Tutorial 3, Oct 6, 2023

Augmenting Data Structures – Interval Trees

- Given a time interval, how can we efficiently tell whether it conflicts with any existing intervals in a set?
- We need a data structure that supports:
  - Insert: insert the new interval
  - Delete: delete the interval
  - Search: find an interval that conflicts with x, or return null
- For two intervals to conflict, we require that the end time of an interval is greater than or equal to the start time of the other interval, and the start time of the interval is less than or equal to the end time of the other interval
- We will augment an AVL tree; in the keys, we will store the starting time (the ending time is also stored, but nodes are ordered by starting time)
  - If the starting time of the node is greater than the end time of the query, we can eliminate this node and the entire right subtree
- As additional information, we store m_{hi}, the maximum end time in the subtree
  - This information can be cheaply maintained; just take the maximum of the end times of the two children and the node itself
- Invariant: if there is an interval intersecting q, then the subtree being processed will contain one
  - This ensures at each step we divide the number of potential nodes by half
- Algorithm: searching for conflict with query l, h in subtree u:
  - If u is null, return null
  - If $[l, h]$ intersects with the interval in u, return this interval
  - Otherwise there are 2 cases:
    * The max end time of the left subtree is greater than l: search the left subtree
      - Consider the interval with the max end time in the left subtree; either this interval starts before h, or it starts after h
      - If it starts before h, then we have a conflict in the left subtree
      - If it starts after h, we might not have a conflict, but we also know that the right tree cannot have a conflict either, since no time in the right tree can start before h
    * Otherwise, search the right subtree
      - In this case, it is impossible for there to be a conflict in the left subtree, since we need an interval that ends after l