Some More Geometric Data Structures (Winowing cont.)
Windowing (reminder)

- We have seen how to find axis-aligned lines intersecting an axis-aligned window.
Interval trees (reminder)

• We have used interval trees:

• In the relevant nodes we searched for the end points contained in a rectangle unbounded from one side.

• For this we have used 2d-Range Trees and then improved to Priority Search Trees.
Non Axis-Aligned segments

• What about general segments, that is, not axis-aligned?
  • We will restrict the problem to non-intersecting segments.
• Can we use the solution we already have?
• Use segment bounding box instead!
• Works quite well in practice.
• Worst case is bad:
Non Axis-Aligned segments

• Can we adopt interval trees?
• The key point in interval trees is knowing that one side of the segment is to the right (or left) of $q$.
• This doesn’t help much if we allow arbitrary orientation.
Non Axis-Aligned segments

• Can we adopt interval trees?
• The key point in interval trees is knowing that one side of the segment is to the right (or left) of $q$.
• This doesn’t help much if we allow arbitrary orientation.
Segment trees

• Let’s remember what interval trees solves in the first place:
• Finding the 1d-segments that cover a given point $x$.
• Can we devise another data structure for that?
• If the segments doesn’t overlap we can store them in a BST, and looking for the one segment that intersects $x$ is easy.
• But what if they do overlap?
Segment trees

• Given a set $S$ of overlapping segments, we want to find which segments intersects a point $x$.
• Create a new set, of non overlapping segments and store it in a BST.
  • Add zero-size segments for the end points.
• In each leaf store a list of (original) segments that intersects it.
Segment trees

• What is the space complexity of this data structure?
• Each segment can appear in many leaves.
• The space complexity is $O(n^2)$.
• Can we improve it?
• If a segment appear in consecutive leaves, we can store it in the parent node instead.
• $s$ will be stored in $\nu$ and $\mu_5$. 
Segment trees

• The complete data structure:
Segment trees

• What is the space complexity now?
• Each segment can appear at most twice at any level of the tree.
• Assume to the contrary:
• All the leaves between \( v_1 \) and \( v_3 \) contain a segment \( s \).
• Then, all the leaves in the subtree of \( parent(v_2) \) also contain \( s \), thus \( s \) will appear in \( parent(v_2) \) and not in \( v_2 \).
• Conclusion: each segment is stored in \( O(\log n) \) nodes.
• The space complexity is \( O(n \log n) \).
Segment trees

• Building a segment tree can also be done in $O(n \log n)$.
• How do we find all the segments covering $x$?
• Search for $x$ in the tree, report all the segment stored in nodes along the search path.
• Complexity: $O(\log n + k)$ where $k$ is the number of reported segments.
• Notice that a segment tree does the same job as a plain interval tree, but with worse space complexity.
Segment trees

• So how does segment trees help us?
• Given a set of non-intersecting segments, build a segment tree to their projection on the $x$-axis.
• Using that we can find potential segments. Segments that cover the $x$ coordinate of the window edge.
• How does this help?
Segment trees

- Each internal node represents the union of segments of its sons.
- A segment will be stored in a node if it covers the whole node-segment.
- This means that the set of segments stored in the node is well ordered.
Segment trees

• The set of segments in each node is well ordered.
  • Intuition: it looks like a (bended) ladder.
• How can we use this to find which segments intersect the window edge?
• Store the segments in a BST!
Segment trees

- The set of segments in each node is well ordered.
  - Intuition: it looks like a (bended) ladder.
- How can we use this to find which segments intersect the window edge?
- Store the segments in a BST!
Segment trees

- The space complexity is not affected: $O(n \log n)$
- The search in each node is done in $O(\log n)$, thus, the query complexity is $O(\log^2 n + k)$
- Building the tree takes $O(n \log^2 n)$.
- It can be improved to $O(n \log n)$ using some trick.