Lecture 5 and 6

*

September 13, 2010

Abstract

The previous lecture showed various ways of finding points of intersection between a given set of line segments. The lecture 5 analysis an output sensitive algorithm. Lecture 6 introduces yet another interesting problem typical to computational geometry. We are given plane contains which contains many rectangles possibly overlapping.The objective is to calculate the total area coverage.

1 Lecture 5 : Analysis of Line Intersection Problem

1.1 Introduction

Here we will present an analysis of the output sensitive line sweep method for finding the intersection points of given set of line segments. The choice of data structure plays a crucial part in worst running time complexity of the algorithm. The structures in question are L(x) and event queue both f which are sorted sets. Thus we need data structure that efficiently store and update (dynamic) sorted sets.

1.2 Data-Structure

As mentioned a Dynamic dictionary which can efficiently handle changes is required to implement L(x) and the $event - queue$. Following are few viable choices.

- **red-black tree** A red-black tree is a type of self-balancing binary search tree. It can search, insert, and delete in *O(log n)* time, where n is total number of elements in the tree.
- **AVL tree** It is also a self-balancing binary search tree. An the balance is achieved by rotation. The worst time for insertion, search and delete is *O(log n)*.
- **skip List** A skip list is a data structure for storing a sorted list of items, using a hierarchy of linked lists that connect increasingly sparse subsequences of the items. The worst case of updates might be huge, but the possibility of that happening is very low. In average it take *O(log n)*.

*Department of Computer Science and Engineering, IIT Delhi, New Delhi 1100116, India.

1.3 Analysis

The algorithm goes through as many steps as there are event points in the $event-queue$.Suppose there where n line segments in the set. Let I be the set of all intersection. Thus, the total number of event points (where the L(x) changes)

$$= \quad 2n \quad + \quad I$$

The event queue is a dynamic sets it can change under the following situation.

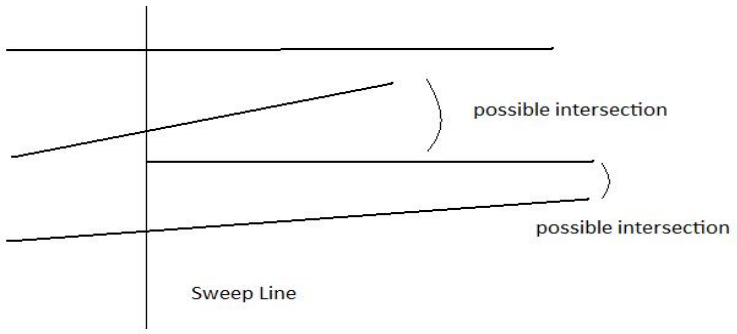
- **Case I** a start of a segment.
- **Case II** a end of a segment.
- **Case III** a Intersection point.

At any event point we update the L(x).

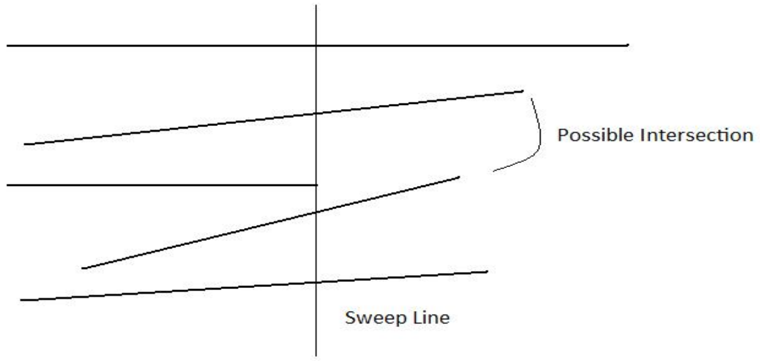
Lemma: *At every event point only constant number update is required.*

Proof

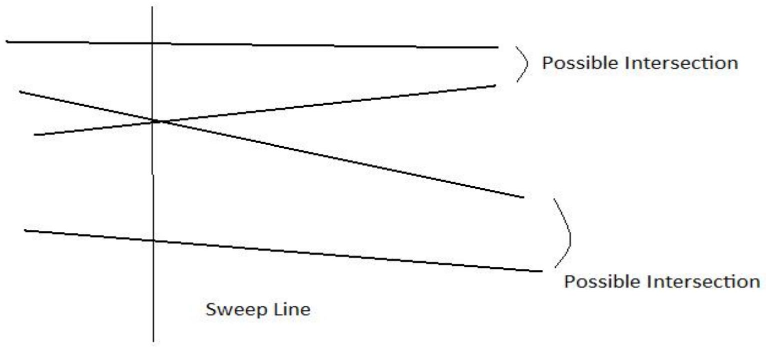
- **Case I:** a start of a line segment.
  We have at most two potential intersection. Thus one insertion and another two possible insertion to L(x).



- **Case II:** a end of a line segment.
  We have at most only one possible intersection. A deletion of object from L(x) and another possible insertion



- **Case III:** an intersection point
  The order of the lines involved in the intersection gets cahnge in L(X). Thus we need two delete and two removal operation. Since now we have possibility of two more intersection , we may had to add two more points to the event queue. Again constant number of operations.



Thus, the time taken in the update at each iteration is

$$O(log\ n)$$

This gives the total complexity of the entire process

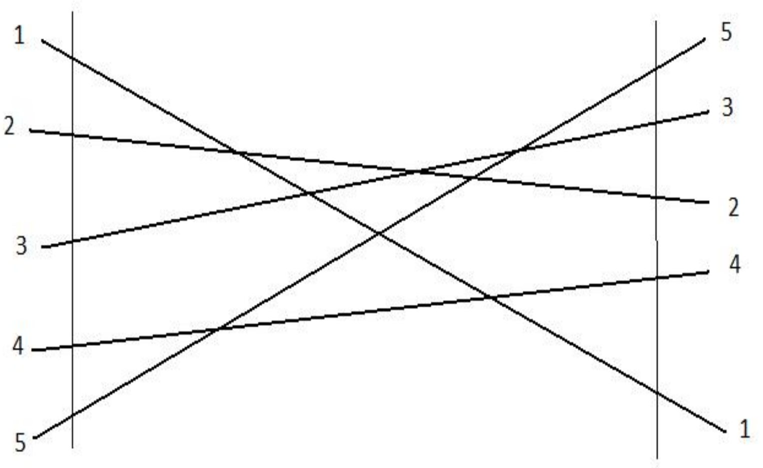
$$O((n + |I|)log\ n)$$

Which shows that the algorithm is actually is output sensitive.

1.4 Conclusion

The algorithm has assumed several simplification which helped in clear understanding. For example, the case of vertical lines. It needed to be mentioned that such situations can be handle by rotation of the axis along which the plane is swiped. another problem we might face is,what happens if more than two line intersect at the same point? We can shift the lines infinitesimally and preserve the condition that no two line intersect at the same point.

Another variation is that instead of reporting all the intersection, it is required to find just the number of intersections. The problem's complexity is no longer bounded from below on the number of intersection, after all the output is just a number. Consider a simplified a scenario. Take two vertical lines and consider the region between them. No line segment start or end with in this region.



The number the line segment in the order in which the intersect the left vertical line. The number of intersection is the number of inversion in the sequence of line segment listed in the order in which they intersect the right line. Thus, it turns down to calculating the number of inversion given a permutation of a sequence. The best method will be to do a binary search on the original sequence and find the difference in position, which gives the number of inversion with respect to the element.(See assignment sheet for complete explanation)

Before we close this section it is worth mentioning few important application of the algorithm. (i) super-imposition of two planar graph produces another planar graph, the algorithm can be used to calculate the new graph. (ii). ATC(Air Traffic Controller) uses similar algorithm to control flight path of aircraft in around the airport air space. Though the problem is now plotted in four dimensions, three of space and one of time.

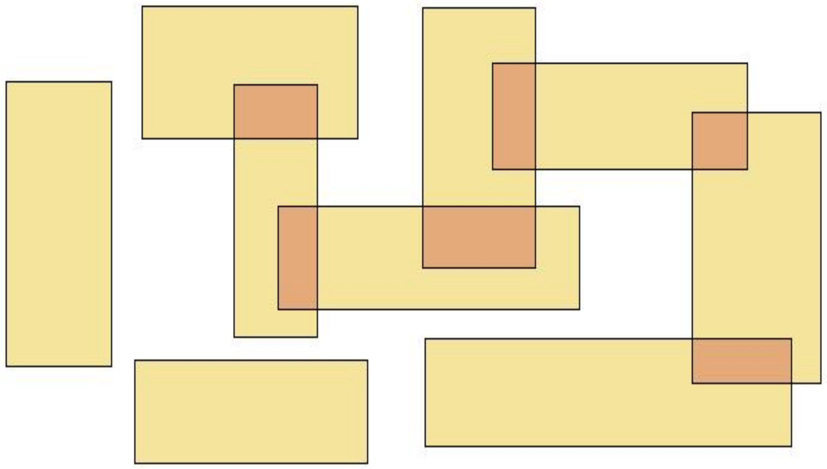
2 Lecture 6: Area of Union of rectangles

2.1 Introduction

We start this lecture by introducing a very instructive problem of computational geometry. It reveals the usefulness planes sweep in its full glory and uses additional data Structure to keep track of information necessary to find the union of areas.

Union of area of rectangles.

Given a layout in which objects are orthogonal polygons with sides parallel to the axises. The task is to the area covered by all the objects.



2.2 Algorithm

The problem is solved by line sweep technique. Consider the sweep line to be a vertical line. We have to some how maintain the intersection of the the polygons with the sweep line. Let y be the sum of all the intercepts. If we consider a sweep of distance $\triangle\ x$, in which the intercepts remain unchanged, then area swiped by the sweep line is

$$\triangle\ A \quad = \quad y.\triangle\ x$$

Total area would then be nothing but the sum of these quantities

$$A \quad = \quad \Sigma y.\triangle\ x$$

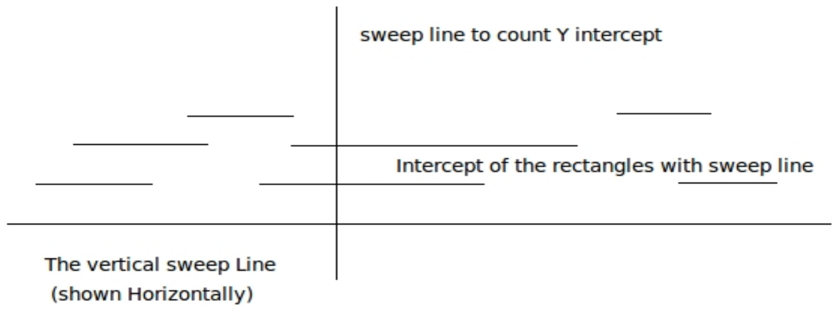
Thus the problem at hand becomes to maintain the intercepts. The y can change only at

1. The beginning of a polygon.
2. The end of the polygon.

These become the event point of the algorithm. Now, we will look into various method of keeping track of these y's. At any moment of the sweep all intercepts are on the sweep line. The total length of the intercept on the line is the our required y. This is what makes line sweeps so remarkable. It has the potential to convert the two dimensional problem into a series of one dimensional problems.

2.3 Naïve Method

So, how do we calculate the intercepts.Well one way of chasing the problem is to execute another line sweep algorithm on the sweep line and add the total intercepts.Here is the simple algorithm


sweep line to count Y intercept
Intercept of the rectangles with sweep line
The vertical sweep Line
(shown Horizontally)

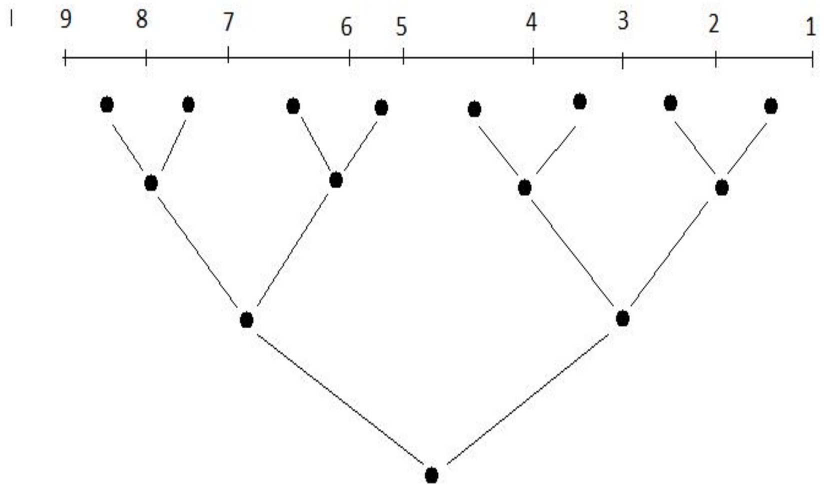
- We sort the end points of the intercepts. All these points become the event points.
- We keep a counter to keep track of the overlaps. Increase the counter when we encounter a start of an intercept and decrease when we encounter the end.
- We add to y if the count is greater than zero.

The line sweep method give us a time complexity of $O(n.log\ n)$. This operation need to be repeated $2n$ (each for the start and end of the polygons) times during the course of the sweep *(the original sweeping of the polygons)*. Thus, the total complexity of the algorithm is

$$O(n^2.log\ n).$$

2.4 Interval Tree Method

Another clever way of tackling the problem is maintain a special data structure called the $interval - tree$ to keep track of the y. The tree is constructed the following way.



1. enumerate all the start and end of the intercepts.
2. sort them.
3. make all the intercepts as the leaves of the interval tree.

4. each node stores the information whether the interval is on the sweep line or not.
5. each node also contain a dividing parameter that stores that stores that end of the right most leaf interval.

So, as the sweep line encounter a starting edge, it adds it to the data structure and increases the value of the y appropriately. And removes an intercept when the end of the rectangle is encountered. The following lemma shows what makes it a more efficient algorithm.

2.5 Analysis

Lemma : *Each interval is stored in at the most 2.log n nodes.*

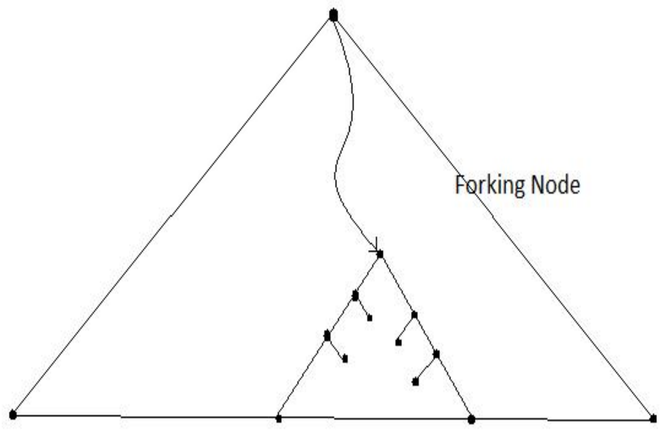
Proof : The following observation justifies this claim

- An interval does not occupy two sibling nodes. The parent node is selected
- If two nodes represent part of an interval the are side by side.
- This make the any two nodes representing parts of the interval adjacent node of different parents. th
- Since an interval is continuous and must occupy adjacent nodes in the same level of different parents, there are at the most two nodes occupied per level.
- There are *log n* levels, thus there can be at the most *2.log n* nodes that are occupied by an interval.

Lemma : *The insertion and deletion of each segment can be done in O(log n)*

Proof: The update requires the interval to be searched first. The search goes along a path through the tree till it forks. The forking condition being that dividing parameter lies between the interval to be updated. There are three cases

1. The interval lies entirely to the left of the node. In which case we only travel along the height of the tree.
2. The interval lies entirely to the right of the node. In which case we only travel along the height of the tree.
3. There is a fork. and the interval gets divided into two , one going to the left subtree and the other to the right. After the first fork any other fork in the left path will include the entire subtree,(other wise there will be gaps) and hence only one node need to be marked. Similarly, after the first fork any other fork in the right path will include the entire left subtree.



Thus at the most two nodes are visited per level. There are *O(log n)* levels. Which gives the required bound.

Similarly deletion of an interval employing the same strategy of finding the nodes will be done in *O(log n)* time. These two lemmas gives us a data structure that maintains the interval information in *O(log n)* time. The line sweep updates the data structure at each event points and there are *O(n)* event points. Thus the time complexity of the entire algorithm is bounded from above by

O(n log n).

References