Insertion of a convex polygon in trapezoidal map and finding point location

A thesis
submitted in partial fulfillment of the requirement for the Degree of Master of Technology in Computer Technology of
Jadavpur University

by

Aadya Rani
Exam Roll No - M6TCT22022B
Registration No - 149855 of 2019-2020
Class Roll No - 001910504021
Session: 2019-2022

Under the Guidance and Supervision of

Dr. Chintan Kumar Mandal
Department of Computer Science and Engineering
Jadavpur University Kolkata-700032
India

2022

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Declaration of Authorship

I, hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of Master in Computer Technology. All information in this document have been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials that are not original to this work.

Sign:_____ Date:_____

Name: Aadya Rani

Examination Roll No. M6TCT22022

Registration No. 149855 of 2019-2020

Thesis Title: Insertion of a convex polygon in trapezoidal map and finding point location

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Recommendation

This is to certify that the dissertation entitled "**Insertion of a convex polygon in trapezoidal map and finding point location**" has been carried out by Aadya Rani (University Registration No: 149855 of 2019-2020, Examination Roll No: M6TCT22022) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the Degree of Master in Computer Technology. The research results presented in the thesis have not been included in any other paper submitted for the award of any degree in any other University or Institute.

Dr. Chintan Kumar Mandal, Asst. Professor
Dept. Of Computer Science Engineering
Jadavpur University, Kolkata- 700032
Signature: _____

Prof. Nandini Mukherjee
Head of Department
Dept. Of Computer Science and Engineering
Jadavpur University, Kolkata-700032
Signature: _____

Prof. Chandan Mazumdar
Dean
Faculty of Engineering and Technology
Jadavpur University, Kolkata-700032
Signature: _____

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate Of Approval

This is to certify that the thesis "**Insertion of a convex polygon in trapezoidal map and finding point location**" is a bonafide record of work carried out by Aadya Rani **(University Registration No: 149855 of 2019-2020, Examination Roll No: M6TCT22022)** in partial fulfillment of the requirements for the award of the degree of Master in Computer Technology in the Department of Computer Science and Engineering, Jadavpur University during the period of June 2019 to july 2022. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, the opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Examiner:

Signature: _____ Date: _____

Dr. Chintan Kumar Mandal
Dept. Of Computer Science and Engineering
Jadavpur University, Kolkata-700032

Signature:_____ Date: _____

Acknowledgement

I would like to express my sincere gratitude to my advisor **Dr. Chintan Kumar Mandal** for his continuous motivation, guidance and patience throughout my thesis work, I have been very lucky to have a advisor who cared so much about my work. More importantly, the scientific and personnel support along with his valuable suggestion softened the journey.

Aadya Rani

Signature: _____

Table of Contents

Abstract

Point Location Problem is a part of Computational Geometry and it is used to determine the area that contains query point ‘q’.

The main subject of my thesis was to describe and implement the algorithm that is used to insert convex polygon in map and analyze it’s run time.

So ,we have proposed algorithms to perform new operations on an arrangement of line segments in the plane, represented by a trapezoidal map; the dynamic insertion of the convex polygon in the trapezoidal map, and then finding the top polygon, bottom polygon, left polygon and right polygon for a given query point in the map.

1. Introduction

The point location problem and partitioning are two of the core issues in computational geometry. It is the preparation of a plane's polygon subdivision into a data structure so that we may quickly determine which polygon a query point is located in.

These actual situations frequently occur in software programmes like Geographic Information Systems (GIS). Consider the situation where a user wants to select a certain country for research while looking at a digital map of the world on a computer monitor. The computer can identify which polygon the query point was inside and return the relevant information to the user based on the screen coordinates and the data stored in the map when the user clicks on the computer monitor inside the polygon denoting that country.

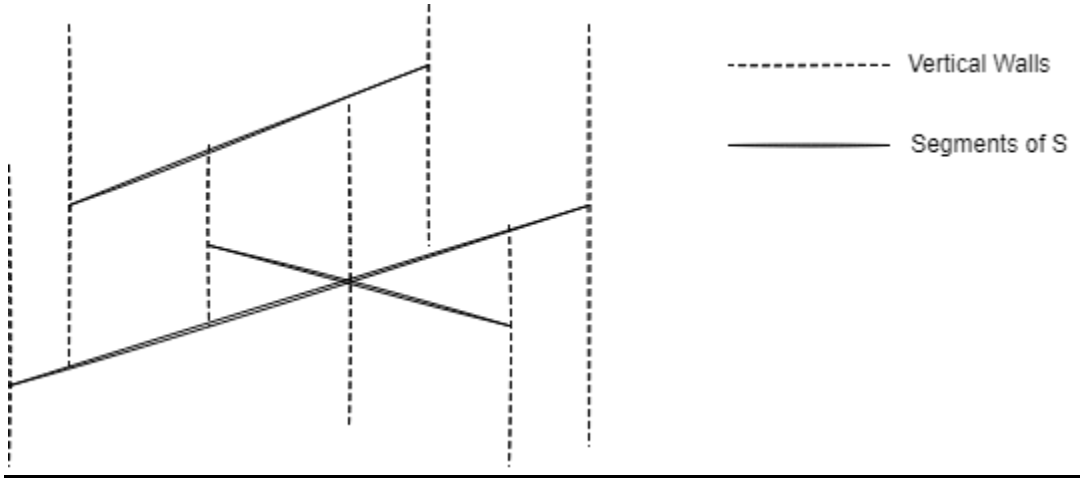
1.1 Problem Statement

This thesis is aimed to insert a convex polygon randomly in a trapezoidal map and then solving the point location problem. We are finding the left ,right ,top and bottom polygon with respect to the given query point .

1.2 Background Study

1.2.1 Trapezoidal map:

To determine the pairs of lines in a set S of n line segments that intersect, trapezoidal maps are frequently utilized. The trapezoidal map T(S) is defined as follows: extend a vertical segment to the first segment in S above and to the first segment in S below from each end point of a line segment in S or each junction point of two line segments in S. (or to infinity if there is no such segment in S). By doing this, we are able to divide the plane into trapezoids, some of which are degenerate. A trapezoid has a floor, a ceiling, and two vertical walls through which it can have at most four neighbouring trapezoids (at most two per wall), known as horizontal neighbours. A trapezoid is defined by at most four segments of S.



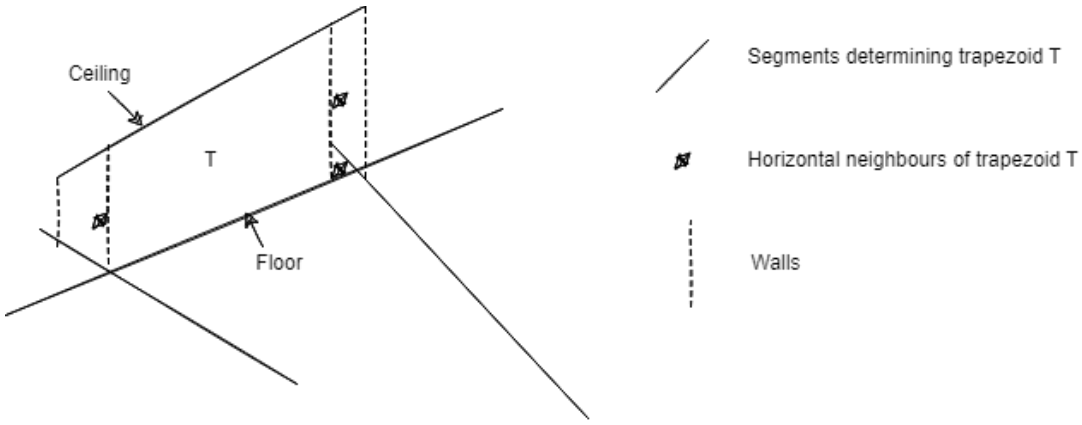


Figure 1

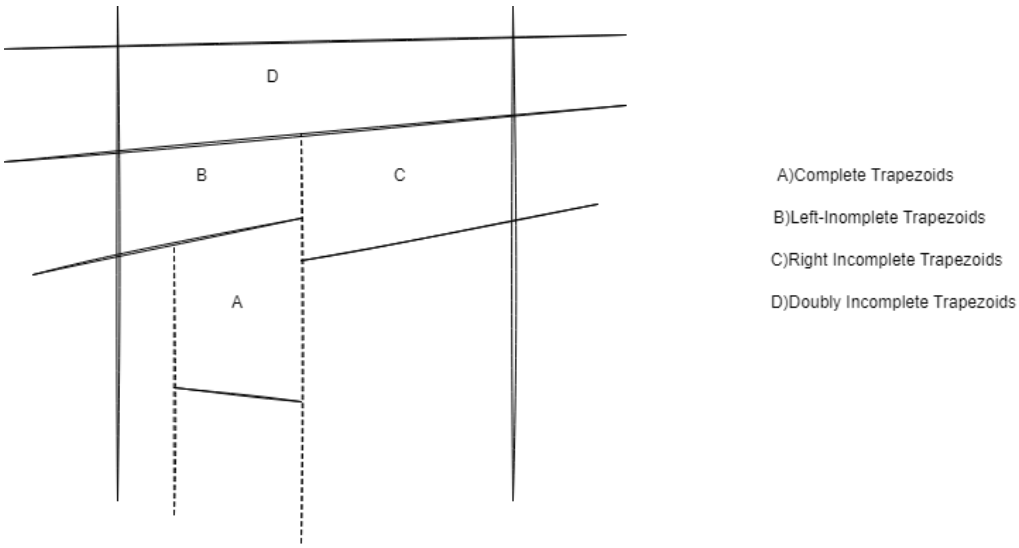


Figure 2

Our issue can be solved by using the definition of a trapezoidal map: a processor P computes the trapezoidal map in its associated slab V = Vab. Then, we must distinguish between various trapezoidal types:

- Usual trapezoids, also called as complete trapezoids.

• Simple incomplete trapezoids with a boundary line of a V on one side. Such a trapezoid T has a floor and a ceiling that both intersect; T is right incomplete unless is the right boundary line of V. The trapezoid in the slab next to "V" with the same boundary line is also determined by

the floor and ceiling of "T." If we were to analyze the trapezoidal map across the entire plane, this trapezoid would join T to create a typical trapezoid. A simply incomplete trapezoid is characterized by a maximum of three segments, has one wall and hence at most two horizontal neighbors, which are the two neighboring trapezoids in the same slab V. We'll also use the phrases left-complete and right-complete in the opposite sense.

• Doubly incomplete trapezoids have no wall and no horizontal neighbor, are specified by two segments traversing V, and are enclosed by the two V boundary lines.

1.2.2 Data Structure

In this case, we first review the Influence Graph's concept and the method by which it may be used to calculate the trapezoidal map of a set S of line segments in the plane, before demonstrating how the structure must be altered to enable splits and unions of trapezoidal maps.

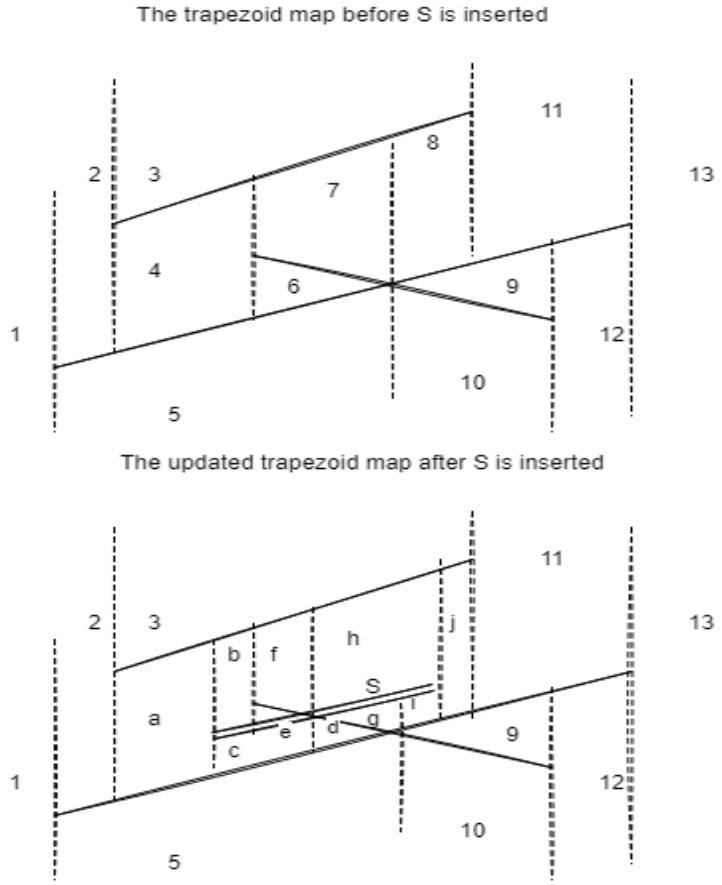

The trapezoid map before S is inserted
11
8
2
3
7
13
4
6
9
1
12
10
5
The updated trapezoid map after S is inserted
11
2
3
j
13
b
f
h
S
a
i
g
e
d
9
c
1
12
10
5

Figure 3

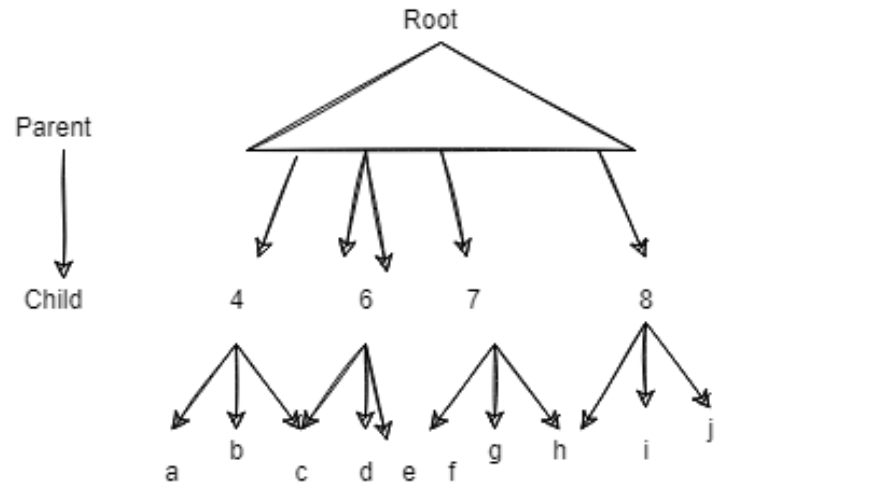

Trapezoids 4,6,7,8 are killed , a-j are created, and influence graph is created
Root
Parent
Child
4
6
7
8
a
b
c
d
e
f
g
h
i
j

Figure 4

1.2.2.1 The Influence Graph:

The concept behind the influence graph is to use an incremental algorithm to keep track of the history of building. In specific situations, it allows for semi-dynamic constructs that only allow insertions, as well as dynamic constructions that allow both insertions and deletions. In a general scenario, it can be supplemented to provide such dynamic structures. It has been demonstrated, nonetheless, that a typical Influence Graph sufficed to permit deletions for the trapezoidal map.

We are particularly interested in trapezoids produced by line segments S. We will state that S defines such trapezoids. The Influence Network is a rooted directed acyclic graph with trapezoids defined by S as its nodes. Its root is the trapezoid defined by an empty set of line segments, which is equivalent to the entire plane. Each new segment S will cross part of the trapezoids defined by the segments added before it if we calculate a trapezoidal map progressively. The modified trapezoidal map must be updated to replace those trapezoids with new ones. However, they will continue to be listed as trapezoids destroyed by S in the Influence Graph. The influence graph preserves the entire construction history in this way. S is credited with creating the new trapezoids. A trapezoid formed by S becomes the parent of a trapezoid slain by S in the influence graph if and only if they overlap. As a result, a trapezoid can have any number of parents but a maximum of four children.

We keep the following information for every node:

Its killer, at most four horizontal neighbours, at most four children, and at most four segments determine it.

The influence graph makes it possible to detect a segment S that needs to be added. Because a child is contained in the union of its parents, we can

recursively traverse the influence graph from the root to the leaves, stopping at all the trapezoids intersected by S.

1.2.2.2 The Modified Influence Graph

A processor P calculates the trapezoidal map in the slab V = Vab that it is associated with by building an Influence Graph progressively and making the following adjustments in each node.

- We store a mark “complete, left-incomplete, and doubly incomplete”.

- In addition to storing links to a node's children, we also need to store ties to its parents. A node may have one or more parents, but the order in which they are arranged along the segment that gave rise to the node is arbitrary. This enables the division of a set of parents or the concatenation of two consecutive lists of parents of two adjacent trapezoids in a logarithmic in the total number of parents. It also allows us to store pointers to the parents of each node in a concatenable queue. A 2-3 tree can be used to implement such a structure. A tree's leaves represent the parents of a node on the Influence Graph, and the tree’s root represent the node itself.

- The links between nodes in the 2-3 tree won't be unidirectional as they often are since we need bidirectional links to be able to reach the root from the leaves as well as the leaves from the branches. When an Influence Graph is browsed using these bi-directional relationships, the children of the node N will be found by accessing the roots of the one to two more trees that N belongs to.

- A node in the influence graph can have up to four offspring, which causes it to appear as a leaf in up to four 2-3 trees. Finding N's children may therefore be done in a time that is logarithmic in the

  amount of elements in the two to three trees it appears in, which is itself constrained by the sum of N's children's parents. Finding a new segment in the changed Influence Graph as a result will be O(log n)-complex, where n is the total number of segments.

1.3 Splitting a trapezoidal map

Let V represent the vertical line that is a part of slab V. Assume that the Influence Graph and the trapezoidal map of the set of line segments intersecting V have previously been calculated. By splitting V along D, we can create the two Influence Graphs that correspond to the slabs V_l and Vr. Both of the new Influence Graphs must match what would have been produced if they had been constructed in new slabs directly.

The goal of the approach is to locate all trapezoids in the past that are split by D by recursively traversing V's Influence Graph. The two new Influence Graphs will be built during this traversal.

The old root becomes the root of the left influence graph after the creation of a new root, which, for example, will be the root of the right influence graph. Then, iteratively, the following procedures are carried out: each node N visited must be divided into two portions, designated N_l and Nr, respectively. N_l is the portion of N that is to the left of D and will be a node of the left Influence Graph.

• If N was left - incomplete, then N_l is doubly incomplete, and if N was right - incomplete, then Nr is left - incomplete. If N was doubly - incomplete, then both Nr and N_l are doubly - incomplete.

• Depending on its position in relation to D, a child that is not split by D becomes a child of either N_l or Nr.

• A child T divided by D is looked at iteratively. It will be divided into two trapezoids: Tr and T_l, the children of Nr and N_l, respectively.

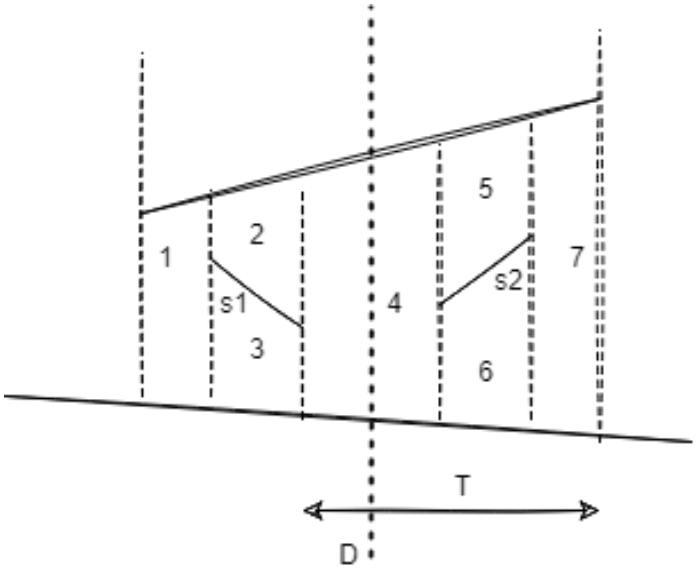
• The parents of N that are not divided by D must also be updated; D divides the ordered set of N's parents into two sets, N_l and Nr. This is accomplished by splitting the concatenable queue that stores the set of parents of N into two concatenable queues in logarithmic time.

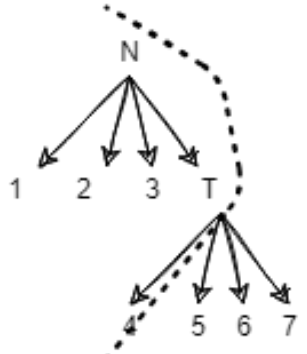
The operations that came before updated N's offspring. However, as the Influence Graphs generated are only temporary, the procedures below must be completed to update the Influence Graph.

• The changed portion of the structure is correct if Nr and N_l have two or three kids each.

• It is clear that Nr and Tr are in fact two identical trapezoids if Nr only has one kid, Tr. Since it doesn't correlate to the insertion of a segment in this slab, the link parent-child between them has no justification and would not have been formed if the Influence Graph for the slab Vr had been directly constructed. The two nodes must then be combined.

• The same is true for both N_l and T_l.





Influence graph before split

Insertion of S1

Insertion of S2

Temporary influence graph

Corrected influence graph

Figure 5

1.4 Union of two trapezoidal maps

The influence graph for two slabs Vr and V_l ,that are next to each other along a vertical line is provided. These two provided influence graphs must be used to infer the influence graph in slab V= Vr U V_l

Beginning at the base of each influence graph, all left-incomplete nodes in Vr and all right-incomplete nodes in V_l, which will be combined, will be visited concurrently. The other nodes don't need to be traversed because they haven't been altered. The recursive traversal involves building a new Influence Graph.

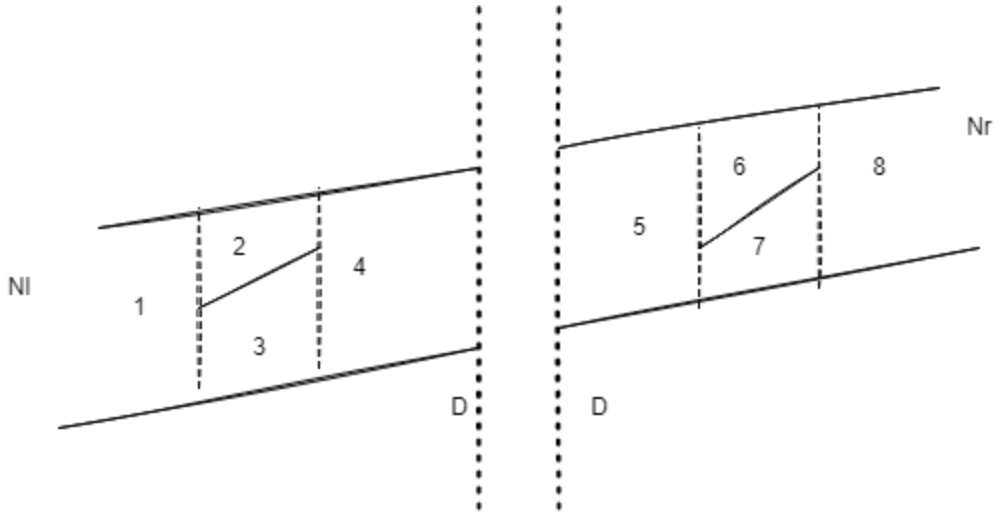
To create a trapezoid N=Nr U N_l in the new influence graph, the roots of the two influence graphs must first be combined. Then, at each step of the recursion, let Nr and N_l be the nodes of the influence graphs of Vr and V_l, respectively. They share the same floor and ceiling. N is a newly generated node. This new node's ceiling and floor are identical to those of Nr and N_l. If there are any horizontal neighbours to its right, they are those of Nr; otherwise, N is right-incomplete if Nr was left-incomplete, if N_l was doubly incomplete, or if Nr was both right- and left-incomplete.

- If the same segment had killed Nr and N_l, then N can adopt their offspring. The two offspring that must be combined among all of N's children continue the recursive process.

- In every other case, if a segment S_l killed N_l before a segment Sr killed Nr. In this instance, N is killed by Sr, and its offspring are the left-complete children of Nr as well as a new trapezoid T that is created by

joining N1 and the left-incomplete child Tr of Nr. By including N in the list of Tr's parents, the whole set of parents for T may be found. After that, the recursion merges N_l and Tr.

- The same approach is taken to tackle the symmetric case when N_l was killed before Nr.



The influence graph after the union, assuming sl was inserted after sr

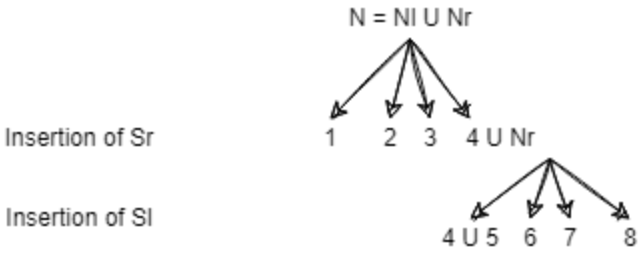


Figure 6

2. Literature Survey

Over the last several years, Point Location Problem has gained significant attention among researchers. Point Location Problem is used to determine the location of a given query point. We solve Point Location Problem with the help of Trapezoidal Map.

Said et al. [1] proposed a system for "The shortest path problem in interval valued trapezoidal and triangular neutrosophic environment" to show that a single valued neutrosophic set may encompass the indeterminacy in real-world decision-making problems. It is an expansion of the neutrosophic set with interval values. Finding the network's shortest path is one of the more well-known real-world challenges since it involves uncertainty in most cases. This work proposes a new score function for interval-valued neutrosophic numbers and solves the SPP problem using these numbers.

Cui Can et al.[2] "Fundamental operations in the GISciences are boolean operations between flat polygons". In this study, Boolean operations are carried out using a novel technique based on trapezoidal decomposition. Boolean operations between polygons are changed into Boolean operations between broken trapezoids using this technique. Comparatively speaking, processing basic trapezoids is simpler than processing random polygons. With this new approach, the complicated computation of the spatial relationship between the polygon edges that is required by the older methods is avoided, making the process quicker and simpler to comprehend. The suggested approach can also be used with concave and holey polygons.

3. Work Done

In this project we are inserting a convex polygon one by one in the map, and then we are trying to find the top, bottom, left and right polygon with respect to a given point. So basically, work done in this project can be explained in three parts:

i) Insertion of polygons in map.
ii) Finding the nearest polygon with respect to the given point.
iii) Then finding the left, right, top and bottom polygon with respect to the point.

3.1 Insertion of convex polygon in map

Here the complete Map is a Directed cyclic graph. Each polygon node will have certain attributes, like left, right, top & bottom. Based on the coordinate location of the polygon, we will store the relative position of polygons with all different polygons already present.

When a certain polygon comes its coordinates are compared with all present polygons and its relative position is calculated. It's finally positioned in the graph based on it. Something like mentioned below.

When the first polygon comes, it gets inserted in map. When the second polygon comes, we check whether this polygon will be inserted in top, left, right or bottom with respect to the first polygon with the help of four equations. And when the third polygon comes it also gets inserted in same way .It gets compared with both the polygon and so on.

We do insertion using the four equations:-

$(x1 - x) < |y - y1|$ ---------- i

$(x1 - x) > |y - y1|$ ---------- ii

$(y1 - y) < |x - x1|$ ---------- iii

$(y1 - y) > |x - x1|$ ---------- iv

If the equation one is fulfilled then the polygon is inserted in left side.

If the equation two is fulfilled then the polygon is inserted in Right side.

If the equation three is fulfilled then the polygon is inserted in top side.

If the equation four is fulfilled then the polygon is inserted in bottom side.

3.1.1 Derivation of equation

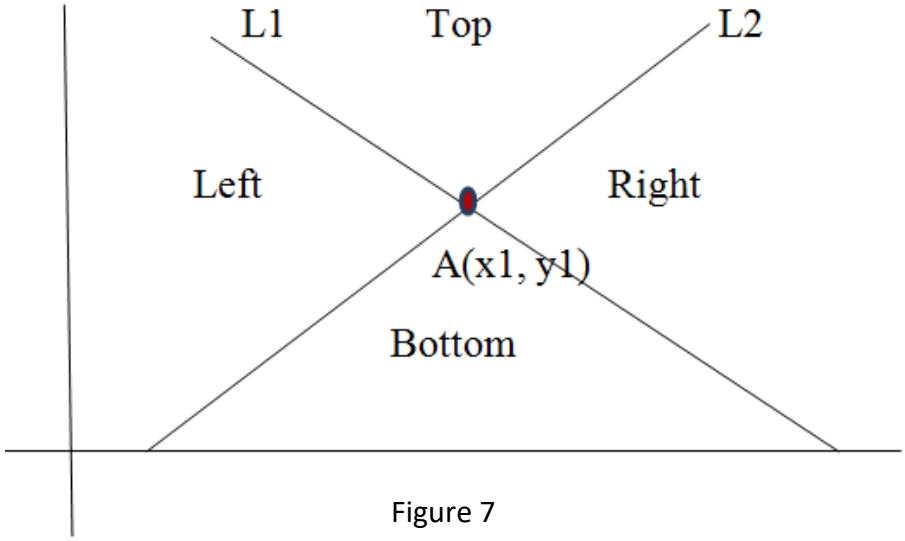


Figure 7

Consider the point is $P(x, y)$

All the points which lie in left segment should satisfy the following equations

Let $A(x1, y1)$ be the mid of polygon.

Left segment = $L1$(bottom side) ^$L2$(top side).

Equation of $L1$:

Slope of $L1$:

$$m = -1 \text{ (as angle with } x - axis \text{ is 135 } degree)$$

So,

$$y - y1 = m(x - x1)$$

$$(x - x1) + (y - y1) = 0$$

So equation of segment above $L1$:

$$(x - x1) + (y - y1) > 0$$

So the equation of segment below $L1$:

$$(x - x1) + (y - y1) < 0$$

$$(x - x1) < (y1 - y)$$

Multiplying both the sides by -1, the above equation becomes

$$(x1 - x) > (y - y1)$$

Equation of $L2$:

Slope of $L2$:

$m = 1$ (as angle with $x - axis$ is 45 $degree$)

$y - y1 = m(x - x1)$

$(x - x1) - (y - y1) = 0$

Equation of portion above $L1$:

$(x - x1) - (y - y1) < 0$

$(x - x1) < (y - y1)$

$(x1 - x) < (y - y1)$ is the equation for left insertion.

Similarly we can find equations for right, top and bottom.

3.1.2 Algorithm for insertion of convex polygon in map

Here, first we will create an object polygonSet(Map), which will store all the polygons. Following is the algorithm for the insertion of one polygon in the map.

Suppose we have a list of polygon vertices to be inserted in polygonSet, then we will insert them one by one (randomly) as mentioned below. The worst time complexity of insertion is O(n).

Step 1: Create a polygon object with given vertices.

Step 2: Using the vertices also calculate the centroid of the polygon, and also set extreme points for it.

Step 3: Iterate through the list of polygonSet and determine the direction of the polygons from the new polygon. The direction is calculated based on the algorithm given above which assigns directions based on the position of the centroid of a certain polygon.

Step 4: Find the nearest polygons in each direction i.e. left, top, right & bottom.

Step 5: Next, we have to add the nearest polygon of each direction to the new polygon. And also review the links of previous present polygons. So we will also review if linked polygon's link have to be updated or not.

3.2 Finding the nearest polygon with respect to the point

The task here is to find the nearest polygon in the map from a given point. For this, we need an algorithm that can find/compare the distance of any polygon from a given point. Also, we need an algorithm that can traverse through the map based on the nearness of a polygon and reach the desired nearest polygon.

To find the nearest polygon with respect to the given point, we first compare whether the given point lies on which side of the first node in graph. When we get the direction , then the nodes on other sides are marked as visited ,then we need to find the nearest polygon ,for that we calculate the distance from point to every nodes that are not visited ,and we store the distance in a list ,and then we find the nearest polygon by comparing the distance.

3.2.1 How to find shortest distance between point and line segment

The length of the perpendicular connecting the point to the line, or the distance between the point and the beginning or end of the line, can be used to determine the shortest distance between two points and a section of a line. Time complexity to calculate distance of a polygon from point is O(1).

For example, the shortest distance, for instance, is the length of the perpendicular green line d2 in figure 8B, which is the boundary of point P. The shortest distance must be determined from either the beginning or the end of the line segment since the points in figures 8a and 8c are outside their respective starting and ending perpendiculars.

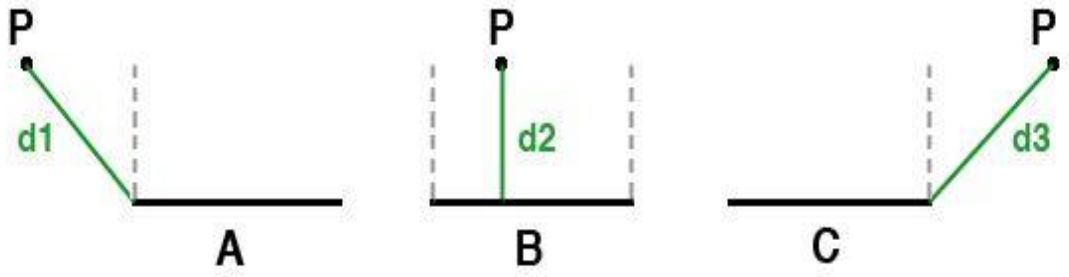


Figure 8

We use vector dot product to find if the shortest distance is the length of the perpendicular line d2 in case B ,or d1 or d3 as shown in case A and case C respectively.

Coordinate inputs -

Line: Start $(1,0,2)$ End $(4.5\ ,0,0.5)$

Point : Pnt $(2\ ,\ 0\ ,0.5)$

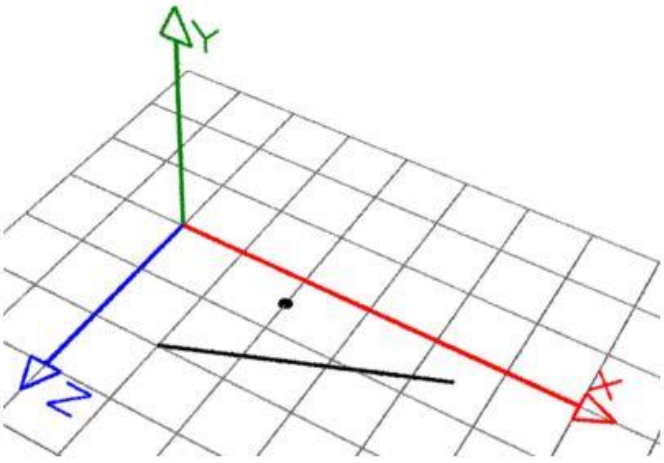


Figure 9

The Y coordinate of the line and point are zero and as such both lie on the XZ plane.

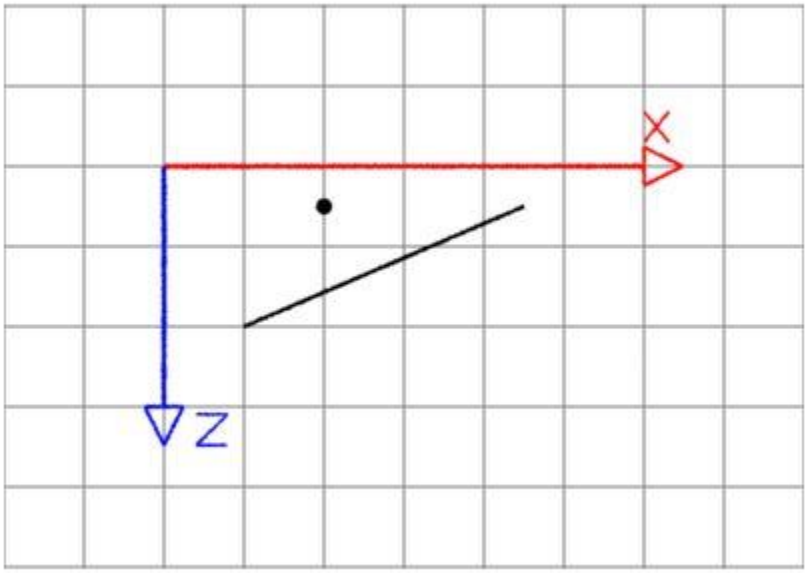


Figure 10

Step 1:

First of all we will convert the line and point to vectors. And the coordinate of the vector representing the point are relative to the start of the line.

$line\_vec = vector(start, end)$ # (3.5,0,-1.5)

$pnt\_vec = vector(start, pnt)$ # (1,0,-1.5)

Step 2:

Scale both vectors by length of the line.

$line\_len = length(line\_vec)$ # 3.808

$line\_unitvec = unit(line\_vec)$ # (0.919,0.0,-0.394)

$$pnt\_vec\_scaled = scale(pnt\_vec, 1.0/line\_len) \text{ \# (0.263,0.0,-0.393)}$$

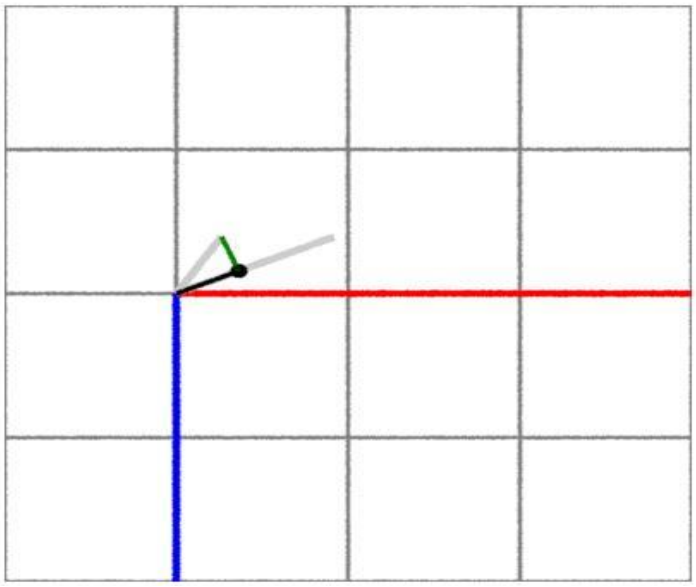


Figure 11

Step 3:

Calculate the dot product of the scaled vectors. The value corresponds to the distance, shown in black, along the unit vector to the perpendicular, shown in green.

$$t = dot(line\_unitvec, pnt\_vec\_scaled) \quad \text{\# 0.397}$$

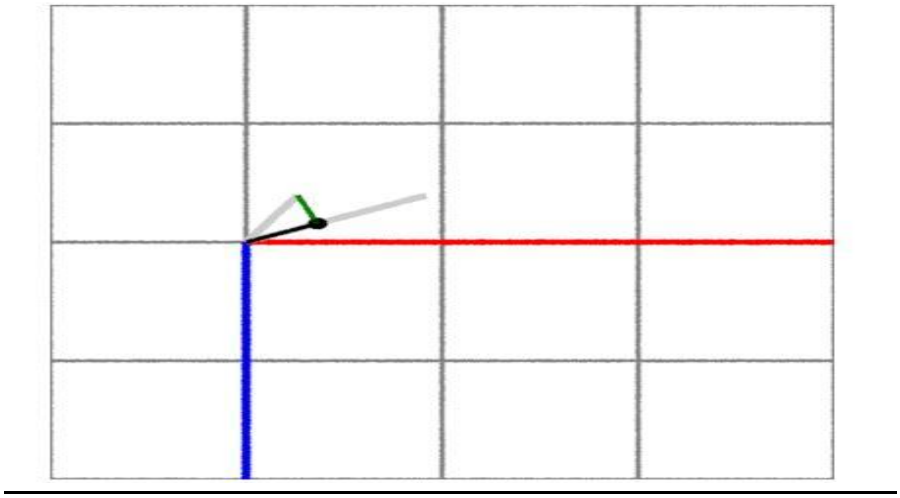


Figure 12

Step 4:

Clamp ‘t’ to the range 0 to 1. Scale the line vector by ‘t’ to find the nearest location, shown in green, to the end of the point vector. We calculate the distance from nearest location to the end of the point vector.

If $t < 0.0$:

$t = 0.0$

elif $t > 1.0$:

$t = 1.0$

$nearest = scale(line\_vec, t)$ # (1.388, 0.0, -0.595)

$dist = distance(nearest, pnt\_vec)$ # 0.985

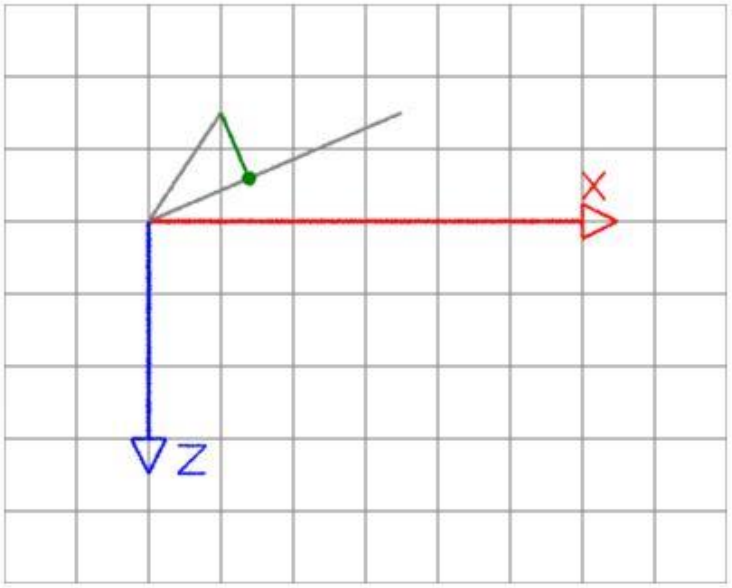


Figure 12

Step 5:

Translate the 'nearest' point relative to the start of the line. This ensures its coordinates "match" those of the line.

$nearest = add(nearest, start)$ # (2.388, 0.0, 1.405)

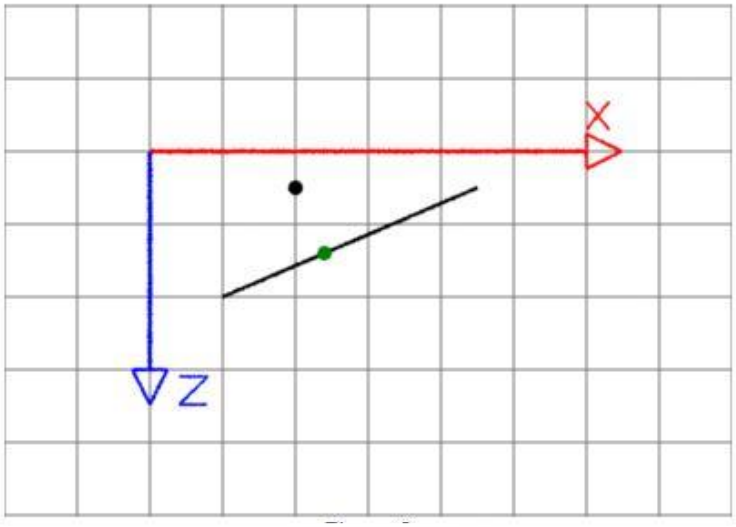


Figure 13

3.2.2 Algorithm to find nearest polygon from the given point

Searching nearest polygon will a worst case time complexity of O(n) and best case time complexity of O(1).

Algorithm

Step 1: Take root as polyset.list[0].

Step 2: dist1 = distance between 'root' and 'point'.

Step 3: poly1=root.

Step 4 while(1)

i) Break if all neighbours (if present)are visited .

ii) Check direction of 'point' from root.

iii) Make root's opposite direction as visited.

e.g. if direction is left , make root -> right as visited.

iv) Check distance of other three neighbor polygons from point.

v) Update 'dist1' with minimum distance got, and update root with nearest polygon.

Step 5: Display nearest polygon.

3.3 Finding left, right, top, bottom polygon with respect to the given point.

To get this we will leverage the results we get from previous algorithms. So, we already have the nearest polygon from the given point. Now we need to find all four sides nearest neighbors of the given point. For this, we will use the below-mentioned algorithm which has a constant time complexity i.e. O(1).

3.3.1 Algorithm :

Algorithm for Clockwise encircle: Direction [l,t,r,b]

Step 1: Suppose we are on ‘x1’, check the direction of ‘p’ from ‘x1’ .Let the direction be ‘d’.

Step 2: Check if any node is present in direction ‘d’ from ‘x1’.

Step 3: If a node is present and direction of ‘p’ from x2 is also ‘d’ then move to ‘x2’ and make ‘x1’ as visited ,add ‘x1’ to the list.

Step 4: If node is not present or direction of ‘p’ from’x2’ is not ‘d’ then , move to the node ‘x2’ in direction (d-1)%4. Add x1 to the list and make it visited, if no node is present in that direction than stop search.

Step 5: Go through the list and check in each direction which is nearest.

Algorithm for Anti-clockwise encircle : Direction [l,t,r,b]

Step 1: Suppose we are on ‘x1’, check the direction of ‘p’ from ‘x1’. Let the direction be ‘d’.

Step 2: Check if any node is present in direction ‘d’ from ‘x1’.

Step 3: If a node is present and direction of ‘p’ from x2 is also ‘d’ then move to ‘x2’ and make ‘x1’ as visited ,add ‘x1’ to the list.

Step 4: If node is not present or direction of ‘p’ from’x2’ is not ‘d’ then , move to the node ‘x2’ in direction (d-1)%4.Add x1 to the list and make it visited. If no node is present in that direction than stop search.

Step 5: Go through the list and check in each direction which is nearest.

3.4 Example

We take an example to explain the insertion of convex polygon, and how to find the left polygon, right polygon, top & bottom polygon for a given point.

Let us take a polygon set.

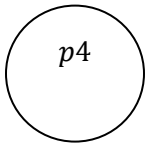
$$P = \{p1, p2, p3, p4, p5, p6\}$$

Here the polygons have 'n' number of vertices and can have any specific number of edges.

First we will do insertion of polygon in map.

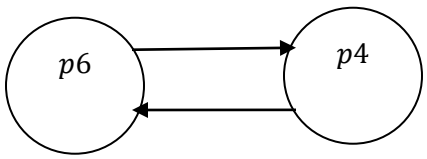
Step 1: Let us assume $p4$ comes first.

So the graph is:



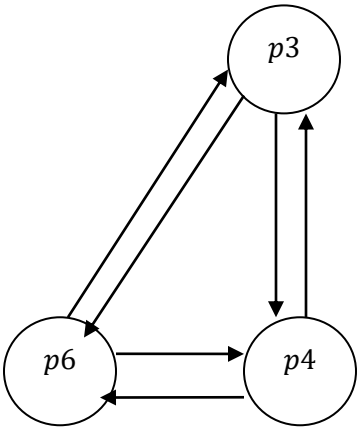
Step 2: Let us assume $p6$ comes after $p4$. Then we check, whether, $p6$ lies on which side of $p4$, using the equations. Let us assume $p6$ lies on left side of $p4$.

So the graph becomes



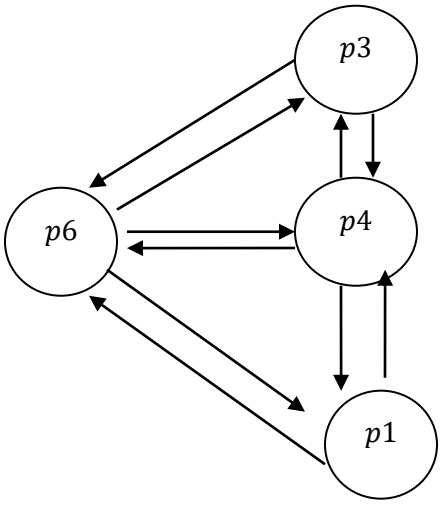
Step 3: Let us assume $p3$ comes next. So we will draw coordinate with respect to $p3$ and we will check the direction in which it will lie and we suppose $p4$ and $p6$ are at mutual direction at $p6$. So first we will compare it with $p6$, let us assume it lies on right side of $p6$. So now we will check with $p4$, (as $p4$ lies on right side of $p6$) whether $p3$ lies left, right, top or bottom of $p4$. Let us suppose it lies on top of $p4$.

So the graph becomes

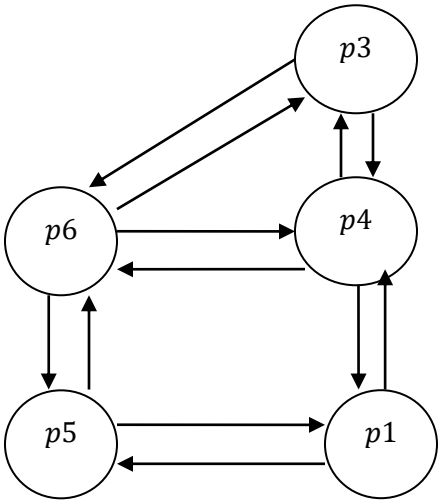


Step 4: Let us assume $p1$ comes next, first we will compare it with $p6$. Let us suppose it lies right to $p6$, then we will compare it with $p4$. Let us suppose $p1$ lies on bottom of $p4$.

So the graph becomes

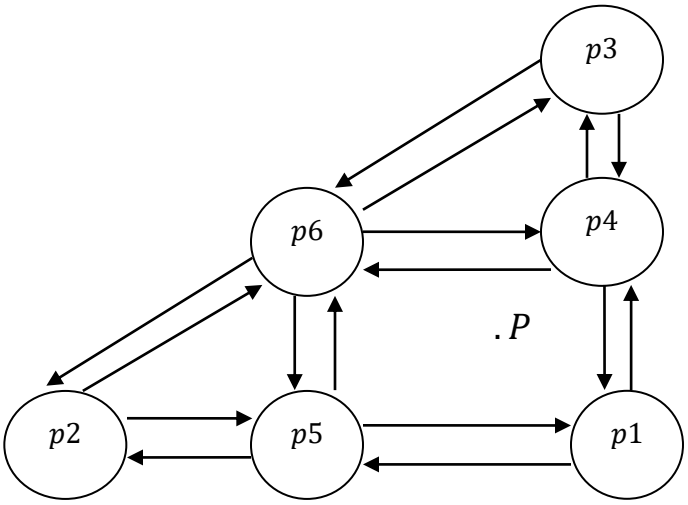


Step 5: Let us assume $p5$ comes next. First we will check with $p6$. Let us suppose it lies on bottom of $p6$.



Step 6: Let us assume $p2$ will be inserted next. First we will check it with $p6$. Let us assume it lies below $p6$. Then we will check with $p5$. Let us assume it lies on left of $p5$.

So the final graph is



Now we will find the nearest polygon to point p

Step 1: First we will check whether point 'p' lies in which direction with respect to $p6$ with the help of equations.

Step 2: Since point lies in right side of $p6$, so all the nodes on left, top, bottom of $p6$ is marked as visited.

Step 3: Now we start checking the distance of the point with rest of the polygons one by one, and we will store the distance in a list, and then we will do sorting and find the nearest polygon.

In this case $p4$ is the nearest polygon.

How to find top, bottom, left & right polygon:

Step 1: We will call encircle for p4 in clockwise direction. The nodes visited will be stored in list.

Step 2: Then we will call encircle for p4 in anti-clockwise direction.

Note-In some cases some nodes may be left, so we need to traverse in both way. We call encircle of order 2

Step 3: Now we will compare the direction and distance of the nodes stored in the list, and will be able to find left, right, top & bottom polygon.

3.5 Data Structure

We have created a graph of polygons (based on Trapezoidal maps), featuring many properties of trapezoidal maps like each node (polygon) of our polygonSet stores 4 neighbor nodes (polygons). Also the insertion of a node (polygon) is done in similar way.

The attributes of polygons are:

i) Sides (Number of edges)
ii) Vertices (list of vertices)
iii) Left (left neighbours i.e. polygon)
iv) Right (Right neighbours i.e. polygon)
v) Top (top side neighbor i.e. polygon)
vi) Bottom (bottom side neighbor i.e. polygon)
vii) Leftmost (Leftmost point of current polygon)
viii) Rightmost (Rightmost point of current polygon)
ix) Topmost (Topmost point of current polygon)
x) Bottommost (Bottommost point of current polygon)
xi) Visited (If visited or not)
xii) Center (It's the centroid of the polygon)

The methods of polygons are:

i) assignseg(obstacles) -> This method will assign values for sides and vertices.
ii) extremes() -> This point will assign all extreme points of the given polygon.
iii) printExtremes() -> This method will print all extreme points.
iv) printPoly() -> Prints all vertices of a given polygon.

The attributes of polygon set are:

i) List (list of polygons)
ii) Size (size of the graph)

4. Tools Used

You have probably noticed that the description of this algorithm is very abstract, especially concerning the steps inside the loop. While I was implementing a working version of this algorithm into the framework, I encountered many little problems to solve, I didn't even imagined to be there in the first place. The framework provided classes for points, segments and faces, geometry functions and a lot of other useful utility. Because the framework was written in Python, I also used Python to implement the algorithm .I learned a lot about object oriented programming, design patterns and the advantages of using them.

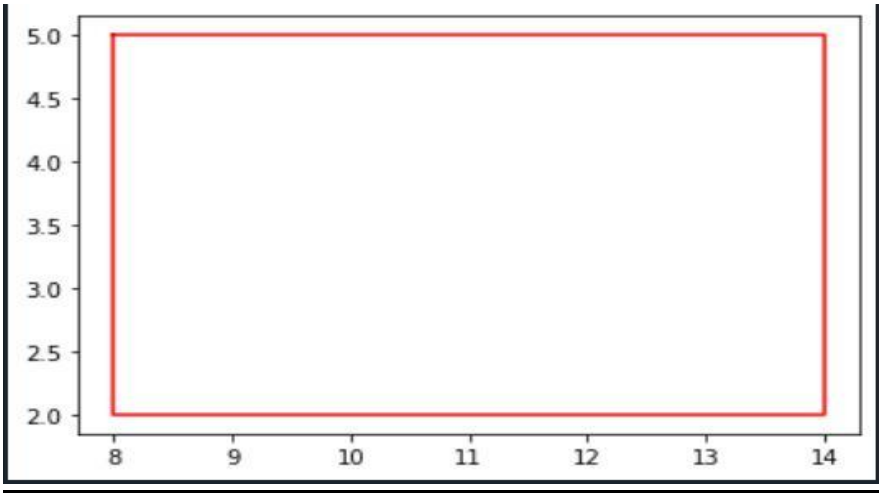
5. Output

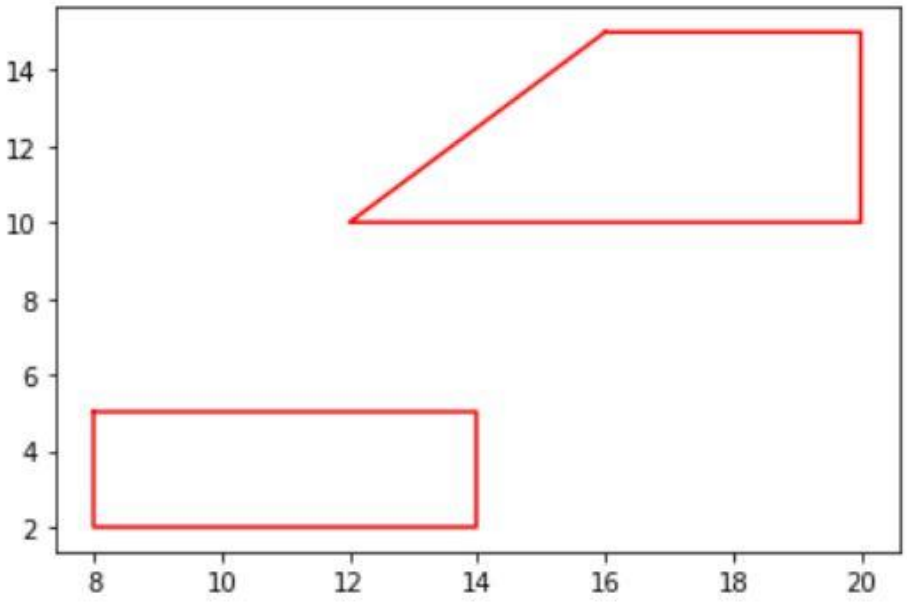
All the polygons from obstacle set are inserted one by one randomly.

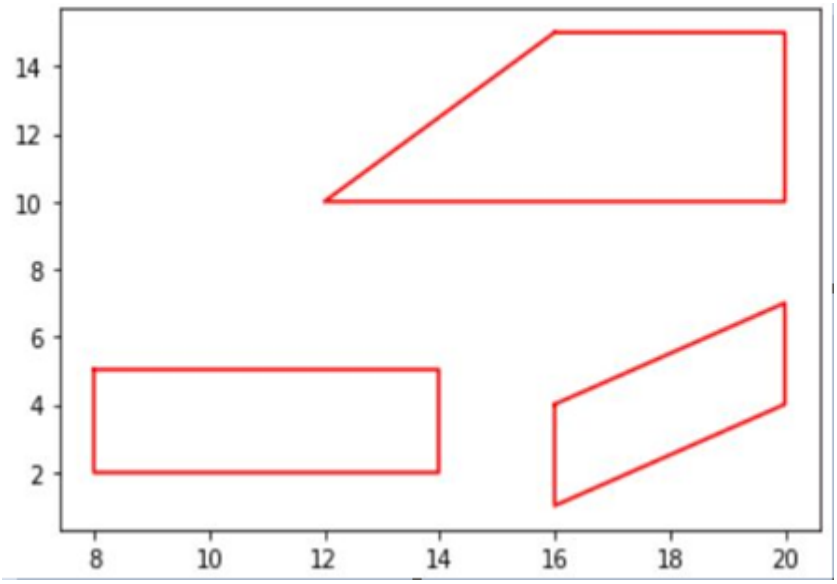
When we run the code second time the sequence in which the polygons

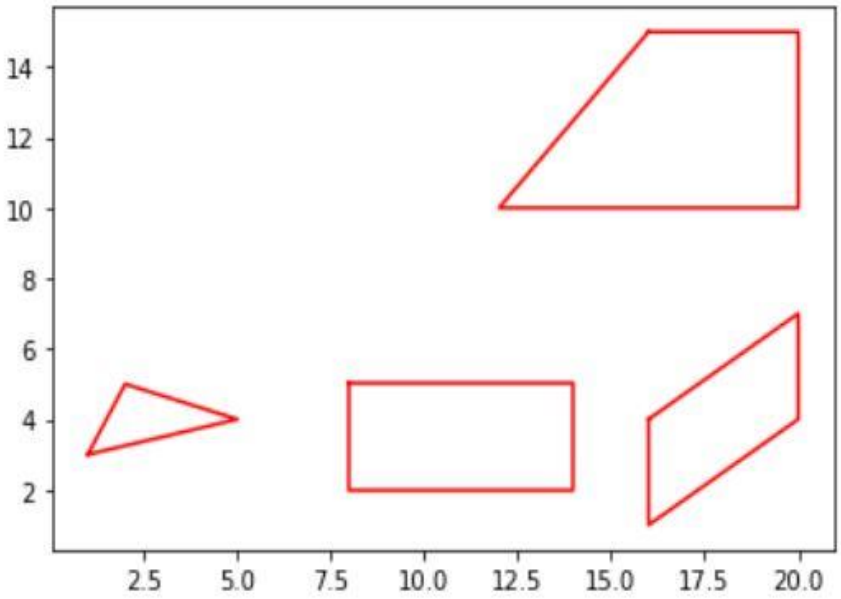
Will be inserted will be different.

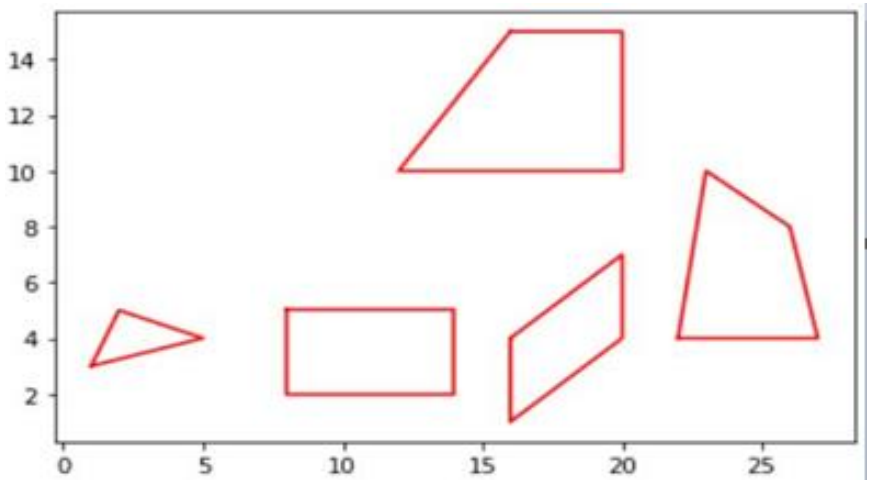
First output:

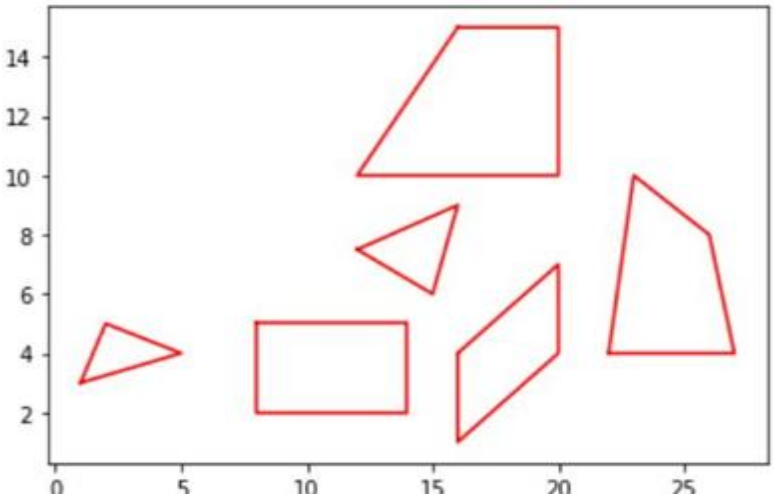




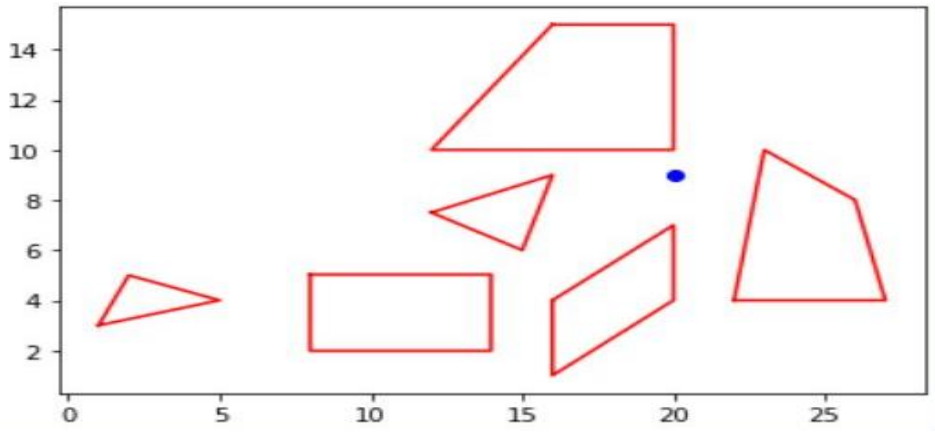

14
12
10
8
6
4
2
8
10
12
14
16
18
20


14
12
10
8
6
4
2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

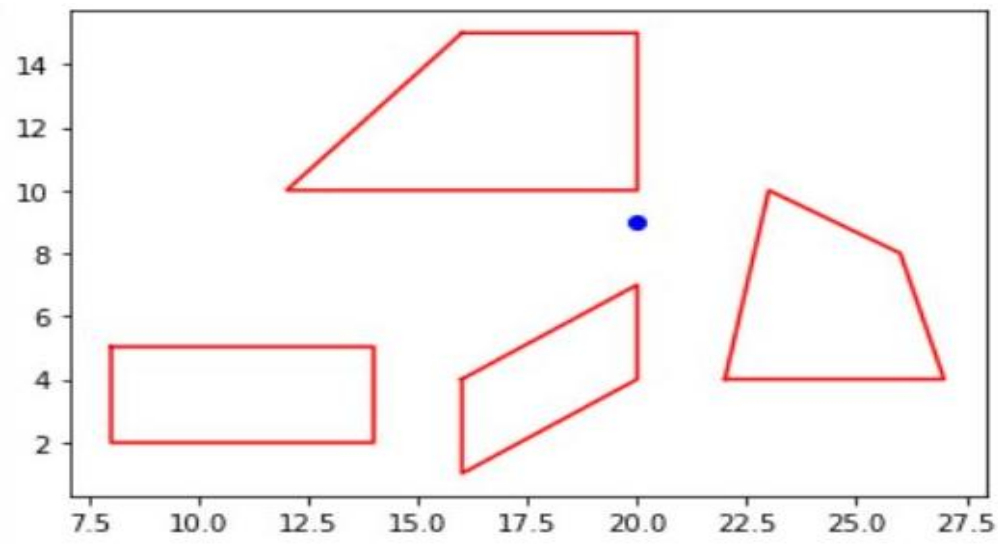




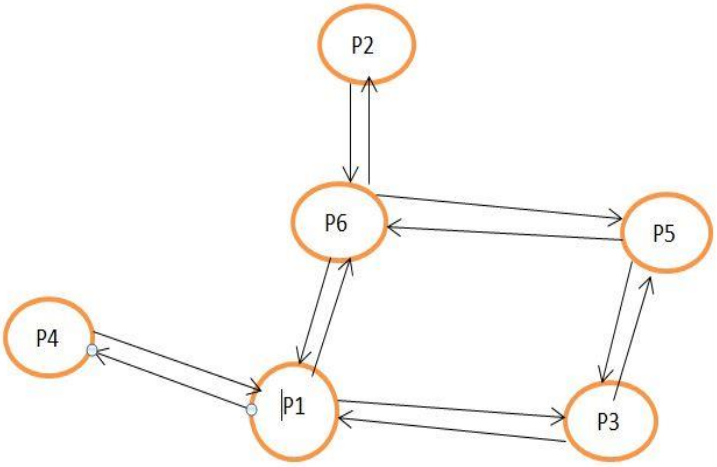
When all the polygons are inserted then we give query point. In this example we give query point as $q(20,9)$.



We get the left polygon, right polygon, top polygon and bottom polygon with respect to the given point.

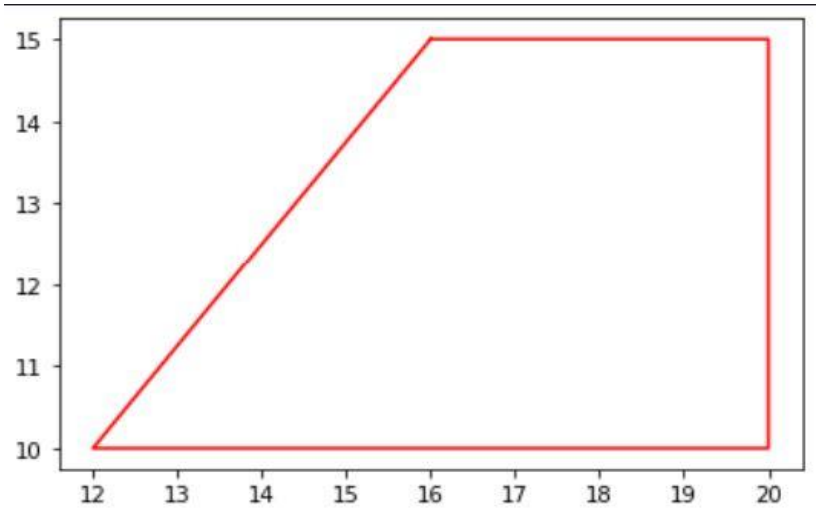


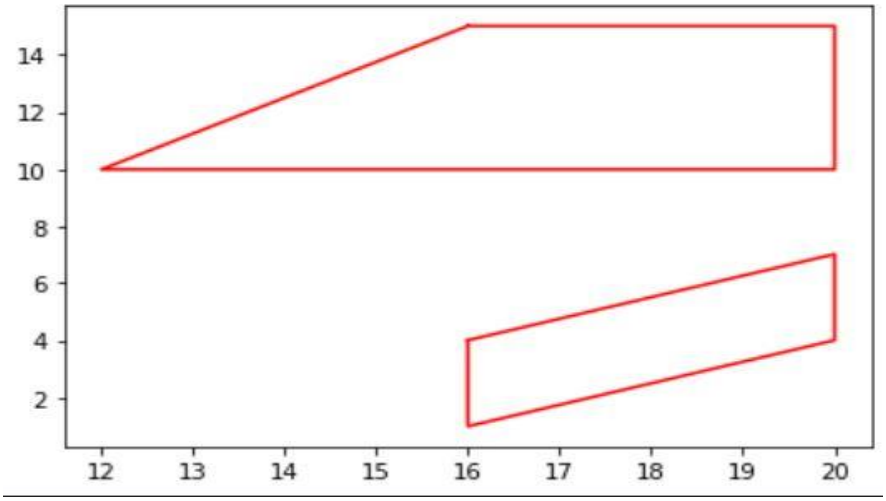
The graph formed for insertion is :

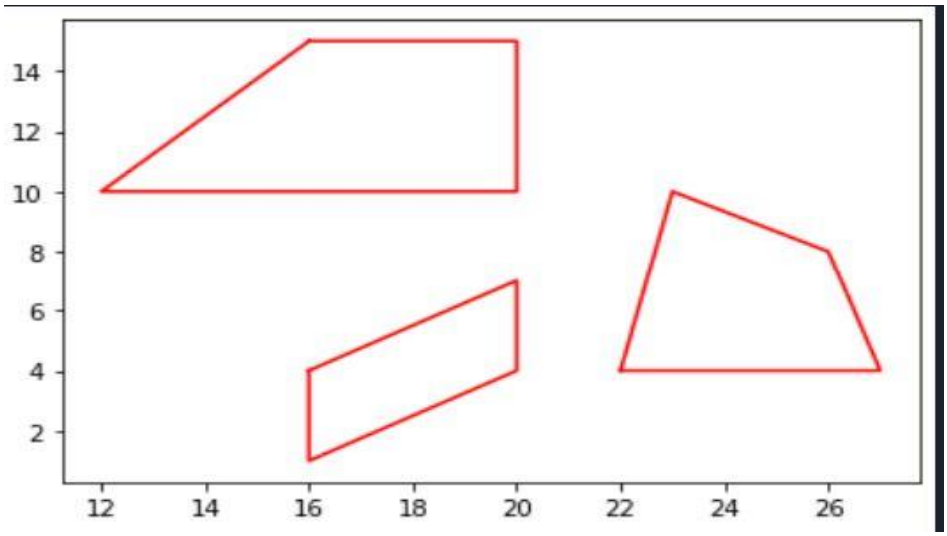


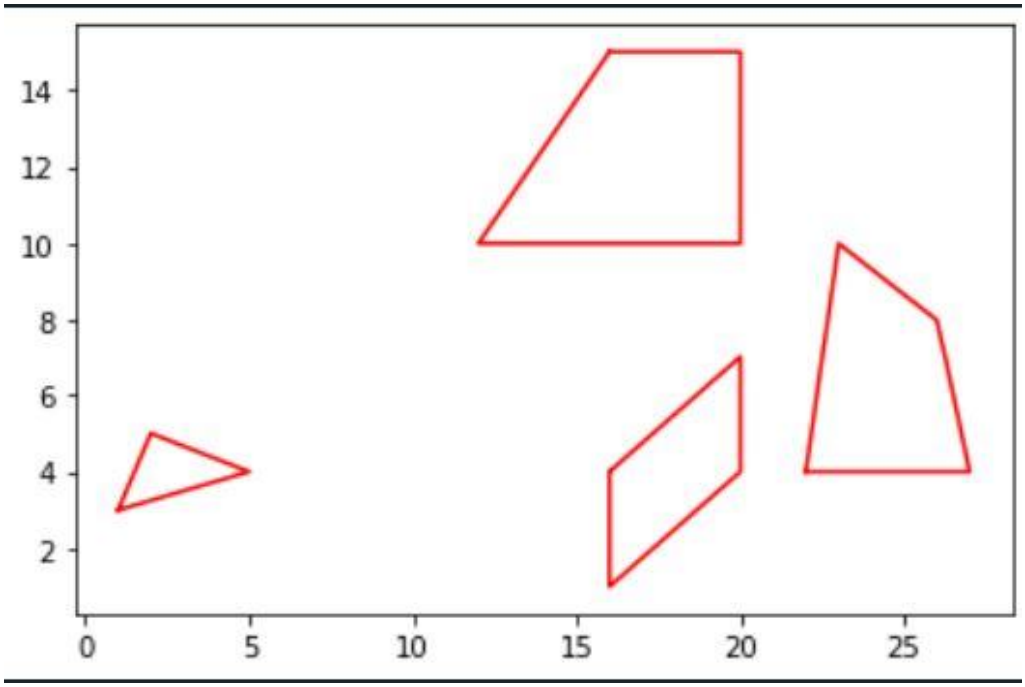
Second output:

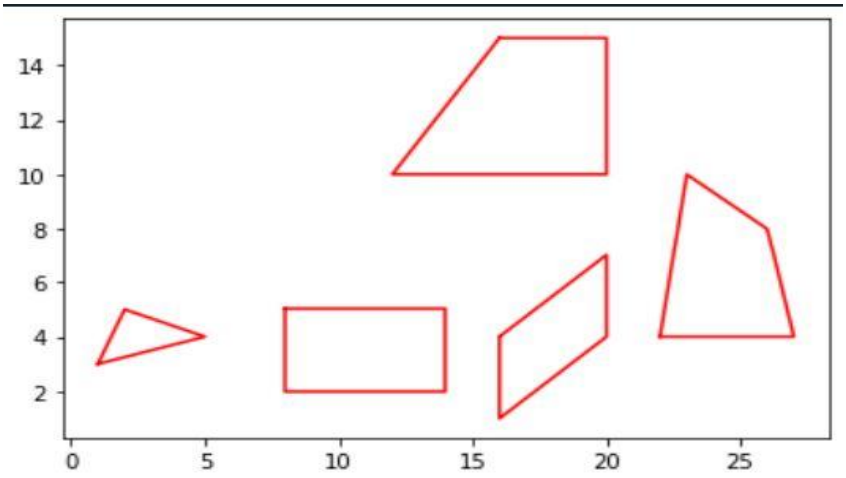
When we run the code second time, the output is:

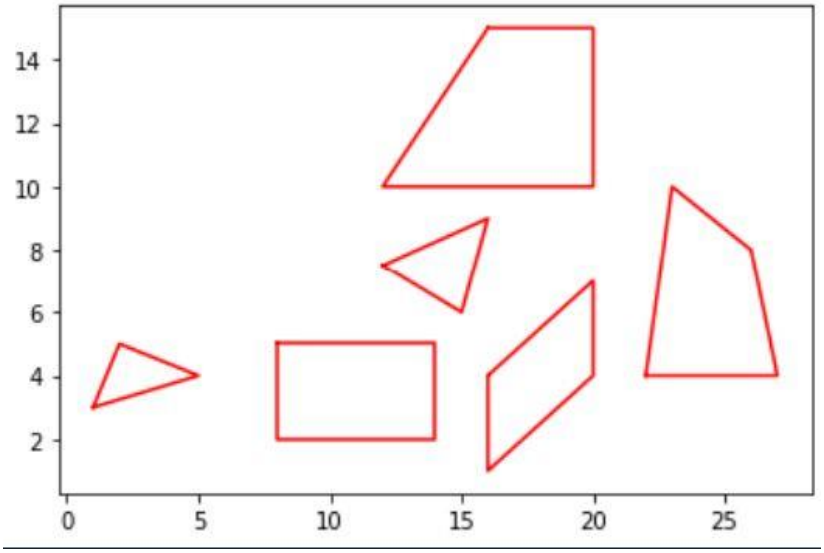


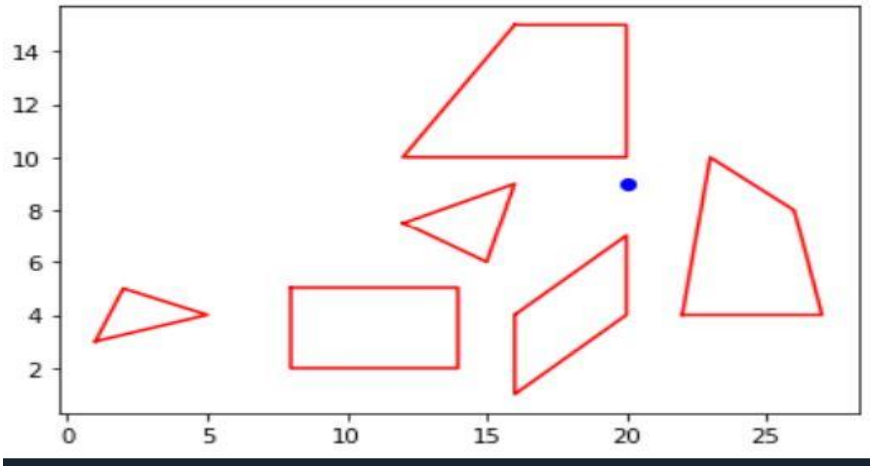

14
12
10
8
6
4
2
12
13
14
15
16
17
18
19
20

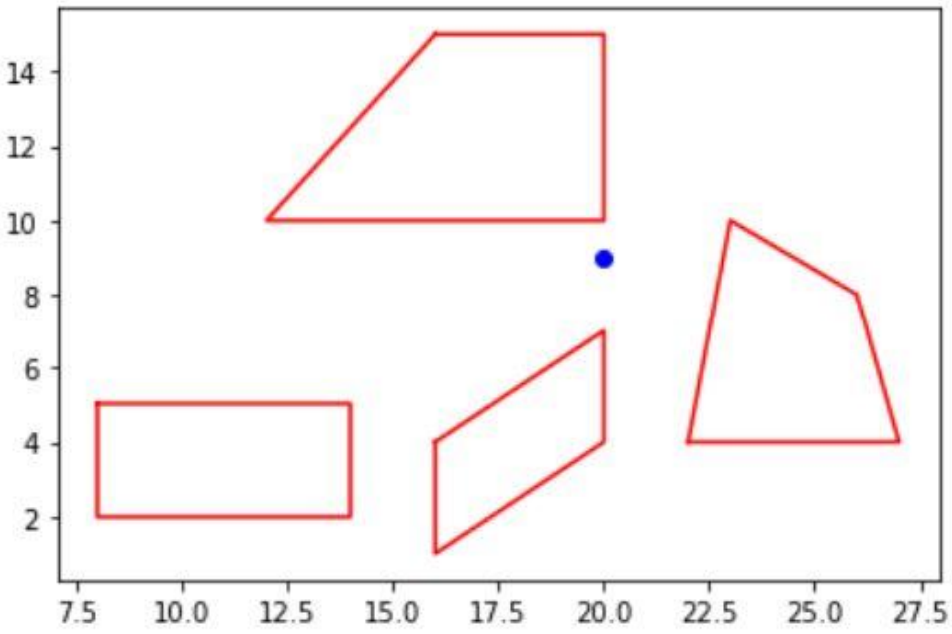

14
12
10
8
6
4
2
12
14
16
18
20
22
24
26


14
12
10
8
6
4
2
0
5
10
15
20
25

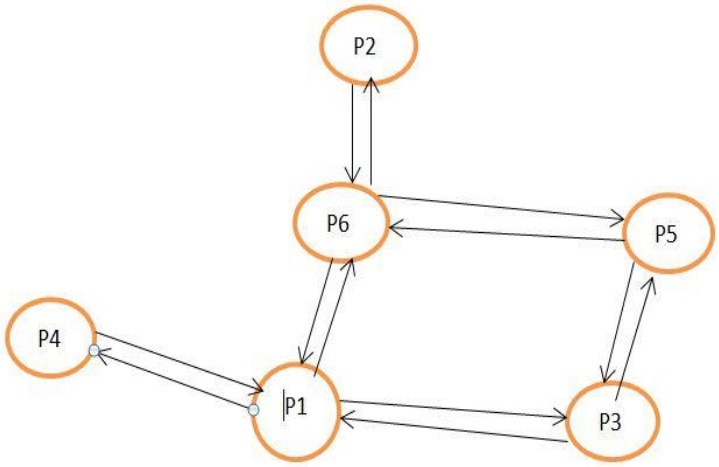

14
12
10
8
6
4
2
0
5
10
15
20
25


14
12
10
8
6
4
2
0
5
10
15
20
25


14
12
10
8
6
4
2
0
5
10
15
20
25



The graph formed for insertion is :



6. Conclusion & Future work

This research aimed how to decompose the Trapezoidal map into polygons. And how to insert polygons in trapezoidal map dynamically and making a search structure for it, given a point ‘q’ use the search structure for it. Further this approach can be tested on some other real life Trapezoidal Map problem and see how it does on various parameters. Also deletion of nodes can be implemented for this.

7. References

1. Said, Broumi & Smarandache, Florentin & Talea, Mohamed & Bakali, Assia. (2016). “Operations on Interval Valued Neutrosophic Graphs. New Trends in Neutrosophic Theory and Applications”.

2. Cui, Can & Wang, Jiechen & Ma, Jinsong. (2010). “A New Method of Applying Polygon Boolean Operations Based on Trapezoidal Decomposition. GIScience & Remote Sensing”. 47. 566-578. 10.2747/1548-1603.47.4.566.

3. Lorenzetto, Gian & Datta, Amitava & Thomas, Richard. (2001). “A Fast Trapezoidation Technique For Planar Polygons. Computers & Graphics”. 26. 10.1016/S0097-8493(01)00180-7.

4. Teillaud, Monique. (1993). “Towards dynamic randomized algorithms in computational geometry”. 10.1007/3-540-57503-0.

5. Amato, Nancy & Goodrich, Michael & Ramos, Edgar. (2001). “A Randomized Algorithm for Triangulating a Simple Polygon in Linear Time. Discrete and Computational Geometry”. 26. 245-265. 10.1007/s00454-001-0027-x.

6. Teillaud, Monique. (2000). “Union and Split Operations on Dynamic Trapezoidal Maps. Computational Geometry.” 8. 153-163. 10.1016/S0925-7721(00)00019-5.