Report for Obligatory Assignment Part 2

Narongrit Unwerawattana

CPR No. 290692-4865

naunw18@student.sdu.dk

This assignment is part of DM819 – Computational Geometry, SDU

24 November 2018

Table of Contents

Introduction

This assignment is based on Point Location problem on chapter 6 of Computational Geometry: Algorithms and Applications book. We implemented a program to locate query point in 2 dimension plane by constructing a searching data structure with input line segments. The requirement is following statement

"A trapez decompostion and the accompanying search structure must be implemented, such that a search for a point returns the trapez the point is located in. You may assume that all x and y coordinates are distinct, i.e., that input is in general position."

Requirements

Golang v1.11.2

dep (golang dependency management tool) v0.5.0

Brief Structure

Program consists of 3 main module as following,

DAG tree: contains tree search structure to traverse information for a query point, where nodes are either x-node or y-node, and leaves are a node consists of a trapezoid

Trapezoid: contains all information for a trapezoid which are leftpoint, upper segment, bottom segment, right point, pointer to neighbors and lastly, a reference to a DAG leaf node. Including methods for handling addition of segments for itself

Pointlocation: contains a DAG tree and list of Trapezoids. Has methods for handling operation of addition of segments and searching for querying point in the DAG tree.

Algorithms

We use algorithm base on book Computational Geometry: Algorithms and Applications Chapter 6 Point Location, the input required for the program is a list of x – y coordinate on 2 dimension plane composing into a Segment object. While constructing the segments, we sort segment start-endpoint to lexicographically order and following pseudo code is execute to establish new PointLocation object for further use in the system

```
/**
        NewPointLocation returns new Pointlocation object for further querying
        input: ss []Segment
*/
function NewPointLocation(ss []Segment)
        boundingBox := compute the bounding trapezoid of segments
        create new DAG tree using boundingBox trapezoid
        randomize order of ss
        for each s in ss
                intersectedTrapezoids := findIntersection(s)
                for each trapezoid in intersectedTrapezoids
                        newTrapezoids :=  trapezoid.addSegment(s)
                        fix neighbors with old trapezoid or adjacent new trapezoids
                        create DAG tree node base on newTrapezoid and replace old node
                        remove old trapezoid from current trapezoid list
                        add newTrapezoids to current trapezoid list in object
/**
        findIntersection returns list of trapezoid that input segment is intersected or lie within the trapezoid
        input: s Segment
        output: []trapezoids
*/
function findIntersection(s)
        startTrapezoid := find s.startPoint in the poinlocation object
        currTrapezoid :=  startTrapezoid
        result := []trapezoid
        while  currTrapezoid.rightPoint.x < s.endPoint.x
```

```
            if s is continue lower  currTrapezoid.rightPoint
                    result.append(currTr.lowerRightN)
            else
                    result.append(currTr.upperRightN)
    return


/**
     addSegment returns new trapezoids that will be replacing current trapezoid after adding s to Pointlocation
    input: t trapezoid, s segment
    output: []trapezoids
*/
function addSegment(t trapezoid, s Segment)
    isFirstTr := if s.startPoint is lie within input trapezoid
    isLastTr := if s.endPoint is lie within input trapezoid
    upperTrapezoid, lowerTrapezoid := t.splitY(s) // split trapezoid vertically
    if isFirstTr
            leftTrapezoid := t.addSegmentLeft(s) // split trapezoid on left part
    if isLastTr
            rightTrapezoid := t.addSegmentRight(s)        // split trapezoid on right part
    return all new trapezoid
```

Complexity Analysis

We computing bounding box trapezoid in line 1 by simply traverse all of given segments input and record maximum and minimum x-y coordinate, thus, expected running time is linear. The iteration executing while adding new segments running time could be break down into 2 problems, the traversal cost of DAG tree and cost of creating new trapezoids, node and removing trapezoid. Traversal cost of DAG tree cost O(logn) due to the randomize incremental approach, and cost of the latter part can be done in linear time. The expected running time is O(nlogn)

Example Input

The program accept comma separate file (csv) to operate. File content shall be construct by lines of x-y coordinate in float format with last line indicate the query point eg.

11.692800521850586,56.20393502509641	point 1
11.690783500671387,56.201189742460066	point 2
11.695590019226074,56.20128523385047	...
11.703529357910156,56.20348147021205	
11.697521209716797,56.20761100886297	
11.69121265411377,56.206799458341905	
11.692800521850586,56.20393502509641	
11.6984224319458,56.203218883353635	query point

The system will construct line segment object by point1 → point2 and so on until second to last line. Afterward, it will generate DAG tree for querying data and using last line coordinate to walk through DAG tree and print the result trapezoid

Manual

The implemented program could be execute in 2 ways, one is by using go command otherwise by using executable file, if “file” flag is not defined by user, program will lookup for file test1.csv, which is included in the package

Using go commands

This method requires host machine to install go version 1.11.2 from the official golang repository by following https://golang.org/doc/install manual and dep, the dependency management tool of go by executing script from dep team (https://github.com/golang/dep for more information)

```
curl https://raw.githubusercontent.com/golang/dep/master/install.sh | sh
```

After installing all dep, we install dependencies by using command

```
dep ensure
```

Lastly we execute program by using go run command, with or without “file” flag

```
go run main.go
go run main.go -file=./test1.csv
```

Using executable file

We have compiled our source code for multiple platform for user who doesn’t wish to install Golang. The implemented program could be execute by using executable file on the project root directory eg. ./pointlocation-linux-amd64 with or without specify path of input

```
pointlocation-linux-amd64
pointlocation-linux-amd64 -file=./test1.csv
```

Input Description

Using command

```
pointlocation-linux-amd64 -file=./test1.csv
```

within project directory, will make program to find and read test1.csv file as an input for all required information. Where data from first line to second last indicate input segments, and last line indicate the query point.

11.692800521850586,56.20393502509641
11.690783500671387,56.201189742460066
11.695590019226074,56.20128523385047
11.703529357910156,56.20348147021205
11.697521209716797,56.20761100886297
11.69121265411377,56.206799458341905
11.692800521850586,56.20393502509641
11.6984224319458,56.203218883353635

Content in test1.csv where last line is querypoint

Output

After the execution complete, terminal will output segment that has been added to program and the trapezoid that querypoint is located in.

```
assignment-geo-2$ ./pointlocation-linux-amd64
clearing image files
adding segment (11.69121265411377, 56.206799458341905) -> (11.692800521850586, 56.20393502509641)
adding segment (11.690783500671387, 56.201189742460066) -> (11.692800521850586, 56.20393502509641)
adding segment (11.690783500671387, 56.201189742460066) -> (11.695590019226074, 56.20128523385047)
adding segment (11.695590019226074, 56.20128523385047) -> (11.703529357910156, 56.20348147021205)
adding segment (11.69121265411377, 56.206799458341905) -> (11.697521209716797, 56.20761100886297)
adding segment (11.697521209716797, 56.20761100886297) -> (11.703529357910156, 56.20348147021205)
---------------------
-------result---------
---------------------
[tr]node tr = trapezoid[bt for 3] lp: (11.697521209716797, 56.20761100886297)
      t: (11.697521209716797, 56.20761100886297) -> (11.703529357910156, 56.20348147021205)
      rp: (11.703529357910156, 56.20348147021205)
      b: (11.695590019226074, 56.20128523385047) -> (11.703529357910156, 56.20348147021205)
      left neighbors: 1
      lowerLeft.leftp: (11.695590019226074, 56.20128523385047)
      lowerLeft.top: (11.69121265411377, 56.206799458341905) -> (11.697521209716797, 56.20761100886297)
      lowerLeft.rightp: (11.697521209716797, 56.20761100886297)
      lowerLeft.bottom: (11.695590019226074, 56.20128523385047) -> (11.703529357910156, 56.20348147021205)
      nil
      has reference to dag, parents = [[y]node segment = (11.697521209716797, 56.20761100886297) -> (11.703529357910156,
56.20348147021205), parent = [x]node xcoor = 11.697521209716797, parent = [y]node segment = (11.695590019226074,
56.20128523385047) -> (11.703529357910156, 56.20348147021205), parent = [x]node xcoor = 11.703529357910156, parent = [x]node xcoor
= 11.695590019226074, parent = [x]node xcoor = 11.692800521850586, parent = [x]node xcoor = 11.69121265411377, parent = <nil>]
---------------------
```

Program output

The trapezoid output is formatted as following

trapezoid[*name_of_trapezoid*] lp: (leftpoint coordinate)

t: (top segment coordinates)

rp: (rightpoint coordinate)

b: (bottom segment coordinates)

left neighbors: number of neighbors

(if there is any)

(trapezoid left information)

right neighbors: number of neighbors

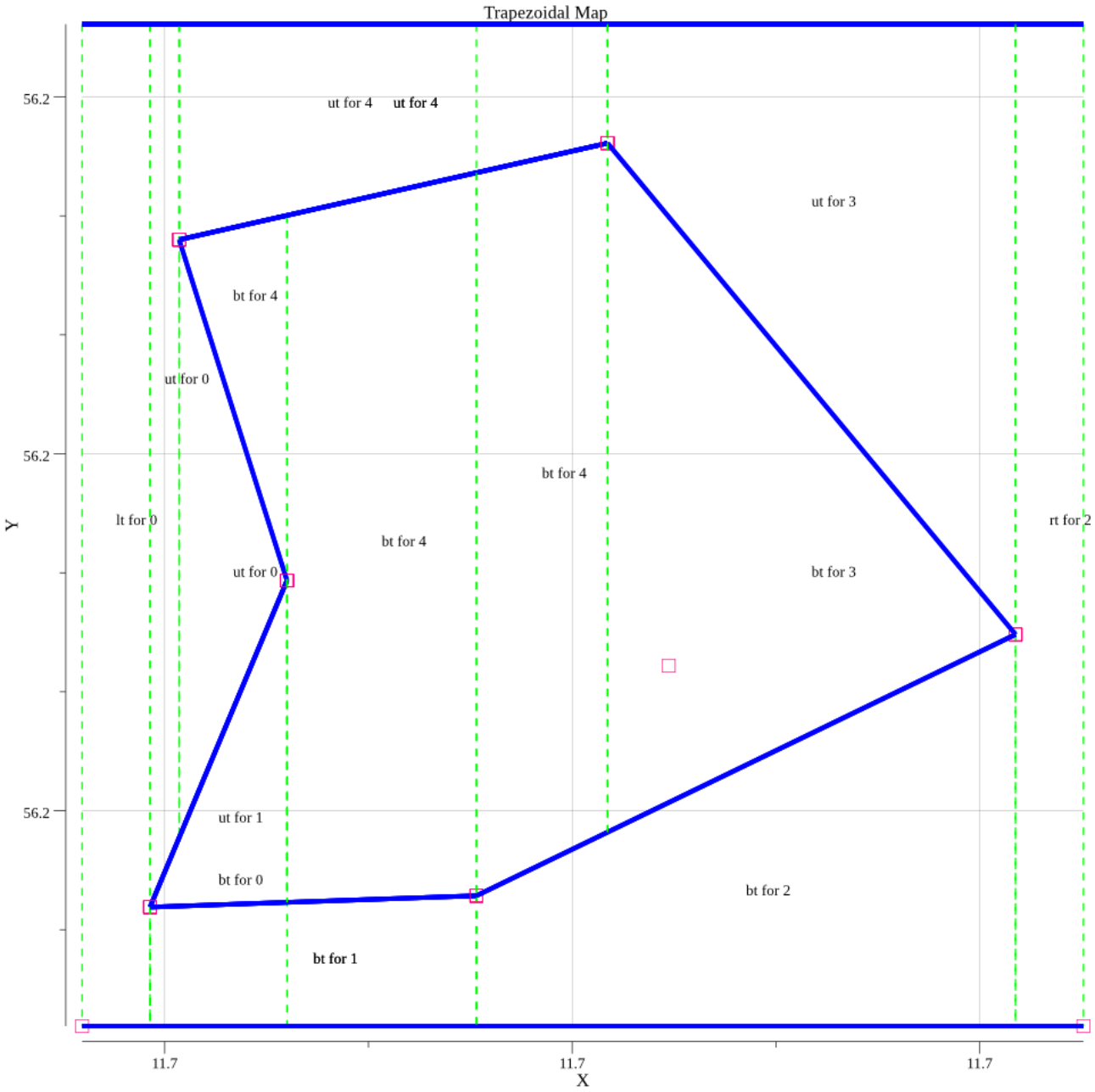
(if there is any)

(trapezoid right information)

DAG : (reference to tree and its parents)

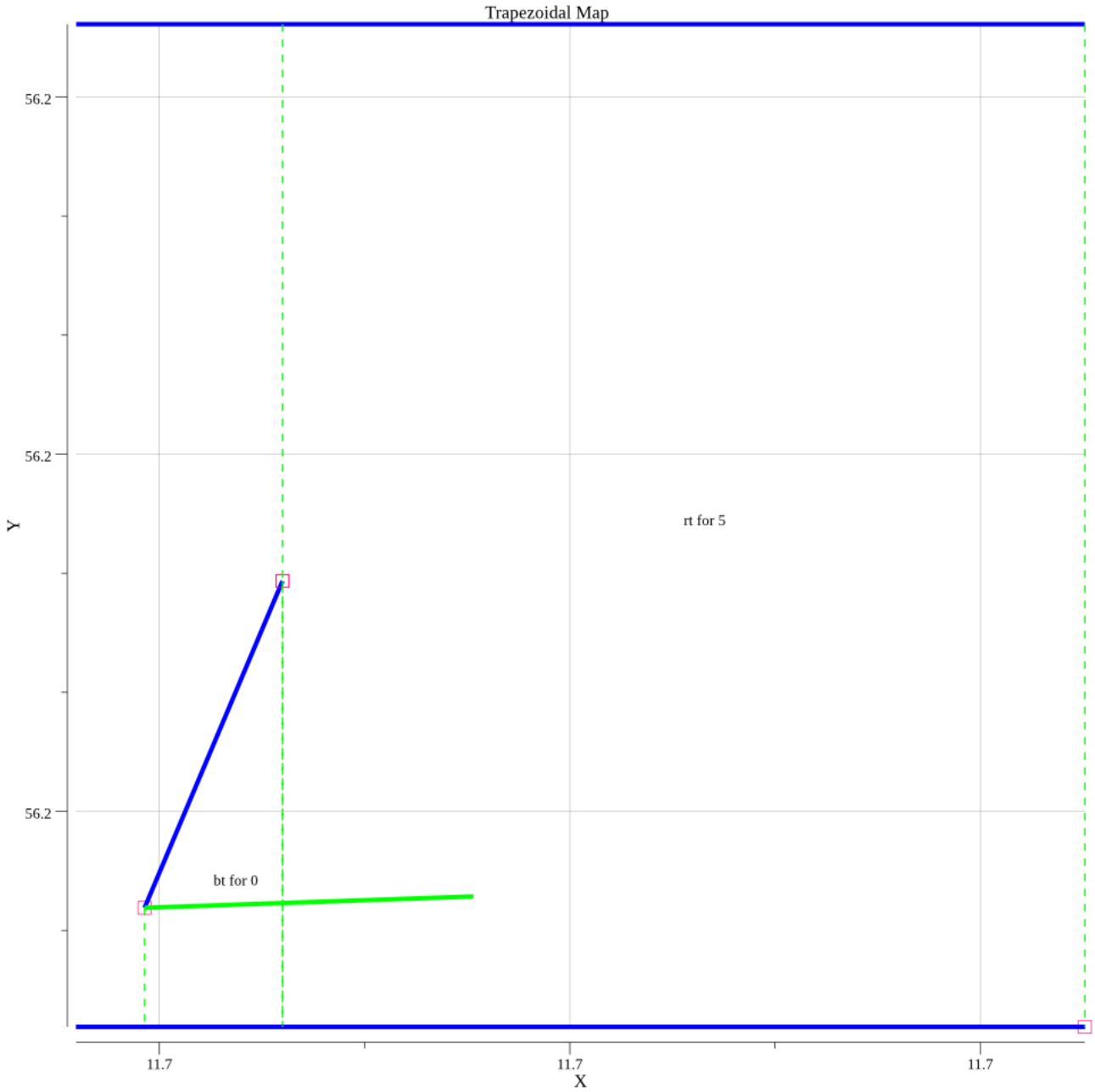
Graphical Output

After the execution complete, the program will also generate png image to current path, namely result.png where result of the execution is showed graphically



result.png

additionally, in “steps” folder, program will generate graphical output for each operation steps of adding segment iteration which includes before adding segment, intersected trapezoid on adding a segments, removing trapezoid and lastly, trapezoid result after adding segment.



Graphical output before adding segment, green is the adding segment