LinelandNavigation: Optimal Navigation Under A Set Of Constraints

Avraham Leff

December 9, 2022

1 Assignment-Specific Packaging

The general packaging is unchanged from the basic "Homework Requirements" (see slides from first lecture and "Homework Policies for COM 2545" on Piazza).

This assignment's "DIR" **must be named** *LinelandNavigation*, and your application's Java class **must be named** *LinelandNavigation.java*. Your write-up file **must be named** *LinelandNavigation.pdf*.

2 Motivation

The assignment is intended to give you more experience applying (and extending) concepts learned in lecture to solve new problems, and more practice explaining an algorithm's design to others.

3 Background

As described in Flatland, *Lineland* is a **one-dimensional** world, and (as adapted by me), its inhabitants consists of Points who can only navigate backwards and forwards on this one-dimensional line. Points are born with an ability to

- Move exactly m (for *move*) positions *forward* from their current position.
- Move any number of positions *backwards* from their current position.

You'd expect that navigation by our client, *LePoint*, would be straightforward: move forward (increasing x-values) and backwards (decreasing x values. (Lineland extends infinitely in both directions: LePoint will never "fall off the edge of his world"). In a civilized Lineland, you'd be correct. Unfortunately, LePoint wants to navigate to a point g (for *goal*) where someone has buried a point of gold, and his enemies are determined to stop him.

Specifically: LePoint's enemies have mined certain *line segments* (each defined by their inclusive endpoints $y_i \leq y \leq y_j, i \leq j$). Navigation to any portion of such mined line segments will result in a very painful death. LePoint knows where these line segments are located, but cannot remove these mined areas. LePoint has reached out to you, requesting an *algorithm that specifies a sequence of forward and/or backward moves that will navigate him to (at least) position* g.

Specifically: the algorithm will either

- **minimize** the number of such navigational moves (a "forward" move counts as one move, as does a "backwards" move) if a successful navigation can be found or
- report that **no successful navigation** is possible.

3.1 The Rules

Here are the notation and constraints on your algorithm. As usual, the skeleton class's Javadoc is dispositive: what follows is only meant to set the stage.

- LePoint begins navigation at Lineland position 0
- You're supplied with the positive integer values of g and m.
- You're supplied with the set of mined line segments.
- The magnitude of LePoint's moves is defined above (note the asymmetry between moving "forwards" and "backwards"). Whether he moves forwards or backwards, he may not navigate to a position in a mined line segment.
- It's OK for LePoint to navigate to any position $\geq g$: this counts as a successful navigation to the goal.

3.2 Suggestions

I suggest that you first focus on devising a correct algorithm, then proceed to improving the algorithm's performance. I'm not specifying an order-of-growth because that specification would "give away" at least the basis of how to devise the algorithm. All I can say is that the algorithm's behavior should be "snappy".

4 Requirements

> Aside from using the JDK, for this assignment, you may **only use code written by yourself or copied from the textbook**!

The assignment consists of both "programming" and "non-programming" components.

4.1 Non-Programming

This portion of the assignment is worth 30%. No more than a **page and a half**!

There are many ways to design a solution to this problem. In this section, you must explain your approach in its entirety.

Note: I plan to grade this part based on: *"with no prior knowledge of this problem, can an CS person understood how you solved the problem?"*. Yes, I always grade based on your *"communication skills"*: clarity, correctness, succinctness, and "convincingness". This assignment will place an even greater weight on your success in this area.

As usual, pseudo-code is the right level of detail. If appropriate, consider incorporating a diagram into your discussion.

You may "explain by reference" to material covered in lecture or textbook or to "well-known" CS-knowledge. If you choose to use a "reference to covered material", you **must offer a convincing explanation** that this material is relevant to solving our problem.

4.2 Programming

This portion of the assignment is worth 70%, and will be graded for "correctness" and "speed" (see below).

Please review the general requirements for a programming assignment! I've tried to reduce the chances of "mistakes" occurring through the use of a "skeleton interface", but ultimately, **you are responsible** for ensuring that I can compile and test your code without incident.

1. Begin by downloading a skeleton `LinelandNavigation.java` class from this git repository.
2. Then, implement the stubbed methods of *LinelandNavigation*.