Старинная задачка о паровозах (2-я редакция)

Исходная постановка

Задана железнодорожная сеть. Сеть состоит из конечного количества станций, соединенных одноколейными железнодорожными перегонами. Для каждого перегона задана длина (положительное целое число).
Задано некоторое количество «паровозов». Паровоз — точечный объект. Для каждого паровоза задан маршрут в виде списка станций, через которые он проходит. По прибытию на последнюю станцию паровоз исчезает. Паровозы стартуют по маршруту одновременно со скоростью 1.

Программа должна считывать из файла начальные данные (структуру железнодорожной сети и маршруты) и отвечать на вопрос: столкнуться ли два каких-либо паровоза? (дополнительно можно реализовать ответ на вопрос какие и где).

Требования к формату файлов не предъявляются. Корректность данных можно не проверять.

Понятие «столкновение»

Данная прога считает «столкновением» только следующее событие:
Два паровоза en1, en2 в некоторый момент времени t достигают одной точки на перегоне между станциями st1, st2. Причем один из en1, en2 движется в этот момент в направлении s1→st2, а другой наоборот — st2→st1.

Другие «сомнительные» ситуации, как то:

- Одновременное прибытие нескольких паровозов (с разных направлений) на одну станцию

- «Слипание» нескольких паровозов в одну точку и совместное движение по одному маршруту

столкновениями не считаются и детектятся этой прогой.

Алгоритм

Алгоритм суперпростой. Функция CrashDetector.collectPass стоит полный список событий типа

class BranchPass(eng: Int, st1:Int, st2:Int, t1:Int, t2:Int)

Каждый объект этого типа отражает факт прохождения паровозом eng от станции st1 к станции st2 в течение времени t1 – t2.

Затем функция CrashDetector.findCrash двойным циклом находит в этом списке все конфликтующие (см метод BranchPass.isCrashed) пары событий, которые и отражают столкновения паровозов. Алгоритм имеет квадратичную сложность.

Заметим, что столкновения не разрушают паровозы и станции. После столкновения паровозы как ни в чем ни бывало продолжают движение по своим маршрутам и могут участвовать в новых столкновениях.

Конфигурация сети

Ж/д сеть определяется файлом в формате XML.

Корневой узел специфицирует число станций и паровозов, например:

<rail_net StatNumber="11" EngineNumber="10">

….

</rail_net>

тут определяется сеть из 11 станций и 10 паровозов.

Перегоны между станциями соответствуют узлам <branch>, например:

<branch From="1" To="10" Length="2"/>

отражает существование перегона между станциями 1 и 10, длина которого равна 2. Атрибуты From, To симметричны, т. е.

<branch From="10" To="1" Length="2"/>

определяет тот же самый прергон.

Узлы <route> и <track> специфицируют маршруты паровозов. Например конструкция:

<route Engine ="3">

    <track Stat="0"/>

    <track Stat="10"/>

    <track Stat="0"/>

    <track Stat="5"/>

</route>

говорит, что паровоз 3 начинает движение со станции 0, идет на станцию 10, возвращается на станцию 0, затем идет на станцию 5, где он «исчезает».

Запуск программы

Под SBT прога запускается командой

run <path to XML network definition>

По содержимому XML файла строятся объекты класса RailNet, Routes, XmlCofig которые отражают конфигурацию данной сети и маршруты паровозов. При этом отлавливаются критически важные ошибки конфигурации (например, отсутствие перегонов между станциями, через которые проходит маршрут паровоза). При обнаружении ошибки, на консоль выдается соответствующее сообщение и выполнение прекращается. Для облегчения читаемости кода поиск ошибок выделен в отдельные методы CheckErrors.

Если сеть признана (статически) корректной, то в ней запускается поиск столкновений (CrashDetector.findCrash). При обнаружении таковых, на консоль выдается инфа о каждой трагедии (время, станции, паровозы).

Примеры

В папку Example я положил несколько примеров сетей с различными результатами выполнения.

Все примеры — маленькие. Я поленился генерить сети с большим (сотни-тысячи) числом объектов. Но если «заказчик» желает, то я сделаю это.

Функциональный стиль

Пытался писать в функциональном стиле. Одним из формальных признаков этого стиля

является почти полный отказ от использования переменных (var) в пользу констант (val). Но эта (светлая) идея вошла у меня в противоречие с архитектурным желанием разделить функциональность проги на несколько object-ов - RailNet, Routes, XmlCofig. Object-ы не имеют конструкторов и, поэтому, константу-член инициализировать нетривиальным значением невозможно. Так у меня образовалось несколько var-ов. Я стыдливо прикрыл их спецификаторами private.

Внутри функций ни одной переменной не определено.

Использовал только immutable коллекции.