Geometric Data Structures

Swami Sarvottamananda

Ramakrishna Mission Vivekananda University

THAPAR-IGGA, 2010

Outline I

1. Introduction
    - Motivation for Geometric Data Structures
    - Scope of the Lecture
2. Range searching
    - Kd-trees
    - Range Trees
3. Interval Queries
    - Interval Trees
    - Segment Trees
4. Sweeping Technique
    - Linear Sweep
    - Angular Sweep
5. Conclusion

Introduction

Motivation-I

- Why do we need special data structures for Computational Geometry?

Motivation-I

- Why do we need special data structures for Computational Geometry?
- Because objects are more complex than set of arbitrary numbers.
- And yet, they have geometric structure and properties that can be exploited.

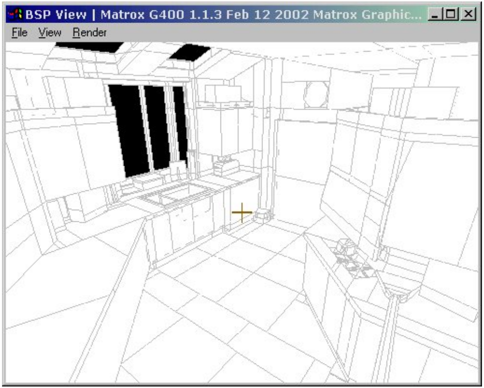
Motivation-I: Visibility in plane/space

Any first-person-shooter game needs to solve visibility problem of computational geometry which is mostly done by Binary Space Partitions (BSP) [4, 5]. (Software: BSPview)

Motivation-I: Visibility in rough terrain

We might not have enclosed space, or even nice simple objects.
(Software: BSPview)

Motivation-I: Visibility in a room



At every step, we need to compute visible walls, doors, ceiling and floor. (Software: BSPview)

Motivation-I: Calculation of Binary Space Partitions

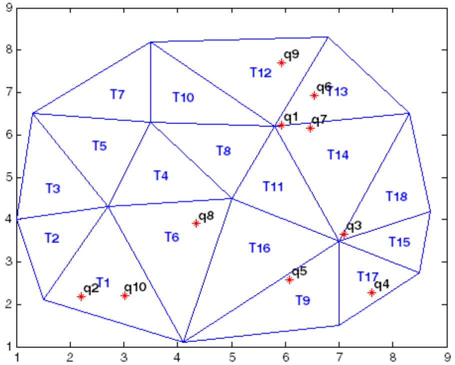
The data structure that is useful in this situation is known as *Binary Space/Planar Partitions*.

Almost every 3D animation with moving camera makes use of it in rendering the scene.

Motivation-II: Locating given objects is geometric subdivisions

Another problem, we might need to locate objects (the elephant) in distinct regions like trees, riverlet, fields, etc. (GPLed game: 0AD)

Motivation-II: Location of objects in subdivision

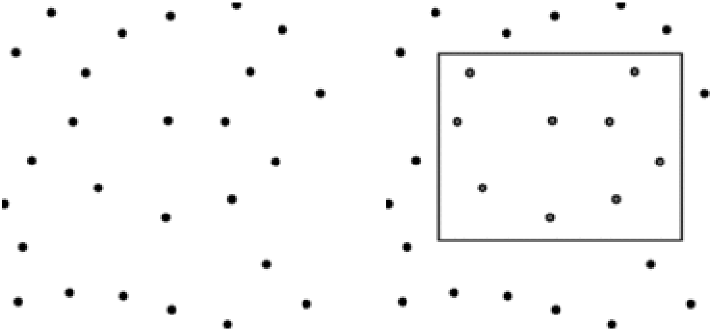


This problem is known as point location problem in its simplest special case.

Motivation-III: Finding objects in a window

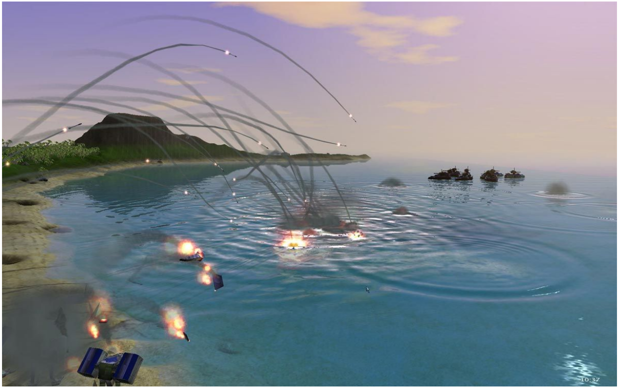
Yet in another case, we need to find all objects in a given window that need to be drawn and manipulated. (GPLed game: 0AD)

Motivation-III: Problem of Rangesearching



This problem is known as 2D/3D range searching.

Motivation-IV: Finding intersections of objects



This is classical collision detection. Intersection of parabolic trajectories with a 3D terrain. (GPLed game: TA-Spring)

Motivation-IV: Problem of Collision Detection/Finding Intersections

This problem is known as collision detection.
In the static case it is just the intersections computation problem.

Scope of the lecture

- *Binary search trees and Kd-trees:* We consider 1-d and 2-d range queries for point sets.

Scope of the lecture

- *Binary search trees and Kd-trees:* We consider 1-d and 2-d range queries for point sets.
- *Range trees:* Improved 2-d orthogonal range searching with range trees.

Scope of the lecture

- *Binary search trees and Kd-trees:* We consider 1-d and 2-d range queries for point sets.
- *Range trees:* Improved 2-d orthogonal range searching with range trees.
- *Interval trees:* Interval trees for reporting all intervals on a line containing a given query point on the line.

Scope of the lecture

- *Binary search trees and Kd-trees:* We consider 1-d and 2-d range queries for point sets.
- *Range trees:* Improved 2-d orthogonal range searching with range trees.
- *Interval trees:* Interval trees for reporting all intervals on a line containing a given query point on the line.
- *Segment trees:* For reporting all intervals in a line containing a given query point on the line.

Scope of the lecture

- *Binary search trees and Kd-trees:* We consider 1-d and 2-d range queries for point sets.
- *Range trees:* Improved 2-d orthogonal range searching with range trees.
- *Interval trees:* Interval trees for reporting all intervals on a line containing a given query point on the line.
- *Segment trees:* For reporting all intervals in a line containing a given query point on the line.
- *Paradigm of Sweep algorithms:* For reporting intersections of line segments, and for computing visible regions.

Not in the Scope yet relevant

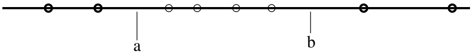
- *Point location Problem:* The elegant solution makes use of traditional data structures such as height balanced trees which are augmented and modified to suite the purpose.

Not in the Scope yet relevant

- *Point location Problem:* The elegant solution makes use of traditional data structures such as height balanced trees which are augmented and modified to suite the purpose.
- *BSP trees:* Trees are usually normal binary trees again (not even height balanced), so we skip it, even though it is quite interesting and needs a lecture by itself to properly treat the subject.

Range Searching

1-dimensional Range searching

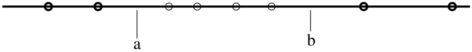


Problem

Given a set P of n points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.

- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where k is the number of points of P in $[a, b]$.

1-dimensional Range searching

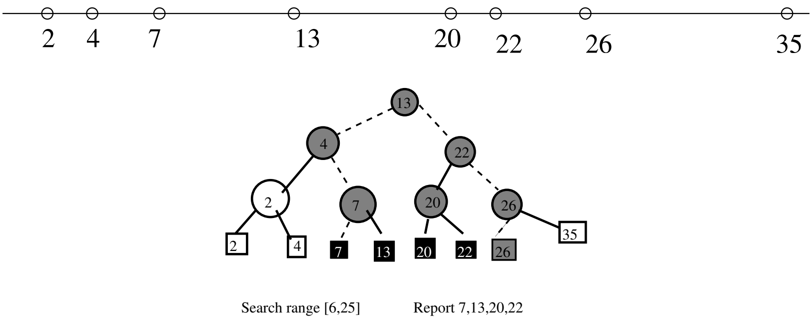


Problem

Given a set P of n points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.

- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where k is the number of points of P in $[a, b]$.
- However, when we permit insertion or deletion of points, we cannot use an array answering queries so efficiently.

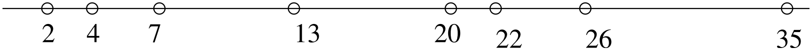
1-dimensional Range searching

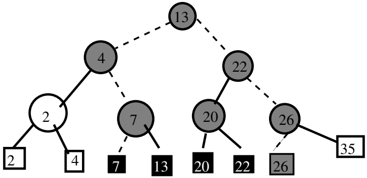


Search range [6,25] Report 7,13,20,22

- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.

1-dimensional Range searching

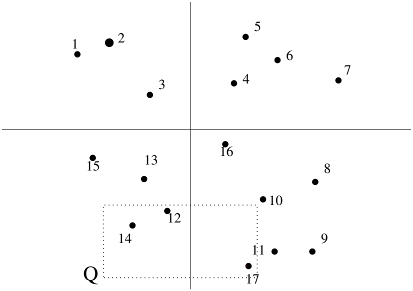




Search range [6,25]   Report 7,13,20,22

- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.
- Each internal node stores the x-coordinate of the rightmost point in its left subtree for guiding search.

2-dimensional Range Searching



Problem

Given a set P of n points in the plane, report points inside a query rectangle Q whose sides are parallel to the axes.

Here, the points inside Q are 14, 12 and 17.

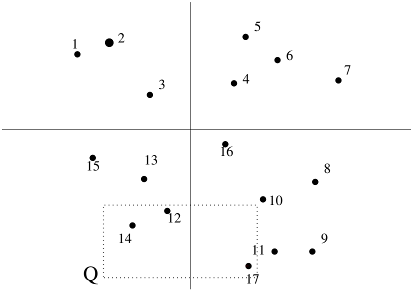
Kd-trees for Range Searching Problem

We use a data structure called Kd-tree to solve the problem efficiently.
What are Kd-trees?

Kd-trees for Range Searching Problem

Kd-trees are basically trees where we divide data first on x-coordinates, then y-coordinates, then z-coordinates, etc. and then cycle.

2-dimensional Range Searching

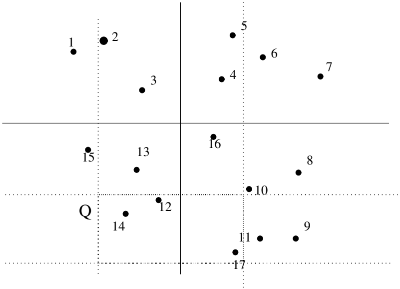


Problem

Given a set P of n points in the plane, report points inside a query rectangle Q whose sides are parallel to the axes.

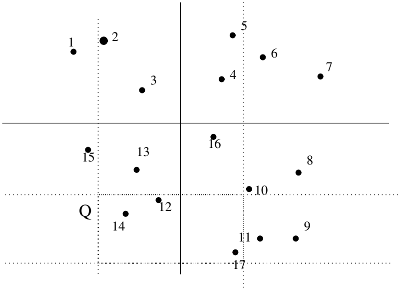
Here, the points inside Q are 14, 12 and 17.

2-dimensional Range Searching - using 1-dimensional Range Searching



- Using two 1-d range queries, one along each axis, solves the 2-d range query.

2-dimensional Range Searching - using 1-dimensional Range Searching

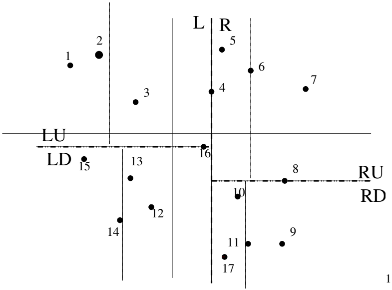


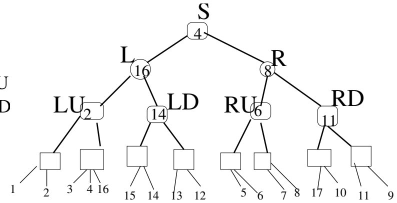
- Using two 1-d range queries, one along each axis, solves the 2-d range query.
- The cost incurred may exceed the actual output size of the 2-d range query (worst case is $O(n)$, $n = |P|$).

2-dimensional Range Searching: Using Kd-trees

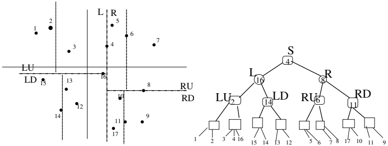
Can we do better?
We can do significantly better using Kd-tree (from $O(n)$ to $O(\sqrt{n})$.

Kd-trees


L
R
5
2
1
6
4
7
3
LU
LD
15
13
16
8
RU
RD
10
12
14
9
11
17

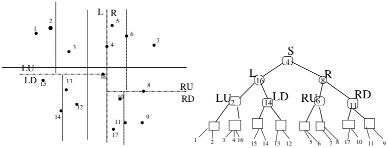

S
4
L
16
R
8
LU
2
14
LD
RU
6
RD
11
1
2
3
4
16
15
14
13
12
5
6
7
8
17
10
11
9

Kd-trees: Some concepts



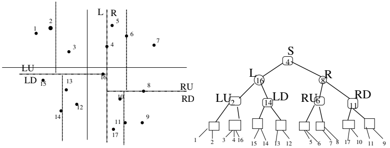
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x- and y- coordinates, respectively as follows.

Kd-trees: Some concepts



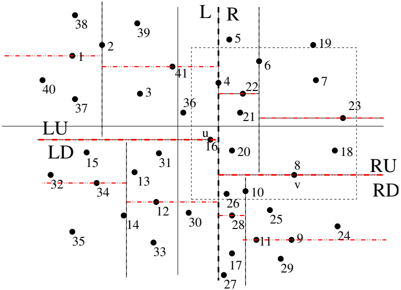
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x- and y- coordinates, respectively as follows.
- The point r stored in the root vertex T splits the set S into two roughly equal sized sets L and R using the median x-coordinate $xmedian(S)$ of points in S, so that all points in L (R) have coordinates less than or equal to (strictly greater than) $xmedian(S)$.

Kd-trees: Some concepts



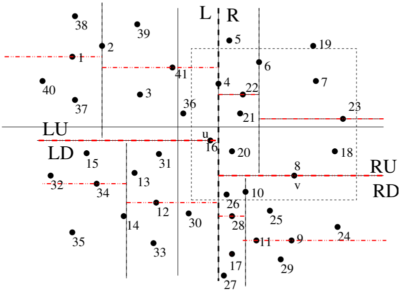
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x- and y- coordinates, respectively as follows.
- The point r stored in the root vertex T splits the set S into two roughly equal sized sets L and R using the median x-coordinate $xmedian(S)$ of points in S, so that all points in L (R) have coordinates less than or equal to (strictly greater than) $xmedian(S)$.
- The entire plane is called the $region(r)$.

Kd-trees: Some Concepts



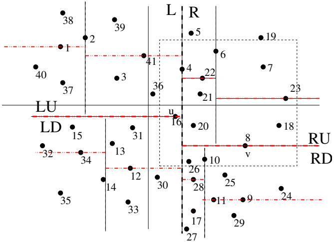
- The set L (R) is split into two roughly equal sized subsets LU and LD (RU and RD), using point u (v) that has the median y-coordinate in the set L (R), and including u in LU (RU).

Kd-trees: Some Concepts



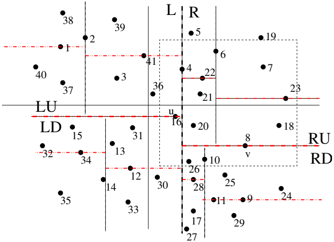
- The set L (R) is split into two roughly equal sized subsets LU and LD (RU and RD), using point u (v) that has the median y-coordinate in the set L (R), and including u in LU (RU).
- The entire halfplane containing set L (R) is called the $region(u)$ $(region(v))$.

Answering rectangle queries using Kd-trees



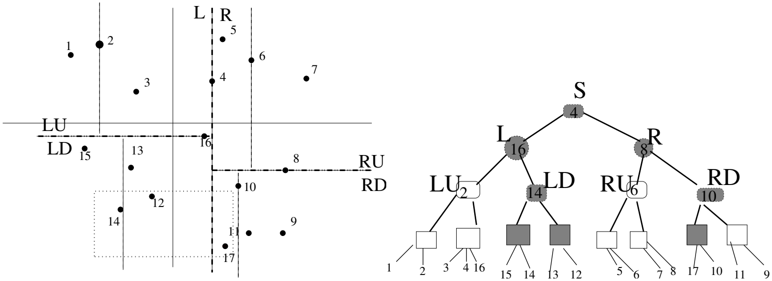
- A query rectangle Q may partially overlap a region, say $region(p)$, completely contain it, or completely avoids it.

Answering rectangle queries using Kd-trees



- A query rectangle Q may partially overlap a region, say $region(p)$, completely contain it, or completely avoids it.
- If Q contains an entire bounded $region(p)$ then report all points in $region(p)$.
- If Q partially intersects $region(p)$ then descend into the children.
- Otherwise skip $region(p)$.

Time complexity of rectangle queries



Time complexity of output point reporting

- Reporting points within Q contributes to the output size k for the query.

Time complexity of output point reporting

- Reporting points within Q contributes to the output size k for the query.
- No leaf level region in T has more than 2 points.

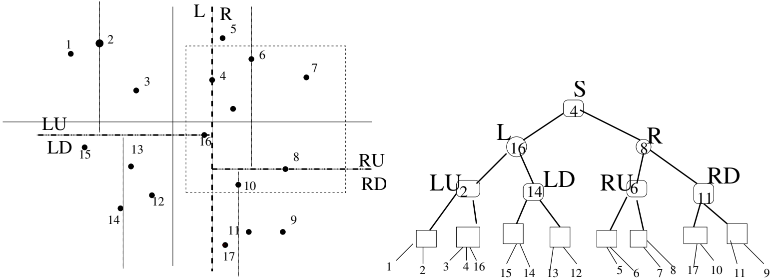
Time complexity of output point reporting

- Reporting points within Q contributes to the output size k for the query.
- No leaf level region in T has more than 2 points.
- So, the cost of inspecting points outside Q but within the intersection of leaf level regions of T can be charged to the internal nodes traversed in T.

Time complexity of output point reporting

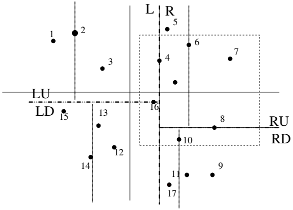
- Reporting points within Q contributes to the output size k for the query.
- No leaf level region in T has more than 2 points.
- So, the cost of inspecting points outside Q but within the intersection of leaf level regions of T can be charged to the internal nodes traversed in T.
- This cost is borne for all leaf level regions intersected by Q.

Time complexity of traversing the tree



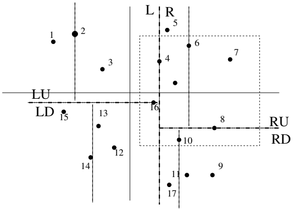
Now we need to bound the number of nodes of T traversed for a given query Q.

Time complexity of traversing the tree



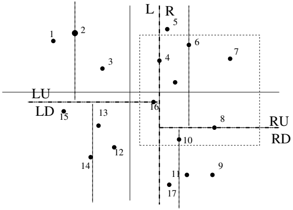
- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.

Time complexity of traversing the tree



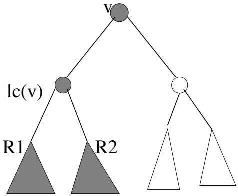
- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.
- Any vertical line intersecting S can intersect either L or R but not both, but it can meet both RU and RD (LU and LD).

Time complexity of traversing the tree



- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.
- Any vertical line intersecting S can intersect either L or R but not both, but it can meet both RU and RD (LU and LD).
- Any horizontal line intersecting R can intersect either RU or RD but not both, but it can meet both children of RU (RD).

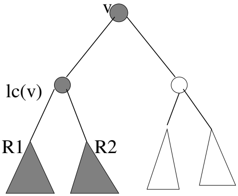
Time complexity of rectangle queries



- Therefore, the time complexity $T(n)$ for an n-vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T(\frac{n}{4}), \quad T(1) = 1$$

Time complexity of rectangle queries



- Therefore, the time complexity $T(n)$ for an n-vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T(\frac{n}{4}), \quad T(1) = 1$$

- The solution for $T(n)$ is $O(\sqrt{n})$ (an exercise for audience!!, direct substitution does not work!!).
- The total cost of reporting k points in Q is therefore $O(\sqrt{n} + k)$.

Summary:Range searching with Kd-trees

Given a set S of n points in the plane, we can construct a Kd-tree in $O(n \log n)$ time and $O(n)$ space, so that rectangle queries can be executed in $O(\sqrt{n} + k)$ time. Here, the number of points in the query rectangle is k.

Range searching with Range-trees

There is another beast called Range-trees.

Range searching with Range-trees

There is another beast called Range-trees.

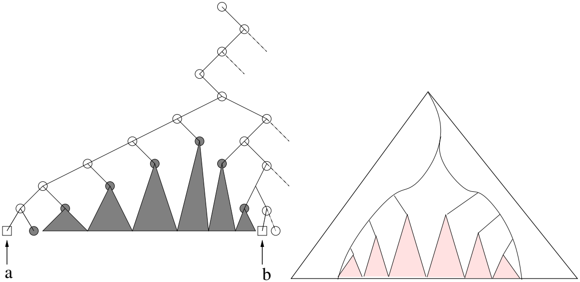
- Given a set S of n points in the plane, we can construct a range tree in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.

Range searching with Range-trees

There is another beast called Range-trees.

- Given a set S of n points in the plane, we can construct a range tree in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.
- The query time can be improved to $O(\log n + k)$ using the technique of *fractional cascading*. We won't discuss this, some deep constructions are involved.

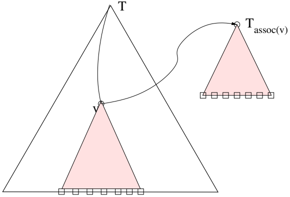
Range trees: Concepts

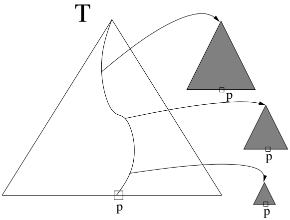


Given a 2-d rectangle query $[a, b] \times [c, d]$, we can identify subtrees whose leaf nodes are in the range $[a, b]$ along the x-direction.

Only a subset of these leaf nodes lie in the range $[c, d]$ along the y-direction.

Range trees: What are these?



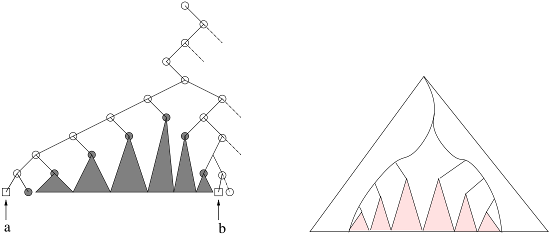


$T_{assoc(v)}$ is a binary search tree on y-coordinates for points in the leaf nodes of the subtree tooted at v in the tree T.

The point p is duplicated in $T_{assoc(v)}$ for each v on the search path for p in tree T.

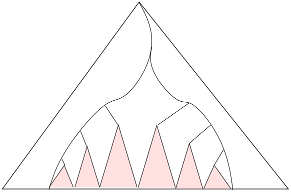
The total space requirements is therefore $O(n \log n)$.

Range trees: Method



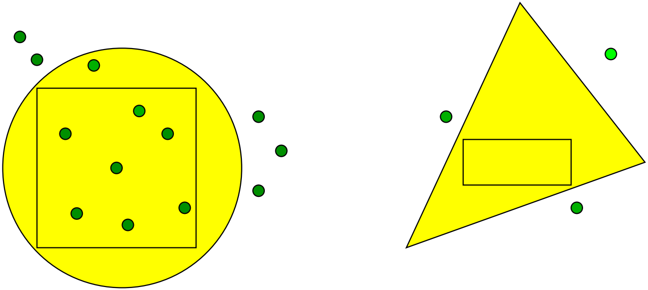
We perform 1-d range queries with the y-range $[c, d]$ in each of the subtrees adjacent to the left and right search paths for the x-range $[a, b]$ in the tree T.

Complexity of range searching using range trees



Since the search path is $O(\log n)$ in size, and each y-range query requires $O(\log n)$ time, the total cost of searching is $O(\log^2 n)$. The reporting cost is $O(k)$ where k points lie in the query rectangle.

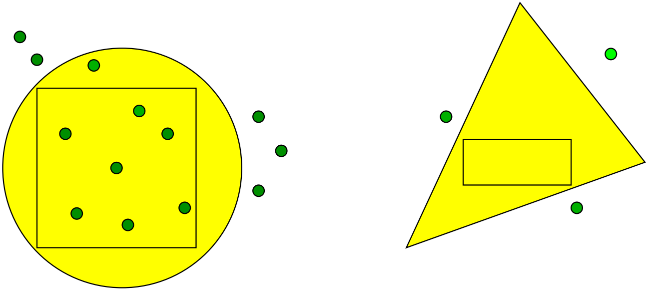
More general queries



General Queries: Points inside triangles, circles, ellipses, etc.

- Triangles can simulate other shapes with straight edges.

More general queries

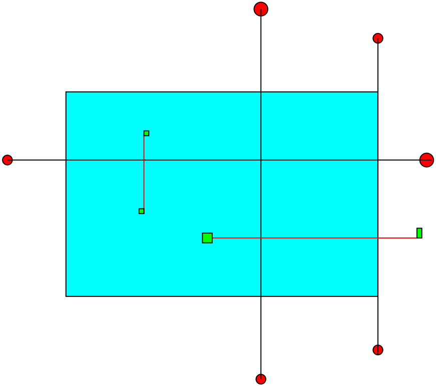


General Queries: Points inside triangles, circles, ellipses, etc.

- Triangles can simulate other shapes with straight edges.
- Circle are different, cannot be simulated by triangles! (or any other straight edge figure!!).

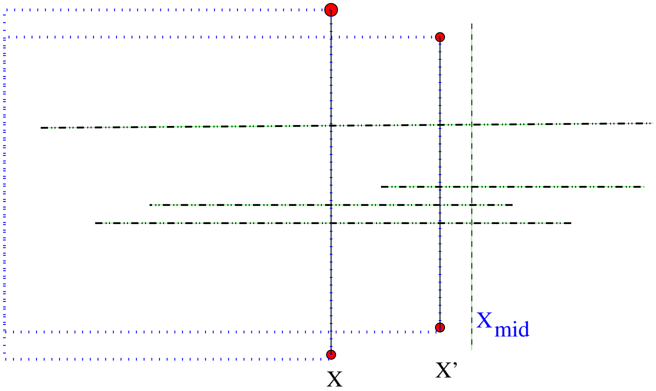
Interval Queries

Motivation: Windowing Problem



Windowing problem is not a special case of rangequery. Even for orthogonal segments.
Segments with endpoints outside the window can intersect it.

Special Treatment: Windowing Problem



We solve the special case. We take the window edges as infinite lines for segments partially inside.

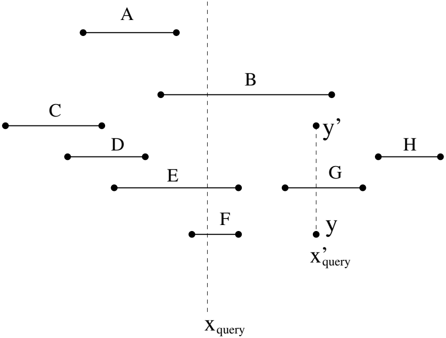
Problem concerning intervals

Problem

Given a set of intervals I and a query point x, report all intervals in I intersecting x.

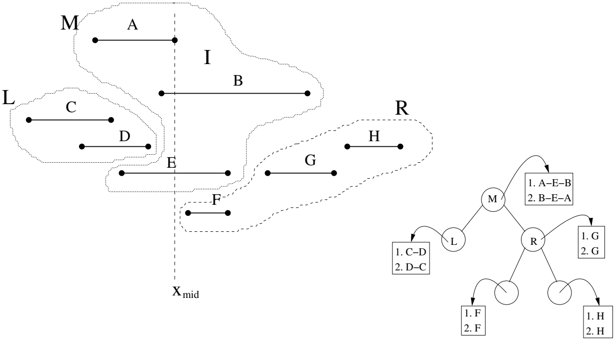
Solution: Obviously $O(n)$, $n = |I|$, algorithm will work.

Finding intervals containing a query point



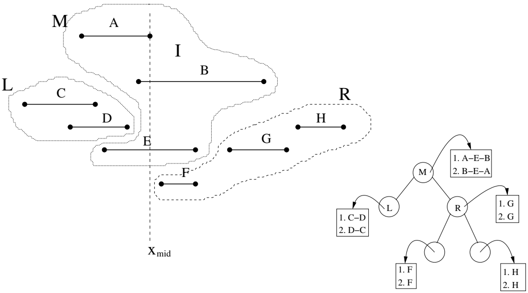
Simpler queries ask for reporting all intervals intersecting the vertical line $X = x_{query}$.
More difficult queries ask for reporting all intervals intersecting a vertical segment joining (x'_{query}, y) and (x'_{query}, y').

Interval trees: What are these?



The set M has intervals intersecting the vertical line $X = x_{mid}$, where x_{mid} is the median of the x-coordinates of the $2n$ endpoints. The root node has intervals M sorted in two independent orders (i) by right end points (B-E-A), and (ii) left end points (A-E-B).

Interval tree: Computing and Space Requirements



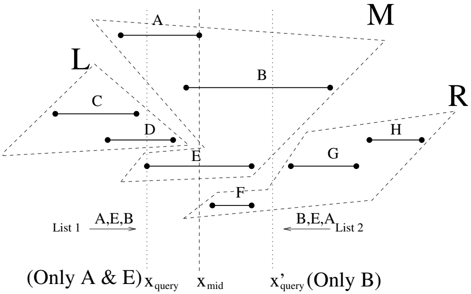
The set L and R have at most n endpoints each.

So they have at most $\frac{n}{2}$ intervals each.

Clearly, the cost of (recursively) building the interval tree is $O(n \log n)$.

The space required is linear.

Answering queries using an interval tree



For $x_{query} < x_{mid}$, we do not traverse subtree for subset R.

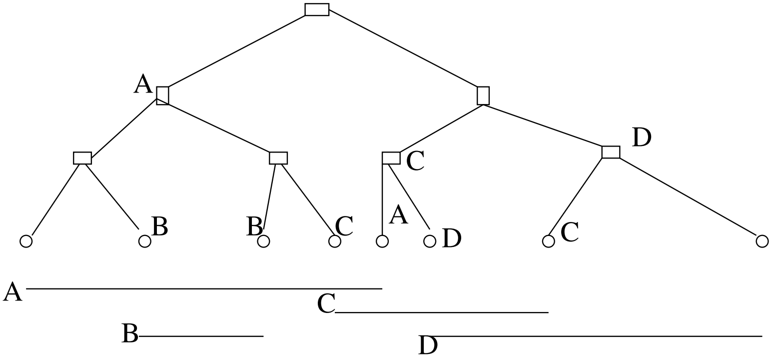
For $x'_{query} > x_{mid}$, we do not traverse subtree for subset L.

Clearly, the cost of reporting the k intervals is $O(\log n + k)$.

Another solution using segment trees

There is yet another beast called *Segment Trees!*.
Segment trees can also be used to solve the problem concerning intervals.

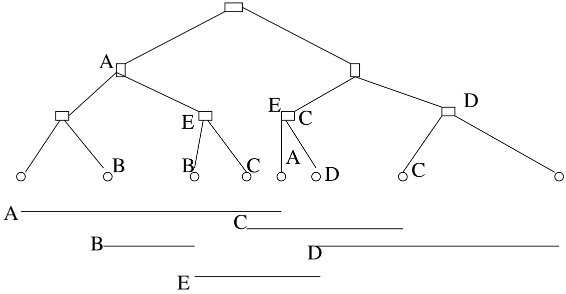
Introducing the segment tree



For an interval which spans the entire range $inv(v)$, we mark only internal node v in the segment tree, and not any descendant of v.

We never mark any ancestor of a marked node with the same label.

Representing intervals in the segment tree

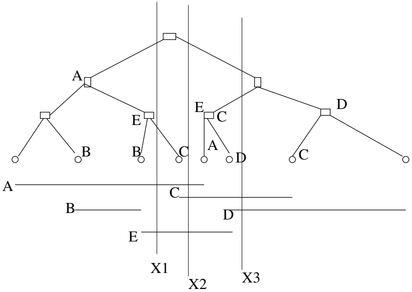


At each level, at most two internal nodes are marked for any given interval.

Along a root to leaf path an interval is stored only once.

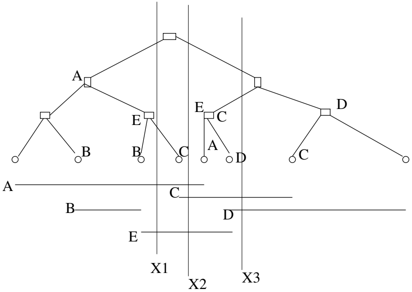
The space requirement is therefore $O(n \log n)$.

Reporting intervals containing a given query point



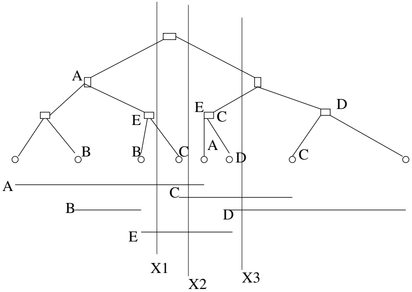
- Search the tree for the given query point.

Reporting intervals containing a given query point



- Search the tree for the given query point.
- Report against all intervals that are on the search path to the leaf.

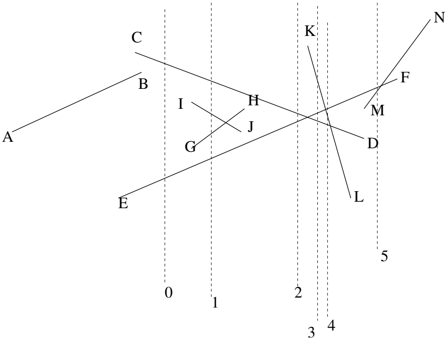
Reporting intervals containing a given query point



- Search the tree for the given query point.
- Report against all intervals that are on the search path to the leaf.
- If k intervals contain the query point then the time complexity is $O(\log n + k)$.

Line Sweep Technique

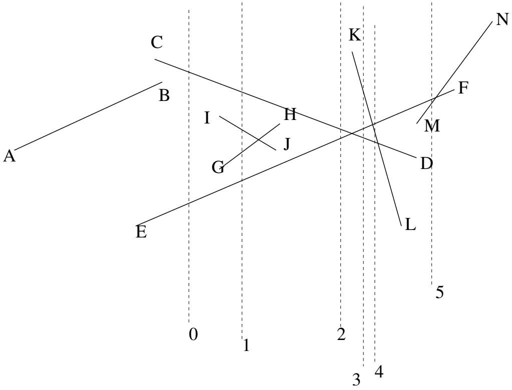
Problem to Exemplify Line Sweep



Problem

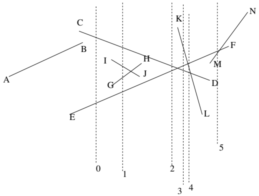
Given a set S of n line segments in the plane, report all intersections between the segments.

Reporting segments intersections



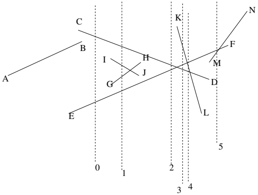
Easy but not the best solution: Check all pairs in $O(n^2)$ time.

Line Sweep: Some observations for Sweeping



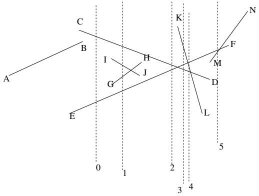
- A vertical line just before any intersection meets intersecting segments in an empty, intersection free segment, i.e. they must be consecutive.

Line Sweep: Some observations for Sweeping



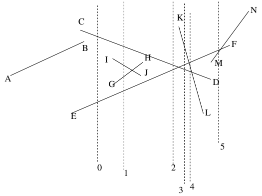
- A vertical line just before any intersection meets intersecting segments in an empty, intersection free segment, i.e. they must be consecutive.
- Detect intersections by checking consecutive pairs of segments along a vertical line.

Line Sweep: Some observations for Sweeping



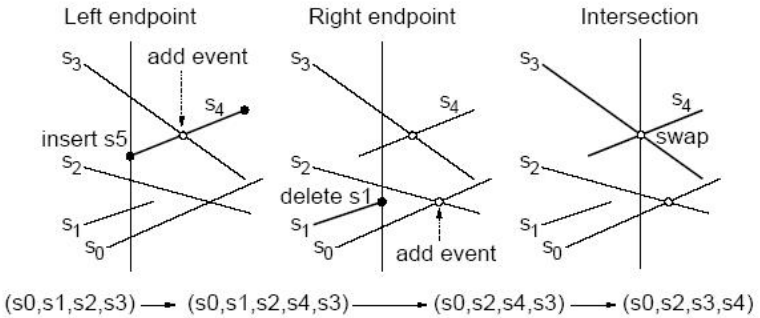
- A vertical line just before any intersection meets intersecting segments in an empty, intersection free segment, i.e. they must be consecutive.
- Detect intersections by checking consecutive pairs of segments along a vertical line.
- This way, each intersection point can be detected. How?

Line Sweep: Some observations for Sweeping



- A vertical line just before any intersection meets intersecting segments in an empty, intersection free segment, i.e. they must be consecutive.
- Detect intersections by checking consecutive pairs of segments along a vertical line.
- This way, each intersection point can be detected. How?
- We maintain the order at the sweep line, which only changes at event points.

Sweeping: Steps to be taken at each event

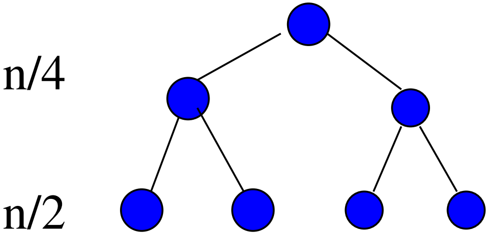


We use heap for event queue.

We use binary search trees (balanced) for segments in the sweep line.

Source (for image): http://research.engineering.wustl.edu/ pless/

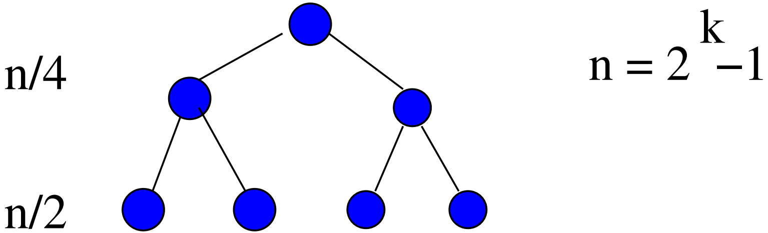
Reporting segments intersections



$$n = 2^{k} - 1$$

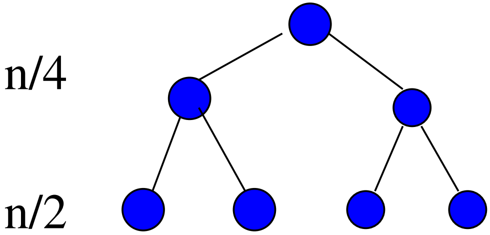
- The use of a heap does nor require apriori sorting.

Reporting segments intersections



- The use of a heap does nor require apriori sorting.
- All we do is a heap building operation in linear time level by level and bottom-up.

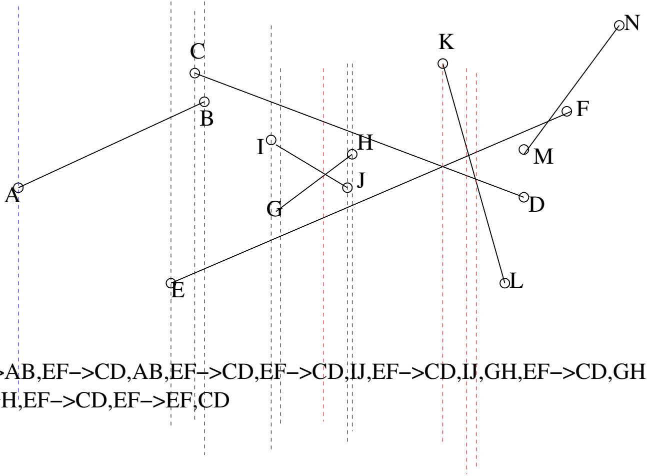
Reporting segments intersections



$$n = 2^k - 1$$

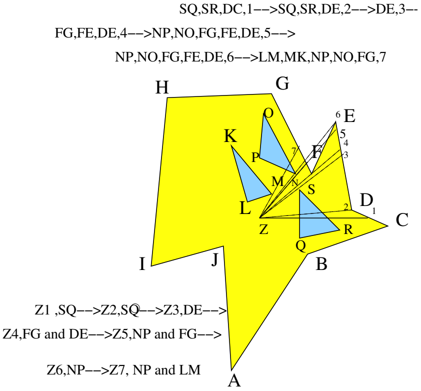
- The use of a heap does nor require apriori sorting.
- All we do is a heap building operation in linear time level by level and bottom-up.
- $1 \times \frac{n}{2} + 2 \times \frac{n}{4} + 4 \times \frac{n}{8} + \ldots$

Sweeping steps: Endpoints and intersection points



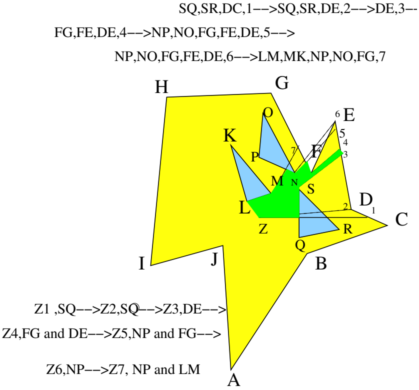
AB–>AB,EF–>CD,AB,EF–>CD,EF–>CD,IJ,EF–>CD,IJ,GH,EF–>CD,GH,IJ,EF
CD,GH,EF–>CD,EF–>EF,CD

Problem for Visibility: angular sweep



The problem is to determine edges visible from an interior point [6]. We can use a similar angular sweep method.

Visibility polygon computation



SQ,SR,DC,1−−>SQ,SR,DE,2−−>DE,3−−

FG,FE,DE,4−−>NP,NO,FG,FE,DE,5−−>

NP,NO,FG,FE,DE,6−−>LM,MK,NP,NO,FG,7

Z1 ,SQ−−>Z2,SQ−−>Z3,DE−−>

Z4,FG and DE−−>Z5,NP and FG−−>

Z6,NP−−>Z7, NP and LM

Final computed visibility region.

Conclusion

Open Problems and Generalisations

- Generalizations of each of the problems in space and higher dimensions
- Counting points/objects inside different type of objects such as triangles, circles, ellipses, or, tetrahedrons, simplexes, ellipsoids in higher space and higher dimensions
- Good data structures for computing and storing visibility information in 3D

Summary

- We studied the concept of Kd-trees and Range trees.

Summary

- We studied the concept of Kd-trees and Range trees.
- Next we saw how we can answer queries about intervals using interval trees and segment trees.

Summary

- We studied the concept of Kd-trees and Range trees.
- Next we saw how we can answer queries about intervals using interval trees and segment trees.
- We looked into line-sweep technique.

Summary

- We studied the concept of Kd-trees and Range trees.
- Next we saw how we can answer queries about intervals using interval trees and segment trees.
- We looked into line-sweep technique.
- Now we are for the final concluding remarks.

You may read

- The classic book by Preparata and Shamos [7], and
- The introductory textbook by Marc de Berg et al. [2],
- The book on algorithms by Cormen et al. [1] contains some basic geometric problems and algorithms,
- For point location Edelsbrunner [3], though a little hard to understand, is good,
- And lots of lots of web resources.

Acknowledgments

Thanks to Dr Sudep P Pal, IIT Kharagpur, for providing most of the material and pictures of the presentation.

References I

Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson.
Introduction to Algorithms.
McGraw-Hill Higher Education, 2001.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf.
Computational Geometry: Algorithms and Applications.
Springer-Verlag, second edition, 2000.

Herbert Edelsbrunner.
Algorithms in Combinatorial Geometry.
Springer-Verlag, New York, 1987.

References II

- H. Fuchs, Z. M. Kedem, and B. Naylor.
  Predetermining visibility priority in 3-D scenes (preliminary report).
  13(3):175–181, August 1979.
- H. Fuchs, Z. M. Kedem, and B. F. Naylor.
  On visible surface generation by a priori tree structures.
  14(3):124–133, July 1980.
- Subir K Ghosh.
  Visibility Algorithms in the Plane.
  Cambridge University Press, Cambridge, UK, 2007.
- F. P. Preparata and M. I. Shamos.
  Computational Geometry: An Introduction.
  Springer-Verlag, 1985.

At Last . . .

Thank You

`shreesh@rkmvu.ac.in`
`sarvottamananda@gmail.com`