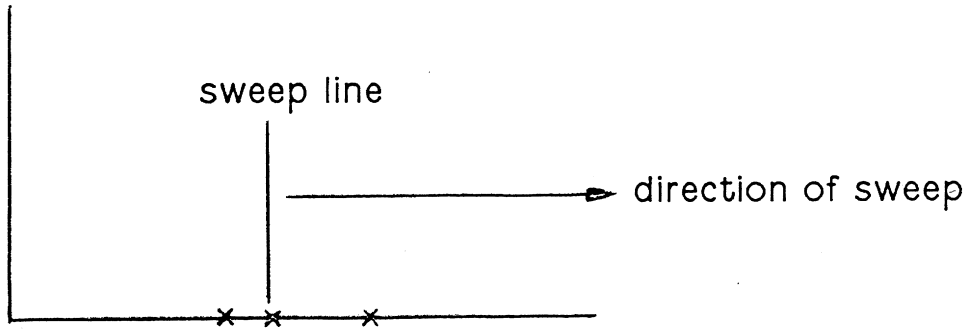
PLANE-SWEEP METHODS

Hanan Samet
Computer Science Department and
Institute for Advanced Computer Studies and
Center for Automation Research
University of Maryland

College Park, MD 20742 USA
e-mail: hjs@umiacs.umd.edu

PLANE-SWEEP METHODS

- Solve geometric problems by sweeping a line (plane in 3-d) across the plane (space in 3-d) and halting at points of intersection with the objects being processed



X≡halting points

- Compute a partial solution so that at the end of the sweep we have a final solution

- Assume 2-d data

- Organize two sets of data:

  1. Halting points of the sweep line (i.e., x coordinates) which are sorted in ascending order
  2. Description of the status of the objects intersected by the current position of the sweep line
     - Status is problem-dependent
     - Efficiency of the solution depends on the data structure

- Basically, a multidimensional sort!

- Efficiency of solutions employing plane sweep are constrained by how fast we can sort - i.e., $o(N \cdot \log_2 N)$

RECTANGLE INTERSECTION PROBLEMS

- Naive solution is $o(N^2)$ - checks each rectangle against every other rectangle

- Rectangles are specified by the 4 values of their boundaries

Pass 1:
- Sort left/right boundaries of rectangles
- x coordinate values

Pass 2:
- Sweep vertical line
- Halt every time a rectangle becomes active - insert
- Halt every time a rectangle becomes inactive - delete

Key: How to represent active rectangles?

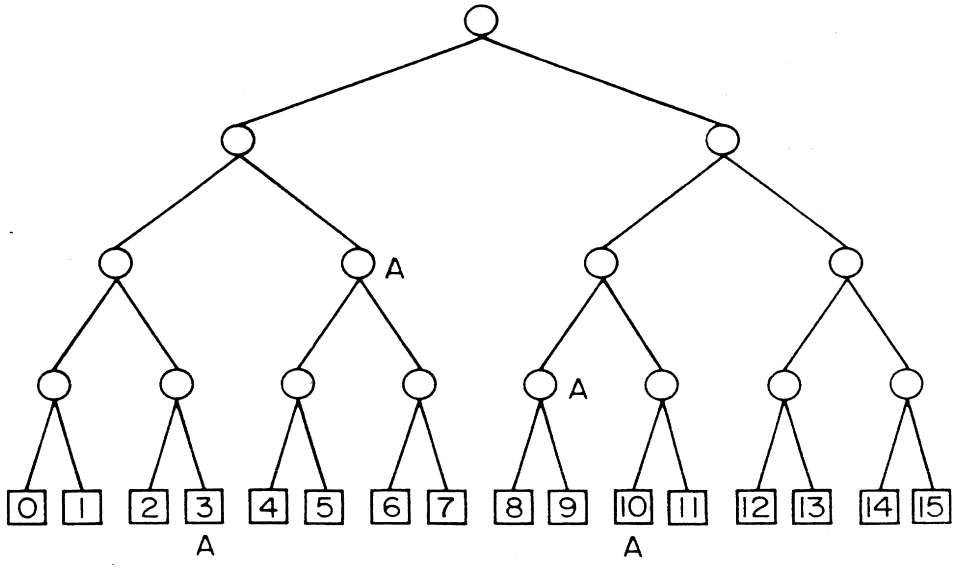
- If just want to determine intersections of line segments (i.e., vertical boundaries with horizontal boundaries), then can use a balanced binary tree for the top and bottom y-coordinate values of active boundaries

- Balanced binary tree method does not work for rectangles that are totally contained within other rectangles since the principle is that upon insertion of a rectangle with vertical boundaries [b,t] a search is made for all rectangles in the binary tree that have endpoints between b and t

UNIT SEGMENT TREE

- Represents one line segment
- Complete binary tree of depth h
- Leaf nodes correspond to unit intervals $[j,j+1)$ $0 \leq j < 2^h$
- If the root is at level h, then a node at level i corresponds to $[p,p+2^i)$
- A node is marked if its interval is in the line segment being represented

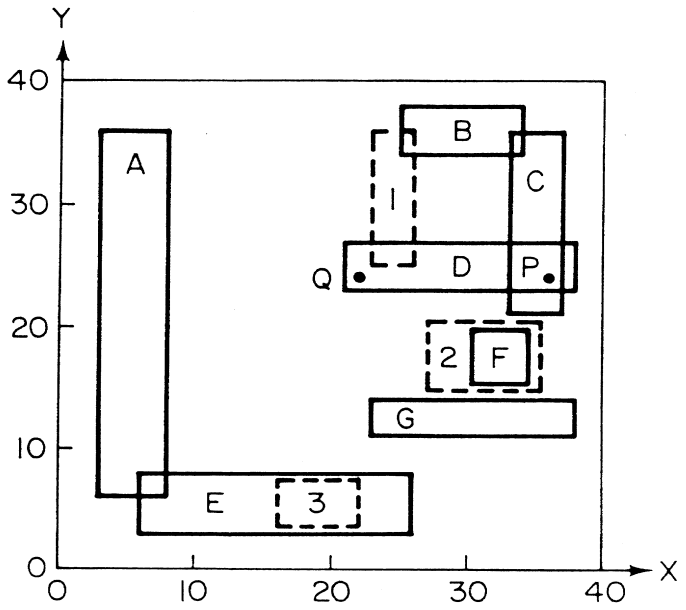
Ex: Line segment [3,11) in the range [0,16)

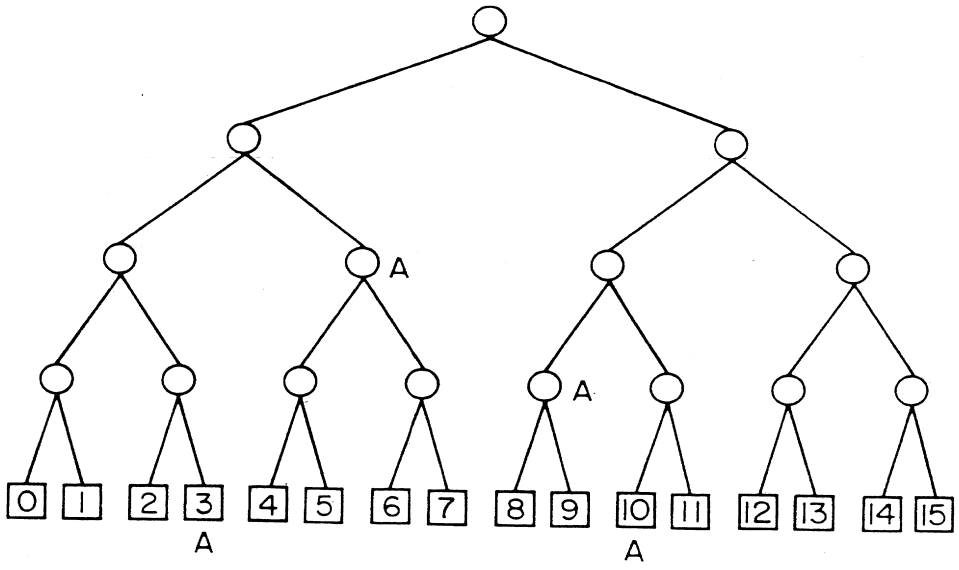


- Insertion is o(h)
- Like a 1-d region quadtree
  BLACK ≡ marked (i.e., labeled)
  WHITE ≡ unmarked
  Nodes of the same color are merged

SEGMENT TREE

- Represent a collection of line segments with arbitrary endpoints
- Intervals need not be of uniform width
- Sort endpoints y_i with duplicates removed
- Formed in the same way as the unit segment tree
- Each node is marked with the names of the segments that cover its interval





UPDATING SEGMENT TREES

1. Insertion

   - Just like unit segment tree
   - $o(\log_2 N)$

2. Deletion

   - Remove line segment from each node that contains it
   - Use doubly-linked lists at each node
   - Auxiliary table indicates for each line segment the nodes containing it
   - Auxiliary table is implemented as a balanced binary tree - $o(\log_2 N)$ to access
   - Each line segment is in a maximum of $2 \cdot \log_2 N$ nodes
   - Doubly-linked lists imply deletion in constant time at each node
   - $o(\log_2 N)$ time
   - $o(N \cdot \log_2 N)$ space

RECTANGLE INTERSECTION WITH SEGMENT TREES

- Can formulate as an ∞ number of point queries - i.e., for each point in S find all line segments that contain it

- Decompose into two problems: $S=[l,r)$

1. Determine all line segments with starting points $<l$ whose intersection with S is non-empty

   - Solve by a point query for l on the segment tree
   - Find the smallest interval that contains l
   - All nodes visited on the path from the root contain l
   - $o(\log_2 N + K_l)$ time
     K_l = number of line segments that contain l
   - $o(N \cdot \log_2 N)$ space

2. Determine all line segments with starting points in $[l,r)$

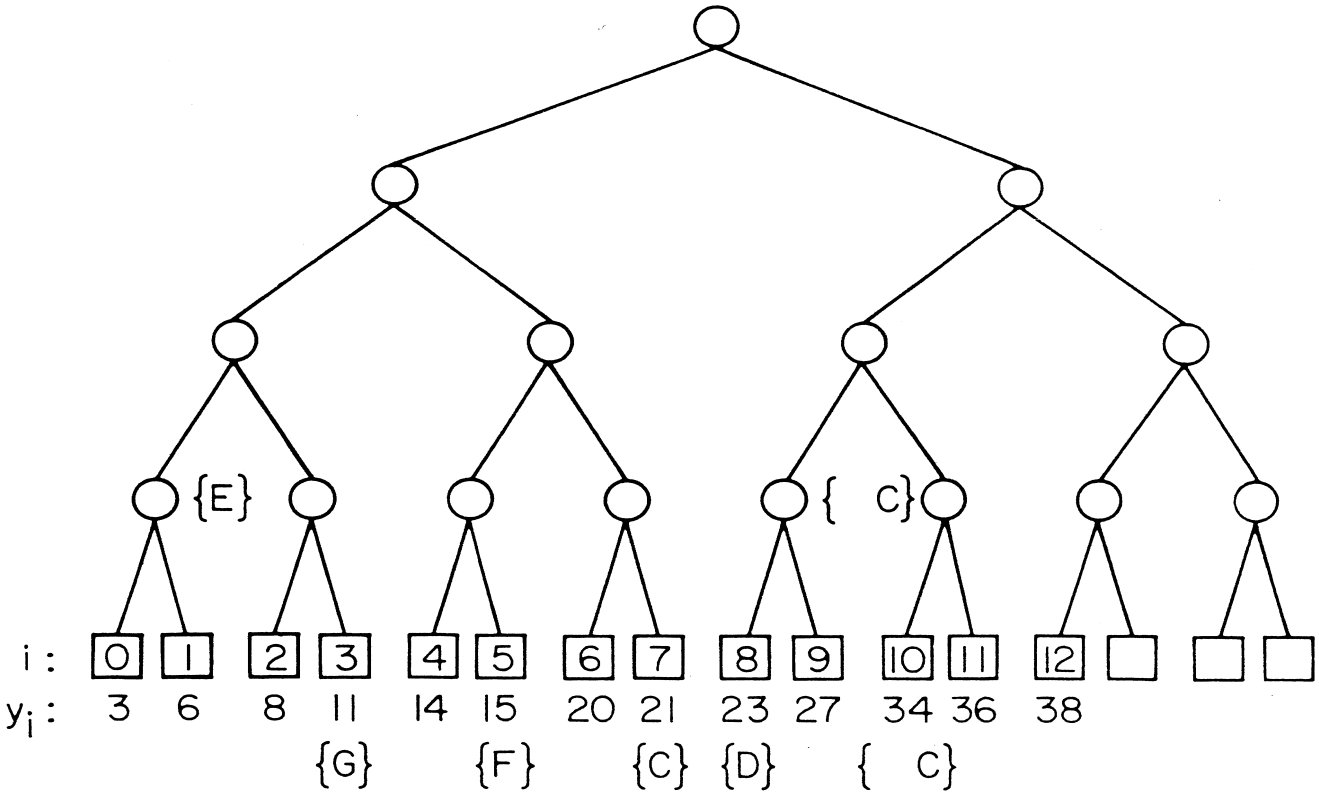
   - Solve by a range query for the range $[l,r)$ on the set of starting points of line segments
   - Use a balanced binary tree to store starting points
   - Link in ascending order
   - Insertion/deletion in a balanced binary tree is $o(\log_2 N)$
   - The range query finds the nodes that are closest to l and r and uses the linked list to output the intervening nodes
     $o(\log_2 N + K_{lr})$ where K_{lr} = number of nodes found
   - Balanced binary tree $\Rightarrow$ $o(N)$ space

- Total solution requires $o(N \cdot \log_2 N)$ space and time

SEGMENT TREE DEFICIENCY

Problem: Without the binary tree, when we insert S, we can't find all existing line segments in the tree that are totally or partially contained in S without examining every node in each subtree that contains S

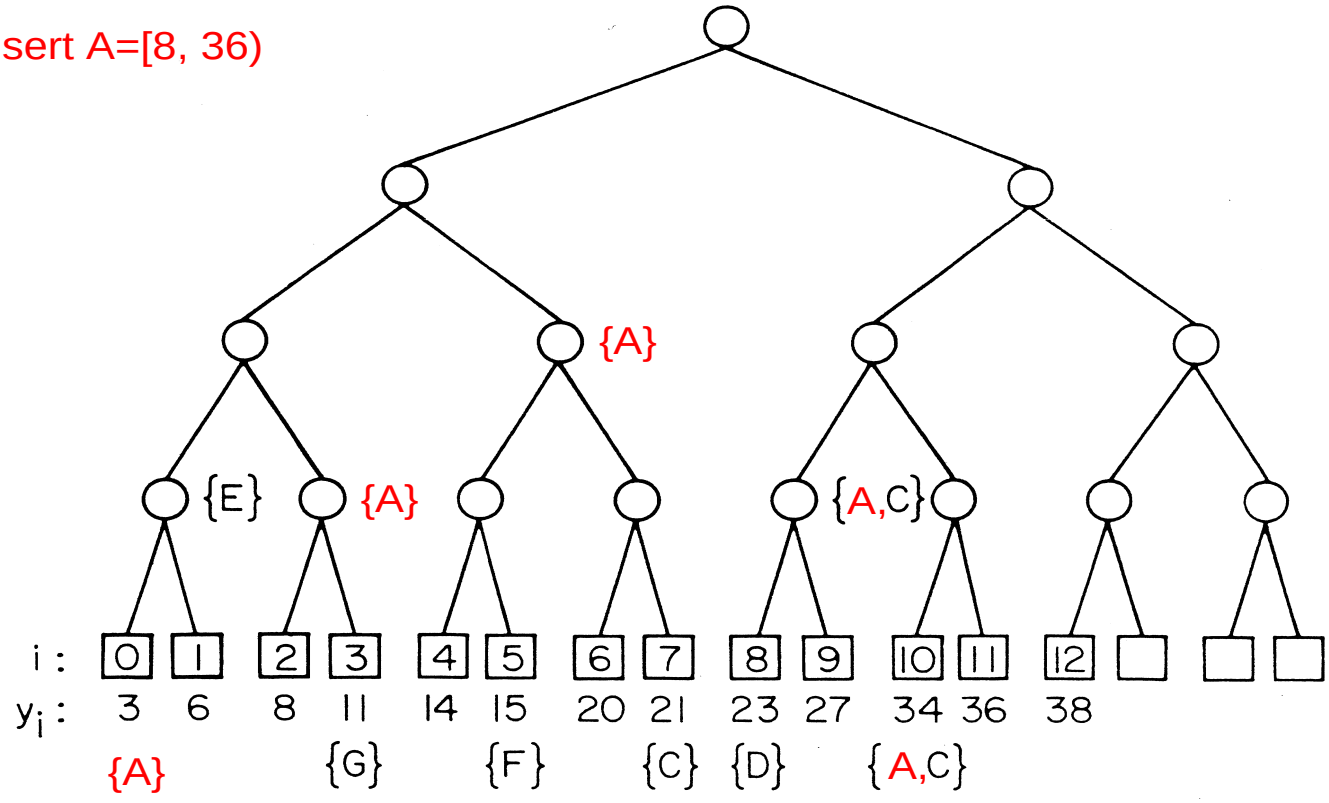
Ex:



SEGMENT TREE DEFICIENCY

Problem: Without the binary tree, when we insert S, we can't find all existing line segments in the tree that are totally or partially contained in S without examining every node in each subtree that contains S

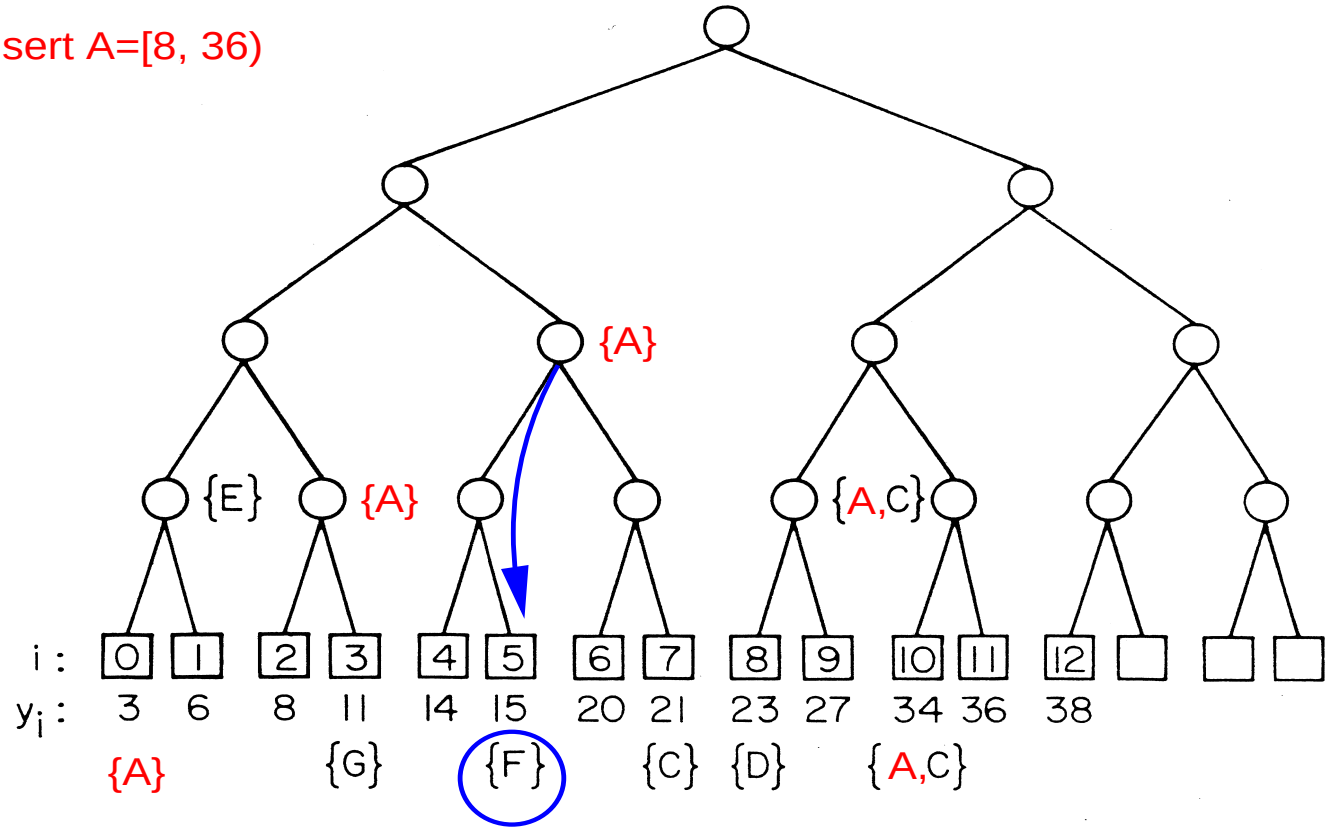
Ex: Insert A=[8, 36)



SEGMENT TREE DEFICIENCY

Problem: Without the binary tree, when we insert S, we can't find all existing line segments in the tree that are totally or partially contained in S without examining every node in each subtree that contains S

Ex: Insert A=[8, 36)

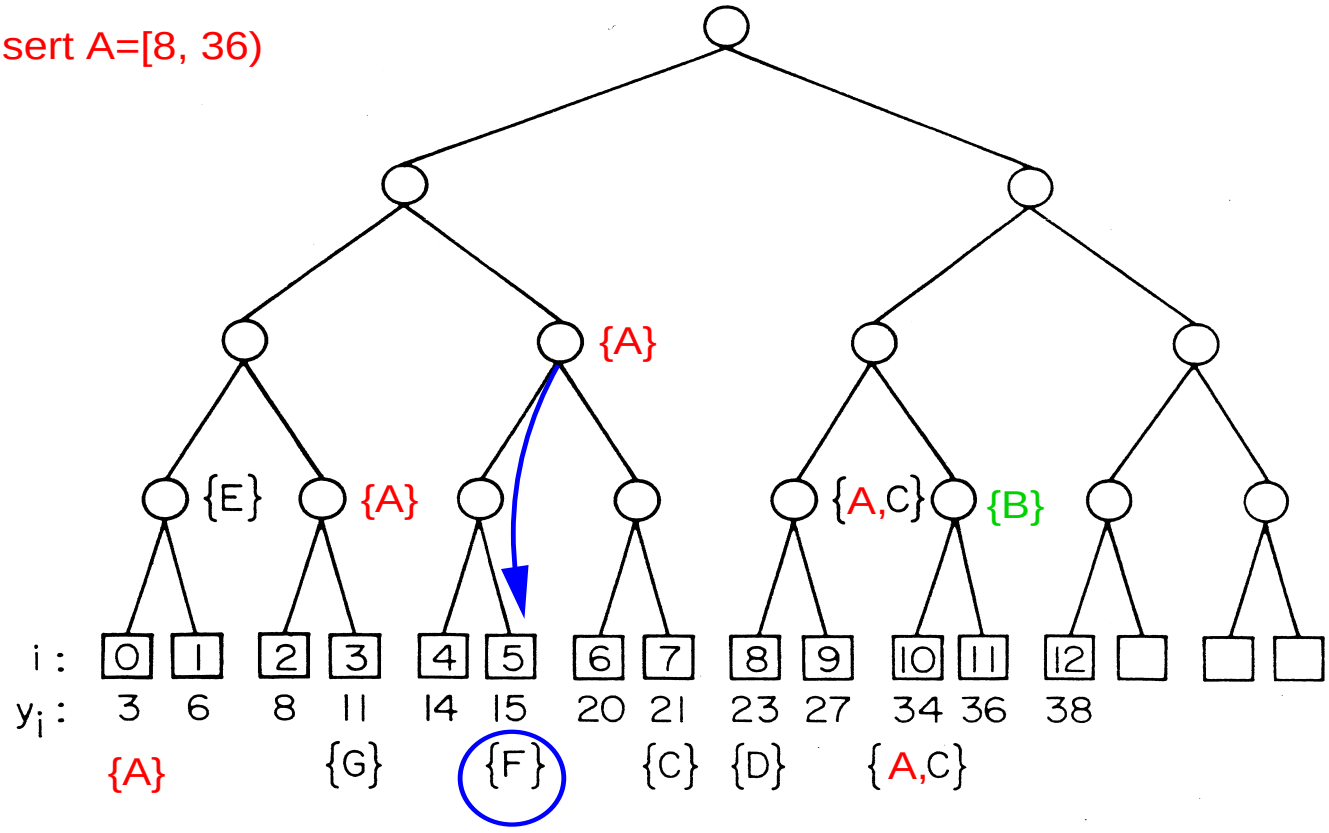


- Can only detect the existence of F=[15, 20) which is totally contained in A by descending to the bottom of the tree rooted at [14, 23).

SEGMENT TREE DEFICIENCY

Problem: Without the binary tree, when we insert S, we can't find all existing line segments in the tree that are totally or partially contained in S without examining every node in each subtree that contains S

Ex: Insert A=[8, 36)



- Can only detect the existence of F=[15, 20) which is totally contained in A by descending to the bottom of the tree rooted at [14, 23).

Ex: Insert B =[34,38)

- Can only detect intersection with A=[8,36) and C=[21,36) in [34,36) by descending to the bottom of the tree rooted at [34,38).

Problem: Checking for partial/total containment in the way requires O(N) time for each segment for a total of $O(N^2)$ -- TOO SLOW

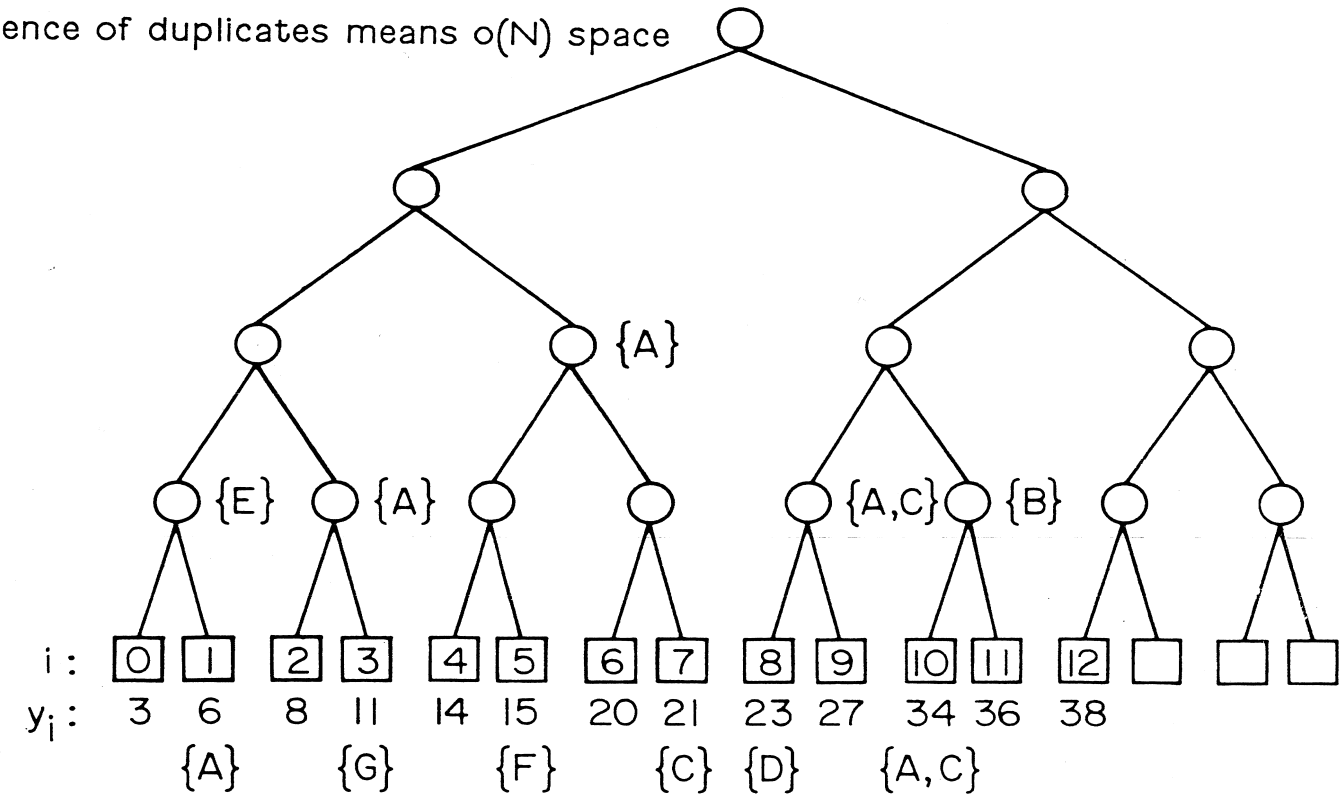
OVERCOMING SEGMENT TREE PROBLEMS

- Use an auxiliary binary tree to link each marked node to all nodes in its subtrees that are marked

Problem: Duplicate intersections are reported since each line segment can be associated with more than one node. A maximum of $o(N^2 \cdot \log_2 N)$ intersections which means at least $o(N^2 \cdot \log_2 N)$ time to remove duplicates by sorting via bin methods (could be even worse!)

Solution: Avoid duplicate entries in each node by redefining the segment tree so each line segment is only associated with one node - the nearest common ancestor of all the intervals contained in the line segment

- Absence of duplicates means o(N) space



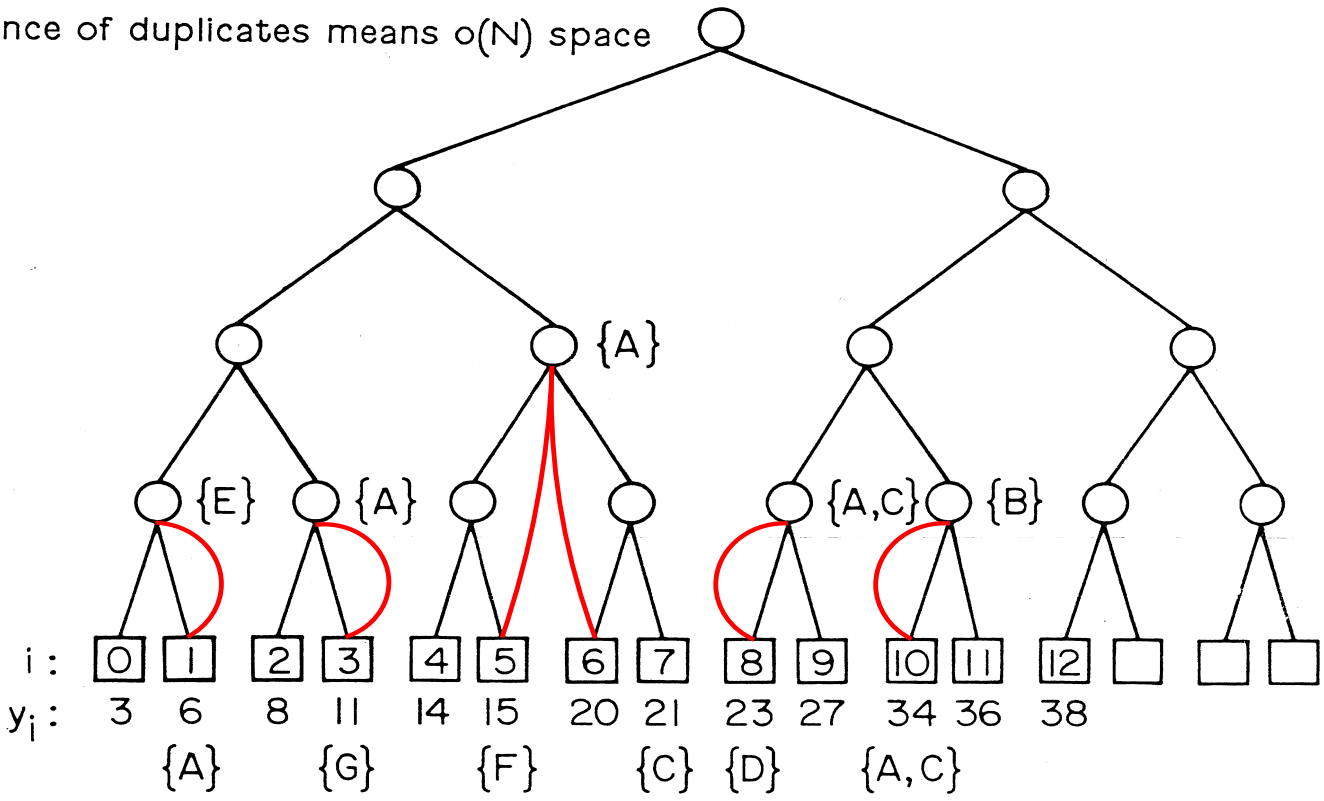
OVERCOMING SEGMENT TREE PROBLEMS

- Use an auxiliary binary tree to link each marked node to all nodes in its subtrees that are marked

Problem: Duplicate intersections are reported since each line segment can be associated with more than one node. A maximum of $o(N^2 \cdot \log_2 N)$ intersections which means at least $o(N^2 \cdot \log_2 N)$ time to remove duplicates by sorting via bin methods (could be even worse!)

Solution: Avoid duplicate entries in each node by redefining the segment tree so each line segment is only associated with one node - the nearest common ancestor of all the intervals contained in the line segment

- Absence of duplicates means o(N) space



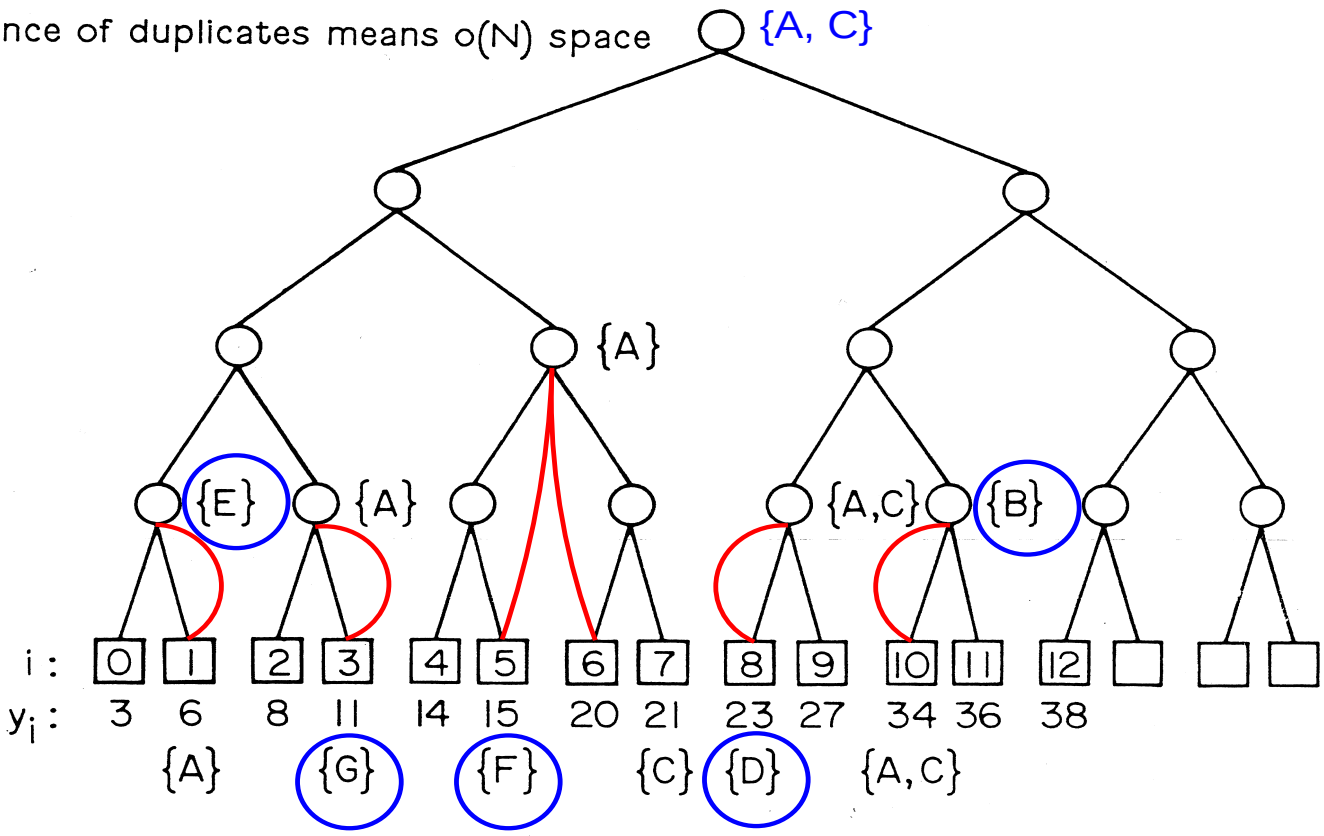
OVERCOMING SEGMENT TREE PROBLEMS

- Use an auxiliary binary tree to link each marked node to all nodes in its subtrees that are marked

Problem: Duplicate intersections are reported since each line segment can be associated with more than one node. A maximum of $o(N^2 \cdot \log_2 N)$ intersections which means at least $o(N^2 \cdot \log_2 N)$ time to remove duplicates by sorting via bin methods (could be even worse!)

Solution: Avoid duplicate entries in each node by redefining the segment tree so each line segment is only associated with one node - the nearest common ancestor of all the intervals contained in the line segment

- Absence of duplicates means o(N) space



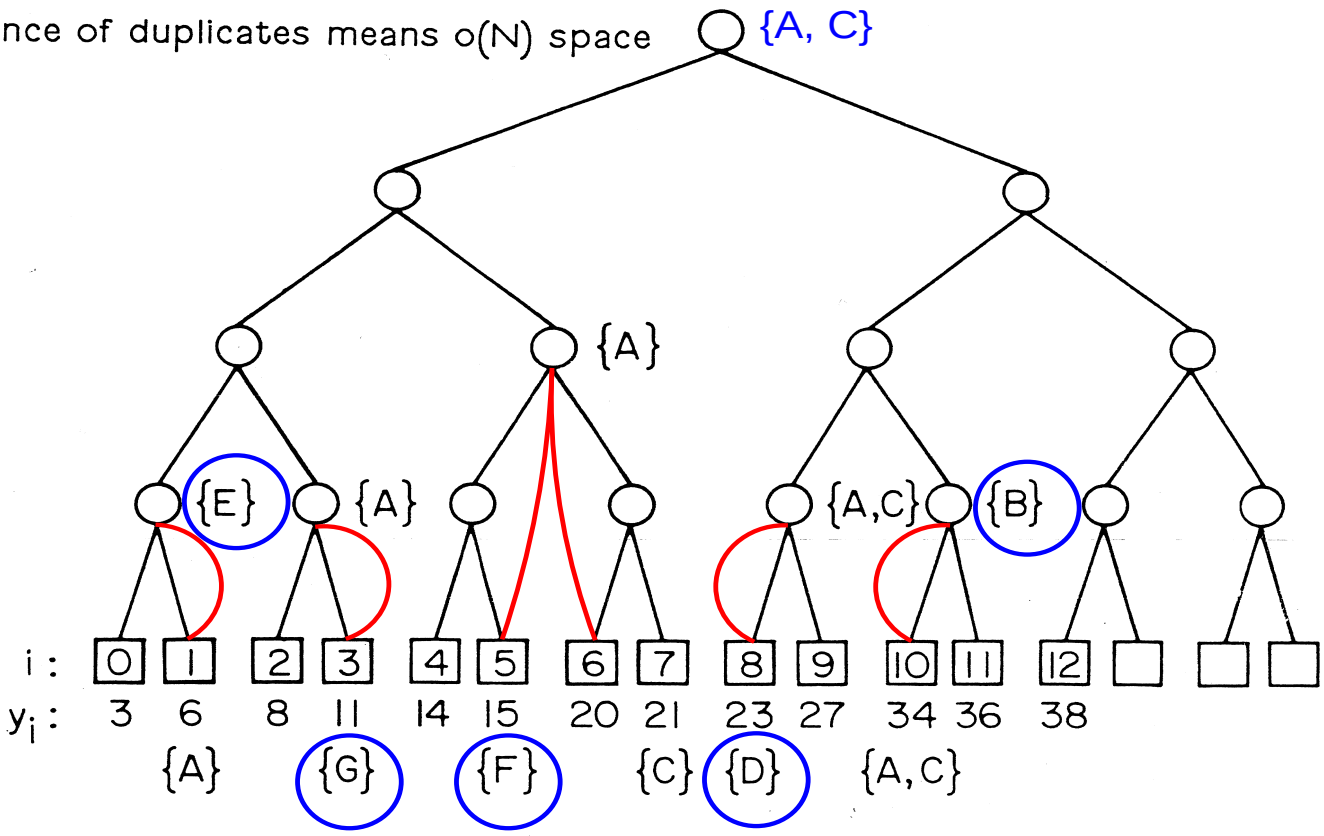
OVERCOMING SEGMENT TREE PROBLEMS

- Use an auxiliary binary tree to link each marked node to all nodes in its subtrees that are marked

Problem: Duplicate intersections are reported since each line segment can be associated with more than one node. A maximum of $o(N^2 \cdot \log_2 N)$ intersections which means at least $o(N^2 \cdot \log_2 N)$ time to remove duplicates by sorting via bin methods (could be even worse!)

Solution: Avoid duplicate entries in each node by redefining the segment tree so each line segment is only associated with one node - the nearest common ancestor of all the intervals contained in the line segment

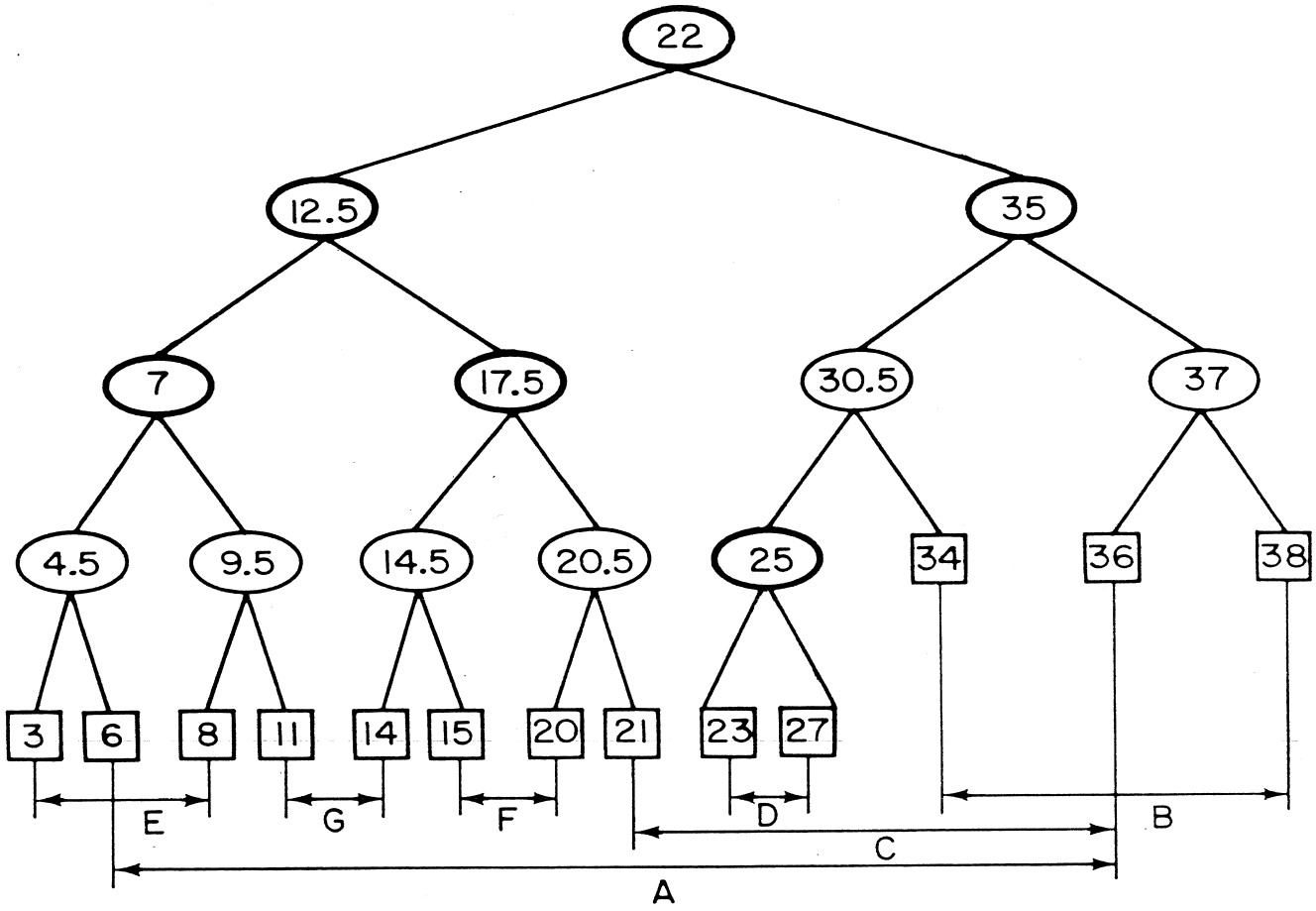
- Absence of duplicates means o(N) space



- These modifications are the basis of the interval tree

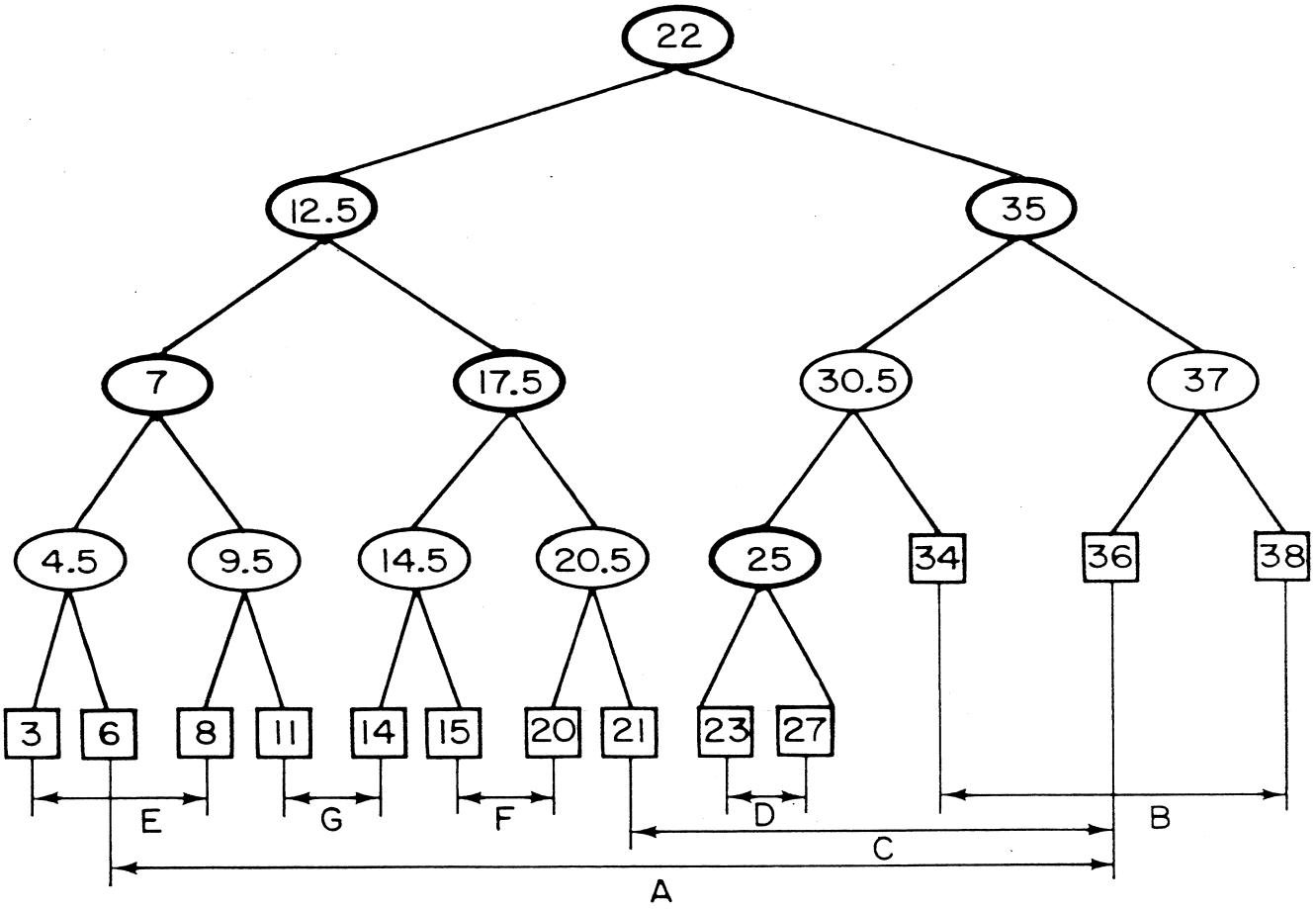
INTERVAL TREE

- Primary structure is a complete binary tree with m+1 external nodes
- Each internal node is assigned a value between the maximum of the left subtree and the minimum of the right subtree
- Each internal node serves as a key to the secondary and tertiary structures



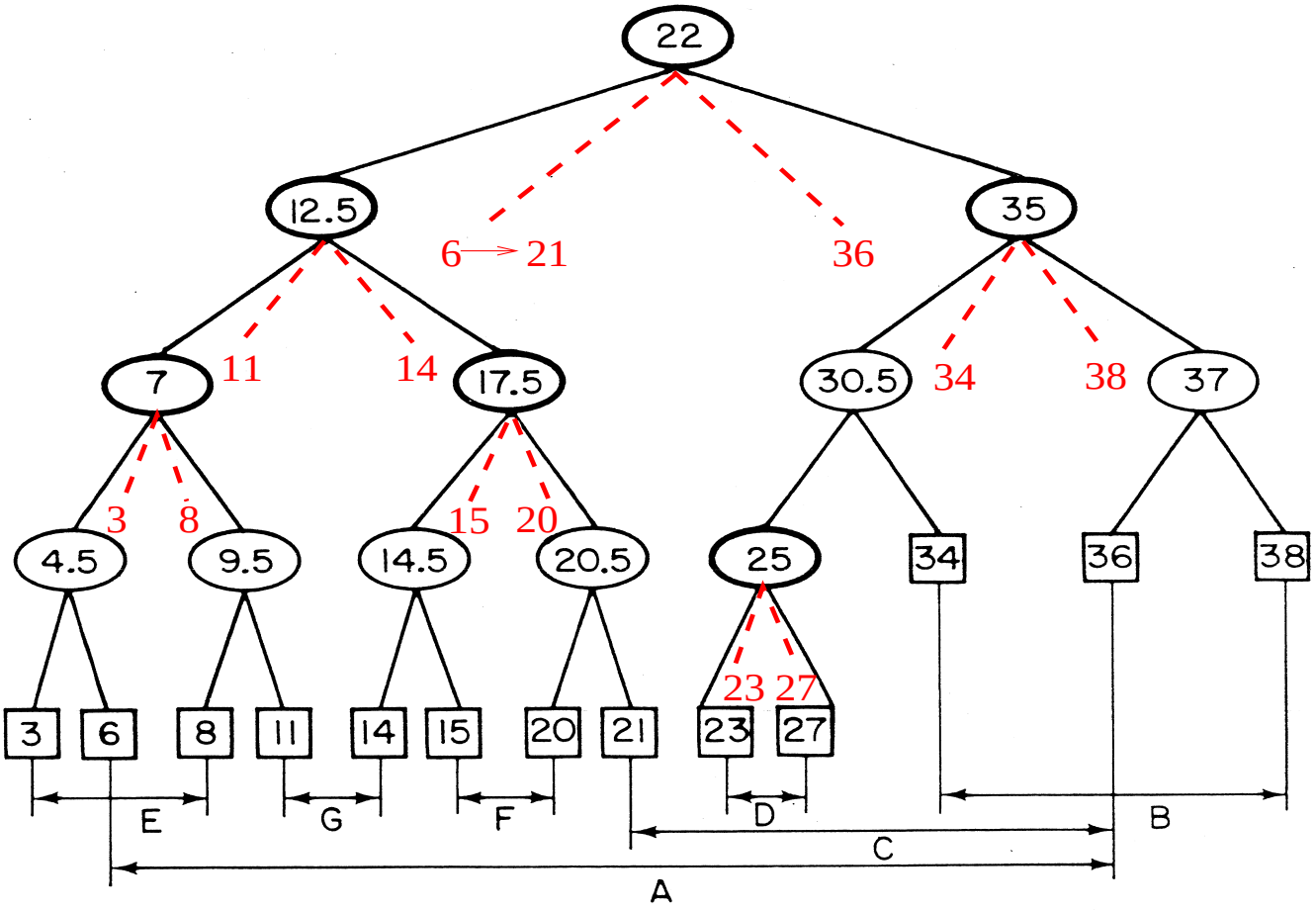
SECONDARY STRUCTURE

- Attached to each internal node
- Contain lists of left (LS) and right (RS) endpoints of intervals that cover the internal node's value
- LS is linked in ascending order
- RS is linked in descending order
- Implement LS and RS as balanced binary trees to support rapid insertion/deletion
- Each internal node points to the roots of LS and RS and to the minimum value in LS and the maximum value in RS



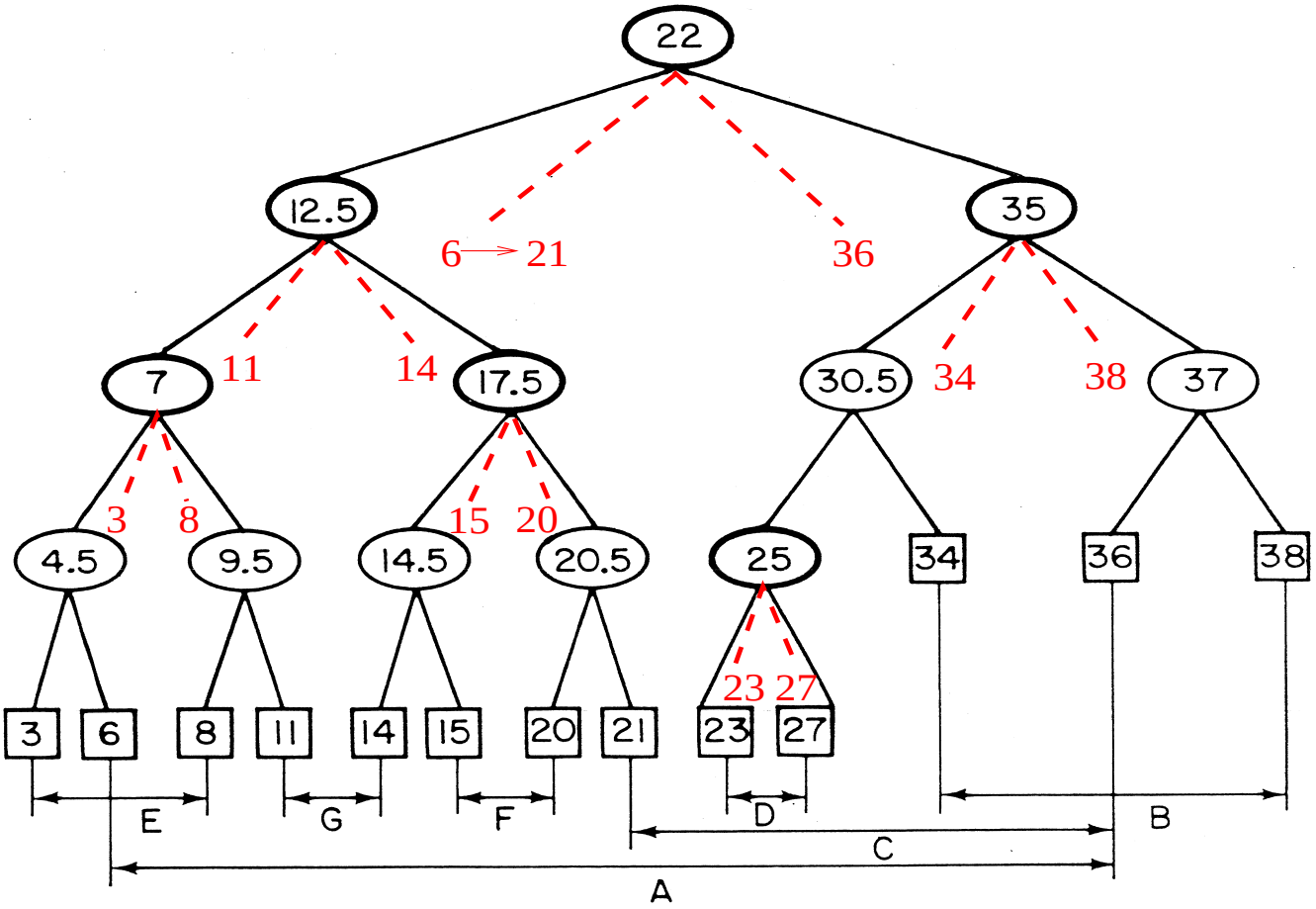
SECONDARY STRUCTURE

- Attached to each internal node
- Contain lists of left (LS) and right (RS) endpoints of intervals that cover the internal node's value
- LS is linked in ascending order
- RS is linked in descending order
- Implement LS and RS as balanced binary trees to support rapid insertion/deletion
- Each internal node points to the roots of LS and RS and to the minimum value in LS and the maximum value in RS



SECONDARY STRUCTURE

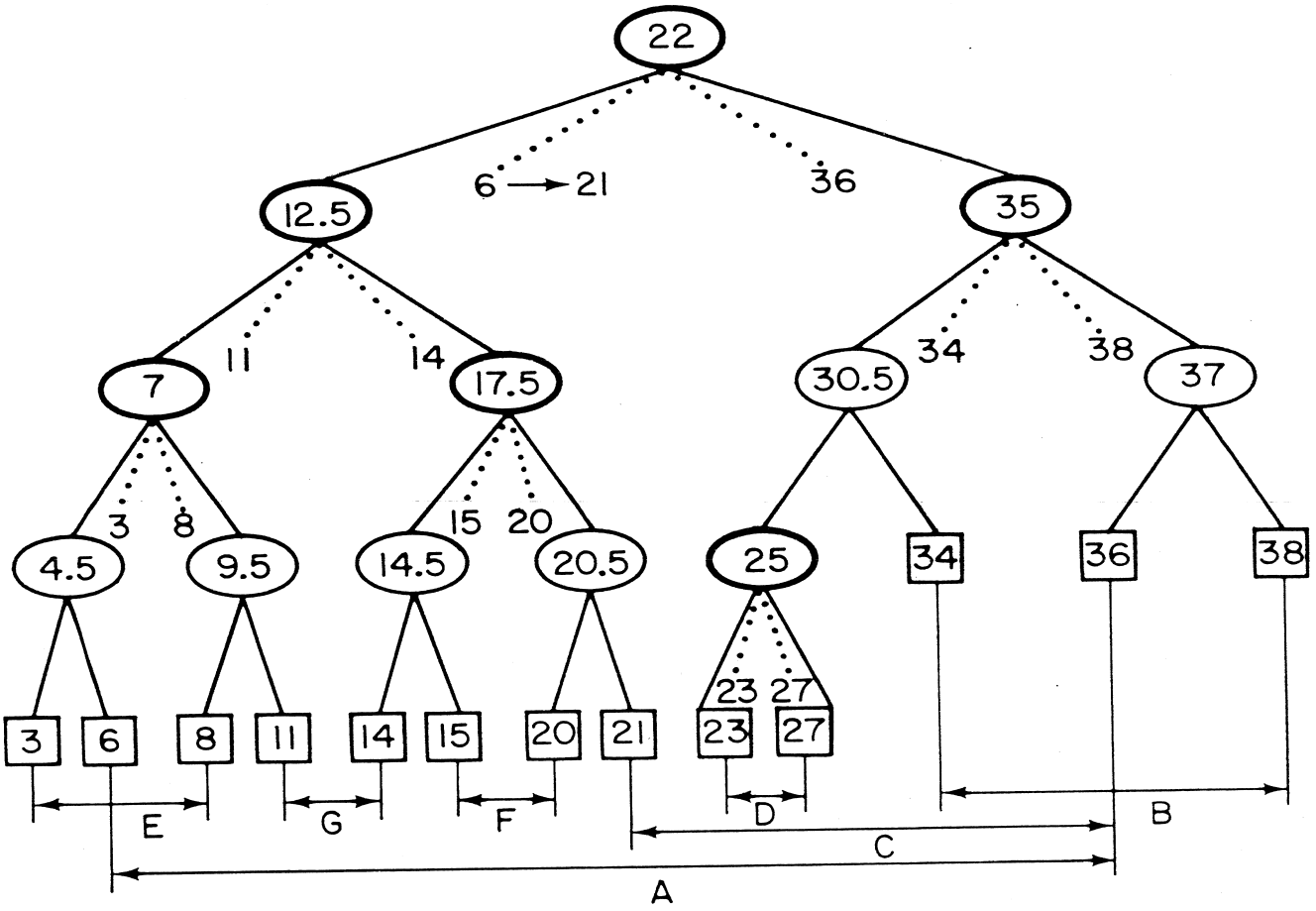
- Attached to each internal node
- Contain lists of left (LS) and right (RS) endpoints of intervals that cover the internal node's value
- LS is linked in ascending order
- RS is linked in descending order
- Implement LS and RS as balanced binary trees to support rapid insertion/deletion
- Each internal node points to the roots of LS and RS and to the minimum value in LS and the maximum value in RS



- Like a 1–d MX–CIF quadtree

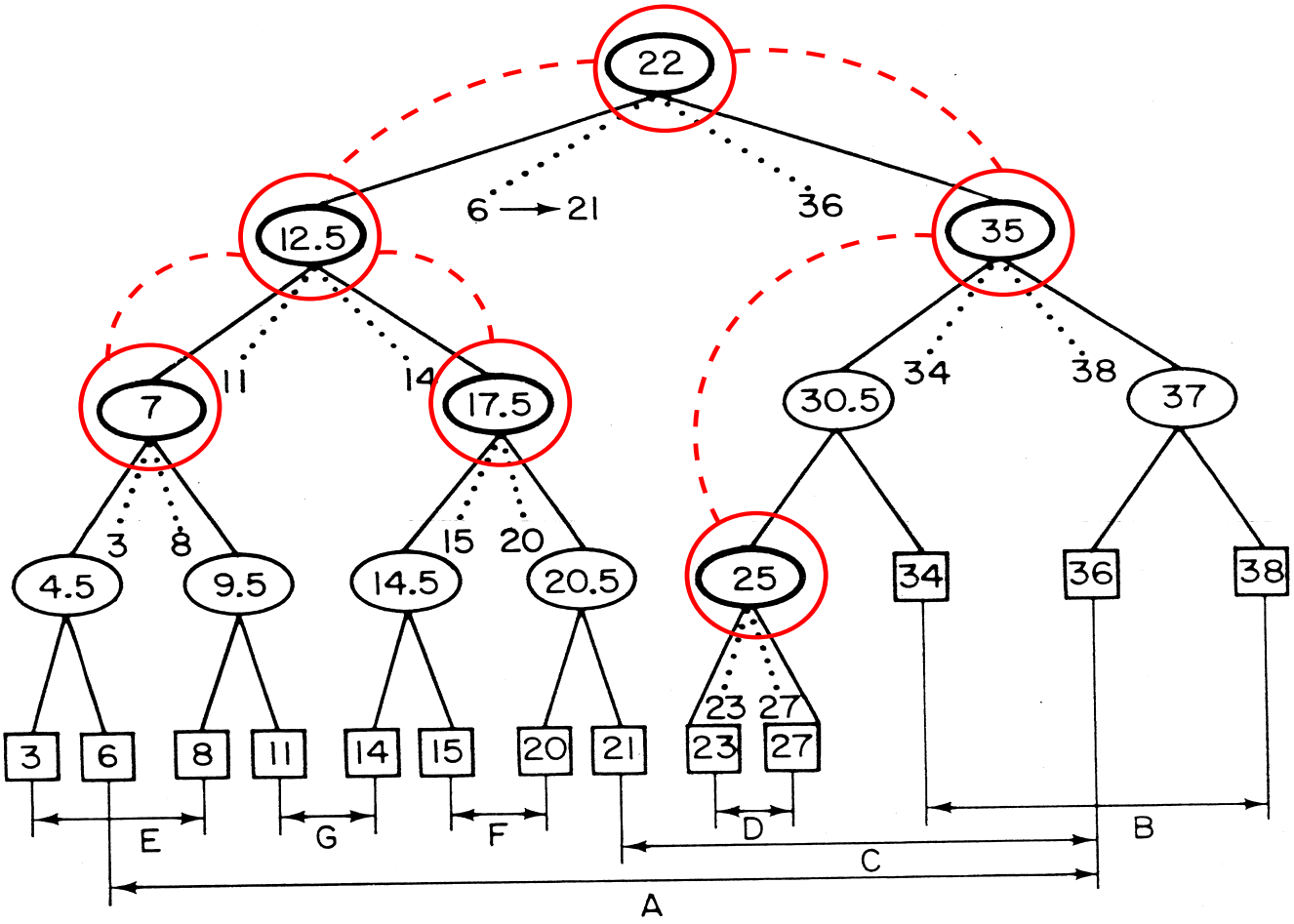
TERTIARY STRUCTURE

- Mark an internal node of the primary structure as active if its secondary structure is non-empty or both of its sons are active; otherwise inactive
- A binary tree consisting of the active nodes
- LT(V) and RT(V) point to the nearest active nodes in the left and right subtrees of internal node V
- LT(V) and RT(V) are Ω if there are no active nodes in their subtrees
- Useful in collecting all line segments that intersect a given line segment without looking at primary nodes whose corresponding line segments do not



TERTIARY STRUCTURE

- Mark an internal node of the primary structure as active if its secondary structure is non-empty or both of its sons are active; otherwise inactive
- A binary tree consisting of the active nodes
- LT(V) and RT(V) point to the nearest active nodes in the left and right subtrees of internal node V
- LT(V) and RT(V) are Ω if there are no active nodes in their subtrees
- Useful in collecting all line segments that intersect a given line segment without looking at primary nodes whose corresponding line segments do not



UPDATING INTERVAL TREES

- Inserting $[l,r)$

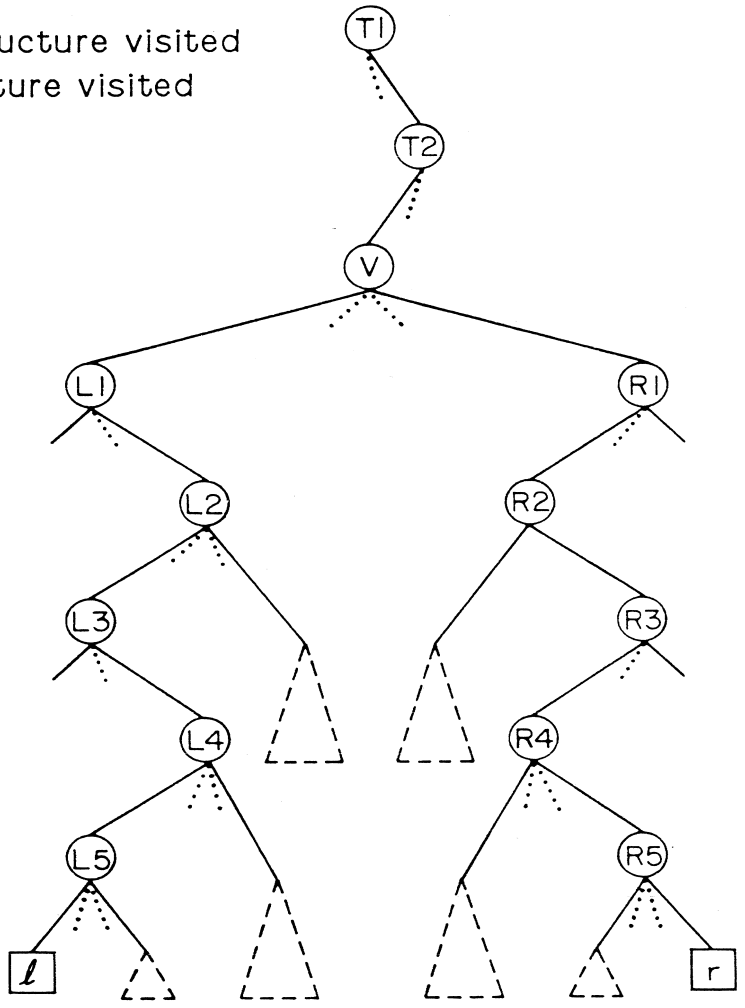
  1. Start at the root and locate the first node V where $l < VAL(V) < r$
  2. Insert l in LS(V) and r in RS(V)
  3. Update the tertiary structure while traversing the primary structure
  4. $o(\log_2 N)$ time

- Deletion in an analogous manner

RECTANGLE INTERSECTION WITH INTERVAL TREES

- Perform task while inserting vertical line segment S=[l,r)
- 3 stage search: report all line segments encountered that overlap S
- Each line segment is only reported once as it is associated with the secondary structure of just one node in the primary structure

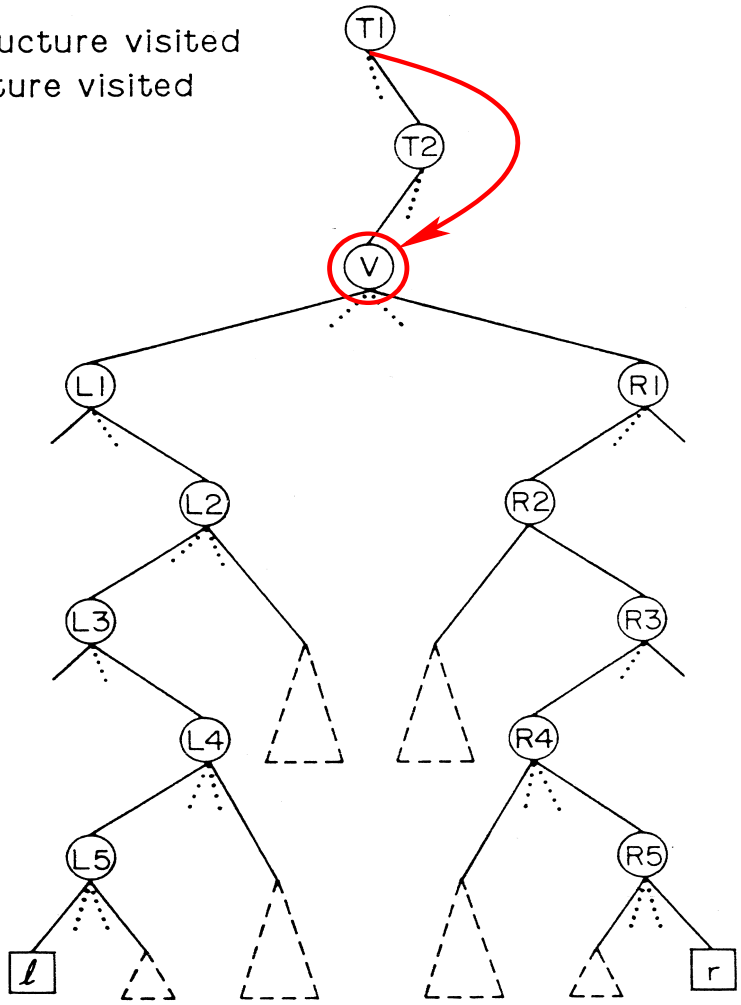
● ● ● ● ● secondary structure visited
- - - - - tertiary structure visited



RECTANGLE INTERSECTION WITH INTERVAL TREES

- Perform task while inserting vertical line segment S=[l,r)
- 3 stage search: report all line segments encountered that overlap S
- Each line segment is only reported once as it is associated with the secondary structure of just one node in the primary structure

● ● ● ● ● secondary structure visited
- - - - - tertiary structure visited

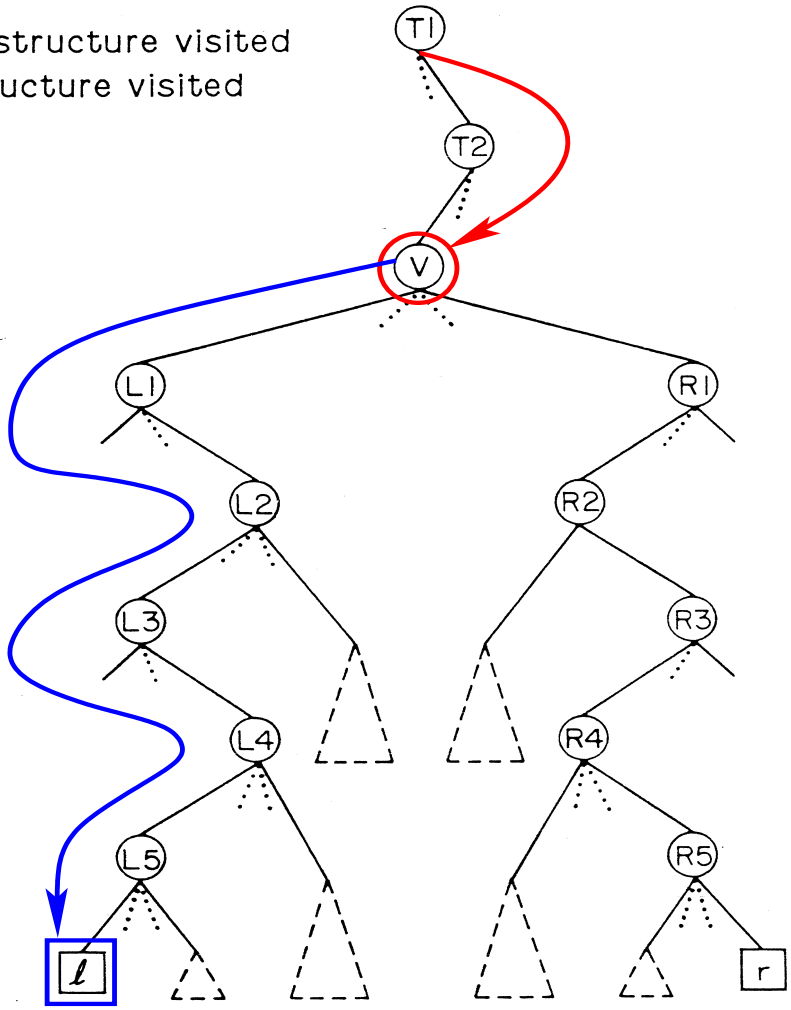


1.Start at root, T_1 and located first node V where l < VAL(V) < r

RECTANGLE INTERSECTION WITH INTERVAL TREES

- Perform task while inserting vertical line segment S=[l,r)
- 3 stage search: report all line segments encountered that overlap S
- Each line segment is only reported once as it is associated with the secondary structure of just one node in the primary structure

● ● ● ● ● secondary structure visited
- - - - - tertiary structure visited

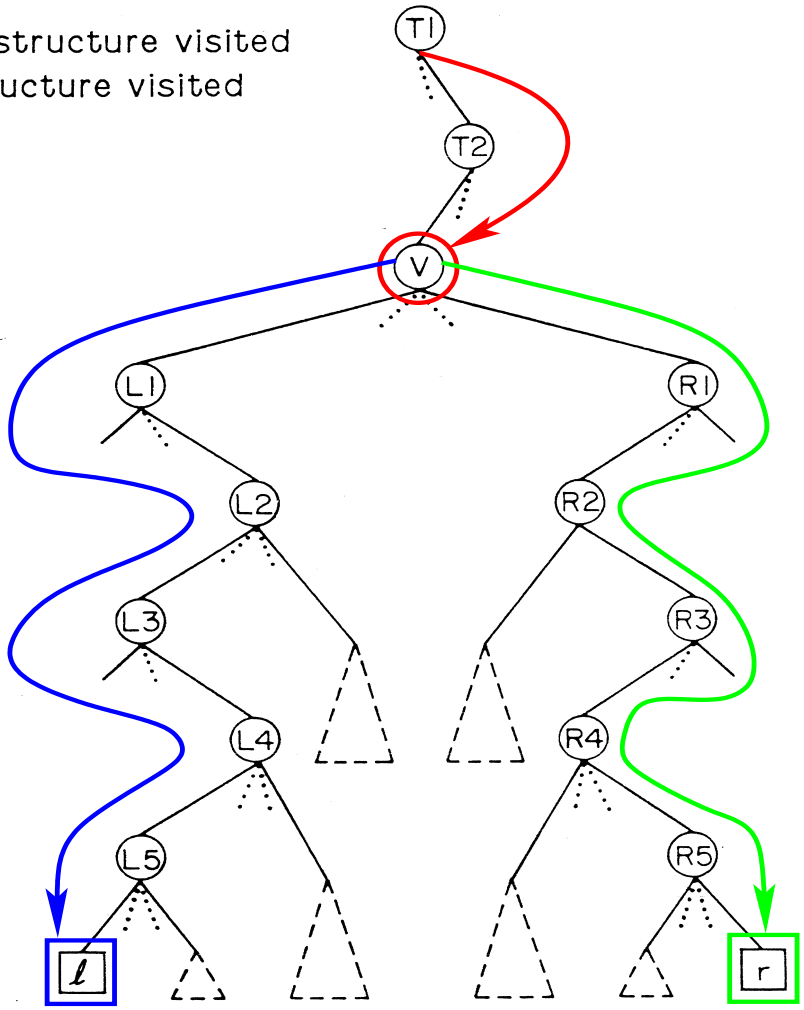


1.Start at root, T_1 and located first node V where l < VAL(V) < r
2.Start at V and locate l in the left subtree of V.

RECTANGLE INTERSECTION WITH INTERVAL TREES

- Perform task while inserting vertical line segment S=[l,r)
- 3 stage search: report all line segments encountered that overlap S
- Each line segment is only reported once as it is associated with the secondary structure of just one node in the primary structure

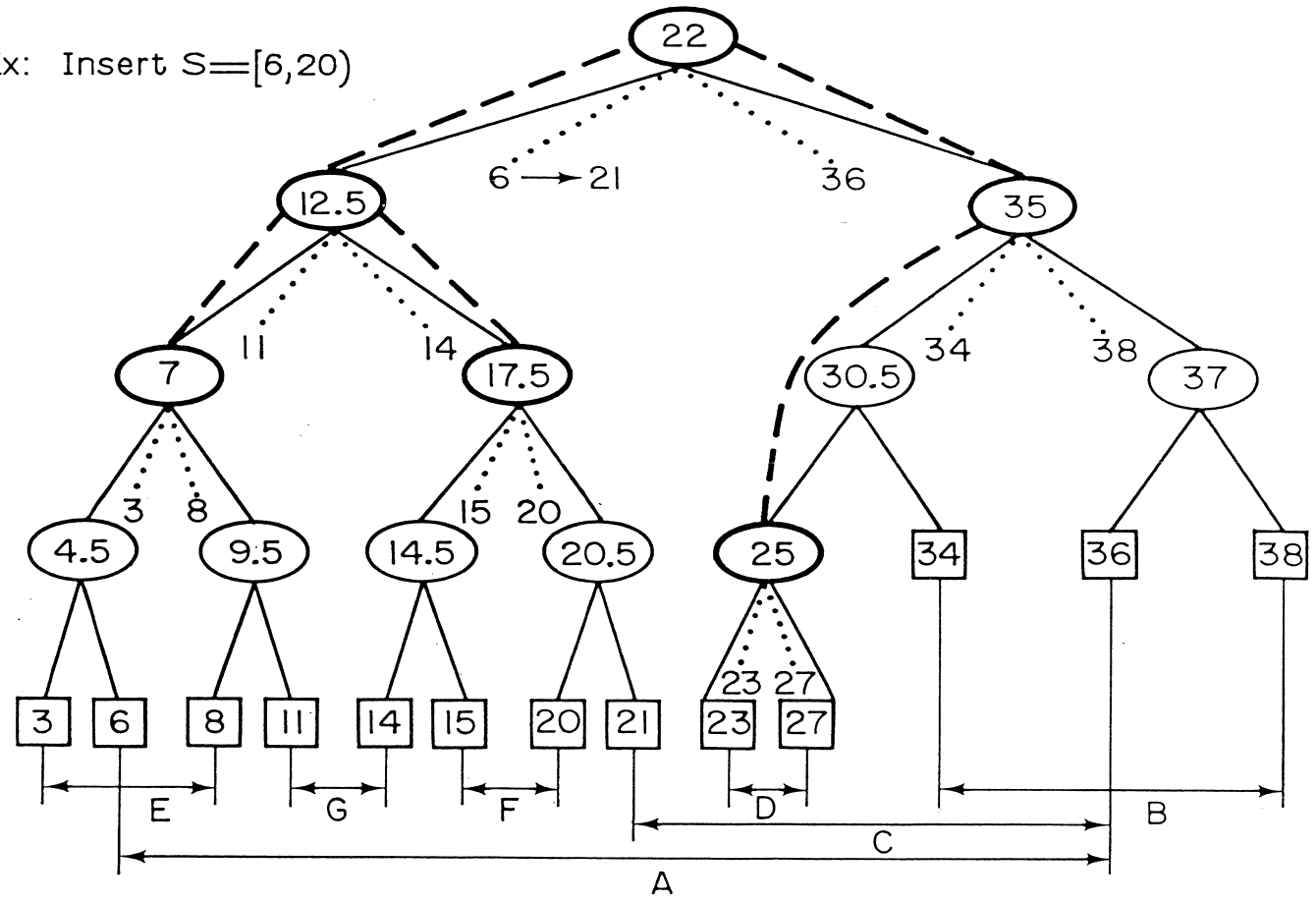
● ● ● ● ● secondary structure visited
- - - - - tertiary structure visited



1.Start at root, T_1 and located first node V where l < VAL(V) < r
2.Start at V and locate l in the left subtree of V.
3.Start at V and locate r in the right subtree of V.

STAGE ONE

Ex: Insert S=[6,20)



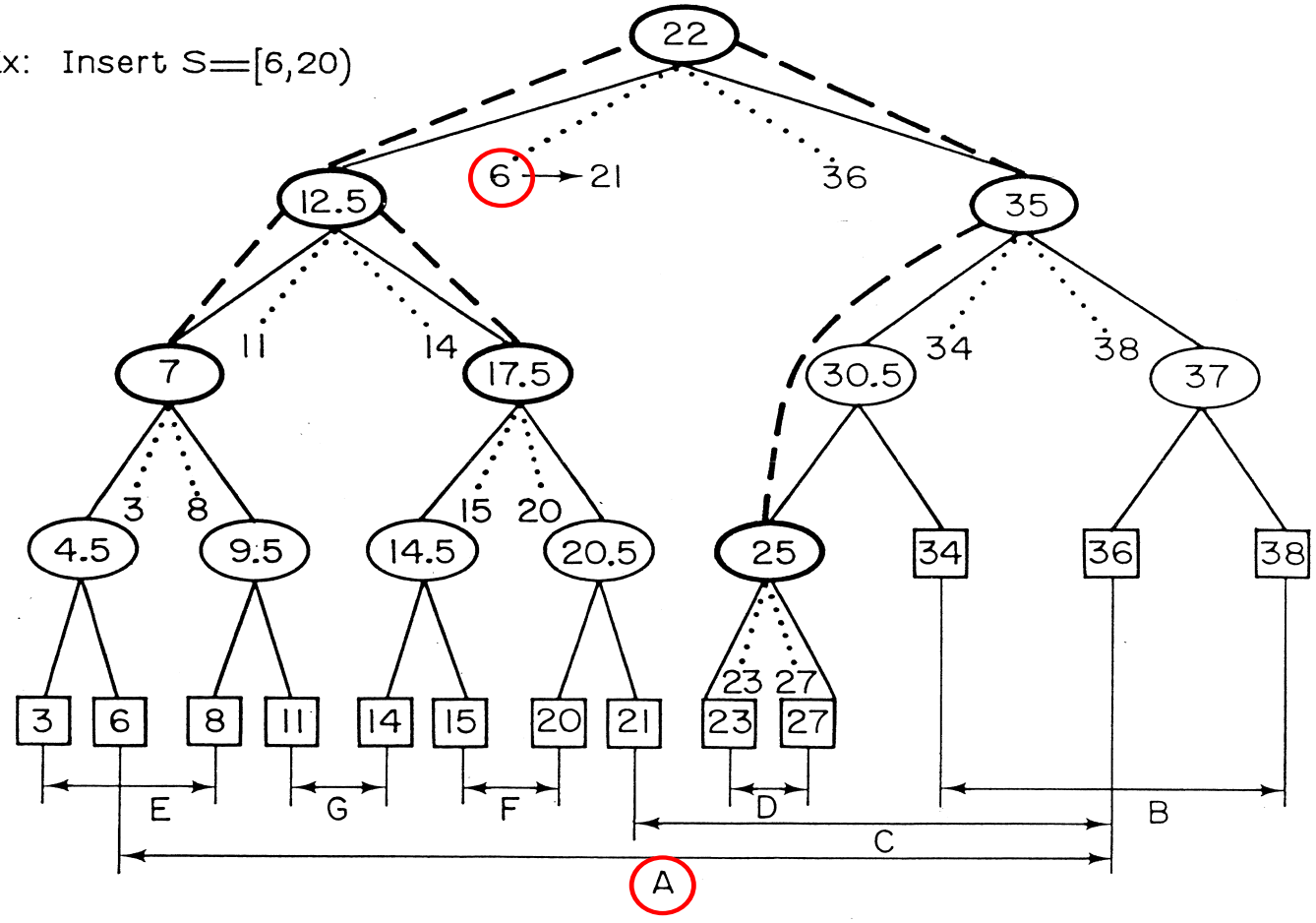
- Check each secondary structure of T for overlap with S=[l,r)
- Either l<r<VAL(T) or VAL(T)<l<r

- If r<VAL(T) then just report all line segments in LS(T) with starting points <r (e.g., A)
- To find them, visit LS(T) in ascending order until encountering a line segment with starting point >r

- Continue in the left subtree of T

STAGE ONE

Ex: Insert S=[6,20)



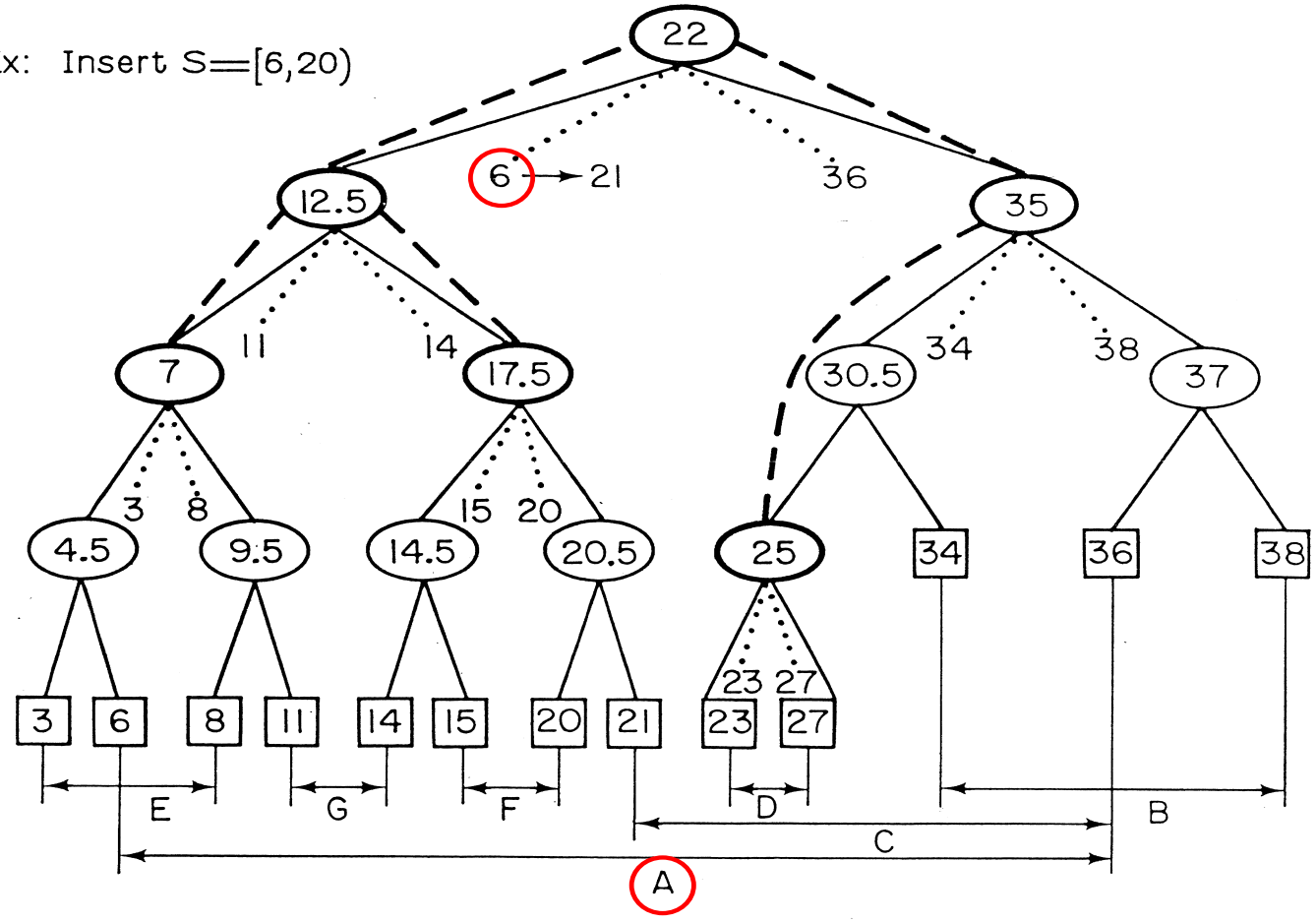
- Check each secondary structure of T for overlap with S=[l,r)
- Either l<r<VAL(T) or VAL(T)<l<r

- If r<VAL(T) then just report all line segments in LS(T) with starting points <r (e.g., A)
- To find them, visit LS(T) in ascending order until encountering a line segment with starting point >r

- Continue in the left subtree of T

STAGE ONE

Ex: Insert S=[6,20)



- Check each secondary structure of T for overlap with S=[l,r)
- Either l<r<VAL(T) or VAL(T)<l<r

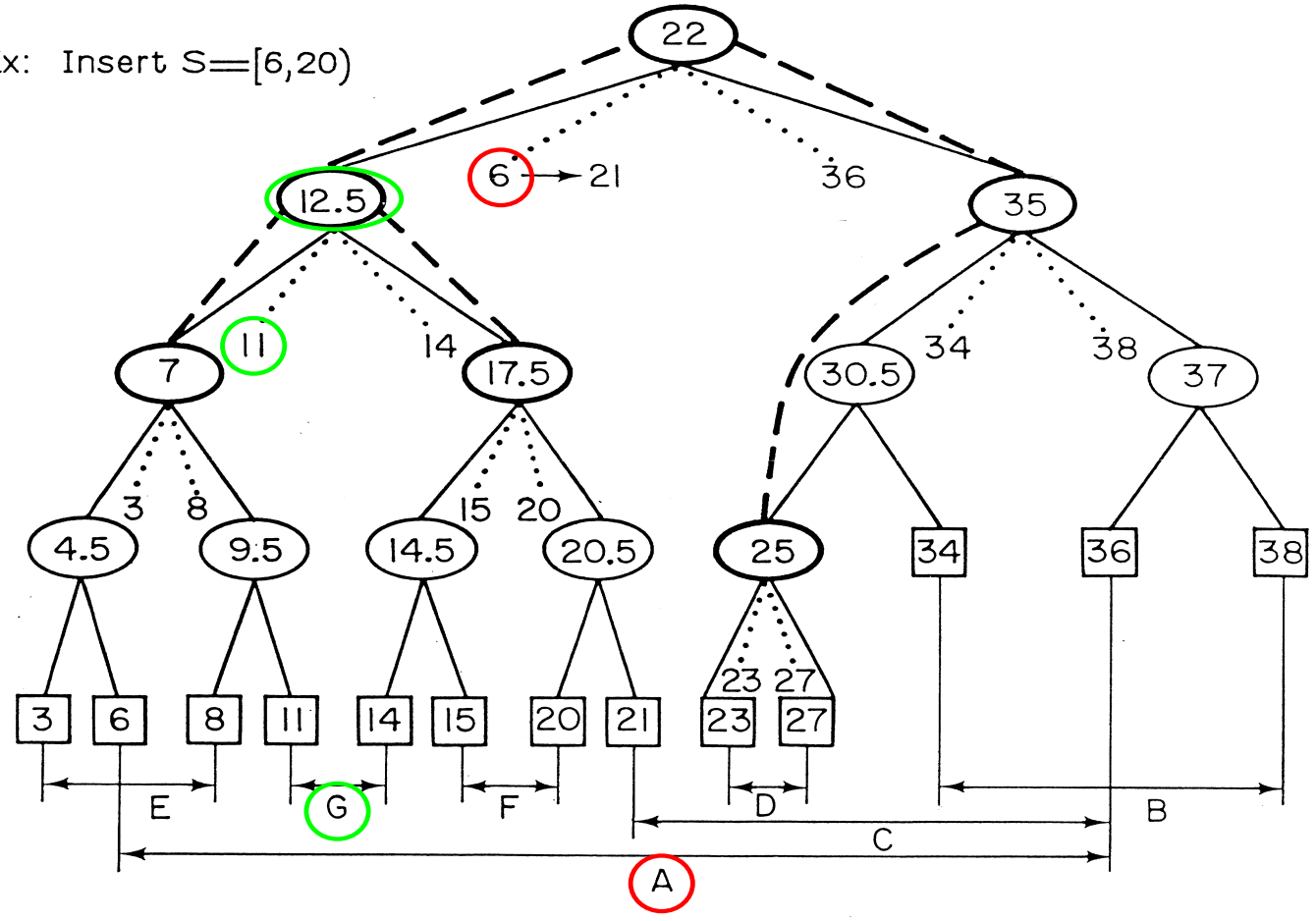
- If r<VAL(T) then just report all line segments in ~~LS(T)~~ RS(T) with ~~starting~~ ending points ~~<~~ r > l (e.g., A)
- To find them, visit ~~LS(T)~~ RS(T) in ~~ascending~~ descending order until encountering a line segment with ~~starting~~ ending point ~~>~~ r < l

- Continue in the ~~left~~ right subtree of T

- If VAL(T) < l then analogous process

STAGE ONE

Ex: Insert S=[6,20)



- Check each secondary structure of T for overlap with S=[l,r)
- Either l<r<VAL(T) or VAL(T)<l<r

- If r<VAL(T) then just report all line segments in ~~LS(T)~~ RS(T) with ~~starting~~ ending points ~~<~~ r > l (e.g., A)
- To find them, visit ~~LS(T)~~ RS(T) in ~~ascending~~ descending order until encountering a line segment with ~~starting~~ ending point ~~>~~ r < l

- Continue in the ~~left~~ right subtree of T

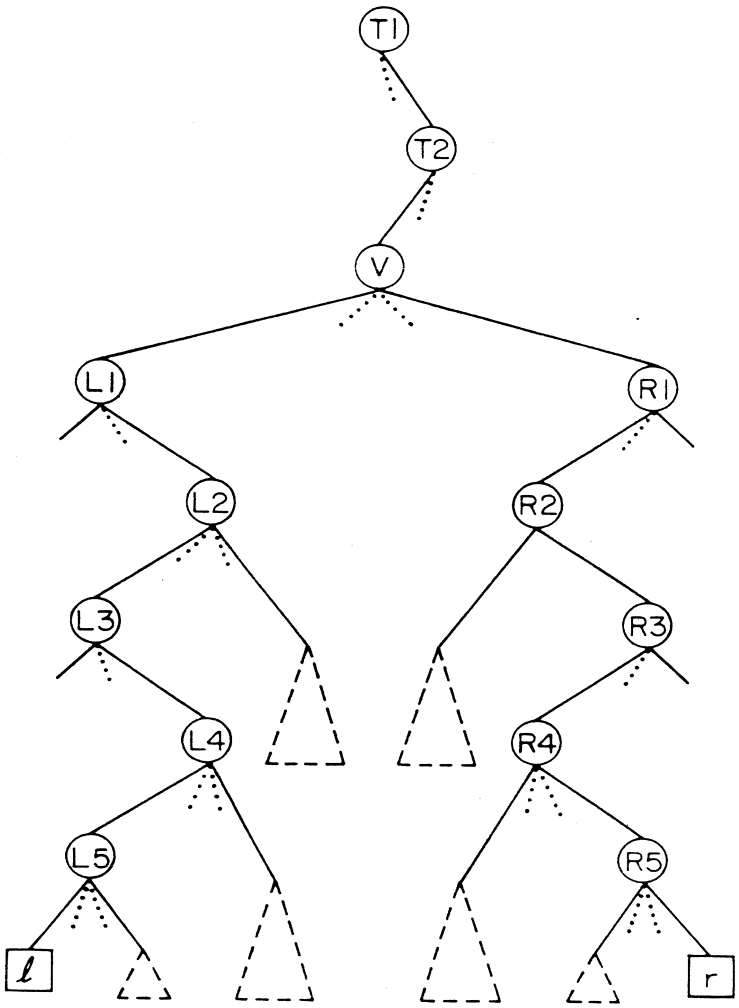
- If VAL(T) < l then analogous process

- Terminate when locate V. Now all elements of LS(V) are reported as intersecting S (e.g., G at V=12.5)

STAGE TWO

- $\{L_i\}$ = set of nodes encountered in the search for l
- Either $l < VAL(L_i)$ or $VAL(L_i) < l$

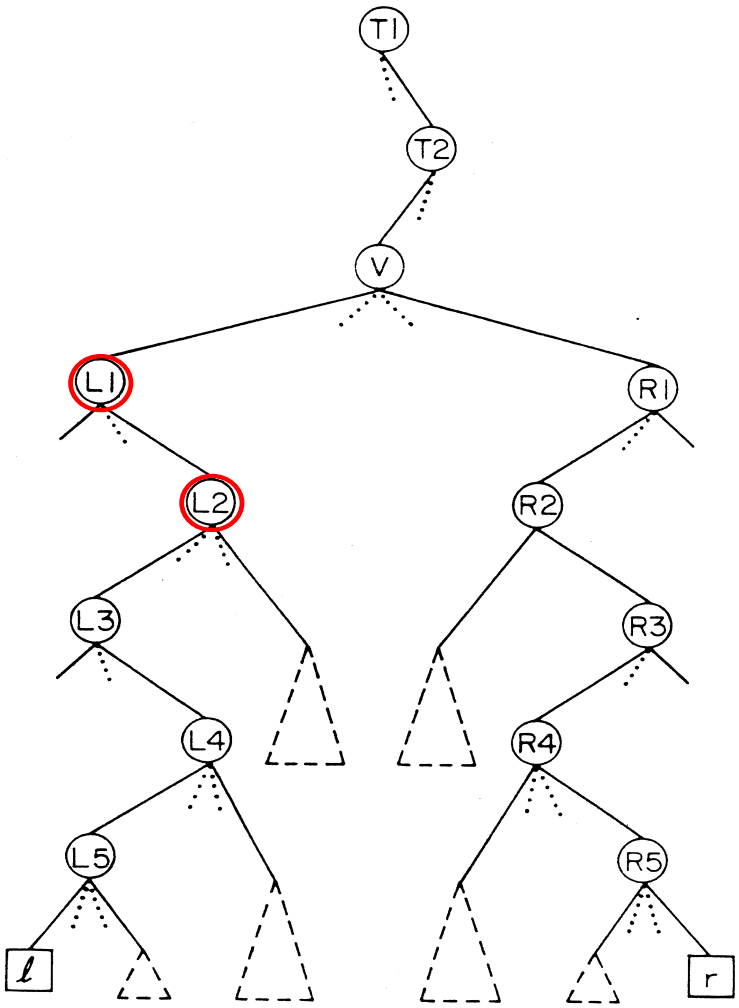
Ex: Insert S=[l,r)



STAGE TWO

- $\{L_i\}$ = set of nodes encountered in the search for l
- Either $l < VAL(L_i)$ or $VAL(L_i) < l$

Ex: Insert S=[l,r)

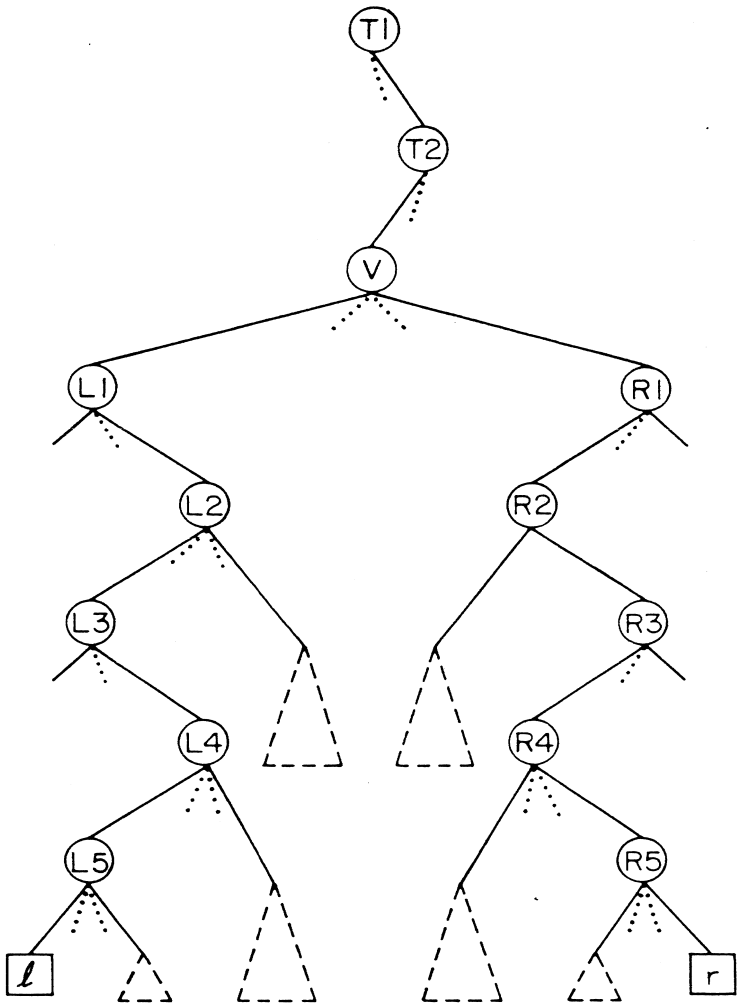


$VAL(L_i) < l$:

- Report all line segments in secondary structure of L_i with ending points > l
  ≡ visit $RS(L_i)$ in descending order until finding a line segment with ending point < l
- Continue to search in right subtree of L_i

$l < VAL(L_i)$:

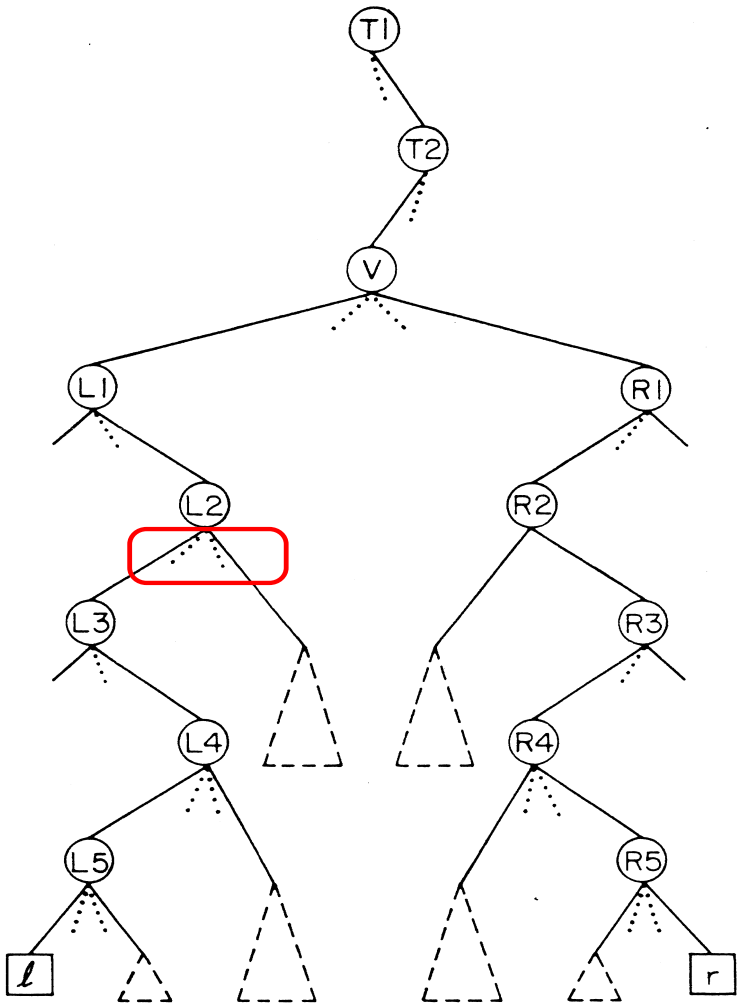
Ex: Insert $S = [l, r)$



1. S intersects every line segment in the secondary structure of L_i
2. and all line segments in the secondary structures of the right subtrees of L_i.
   - Find by visiting all nodes in the right subtree of L_i
   - Tertiary structure assures that we don't visit irrelevant nodes
   - More than one half of the active nodes have non-empty secondary structures
3. Continue search in the left subtree of L_i

$l < VAL(L_i)$:

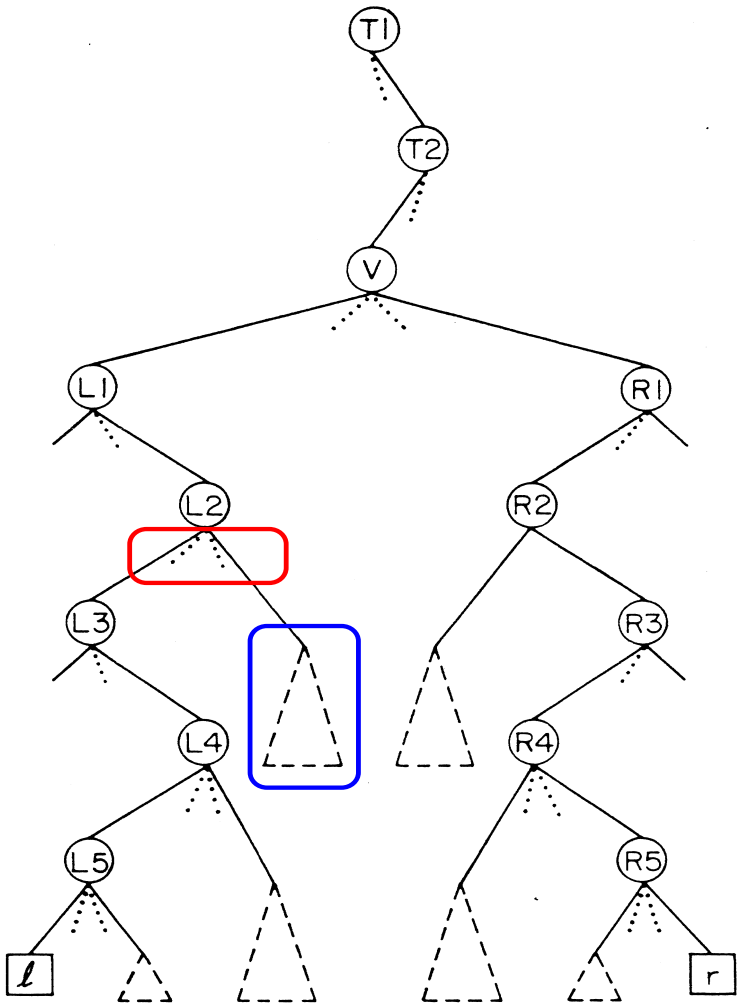
Ex: Insert $S = [l, r)$



1. S intersects every line segment in the secondary structure of L_i
2. and all line segments in the secondary structures of the right subtrees of L_i.
   - Find by visiting all nodes in the right subtree of L_i
   - Tertiary structure assures that we don't visit irrelevant nodes
   - More than one half of the active nodes have non-empty secondary structures
3. Continue search in the left subtree of L_i

$l < VAL(L_i)$:

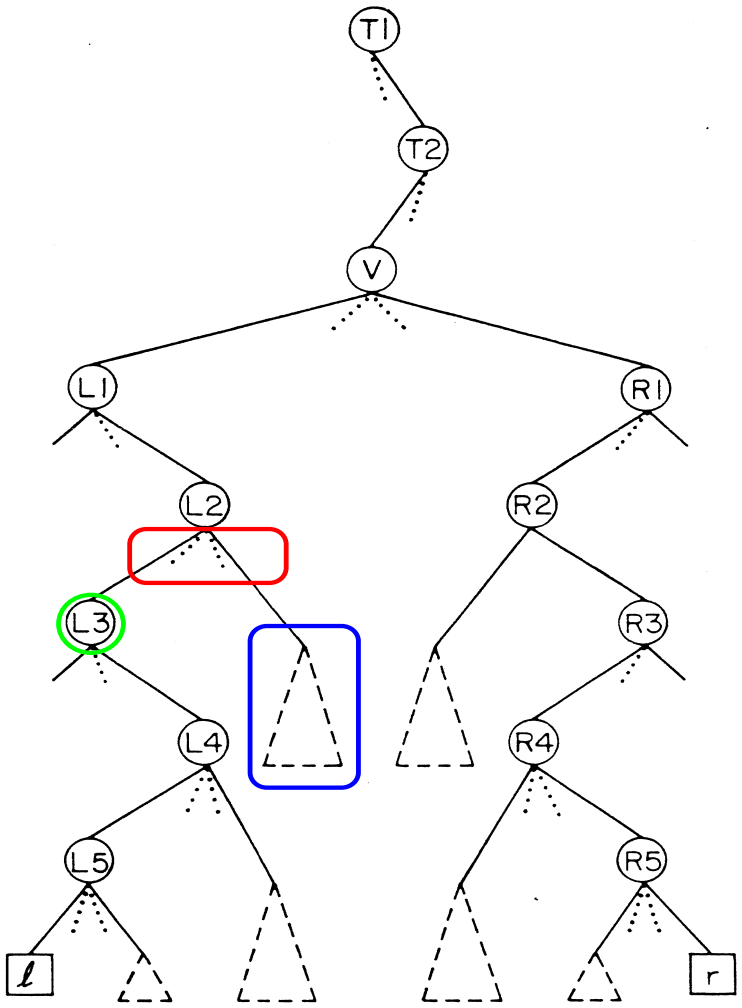
Ex: Insert $S = [l, r)$



1. S intersects every line segment in the secondary structure of L_i
2. and all line segments in the secondary structures of the right subtrees of L_i.
   - Find by visiting all nodes in the right subtree of L_i
   - Tertiary structure assures that we don't visit irrelevant nodes
   - More than one half of the active nodes have non-empty secondary structures
3. Continue search in the left subtree of L_i

$l < VAL(L_i)$:

Ex: Insert $S = [l, r)$



1. S intersects every line segment in the secondary structure of L_i
2. and all line segments in the secondary structures of the right subtrees of L_i.
   - Find by visiting all nodes in the right subtree of L_i
   - Tertiary structure assures that we don't visit irrelevant nodes
   - More than one half of the active nodes have non-empty secondary structures
3. Continue search in the left subtree of L_i

ANALYSIS

1. o(N) space

   - 2·N terminal nodes in the primary structure
   - 2·N terminal nodes in the secondary structure
   - Tertiary structure consists of nodes from the primary structure and only needs extra space for the pointer fields

2. $o(N \cdot \log_2 N)$ time

   - $o(\log_2 N)$ time to search primary structure for start and end points of a line segment
   - Number of nodes visited in the secondary structure is proportional to the number of rectangle intersections
   - At least one half of the active nodes have non-empty secondary structures so that the number of nodes visited in the tertiary structure is no more than 2 times the number of nodes visited in the secondary structure
   - $o(N \cdot \log_2 N)$ time to initially sort the endpoints of the line segments

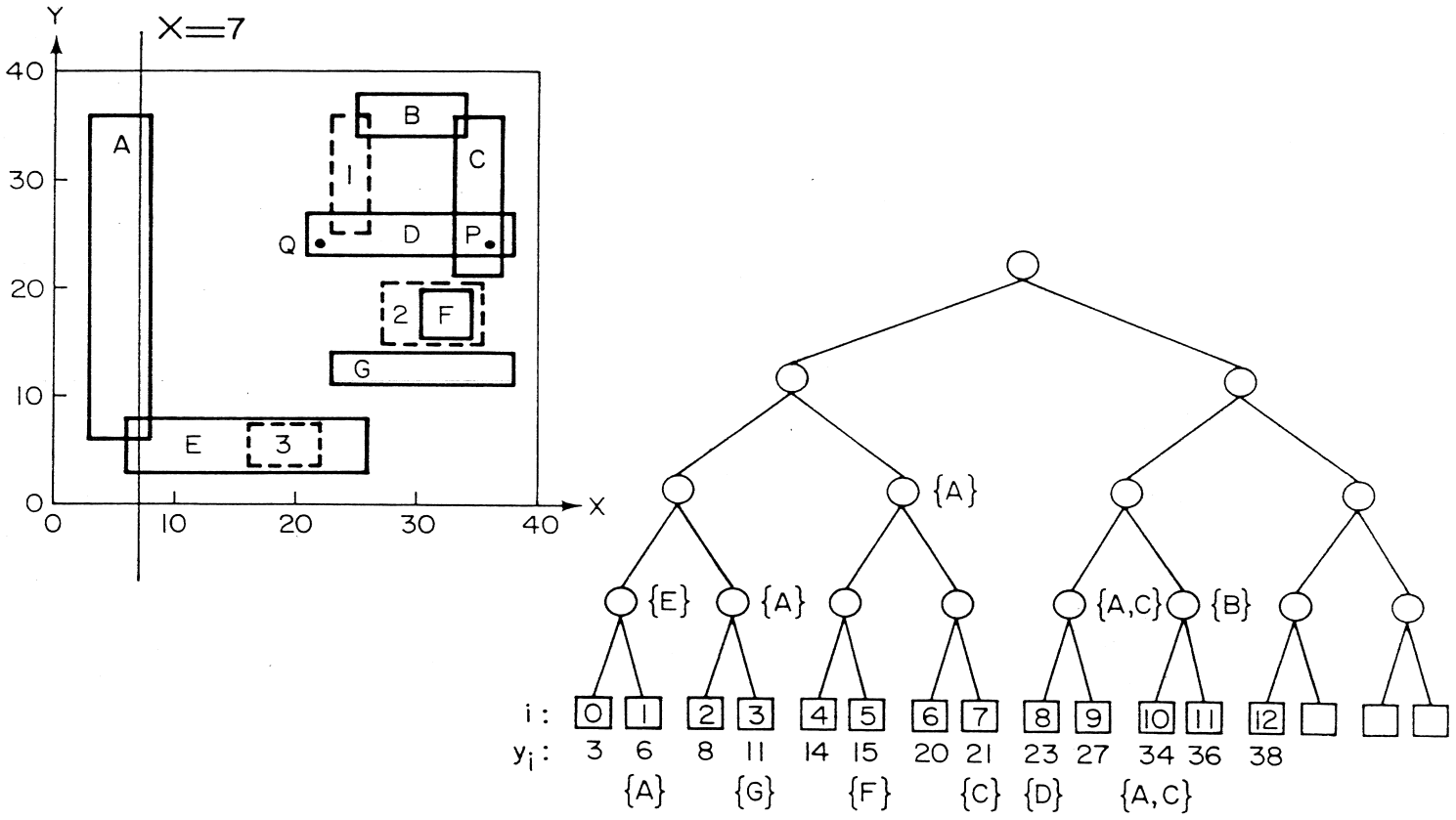
APPLICATION OF PLANE-SWEEP METHODS

- Can use a segment tree to solve measure problems (e.g., area, perimeter)
- 3-d measure problems are solved by use of a 2-d region quadtree to organize the data in the swept plane

Ex: Compute area

- Keep track of the total length of the vertical scan line, say L, at halting point X_i that overlaps vertical boundaries of rectangles active at X_i
- Adjust L at each halting point
- Total area is accumulating product of $L \cdot (X_i - X_{i-1})$
- Each node of the tree contains the length of the overlap of its marked components
- Record the number of times each node is marked - good for deletion

Ex: Status when X=7



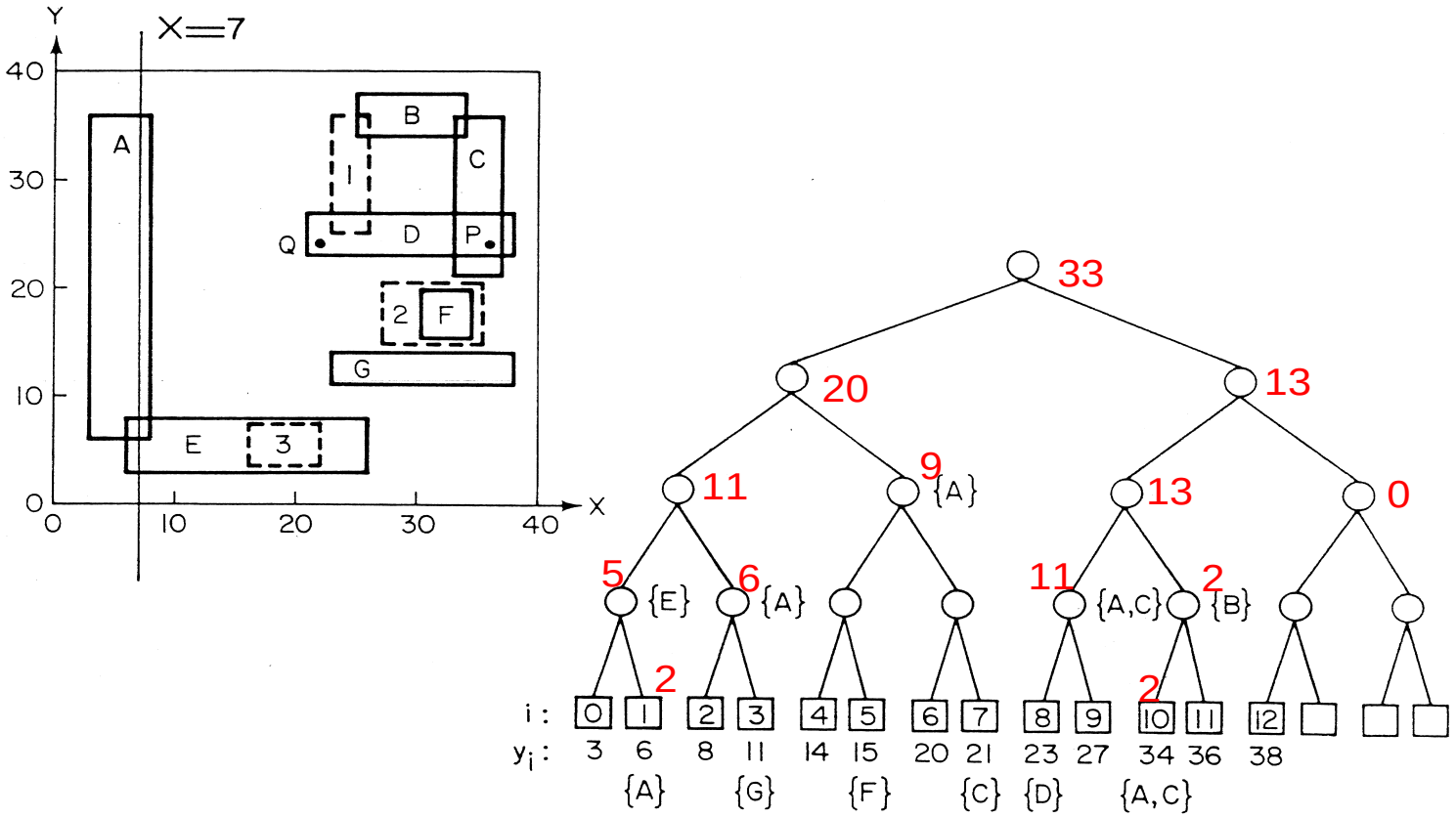
APPLICATION OF PLANE-SWEEP METHODS

- Can use a segment tree to solve measure problems (e.g., area, perimeter)
- 3-d measure problems are solved by use of a 2-d region quadtree to organize the data in the swept plane

Ex: Compute area

- Keep track of the total length of the vertical scan line, say L, at halting point X_i that overlaps vertical boundaries of rectangles active at X_i
- Adjust L at each halting point
- Total area is accumulating product of $L \cdot (X_i - X_{i-1})$
- Each node of the tree contains the length of the overlap of its marked components
- Record the number of times each node is marked - good for deletion

Ex: Status when X=7



APPLICATION OF PLANE-SWEEP METHODS

- Can use a segment tree to solve measure problems (e.g., area, perimeter)
- 3-d measure problems are solved by use of a 2-d region quadtree to organize the data in the swept plane

Ex: Compute area

- Keep track of the total length of the vertical scan line, say L, at halting point X_i that overlaps vertical boundaries of rectangles active at X_i
- Adjust L at each halting point
- Total area is accumulating product of $L \cdot (X_i - X_{i-1})$
- Each node of the tree contains the length of the overlap of its marked components
- Record the number of times each node is marked - good for deletion

Ex: Status when X=7