Logo, company name

Description automatically generated

CS 478 HOMEWORK2  
  
ALP TUĞRUL AĞÇALI

21801799

Q1-)To begin, we need to find the rightmost and leftmost point of one of the polygons (P). Then let's find the points one right and one left of the left vertex we found. (The starting point of the edge coming from the vertex and the ending point of the edge going from the vertex.) Let's call these points the right child and the left child. Then let's do the same actions for the Q polygon.

1-) First, let's see if there is an intersection between the points to the right and left of the leftmost vertex of the P polygon and the points to the right and left of the leftmost vertex of the Q polygon. Let's add these intersection points and intersecting lines to J, if any. Let's equate the right or left child of the ppolygon to the current node (whichever has the smaller x coordinate) and then equate the x point of the child we equated to the sweepLine. Next, we can start a while loop, which should continue if Pleftchild and Prightchild are not equal to Prightmost.

2-) The first thing we will check in the loop is whether the sweppline is ahead of the point to the right of its child to the right or left of the leftmost of Q . If it is not in front, checking whether the edge on the right side of the current Node and the edge on the two edges of the root intersect. If they intersect, let's assign the intersection points and and intersecting lines to j and skip step 3.

3-) If the right or left child of Q's leftmostU is ahead of the sweepline, then it is checked whether the right and left children of current Q (smaller x child) and the edge on the right of Current'node intersect. If there is an intersection, it is added to J.

4-) In this interval, it is checked whether the point to the right of the right or left child of Q is in front of the sweepline. next step if not ahead. If it is in front of it, the point to the right of whichever point is ahead, equates that point to the point in front and returns to the beginning of the loop (2nd step).

5-) In order to avoid infinite loop in the loop, the X points of the right and left children of P are compared. The smaller X becomes the current node and the selected child is synced to a point one to its right. The sweep line is updated.

In this algorithm, the edges of the Leftmost vertex are examined separately. Because it is important to compare two sides of them and only the right side of the others. In addition, the point events of P and Q are propagated separately. This is done in the Q polygon (which does not produce the currentNode) to avoid forgetting the lagging edges.

Selected vertices can be used to find edges. The algorithm goes over these points on its second turn. From each intersection to the next intersection, the edges of the polygon on the left at the right child and on the right at the left child form the border.

Q2-) The best way to do this is to use an algorithm like the combination of Interval tree insertion and Query algorithm. Thus, each of the N intervals can be calculated in O(logN) time.

InsertAndReport Function

For all intervals, if start ≤ M(v) ≤ end, add the start, end pair to L(v) and R(v). Adjust the secondary search tree pointers as needed. Report all intervals in L(v) with the newly added interval. Then call the known QueryIntervalTree function for left pointer and right pointer. (Because the addition has been made, now only reporting remains).

If end < M(v), report every point in L(v) whose left point is less than the end then call InsertAndReport(This function) for leftChild (because the insertion hasn't been done yet).

If M(v) < start, report every function whose right point is greater than the start and call the InsertAndReport function for the Right child.

Function InsertAndReport(start, end, root):

If (start ≤ M(v) ≤ end):

Add [start, end] to L(v) and R(v)

adjust the secondary search tree pointers as needed

report all points in the L(v)

QueryIntervalTree(start, end, Lptr(v))

QueryIntervalTree(start, end, Rptr(v))

Else:

If((e < M(v)):  
 i = 1

while (the ith interval [li , ri ] in L(v) has li ≤ end) :

report [li , ri ]

i++

InsertAndReport(start, end, Lchild(v))

Else: // M(v) < start

i = 1

while (the ith interval [li , ri ] in R(v) has start ≤ ri) :

report [li , ri ]

i++

InsertAndReport(start, end, Rchild(v))

Q3-) We can use Priority queue for events and Interval tree for rectangles. Add all events to the priority queue. A point is event if it has an edge goes to upside. Start event if an edge from it goes to right and end event if an edge goes from it to left. All non-event points goes in non-event priority queue. Create rectangle structure for tree node (it is going to be in AVL trees, with startY and endY). Create a rectangle list halfRectangles. Pop from eventqueue, if it is a start event add it to halfrectangles. Pop from non-event queue and set added half rectangles upper start point. If the event finish event find related start event and finish the rectangle with a pop from non-eventqueue. After finishing Add it to tree according to it’s x values (start-end). Then calculate nodes of the tree. For each node, calculate all rectangles and remove common parts of rectangles. ( as it calculates the set sum)  
  
In this algorithm, a certain interval can be given for the Interval tree and the Node can be sent as a parameter. But this may not be enough for the Avl Tree inside. There are two solutions: one is to implement a new avl tree and the other is to use a simple List.

Q4-) The best way to do this is to create an interval tree. Segments in S can be placed in the interval tree according to the start and end x coordinates. Later, the segment or segments of the query segment can be found by search.

Segments can be placed in the interval tree according to the range of the X coordinate. Then the queryIntervalTree function runs according to the x range of the query segment and the segments whose x coordinates overlap are found. This arrangement allows us to test only segments whose X coordinates overlap, instead of testing N segments one by one. reduces the worst-case complexity of the algorithm from O(N) to O(LogN+K). This drop is very useful on large planes with large number of segments.

5-) In order to achieve this algorithm, we first need to record some values. These are the maximum y coordinate and minimum y coordinate of the segment with the smallest x coordinate, the minimum y coordinate of the segment with the largest x coordinate, and finally, the point where the highest point is the minimum value among all the lines except the first and last, and the point whose lowest point has the maximum value. We can find these in a loop that tests all the lines.

Function IsThereLine(Segments):

Minx = infinite

Maxx = -infinite

minxMaxy

minxMiny

maxxMaxy

MaxxMiny

minYAmongHighest

maxYAmongLowest

for each segment in segments:

if (segment.x < minx):

minx = segment.x

minxMiny = segment.miny

minxMaxy = segment.maxy

continue

if (segment.x > maxX):

maxx = segment.x

maxxMiny = segment.miny

maxxMaxy = segment.maxy

continue

if (segment.maxy < minYAmongHighest ):

minYAmongHighest = segment.maxy

If (segment.miny > maxYAmongLowest):

maxYAmongLowest = segment.miny

We need to create a chain using minYAmongHighest and maximum y points of the first and last. We should create one of the same using the lowest Y points. If these two chains intersect, it is impossible for a line to pass through them. If it does not intersect, there is one more test to be done.

UpperChain = createChain (minxMaxy, minYAmongHighest, maxxMaxy)

lowerChain = createChain (maxxMiny, maxYAmongLowest, maxxMiny)

if (intersect(upperChain, lowerChain):

return false

If these lines do not intersect, there must be another Loop that traverses all the lines. The purpose of this loop is to check if there is a maximum point below the lower chain we draw or if there is a minimum point above the upper chain we have drawn. (The purple line shown in the image is an example of one of the lines we mentioned. And it prevents the middle line from being drawn. The green line in the image is to show that even though that point is not actually the lowest between the upper limits, it prevents the line from being drawn. The red and blue chains drawn represent the chains we have just drawn. .) A picture containing line, diagram

Description automatically generated

For each segment in segments:

If(lowerChain.isPointunder(segment.maxY)

Return false

If(upperChain.isPointAbove(segment.miny)

Return false

Return True

If the last test is also successful, a line can be created that cuts all these segments. This algorithm uses two loops that test all segments O(N) (from O(2N)). The createChain function used in the middle is O(1), the intersect function is O(1) (since there are always two chains and 4 segments), and the isPointUnder and isPointAbove functions are o(1). So, we find the total time complexity as o(N).