Representing collections of small rectangles

- Plane sweep algorithms and interval trees
- Point-based representations of rectangles
- Area based methods

Rectangles - 1

Typical problems

- Given N rectangles, report all intersections between pairs of rectangles
- Compute the area of a collection of N rectangles in the plane
- Application areas
  - VLSI design
  - Geographic information systems
- Assumptions
  - all rectangles upright
  - rectangles open on upper and right boundaries, closed on lower and left boundaries

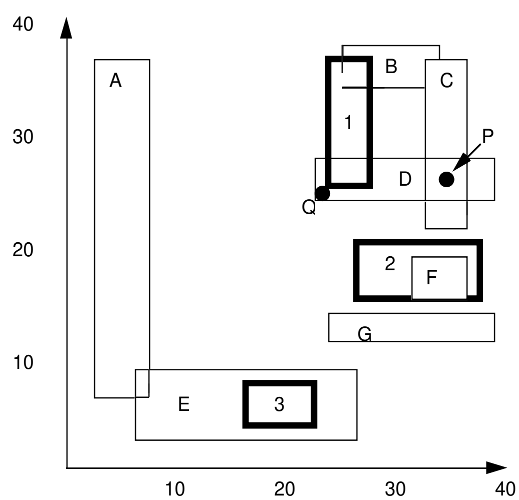
Rectangles - 2

Data set for examples

Rectangle	x_L	x_R	y_L	y_R	
A	3	8	6	36	
B	25	34	34	38	
C	33	37	21	36	
D	21	38	23	27	
E	6	26	3	8	
F	31	35	15	19	
G	23	38	11	14	
1	23	26	25	36	} query rectangles
2	27	35.5	14.5	20.5	
3	16	22	3.5	7.5	
P	36		24		} query points
Q	21		24		

Rectangles - 3

Data set for examples



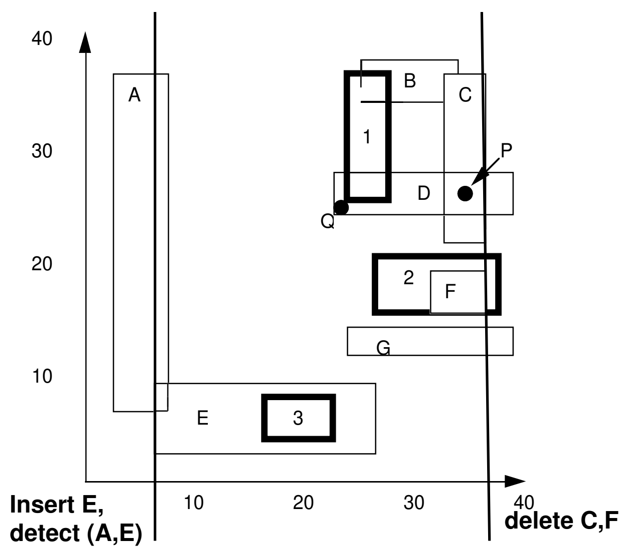
Rectangles - 4

Plane sweep algorithm for rectangle intersections

- Basic idea
  - rectangles A and B intersect if and ony if a corner of A (B) is contained in B (A)
  - sort the x coordinates of the vertices of the rectangles, and sweep through this set from smallest to greatest using a vertical scan line
  - Rectangles that intersect the scan line are called active. The set of active rectangles is changed as the scan line is swept.
  - Efficiently determine the intersections between a newly activated rectangle and the currently active rectangles.
  - Basic computational problem is developing a good data structure with efficient algorithms for insertion, deletion and intersection detection for the active set.

Rectangles - 5

Plane sweep



Rectangles - 6

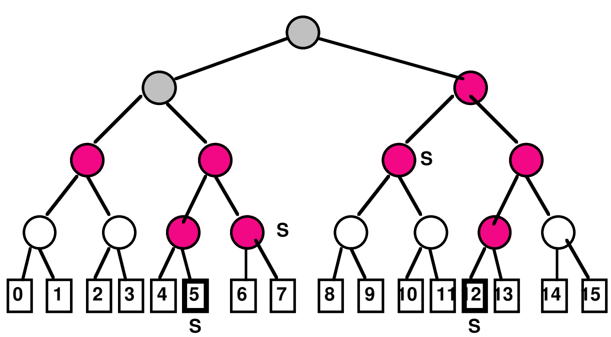
Representing the active set - segment trees

- Unit segment tree for representing a single line
  - endpoints of lines drawn from set of integers {i| $0 \le i \le 2^h$}
  - S is a line segment with end points l and r, and can be regarded as set of unit intervals {[j,j+1) | $l \le j < r$}
- Unit segment tree is a complete binary tree of height h
  - leaf nodes correspond to the unit intervals
  - node at level i represents the interval [p, p + 2^i) where p mod $2^i = 0$
- Represent a line segment in a unit segment tree by marking the minimal number of nodes required to cover the line by the intervals represented by those nodes.

Rectangles - 7

Unit segment trees

- Compare S against interval at node T, initially the root.
- If $S \supset T$, mark T and halt. If $S < T$ or $S > T$, halt.
- Otherwise, repeat process for left and right sons of T.
- Process visits at most 4 nodes per level.
- So, insertion takes O(h) time.



Example: S = [5,12]

Rectangles - 8

Segment trees - many lines, real-valued endpoints

- Given a set of N line segments corresponding to the vertical edges of our rectangles, the segment tree will represent the y-coordinate intervals of each line segment
- Sort the set of y coordinates to obtain y_0, y_1, ..., y_m, $m \leq 2N$.
- The unit segments are replaced by the base segments or base intervals $[y_j, y_{j+1})$, $0 \leq j < 2^h$ and $2^{h-1} \leq m < 2^h$.
- A line segment, S, with endpoints y_b and y_t consists of the consecutive intervals $[y_j, y_{j+1})$, $b \leq j < t$.

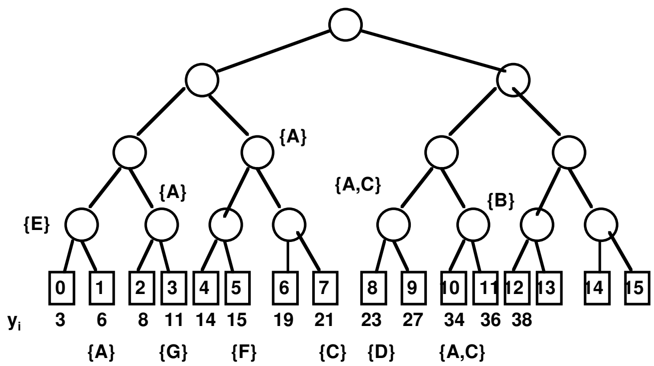
Rectangles - 9

Segment trees

- Nonleaf nodes correspond to sequences of 2^i consecutive basic intervals. Also, associate a "midrange" value with each nonleaf node.
- A node's interval may be contained in many line segments, so we associate with each node a list of the names of the line segments that it is contained in. Call it the segment list.
  - List is represented as a doubly linked list for ease of insertion and deletion during the plane sweep.
- Note that even though we may never store all of the line segments in the segment tree at any one time, its size is determined by the size of the entire data set.

Rectangles - 10

Segment trees



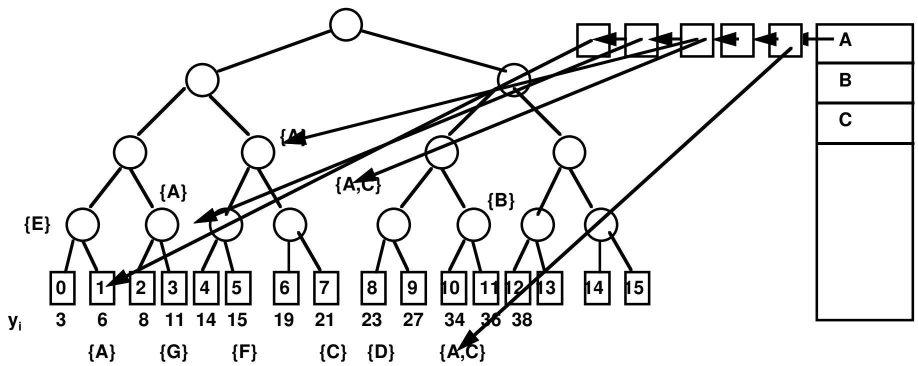
Rectangles - 11

Operations on segment trees

- Insertion: similar to insertion into unit segment tree; must also add name of line segment to lists associated with the segment.
  - For a domain with N line segments, this takes O(logN) time - the height of the segment tree.
- Deletion is more complex
  - Finding the nodes in the tree whose segment lists contain the segment to be deleted is easy - O(logN) operations
  - but each of these lists may contain O(N) items. We do not want to search them, because then the entire algorithm would be worst case $O(N^2)$ - deletion of N lines during the plane sweep, each taking O(N) time to delete.
  - Solved by maintaining a table, with one entry for each line segment in the domain. Each table entry points to a list of pointers to nodes in the segment lists of nodes associated with that line

Rectangles - 12

Deletion from segment trees



Deletion

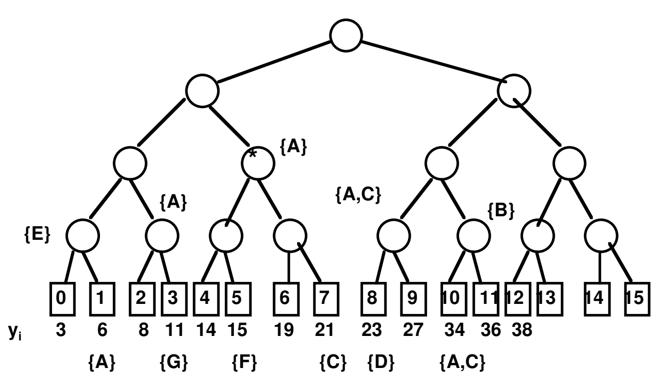
- Table can be implemented as a balanced binary tree (then O(logN) to find the entry for a line segment) or even as an array since the set of line segments is known beforehand (then O(1) time to find an entry)
- The lists associated with each element of the table can be only O(logN) long, since that is the maximum number of nodes marked in the segment tree for any line segment.
- Deletion of a specific item in one segment tree node's segment list is then O(1) because we have the doubly linked list.
- So total cost of deletion is O(logN) and we are still on track for an O(NlogN) algorithm

Rectangle intersections

- The plane sweep halts at each rectangle x-coordinate.
  - » first delete from the segment tree all rectangles that end at that x-coordinate
  - » then insert into the segment tree the rectangles that begin at that x-coordinate. As each rectangle is inserted, we check it for intersections with all rectangles in the segment tree.
- Segment tree, itself, is not good enough to detect intersections
  - » When S is inserted into the segment tree, need to find all existing line segments that are either entirely contained in S or overlap S.
  - » But this would require searching every node in every subtree containing S

Rectangles - 15

Examples



- Consider inserting A into segment tree containing {B, ..., G}
  - » To detect the intersection with F, we have to traverse the subtree rooted at *.
- This can take O(N) time for each of the n line segments, giving us an O(N^2) algorithm

Rectangles - 16

Computing intersections

- Split problem into two cases for detecting all intersections with rectangle having y-coordinates in the range [b,t)
- Find all rectangles with starting point < b that overlap [b,t). This is equivalent to finding all active rectangles that contain b and is done with the segment tree.
- Find all rectangles with starting points in [b,t). This is done with a range tree.

Rectangles - 17

Computing intersections

- Case I
  - Traverse the path in the segment tree from the root to the leaf whose interval contains b.
  - For each node on that path, report the line segments on the associated lists.
  - Only log(N) nodes will be visited
  - Each line segment will be encountered only once, since a node and its sons are never labeled with the same line segment.
  - So, total time complexity is (logN + F_b) using space O(NlogN). F_b is the number of active rectangles that contain b.

Rectangles - 18

Computing intersections

- Case II:
  - Perform a range query for [b,t) against the starting points of the current line segments.
  - Can be done using a range tree
  - This requires $O(\log N + F_t)$ time but only O(N) space.
  - Note that the rectangles found are disjoint from those found under Case I.
- The combination of the two cases takes O(NlogN + F) time and O(NlogN) space - dominated by the segment tree.
- How can we improve on the space bound?

Rectangles - 19

Interval trees

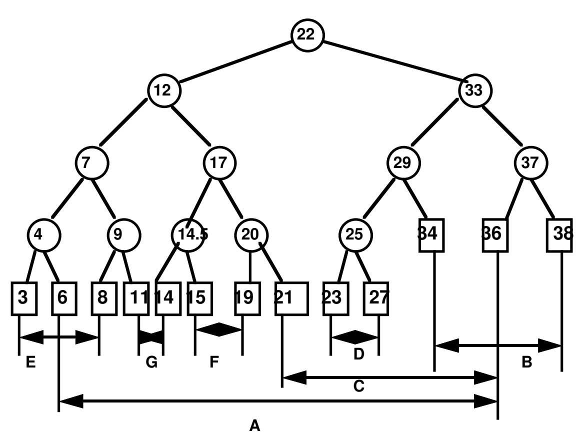
- Replacment for segment tree that only requires O(N) storage
  - a line segment will only be associated with one node in the tree, rather than log(N).
  - that node is the nearest common ancestor of all nodes contained in the line segment
  - some nodes will have additional information associated with them (lists of coordinates and pointers that jump through the tree) but not enough to raise the overall storage complexity past O(N).
  - basic algorithms will have same time complexity as for segment tree approach
- But is is complicated . The tree has primary, secondary and tertiary structure!

Rectangles - 20

Primary structure of interval trees

- As before, y_0, y_1, ..., y_m are the y-coordinates of the segment in ascending order.
- Primary structure is a complete binary tree with m+1 leaf nodes, one for each of the y_i.
- Nonleaf nodes have associated midrange values that lie between the maximum value in their left subtrees and minimum in their right subtrees.
- Just like a segment tree.

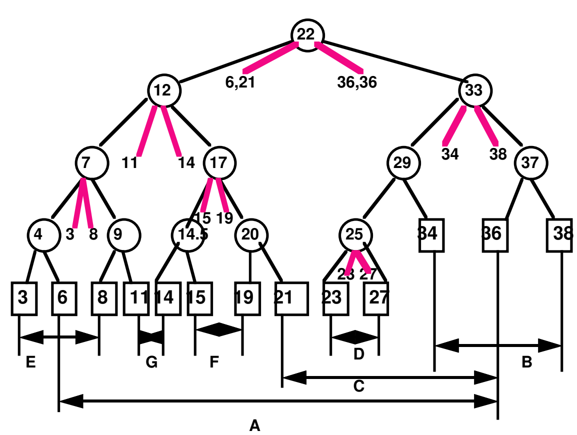
Primary structure



Secondary structure of segment trees

- Given a line segment [b,t) its nearest common ancestor is the nonleaf node containing b in its left subtree and t in its right subtree
  - node with midrange value of 17 is nearest common ancestor of end points of segment F.
- Each non-lead node in primary structure has two lists associated with it that constitute the secondary structure:
  - LS - left end points of those segments for which the node is the nearest comnmon ancestor. End points are sorted in ascending order, with duplicates included. Implementation is balanced binary tree and a doubly linked list.
  - RS - similar structure for the right end points.

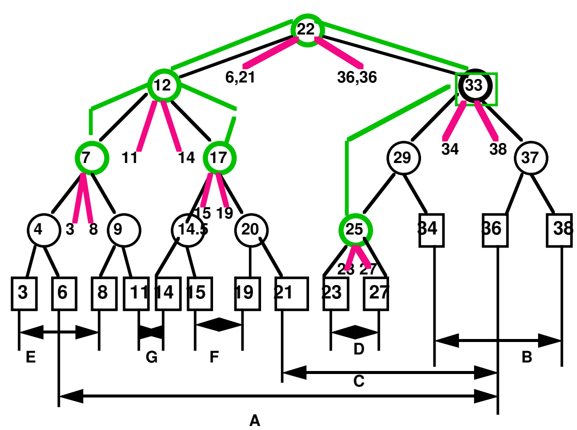
Secondary structure - example



Tertiary structure

- A nonleaf node in the primary structure is active if either:
  - its secondary structure is nonempty, or
  - both of its sons have active descendants.
- Other nodeas are called inactive
- Active nodes organized as a binary tree
  - Root is the root of the primary structure
  - If v is active, then its left son in the tertiary structure, LT, is the closest active node in its left subtree
  - Right son, RT, is closest active node in the right subtree
  - Inactive nodes are given null LT and RT values.

Tertiary structure - example



Operations on interval trees

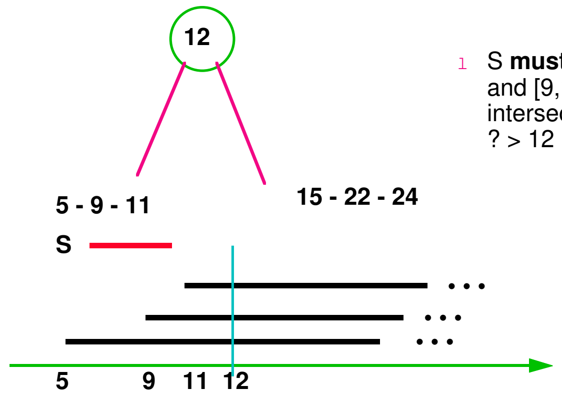
- Insertion of segment S = [b,t).
  - start at root and locate first node v with b < val(v) < t
  - insert b into LS(v) and t into RS(v)
  - must also traverse and update tertiary structure at same time
  - all operations take O(logN) time
- Deletion - similar to insertion
- Reporting all rectangles that intersect [b,t). Three stage search
  - Start at root and find first node v with b < val(v) < t
  - Start at v and locate b in left subtree of v in primary structure
  - Start at v and locate t in right subtree of v in primary structrure
  - Last 2 steps use tertiary structure to avoid nodes whose Ls and RS sets are null.

Intersections in interval trees

- Let $\{T_i\}$ denote nodes encountered in search for v.
- We check the secondary structure of each T_i for overlap with S. Two possibilities:
  - $t < val(T_i)$ - S to the left of val
  - $val(T_i) < b$ - S to the right of val
- If t < val, then we need to find all segments in the secondary structure of T_i whose starting points are < t.
  - These are guaranteed to overlap S, since all segments in secondary structure of T_i must end at $t' > val(T_i) > t$
  - We find these segments by traversing $LS(T_i)$ in ascending order, terminating when we encounter a value >t

Example

S = [6, 10)



- S **must** intersect [5,?) and [9,?) but **cannot** intersect [11,?), where ? > 12

Rectangles - 29

Intersection in interval trees

- If val < b, then we need to find all segments in the secondary structure of T_i whose ending points are > b.
  - These are guaranteed to overlap S, since all segments in secondary structure of T_i must begin at b' < val(T_i) < b
  - We find these segments by traversing RS(T_i) in descending order, terminating when we encounter a value < b
- Time required is proportional to number of intersections reported.

Rectangles - 30

Intersection - from v to the leaves

- Let $\{L_i\}$ be the sequence of nodes encountered in searching for b from node v. Two cases to consider:
- First case is $b < val(L_i)$.
  - Then S intersects every segment in the secondary structure of L_i because **all** of these segments contain $Val(L_i)$ and $Val(L_i)$ ε [b,t), sinced we also have that $t > val(v) > val(L_i)$.
  - S also intersects every segment in the secondary structure of the right subtree of L_i. This is because the val's of these nodes all fall in [b,t) (all greater than b because they are in the right subtree of L_i, and $val(L_i) > b$; all < t because they are in the left subtree of v and $val(v) < t$.), and every segment contains one of those val's as a point.
  - The right subtree is traversed using the tertiary structure to avoid visiting "too many" worthless nodes (only half of them can be worthless - i.e., have no secondary structure!
  - search is continued in $LS(L_i)$ in primary structure

Rectangles - 31

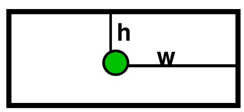
Intersection - from v to the leaves

- Second case is $val(L_i) < b$
  - all segments in the secondary structure of L_i begin to the left of b.
  - So, those that end to the right of b must intersect S.
  - Search $RS(L_i)$ in descending order, stopping when we encounter a value < b. This would correspond to a segment completely to the left of S.
  - Continue search in $RS(L_i)$ in primary structure.
- Overall, process require O(NlogN + F) time, as before, but only O(N) space.
  - primary structure has only 2N leaf nodes
  - secondary structure represents each segment once, at some node in the primary structure, so it uses O(N) space also
  - tertiary structure only adds at most 2 pointers per node to the primary structure - also O(N).

Rectangles - 32

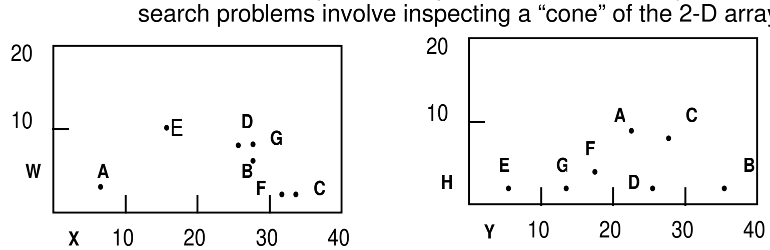
Representing rectangles by points

- Interval trees represent rectangles by intervals
- Can also represent an rectangle as a point in 4 dimensional space
  - (x,y) coordinates of opposite corners
  - (x,y) coordinates of one corner, and 2 side lengths
  - (x,y) coordinates of rectangle “center” and horizontal and vertical “radii”



Representing rectangles by points

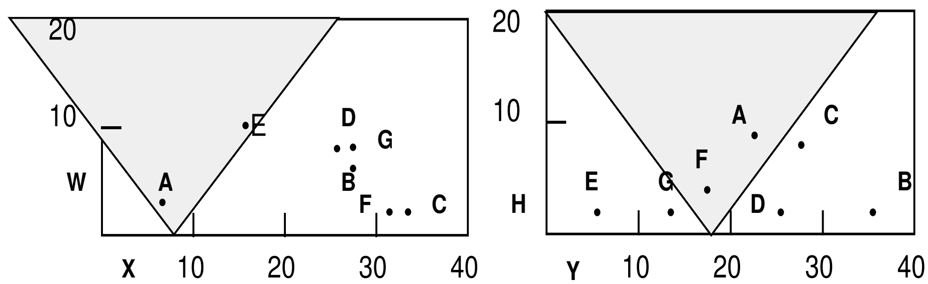
- Implementation of these representations decompose the 4-D space into a product of 2 2-D spaces
  - e.g., take the 4-tuple (x_c, y_c, w,h) and represent it in two 2-D planes, (x_c,w), (y_c,h)
  - simplest data structure would be an array who x-axis corresponds to x_c (y_c) and y-axis corresponds to w(h)
  - When we decompose the representation in this way, most search problems involve inspecting a “cone” of the 2-D arrays



Point in rectangle queries

- How do we determine which rectangles contain a given point (x,y)
  - in the (x_c,w) space, this corresponds to all rectangles in the 45º cone centered at (x,0)
  - in the (y_c,h) space, this corresponds to all rectangles in the 45º cone centered at (y,0)
  - The intersection of these two sets are the enclosing rectangles.

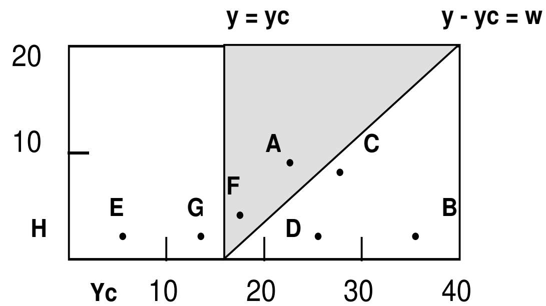
Example - rectangles containing (8,18)



- In the (x,w) plane, we find candidate rectanges A and E
- In the (y,h) plane we find candidates A and F
- Their common subset is {A}

Why the 45º cone?

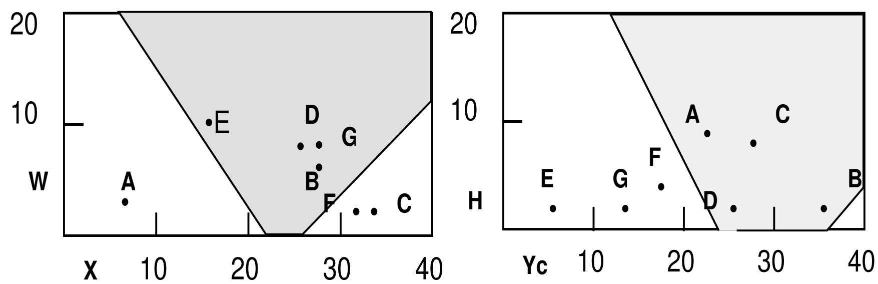
- x is in (x_c,w) if either:
  - » $x \geq x_c$ and $x - x_c \leq w$ **or**
  - » $x \leq x_c$ and $x_c - x \leq w$
- Similar conditions for y



Find all rectangles that intersection a query rectangle

- This problem can also be solved by inspecting "cones" in the 2-D representations
- Now, the base of the cone in the (x_c,w) space is the x-interval of the query rectangle
  - » think of this as the union of all the cones for the points in the interval
- The base of the cone in the (y_c,h) space is the y-interval of the query rectangle.

Example - query rectangle 1

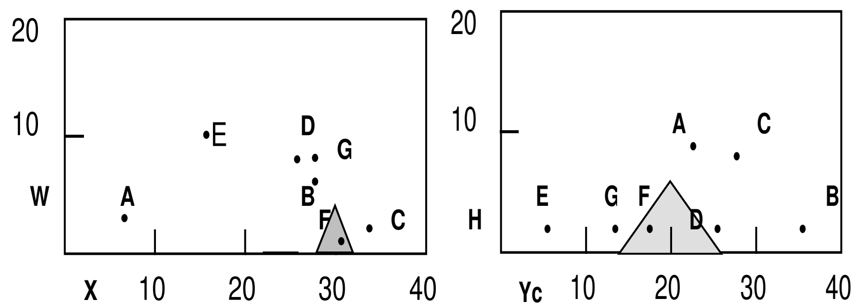


- In the (x_c,w) space, rectangle 1 intersects B,D,E,G
- In the (y_c,h) space, it intersects A,B,C,D
- So, answer to query is {B,D}

Finding all rectangles contained within a query rectangle

- Search regions are again cones in the 2-D spaces
- Cones open downward at 45°
- Centered at the (x_c ,w), (y_c,h) representation for the query rectangle.

Example - rectangle 2



Rectangles - 41

Implementation issues

- We do not want to implement rectangle queries using the 2-D array approach
  - Most algorithms will be $O(n^2)$, since the lists of rectangles identified in the 2-D queries can be $O(n)$ long and we have to compuote their intersection
  - Representation is wasteful of space
- Possible approaches
  - corner stitching
  - k-d trees
  - grid files

Rectangles - 42