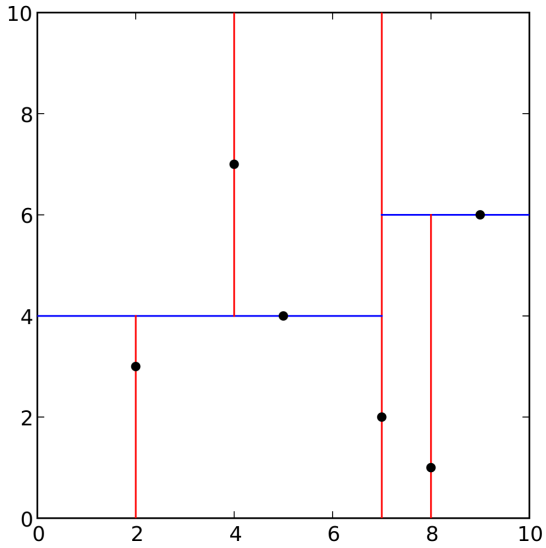

Figure 2: k-d tree decomposition of the fleet. Dots represents ships; vertical red (horizontal blue) lines correspond to splitting lines along the $x(y)$-coordinate.

Strategy

Let us now analyze how to proceed. The input to the problem is the planar coordinates of each vessel in the fleet, so you need to read the coordinates and appropriately keep the information. Next, remembering what you learned in Computational Geometry, you know that you need to build a computational model of the fleet so that you can efficiently process information about navigation and implement the functionality discussed above. A good candidate geometric model is a *k-d tree*, which is a geometric data structure that partitions space. Other pieces of information might also be relevant to keep nearby.

With this understanding, we now have a clear picture of how to proceed with the construction of the class. You may also want to keep in mind the range searching, the nearest-neighbor and the closest-pair of points problems, which can be efficiently solved in two dimensions using k-d trees. The methods to be implemented are listed in increasing order of difficulty.

References

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Third edition. Springer-Verlag, 2008.