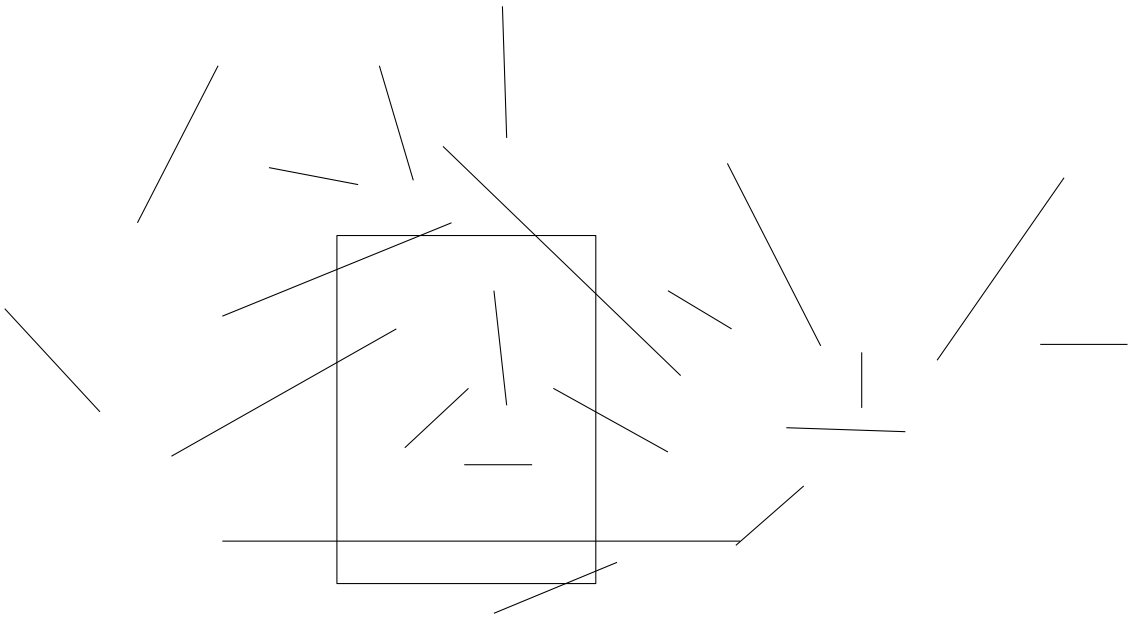
Windowing problem for a set of Tilted line segments

Kapil Bari

Chaitanya Reddy T

Problem Statement ::

Using Priority Search Trees and Segment Trees Answering Windowing/ Rectangle Queries for a set ofTilted Line Segments.



There are two cases, first, atleast one segments endpoint lie inside the window. Second, no segements endpoint inside window but intersecting with window. First case is trivial can be solved using range Trees in O(log2n + k') time. Here, I discussed second case using segment trees.

Steps for Preprocessing Tilted Line Segments ::

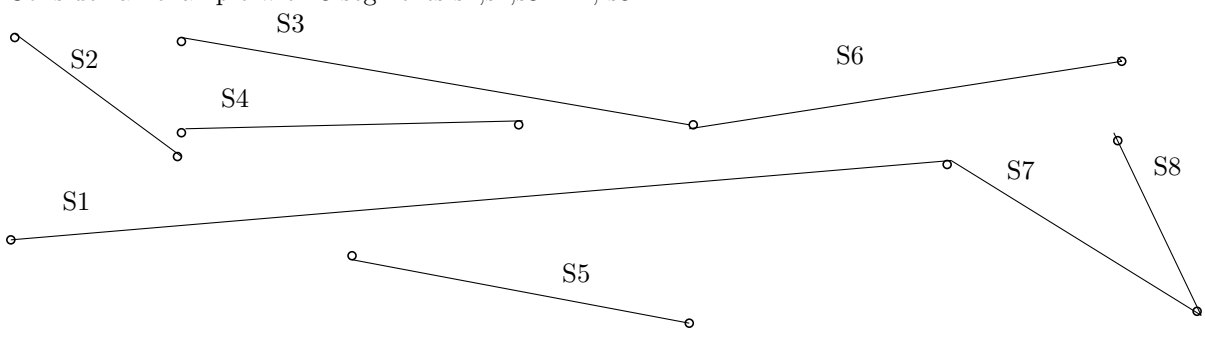
1. Building Segmenmt Tree for Tilted line segments considering only x- cordinates of line Segments.
2. Building Priority Search Tree in every node of Segment Tree for existing segments on nodes.

Building Segment Tree

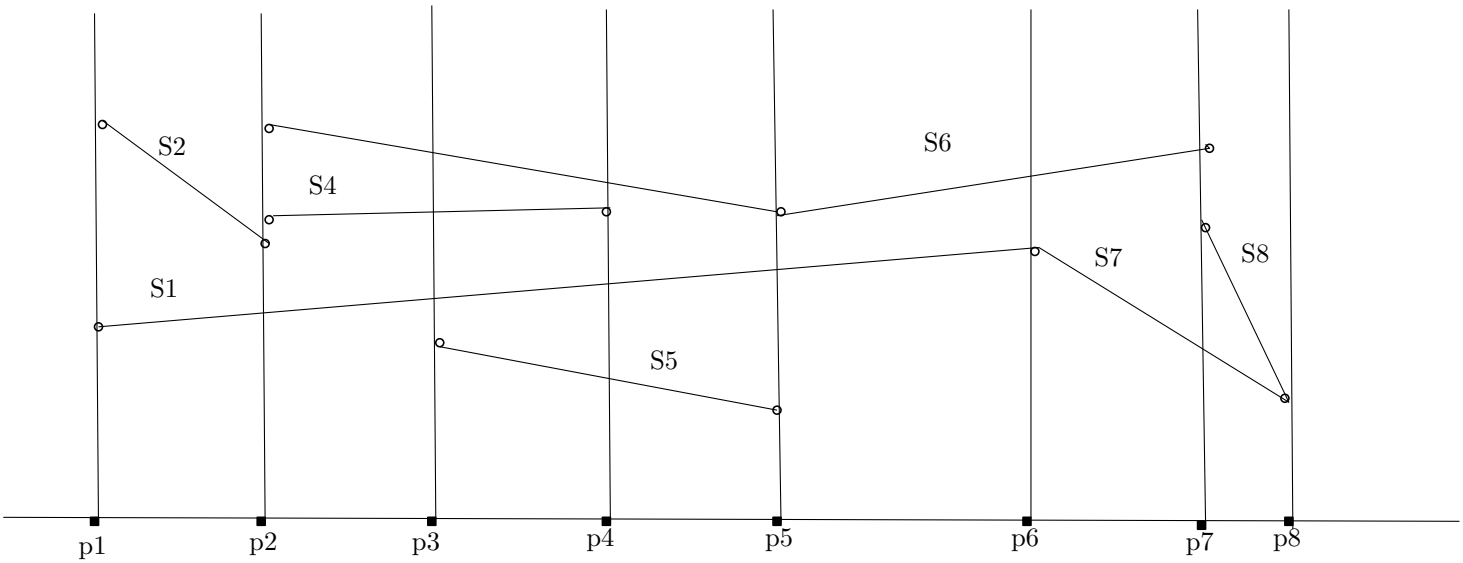
Let I := [x1 : x'1], [x2 : x'2], . . . , [xn : x'n] be a set of n intervals on the real line. The data structure that we are looking for should be able to report the intervals containing a query point qx . Our query has only one parameter, qx , so the parameter space is the real line. Let p1 , p2 , . . . , pm be the list of distinct interval endpoints, sorted from left to right. The partitioning of the parameter space is simply the partitioning of the real line induced by the points pi . We call the regions in this partitioning elementary intervals. Thus the elementary intervals are, from left to right,

(:p1),[p1:p1],(p1:p2),[p2:p2]..... (pm-1:pm),[pm,pm],(pm:)

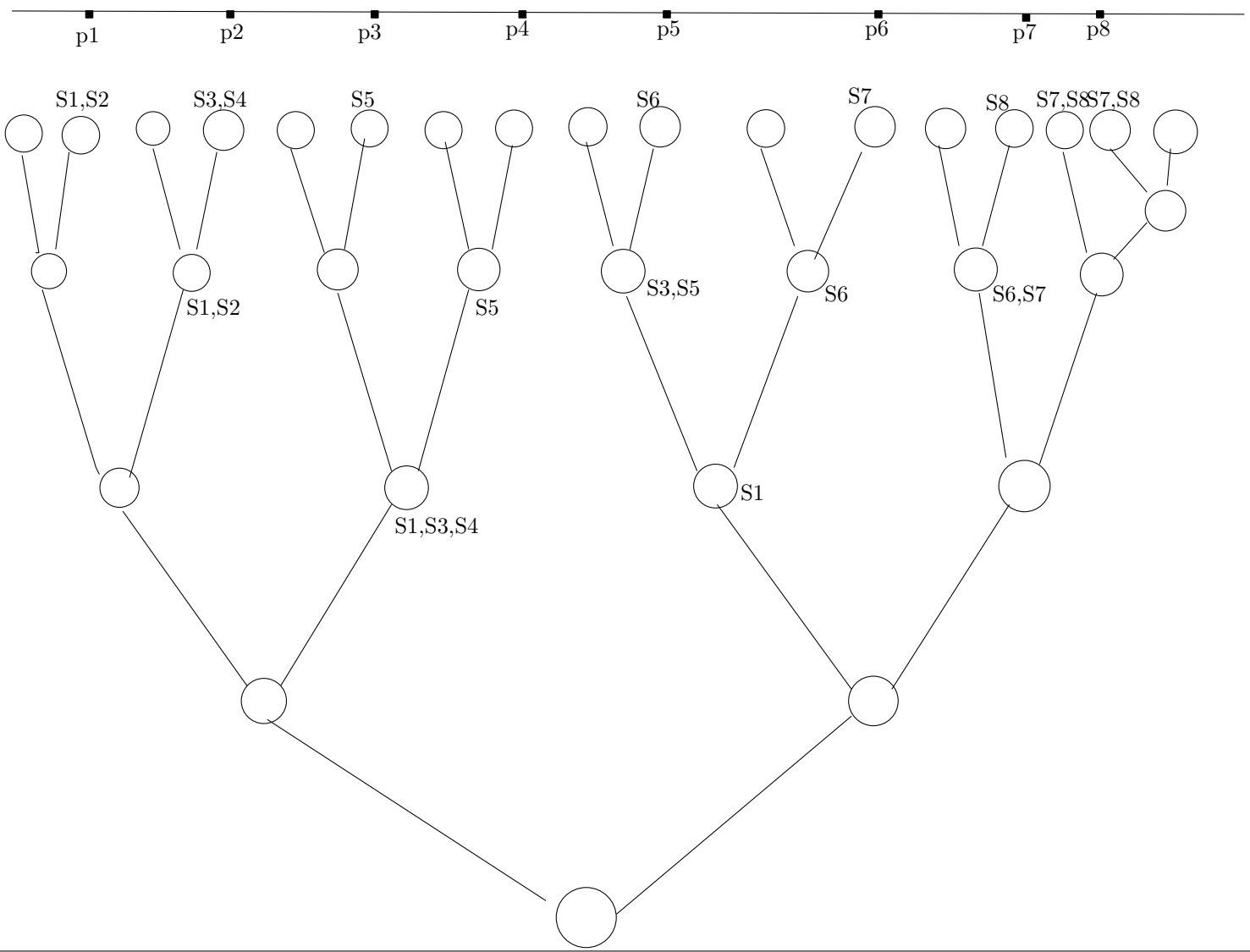
Consider an example with 8 segments s1,s2,s3....., s8



The list of elementary intervals consists of open intervals between two consecutive endpoints pi and pi+1 , alternated with closed intervals consisting of a single endpoint. The reason that we treat the points pi themselves as intervals is, of course, that the answer to a query is not necessarily the same at the interior of an elementary interval and at its endpoints. To find the intervals that contain a query point qx , we must determine the elementary interval that contains qx . To this end we build a binary search tree T whose leaves correspond to the elementary intervals.
We denote the elementary interval corresponding to a leaf U by Int(U).

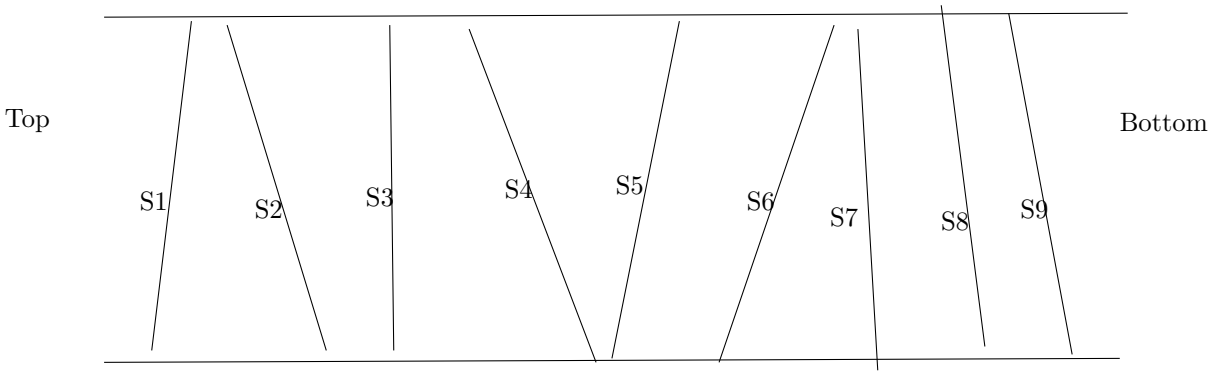


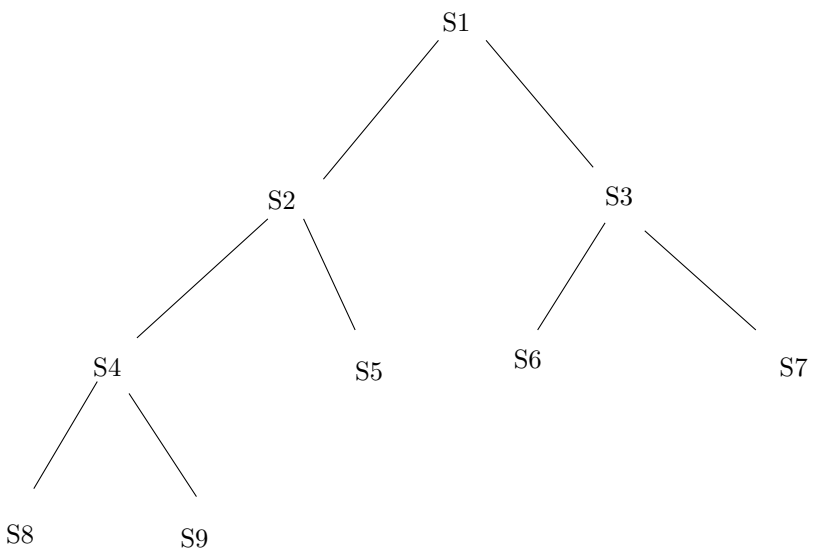
Segment Tree for these segments:



Building Priority Search Tree

Consider a slab of Segment Tree, the slab represents a path from root to the child. Let S1,S2,S3,S4.....,S9 are the segments are in a node N of segment tree. The node N must belongs to one/more slabs, Let us consider one slab with the segments of the node N.



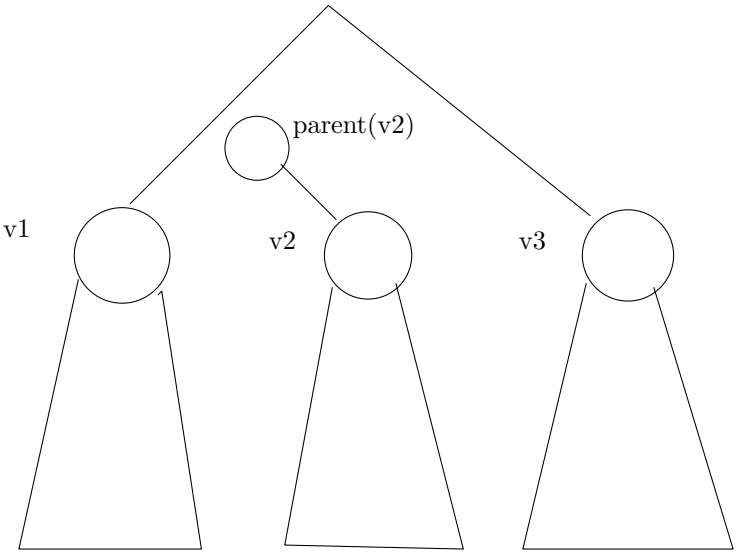


We can find the order of segments in all slabs by using Line sweep in O(nlogn). By that top to bottom order we can build Priority Search tree on every Node.

Pre-processing Time and Space ::

Th1 : A segment tree on a set of n intervals uses O(n log n) storage.

Proof. Because T is a balanced binary search tree with at most 4n + 1 leaves, its height is O(log n). We claim that any interval [x : x'] belongs to I is stored in the set I(v) for at most two nodes at the same depth of the tree. To see why this is true, let v1 , v2 , v3 be three nodes at the same depth, numbered from left to right. Suppose [x : x'] is stored at v1 and v3 . This means that [x : x'] spans the whole interval from the left endpoint of Int(v1) to the right endpoint of Int(v3). Because v2 lies between v1 and v3 , Int(parent(v2)) must be contained in [x : x']. Hence, [x : x'] will not be stored at v2 . It follows that any interval is stored at most twice parent(v2) at a given depth of the tree, so the total amount of storage is O(n log n).



From above therom 1 we can under stand that segment tree will be taking O(nlogn) space for n intervals(n segments). In our case we are even storing priority search trees in every node. But the storage wont be increasing because we are not using any extra space, we are finding only order of segments and building the segment.

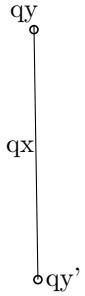
Th2 : The Segment Tree will be constructed in O(nlogn) time.

Proof :: To insert an interval [x : x'] into the segment tree, At every node that we visit we spend constant time (assuming we store I(v) in a simple structure like a linked list). When we visit a node v, we either store [x : x'] at v, or Int(v) contains an endpoint of [x : x']. We have already seen that an interval is stored at most twice at each level of T. There is also at most one node at every level whose corresponding interval contains x and one node whose interval contains x'. So we visit at most 4 nodes per level. Hence, the time to insert a single interval is O(log n), and the total time to construct the segment tree is O(n log n).

After Building the Segment tree T, we will insert every segment interval into the tree T. After all insertions we will get some segments on some of the nodes. The number of segments on all nodes in the tree T will be O(n). So, the construction of priority Serch Tree will take O(nlogn). So, the total complexity for preprocessing not change. i.e., O(nlogn).

Window Querying ::

We already said that, the segments whose endpoint(s) is/are inside the window can be reported using the range trees. The time we need for reporting those segments is O(log2n + k'). But the segments which are intersecting with window and their endpoints are outside of window can be reported using segment trees with priority search trees. We will see how those segments can be reported in logarithmic time.



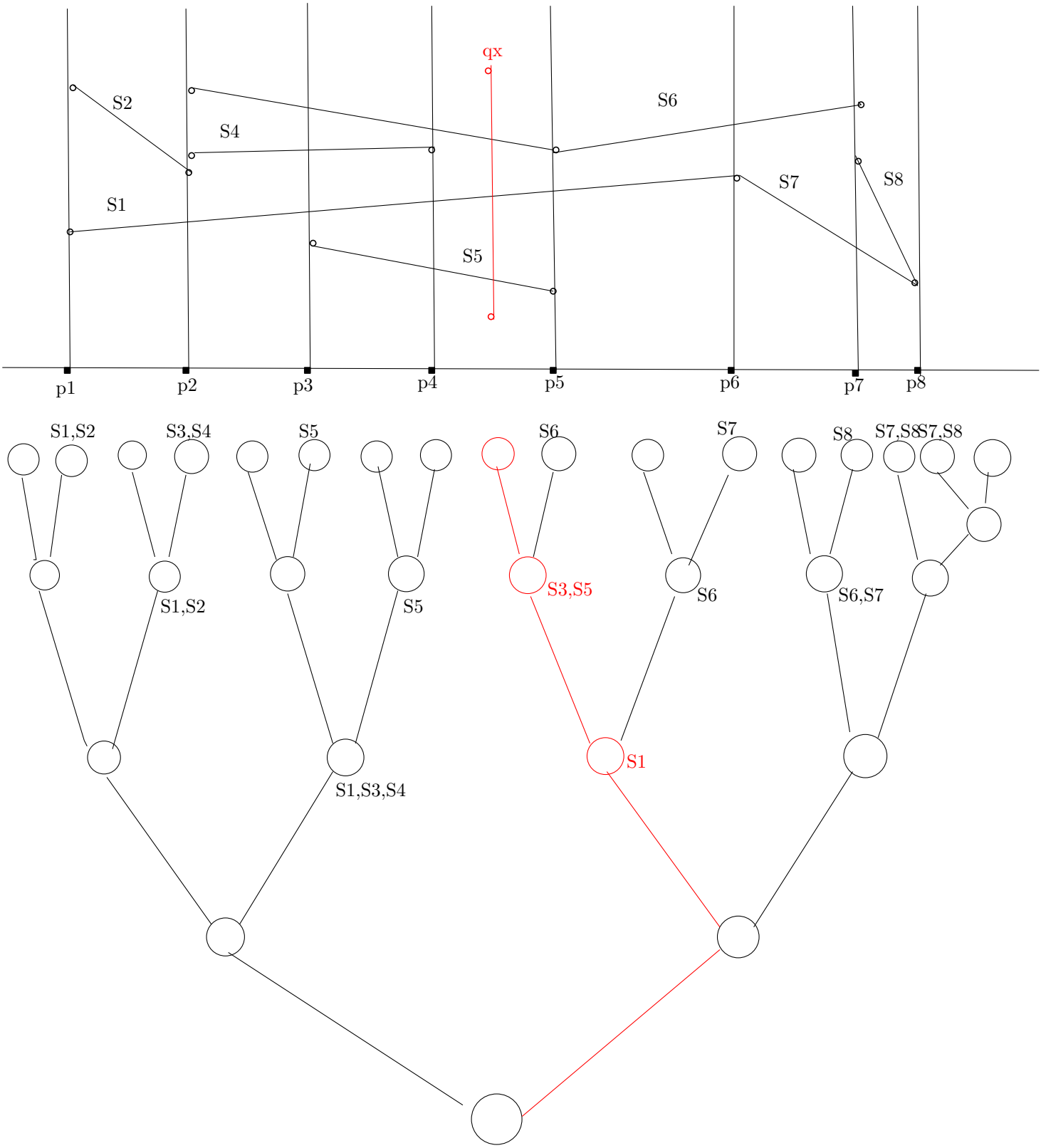
Consider the query qx X [qy : qy'] is a vertical segment.

- We traverse the Segment Tree from root using qx. On the path of traversal we find O(logn) nodes (i.e., Depth). In that path some nodes may consists the Segments in priority Search Tree. By using [qy : qy'] interval we report the segments intersecting the query line.

Reporting Segments ::

Let us assume the segment line equation ax+by=c. Let Y=aqx+bqy and Y'=aqx+bqy'. If Y lessthan c and Y' greaterthan c or Y greaterthan c and Y' lessthan c then the segment will be reported. If Y lessthan c and Y' lessthan c i.e., both points on the same side to the segment and those points are present in Upside of the segment , So, We can end the Serch in priority search tree. If Y greaterthan c and Y' greaterthan c i.e., both points on the same side to the segment and those points are present at downside of the segments, So, We can continue the continue the Search on children.

Consider the Same example initially we build segment tree.



Querying Time ::

Traversing will be taking O(logn) time. At some nodes we spent O(1+k") time to report segments. So, the time Complexity for reporting the segments which are intersecting with the query line will be O(logn + k). There are four line segments for a window so, for reporting the sements which are intersceting the window will be taking O(logn+k).

- Finally, to answer the window queries for a set of tilted line segments we need to spent O(lon2n + k) time.