Geometric data structures

Sudebkumar Prasant Pal
Department of Computer Science and Engineering
IIT Kharagpur, 721302.
email: spp@cse.iitkgp.ernet.in

March 14-16, 2012 - DAIICT Gandhinagar
Introduction to Graph and Geometric Algorithms

Scope of the lecture

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

Scope of the lecture

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

Scope of the lecture

- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.
- **Interval trees**
  Interval trees for reporting all (horizontal) intervals containing a given (vertical) query line or segment.

Scope of the lecture

- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.
- **Interval trees**
  Interval trees for reporting all (horizontal) intervals containing a given (vertical) query line or segment.
- **Segment trees**
  For reporting (portions of) all segments inside a query window.

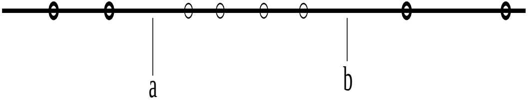
Scope of the lecture

- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.
- **Interval trees**
  Interval trees for reporting all (horizontal) intervals containing a given (vertical) query line or segment.
- **Segment trees**
  For reporting (portions of) all segments inside a query window.
- **Planar point location**
  Using triangulation refinement and monotone subdivisions.

Scope of the lecture

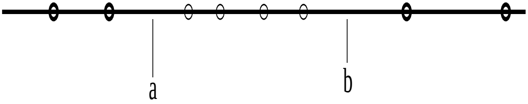
- **Binary search trees and 2-d range trees**
  We consider 1-d and 2-d range queries for point sets.
- **Range searching using Kd-trees**
  2-d orthogonal range searching with Kd-trees.
- **Interval trees**
  Interval trees for reporting all (horizontal) intervals containing a given (vertical) query line or segment.
- **Segment trees**
  For reporting (portions of) all segments inside a query window.
- **Planar point location**
  Using triangulation refinement and monotone subdivisions.
- **Hierarchical representation of a convex polygon**
  Detecting the intersection of a convex polygon with a query line..

1-dimensional Range searching



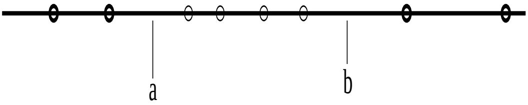
- Problem: Given a set P of n points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.

1-dimensional Range searching



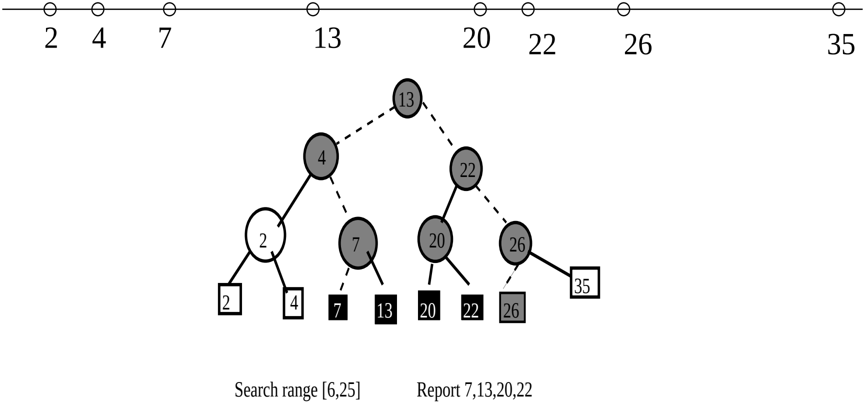
- Problem: Given a set P of n points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.
- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where k is the number of points of P in $[a, b]$.

1-dimensional Range searching



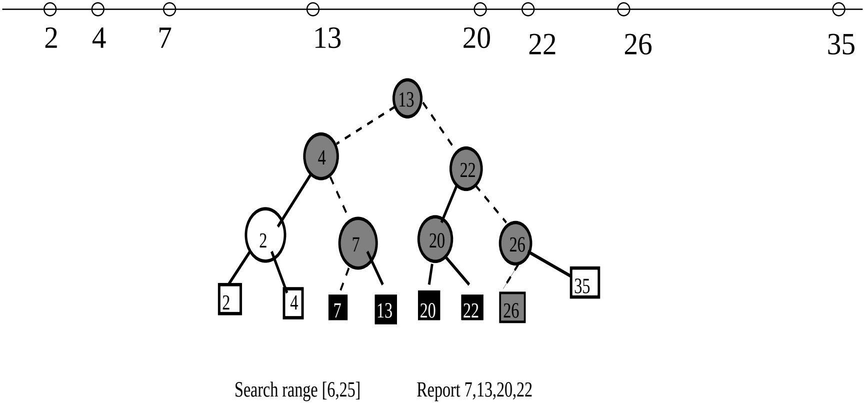
- Problem: Given a set P of n points $\{p_1, p_2, \cdots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.
- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where k is the number of points of P in $[a, b]$.
- However, when we permit insertion or deletion of points, we cannot use an array answering queries so efficiently.

1-dimensional Range searching



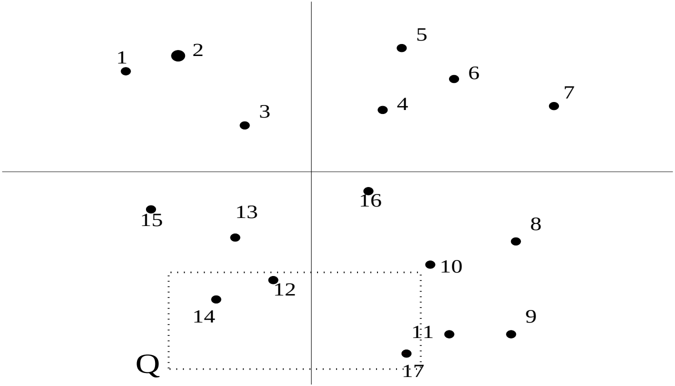
- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.

1-dimensional Range searching



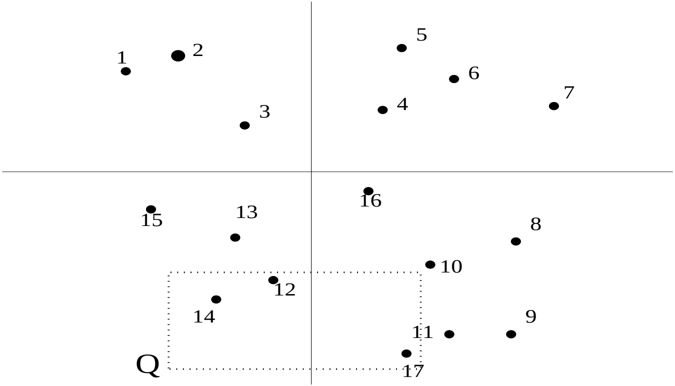
- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.
- Each internal node stores the x-coordinate of the rightmost point in its left subtree for guiding search.

2-dimensional Range Searching



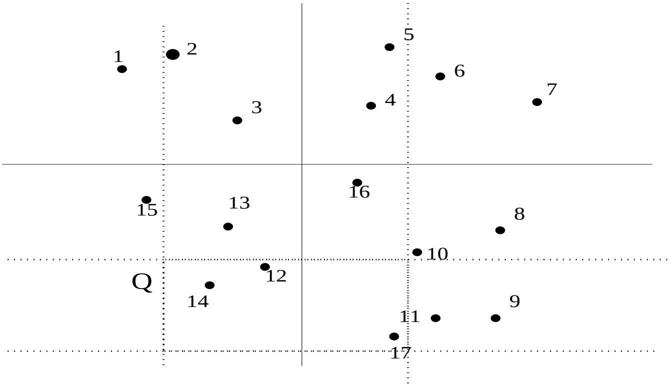
- Problem: Given a set P of n points in the plane, report points inside a query rectangle Q whose sides are parallel to the axes.

2-dimensional Range Searching



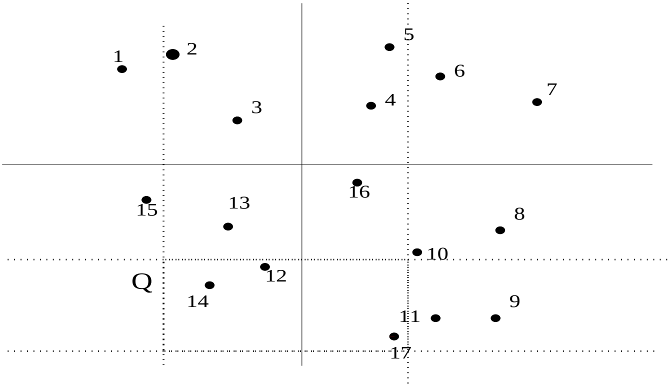
- Problem: Given a set P of n points in the plane, report points inside a query rectangle Q whose sides are parallel to the axes.
- Here, the points inside R are 14, 12 and 17.

2-dimensional Range Searching



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

2-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.

Range searching with range trees and Kd-trees

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

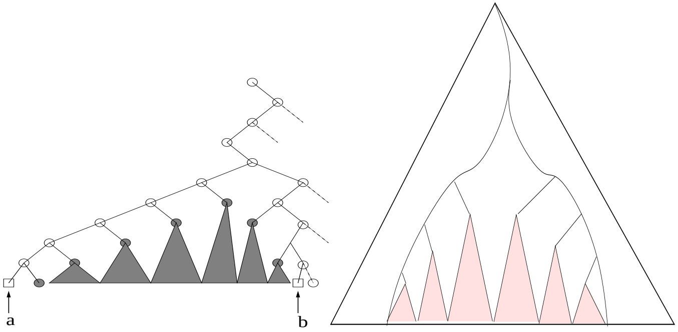
Range searching with range trees and Kd-trees

- Given a set S of n points in the plane, we can construct a *2d-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*.

Range searching with range trees and Kd-trees

- Given a set S of n points in the plane, we can construct a *2d-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*.
- 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.

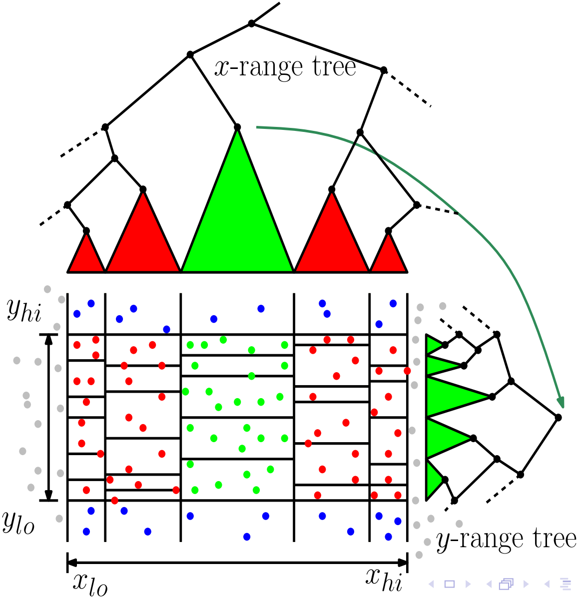
Range searching in the plane using range trees



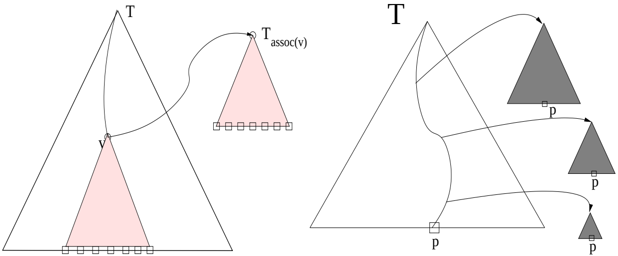
Given a 2-d rectangle query $[a, b]X[c, d]$, we can identify subtrees whose leaf nodes are in the range $[a, b]$ along the X-direction.

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

Range searching in the plane using range trees



Range searching in the plane using range trees

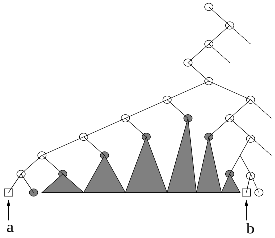


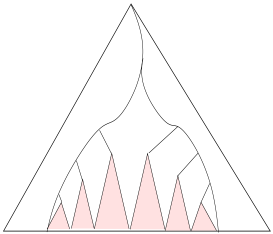
$T_{assoc(v)}$ is a binary search tree on y-coordinates for points in the leaf nodes of the subtree tooted at v in the tree T.

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

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

Range searching in the plane using range trees

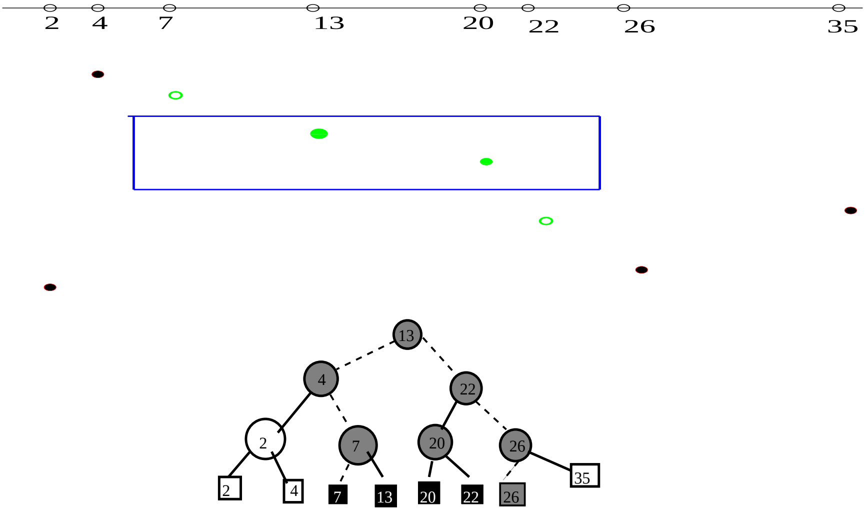




We perform 1-d range queries with the y-range $[c, d]$ in each of the subtrees adjacent to the left and right search paths within the x-range $[a, b]$ in the tree T.

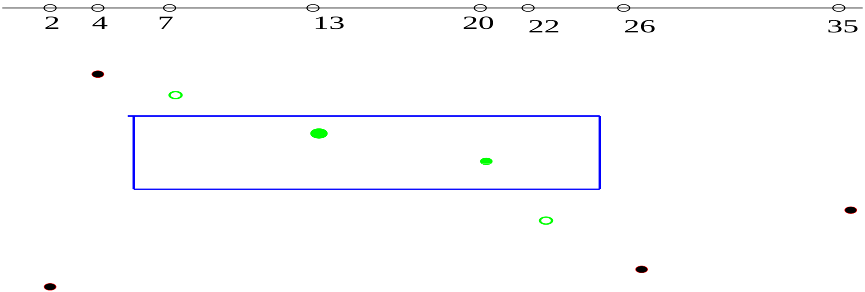
Since the search path is $O(\log n)$ in size, and each y-range query requires $O(\log n)$ time, the total cost of searching is $O(\log^2 n)$. The reporting cost is $O(k)$ where k points lie in the query rectangle.

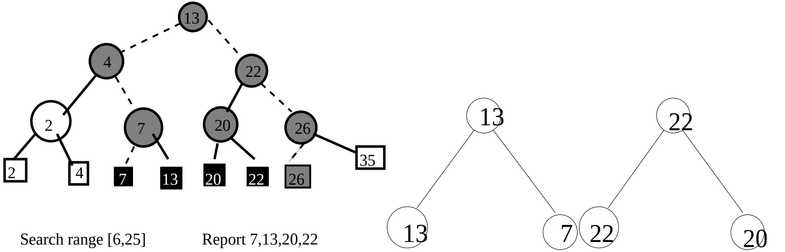
2-RANGE TREE SEARCHING



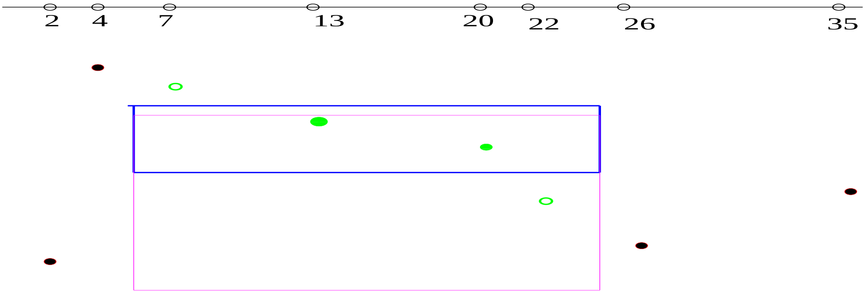
Search range [6,25]   Report 7,13,20,22

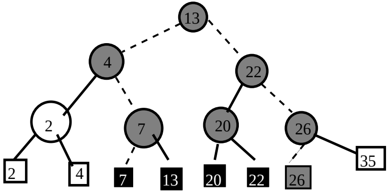
2-RANGE TREE SEARCHING





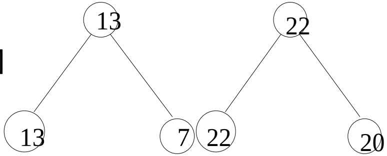
2-RANGE TREE SEARCHING



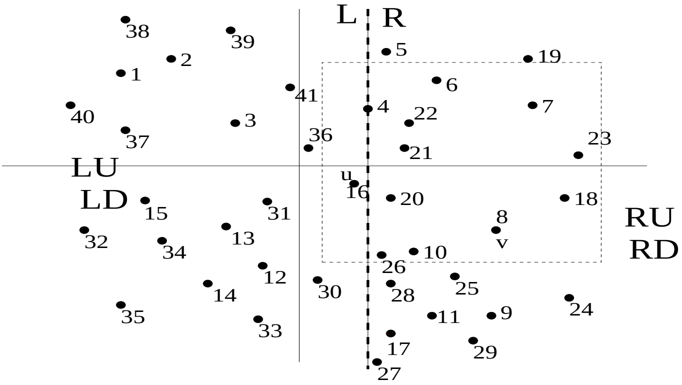


Search range [6,25]

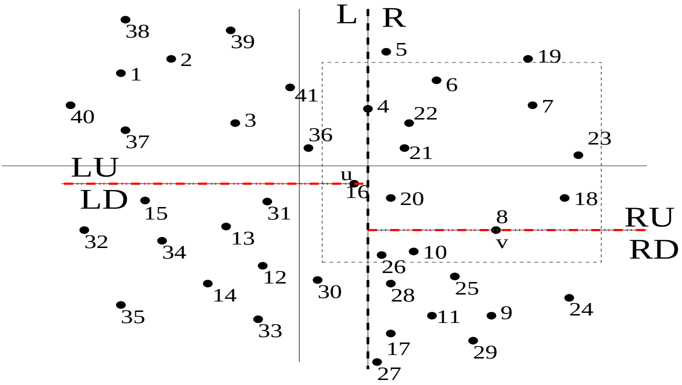
Report 7,13,20,22



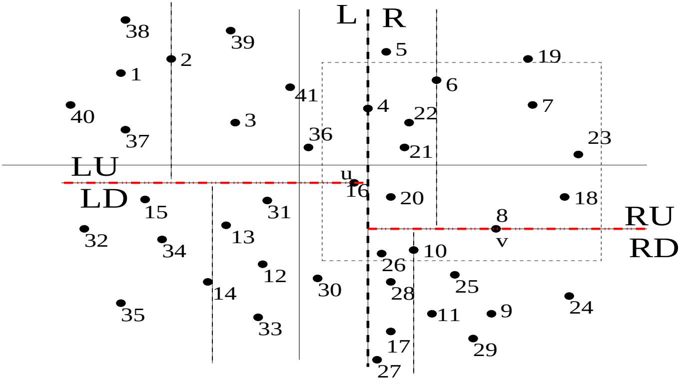
Partition by the median of x-coordinates



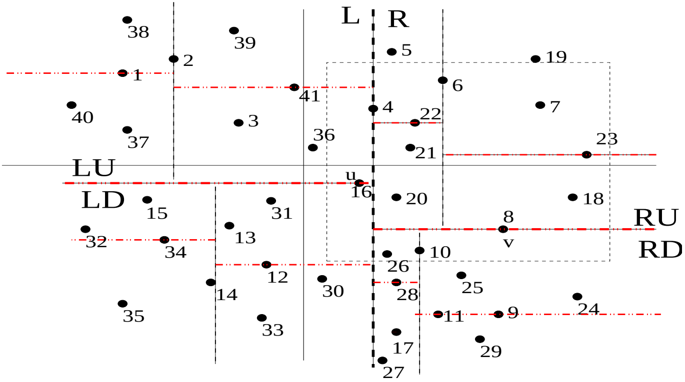
Partition by the median of y-coordinates



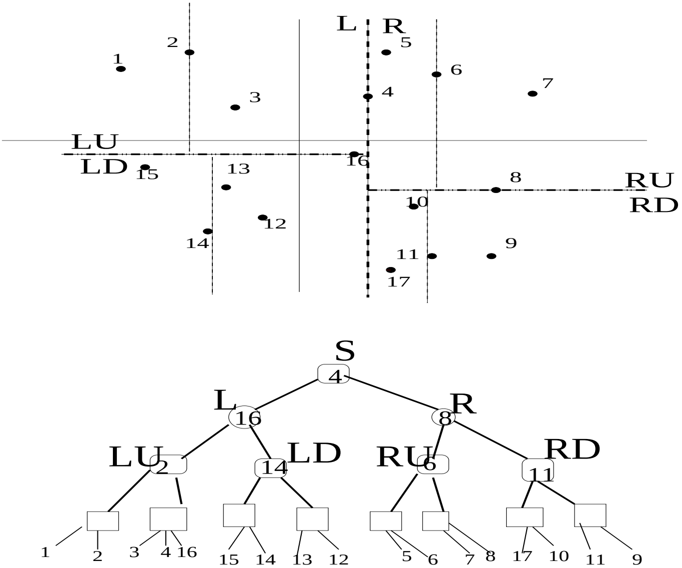
Partition by the median of x-coordinates



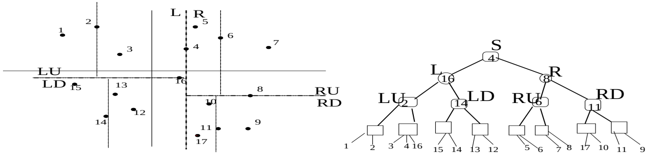
Partition by the median of y-coordinates



2-dimensional range searching using Kd-trees

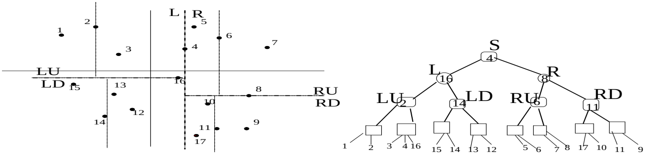


Description of the Kd-tree



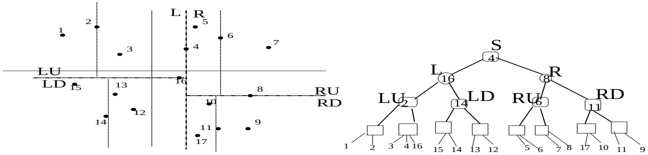
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x- and y- coordinates, respectively as follows.

Description of the Kd-tree



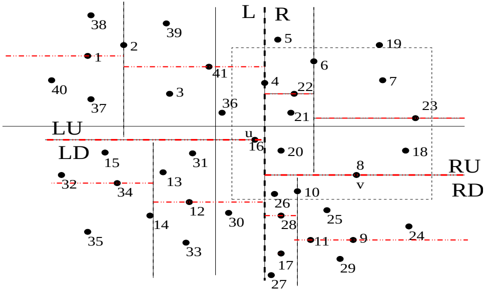
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x- and y- coordinates, respectively as follows.
- The point r stored in the root vertex T splits the set S into two roughly equal sized sets L and R using the median x-cooordinate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

Description of the Kd-tree



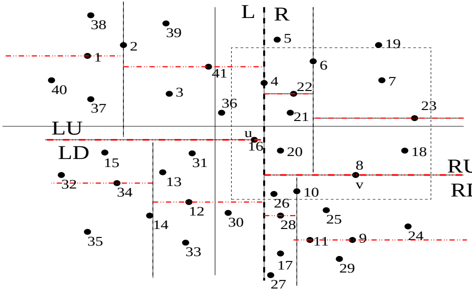
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x- and y- coordinates, respectively as follows.
- The point r stored in the root vertex T splits the set S into two roughly equal sized sets L and R using the median x-cooordinate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

Answering rectangle queries



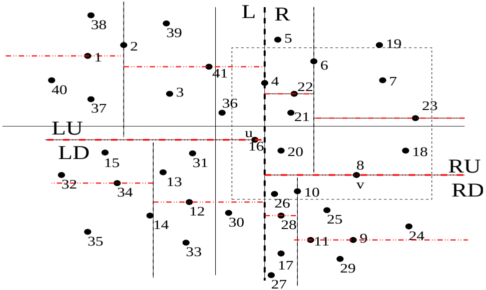
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.

Answering rectangle queries



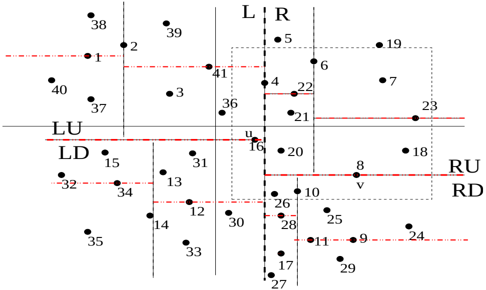
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If R contains the entire bounded $region(p)$ of a point p defining a node N of T then report all points in $region(p)$.

Answering rectangle queries



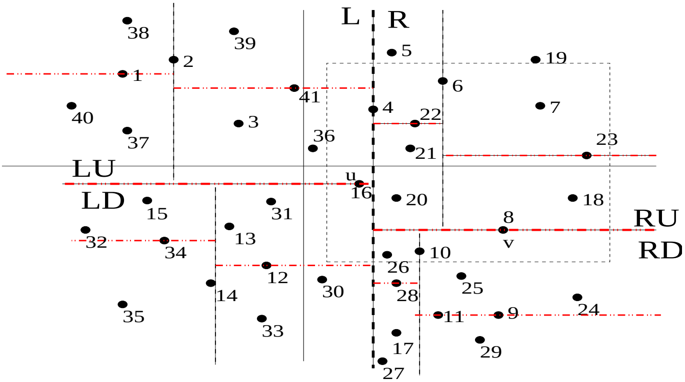
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If R contains the entire bounded $region(p)$ of a point p defining a node N of T then report all points in $region(p)$.
- If R misses the $region(p)$ then we do not treverse the subtree rooted at this node.

Answering rectangle queries



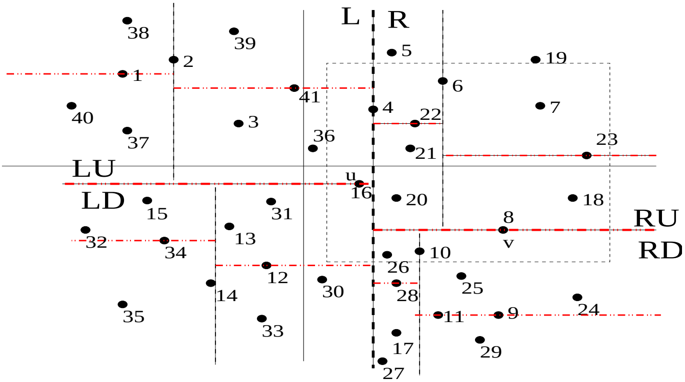
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If R contains the entire bounded $region(p)$ of a point p defining a node N of T then report all points in $region(p)$.
- If R misses the $region(p)$ then we do not treverse the subtree rooted at this node.
- If R overlaps $region(p)$ then we check whether R also overlaps the two regions of the children of the node N.

2-dimensional Range Searching: Kd-trees



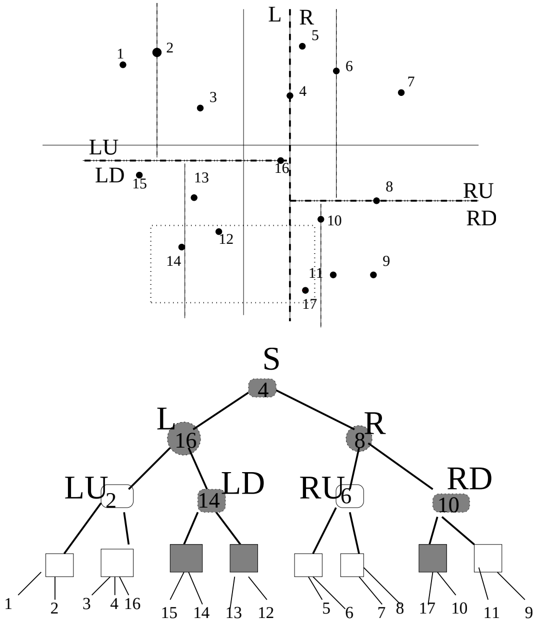
- The set L (R) is split into two roughly equal sized subsets LU and LD (RU and RD), using point u (v) that has the median y-coordinate in the set L (R), and including u in LU (RU).

2-dimensional Range Searching: Kd-trees

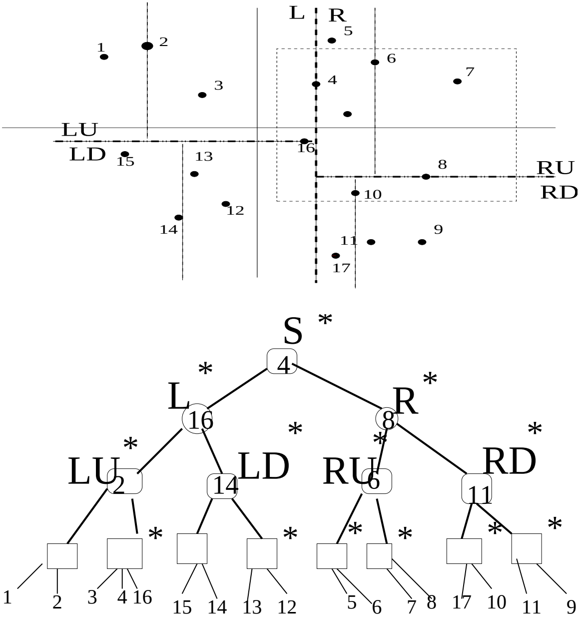


- The set L (R) is split into two roughly equal sized subsets LU and LD (RU and RD), using point u (v) that has the median y-coordinate in the set L (R), and including u in LU (RU).
- The entire halfplane containing set L (R) is called the $region(u)$ ($region(v)$).

Nodes traversed in the Kd-tree



Nodes traversed in the Kd-tree



Time complexity of output point reporting

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

Time complexity of output point reporting

- Reporting points within R 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 R 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 R 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 R 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 R 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 R.

Worst-case cost of traversal

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

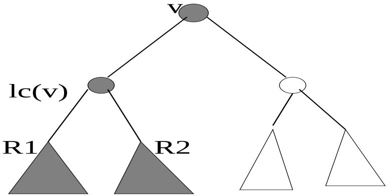
Worst-case cost of traversal

- 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).

Worst-case cost of traversal

- 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 for Kd-trees

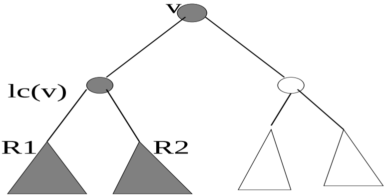


- Therefore, the time complexity $T(n)$ for an n-vertex Kd-tree obeys the recurrence relation

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

$$T(1) = 1$$

Time complexity of rectangle queries for Kd-trees



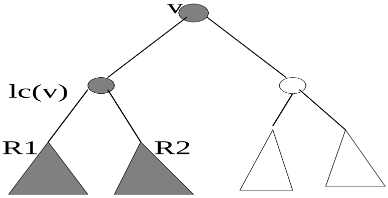
- Therefore, the time complexity $T(n)$ for an n-vertex Kd-tree obeys the recurrence relation

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

$$T(1) = 1$$

- The solution for $T(n) = O(\sqrt(n))$.

Time complexity of rectangle queries for Kd-trees



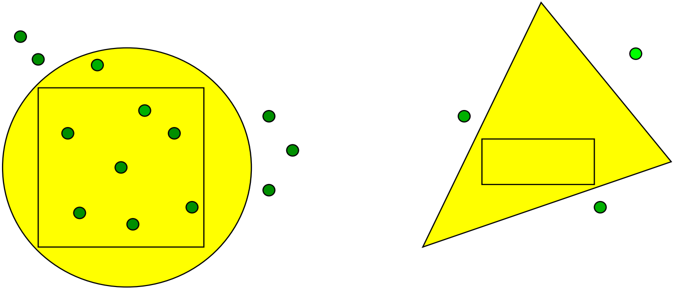
- Therefore, the time complexity $T(n)$ for an n-vertex Kd-tree obeys the recurrence relation

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

$$T(1) = 1$$

- The solution for $T(n) = O(\sqrt(n))$.
- The total cost of reporting k points in R is therefore $O(\sqrt(n) + k)$.

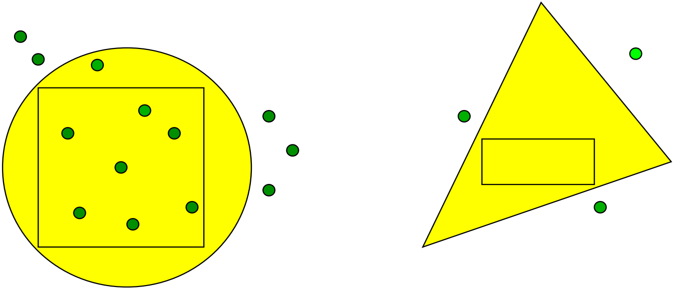
More general queries



General Queries:

- Triangles can be used to simulate polygonal shapes with straight edges.

More general queries



General Queries:

- Triangles can be used to simulate polygonal shapes with straight edges.
- Circles cannot be simulated by triangles either.

Triangle queries

- Using $O(n^2)$ space and time for preprocessing, triangle queries can be reported in $O(\log^2 n + k))$ time, where k is the number of points inside the query triangle.

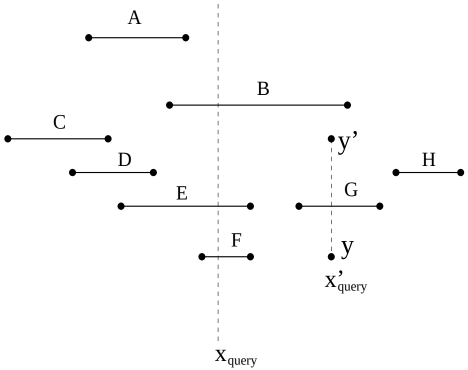
Goswami, Das and Nandy: Comput. Geom. Th. and Appl. 29 (2004) pp. 163-175.

Triangle queries

- Using $O(n^2)$ space and time for preprocessing, triangle queries can be reported in $O(\log^2 n + k))$ time, where k is the number of points inside the query triangle.
- For counting the number k of points inside a query triangle, worst-case optimal $O(\log n)$ time suffices.

Goswami, Das and Nandy: Comput. Geom. Th. and Appl. 29 (2004) pp. 163-175.

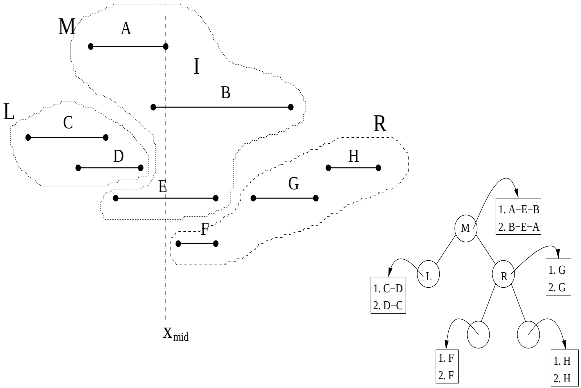
Finding intervals containing a vertical query line/segment



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').

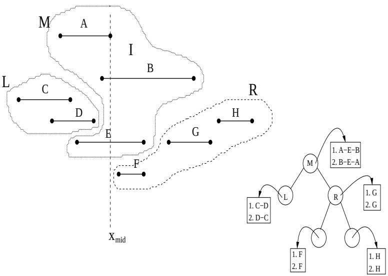
Constructing the interval tree



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).

Answering queries using an interval tree



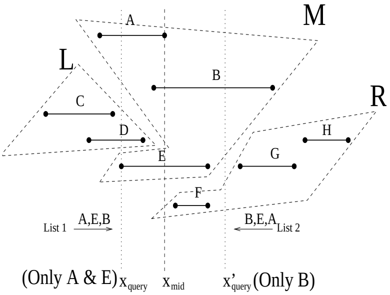
The set L and R have at most n endpoints each.

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

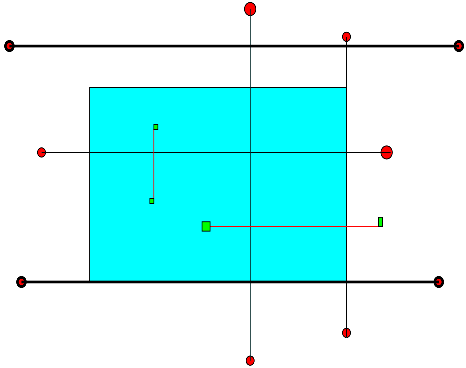


For $x_{query} < x_{mid}$, we do not traverse subtree for subset R.

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

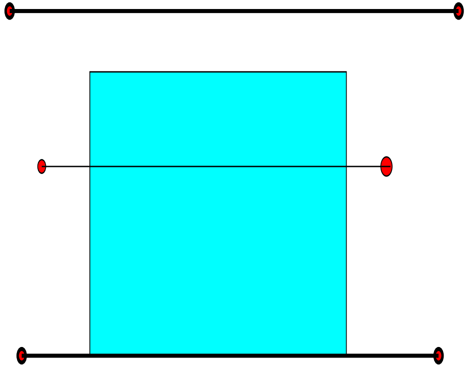
Clearly, the cost of reporting the k intervals is $O(\log n + k)$.

Reporting (portions of) all rectilinear segments inside a query rectangle



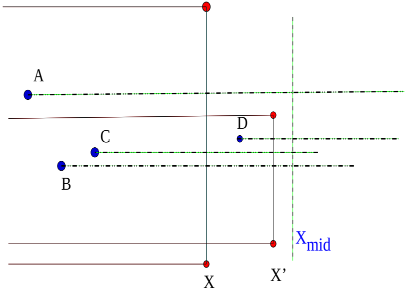
For detecting segments with one (or both) ends inside the rectangle, it is sufficient to maintain rectangular range query apparatus for output-sensitive query processing.

Reporting segments with no endpoints inside the query rectangle

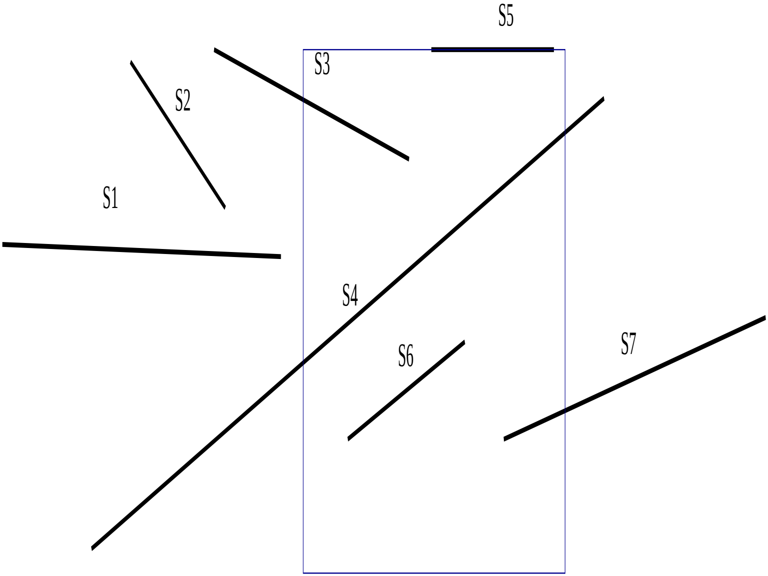


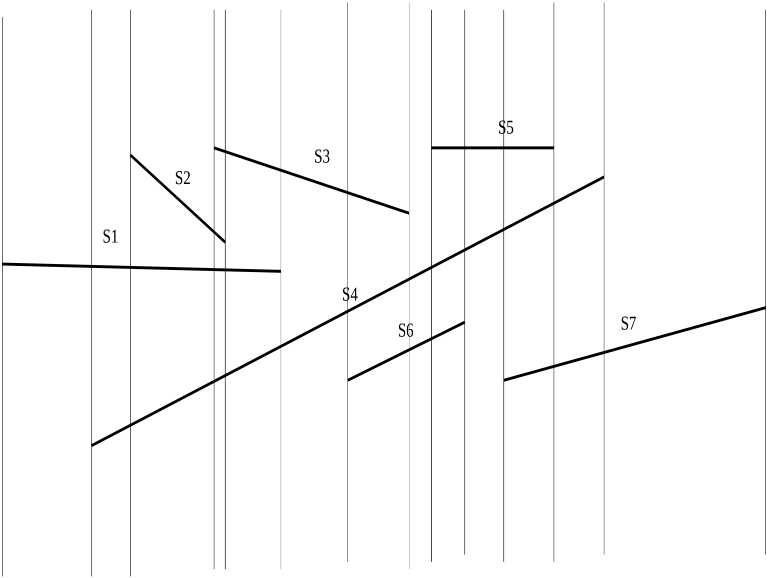
Report all (horizontal) segments that cut across the query rectangle or include an entire (top/bottom) bounding edge.
Use either the right (or left) edge, and the top (or bottom) edge of the query rectangle.

Right edges X and X' of two query rectangles

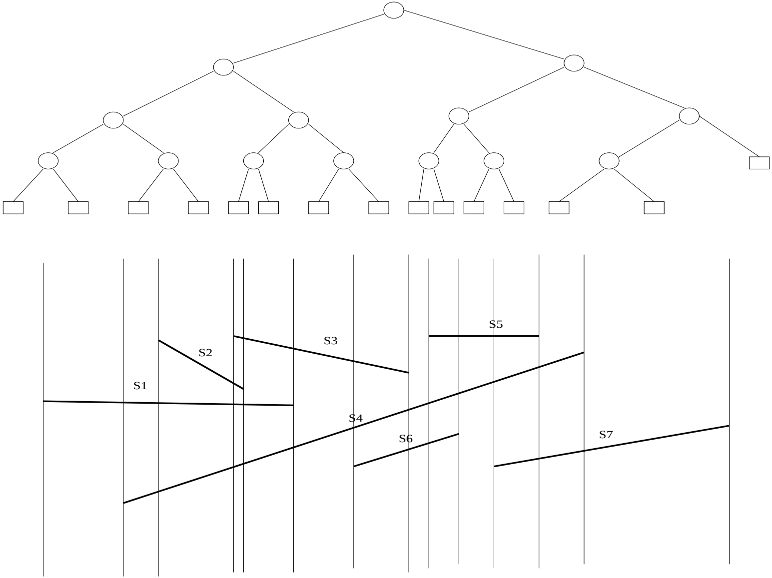


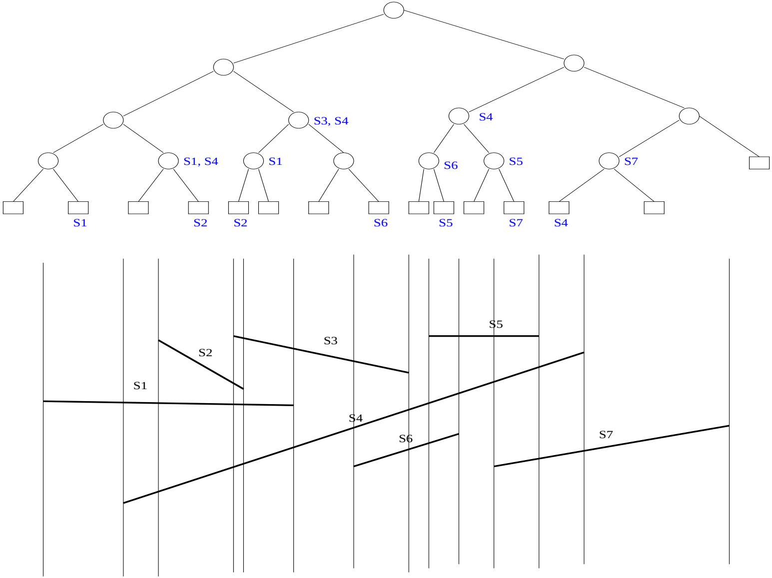
Use an interval tree of all the horizontal segments and the right bounding edge of the query rectangle like X or X'.
This helps reporting all segments cutting the right edge of the query rectangle.
Use the rectangle query for vertical segment X and find points A, B and C in the rectangle with left edge at minus infinity. For X', report B, C and D, similarly.


S5
S3
S2
S1
S4
S6
S7

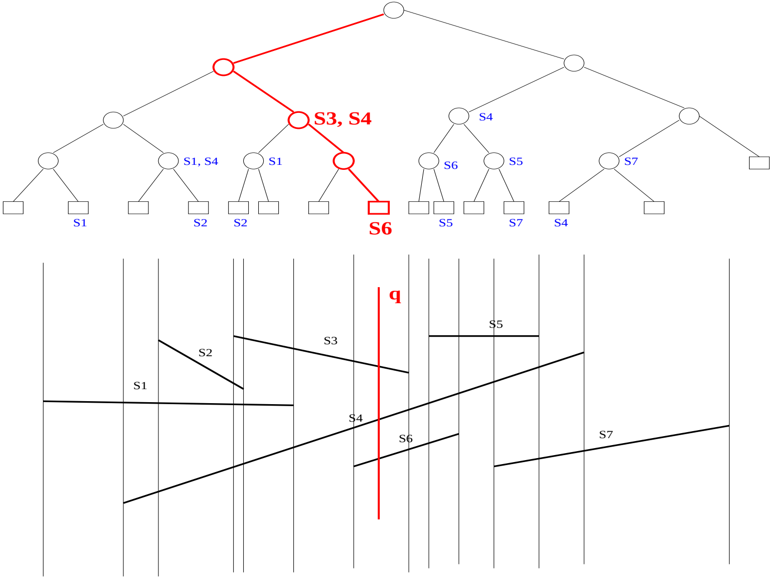


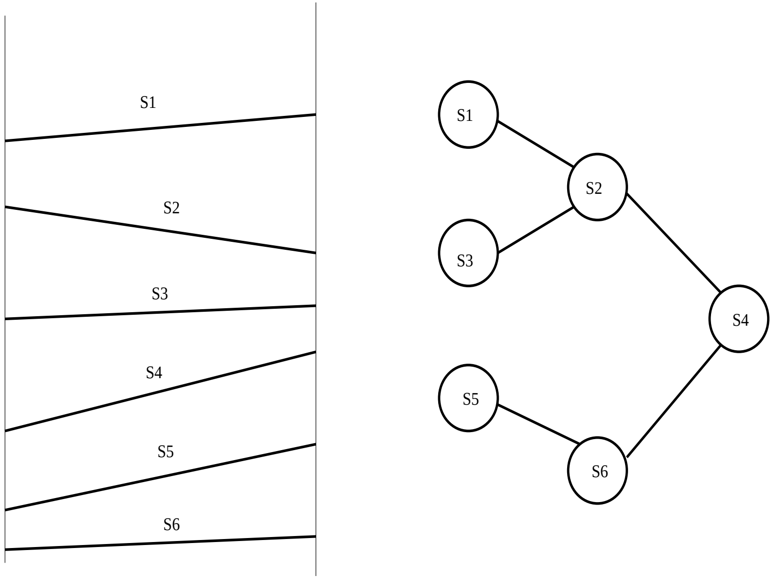
S5
S3
S2
S1
S4
S6
S7


S1
S2
S3
S4
S5
S6
S7



S3, S4
S4
S1, S4
S1
S6
S5
S7
S1
S2
S2
S6
S5
S7
S4
S5
S3
S2
S1
S4
S6
S7


S3, S4
S4
S1, S4
S1
S6
S5
S7
S1
S2
S2
S6
S5
S7
S4
q
S5
S3
S2
S1
S4
S6
S7

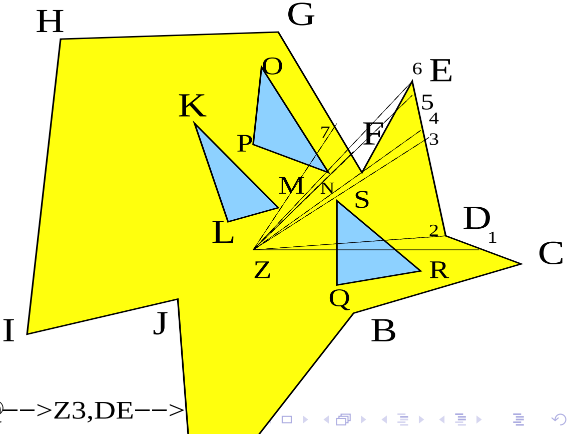

S1
S2
S3
S4
S5
S6
S1
S2
S3
S4
S5
S6

Computing the visible region in a polygon with opaque obstacles

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



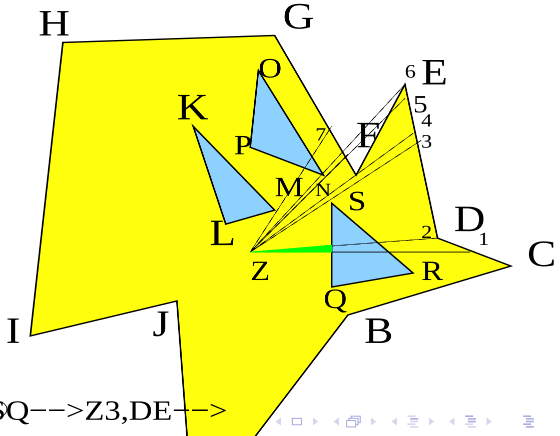
Z1 ,SQ−−>Z2,SQ−−>Z3,DE−−>

Computing the visible region in a polygon with opaque obstacles

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



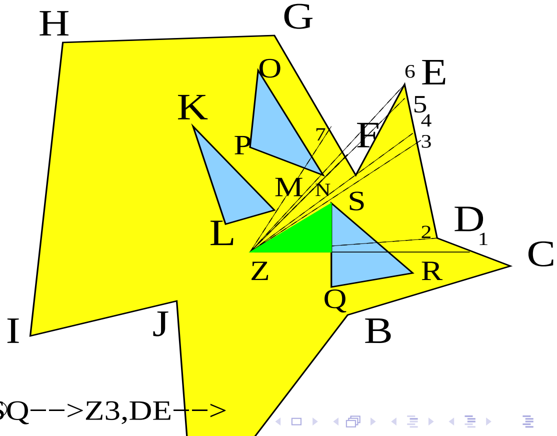
Z1 ,SQ-->Z2,SQ-->Z3,DE-->

Computing the visible region in a polygon with opaque obstacles

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



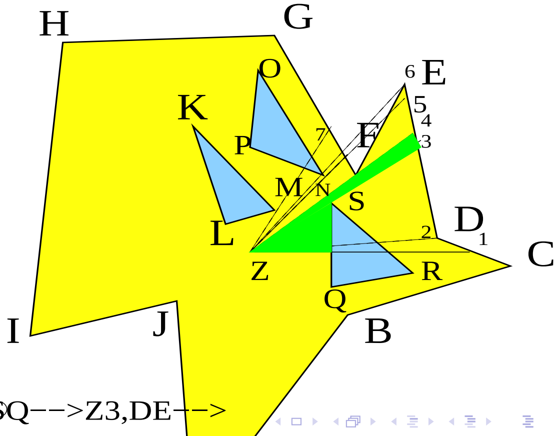
Z1 ,SQ−−>Z2,SQ−−>Z3,DE−−>

Computing the visible region in a polygon with opaque obstacles

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



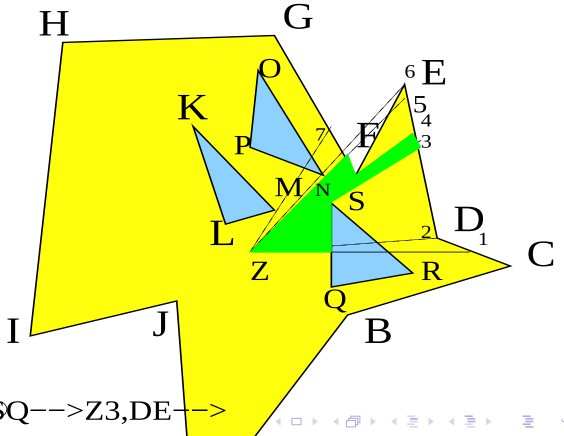
Z1 ,SQ−−>Z2,SQ−−>Z3,DE−−>

Computing the visible region in a polygon with opaque obstacles

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



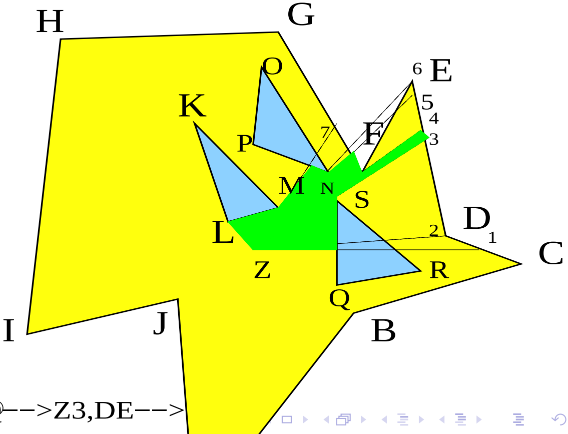
Z1 ,SQ−−>Z2,SQ−−>Z3,DE−−>

Computing the visible region in a polygon with opaque obstacles

SQ,SR,DC,1−−>SQ,SR,DE,2−−>DE,3−-

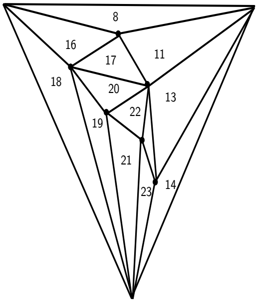
FG,FE,DE,4−−>NP,NO,FG,FE,DE,5−−>

NP,NO,FG,FE,DE,6−−>LM,MK,NP,NO,FG,7

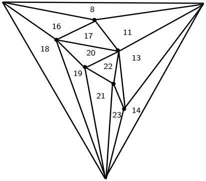


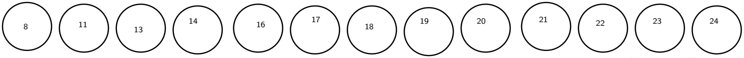
Z1 ,SQ−−>Z2,SQ−−>Z3,DE−−>

Planar point location by triangulation refinement

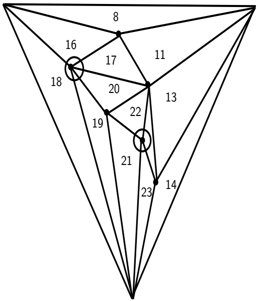


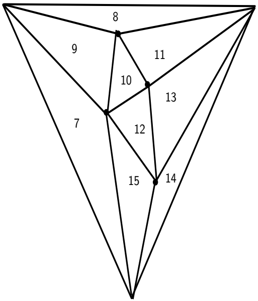
Planar point location by triangulation refinement



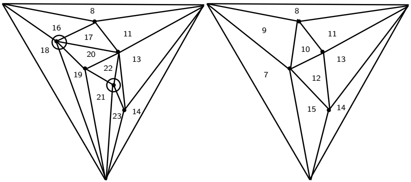


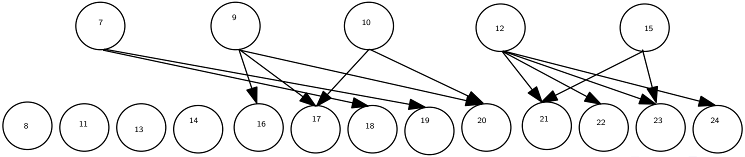
Planar point location by triangulation refinement

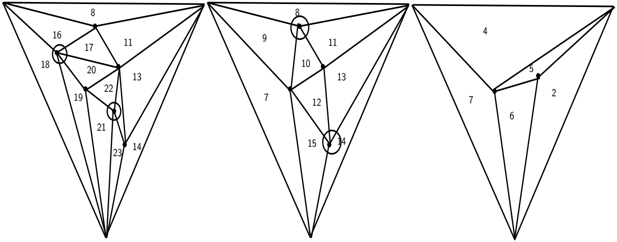


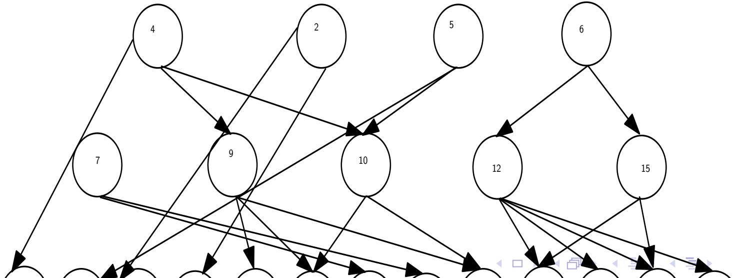


Planar point location by triangulation refinement

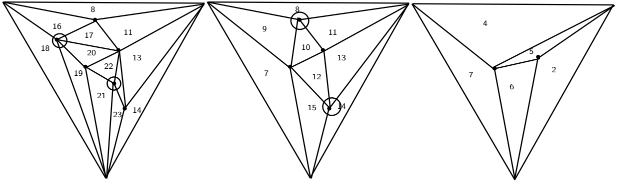


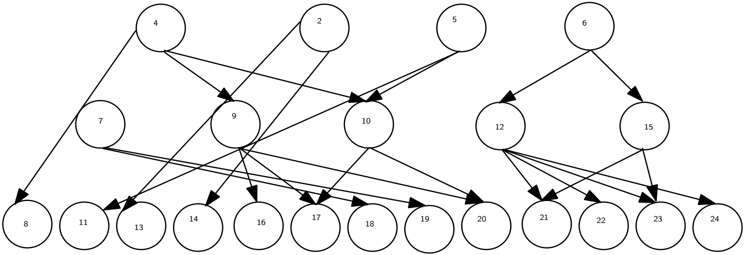



8
16
11
17
18
20
13
22
19
21
23
14
8
9
11
10
13
7
12
15
14
4
5
7
2
6

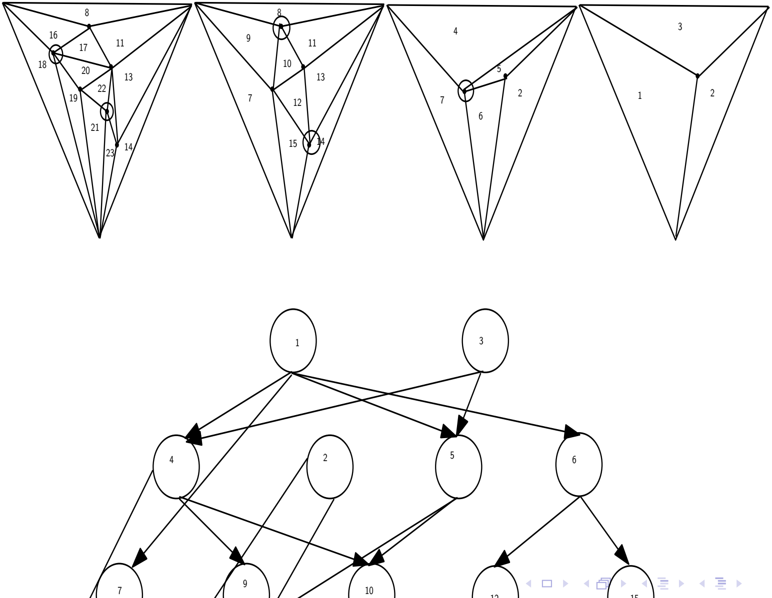

4
2
5
6
7
9
10
12
15

Planar point location by triangulation refinement

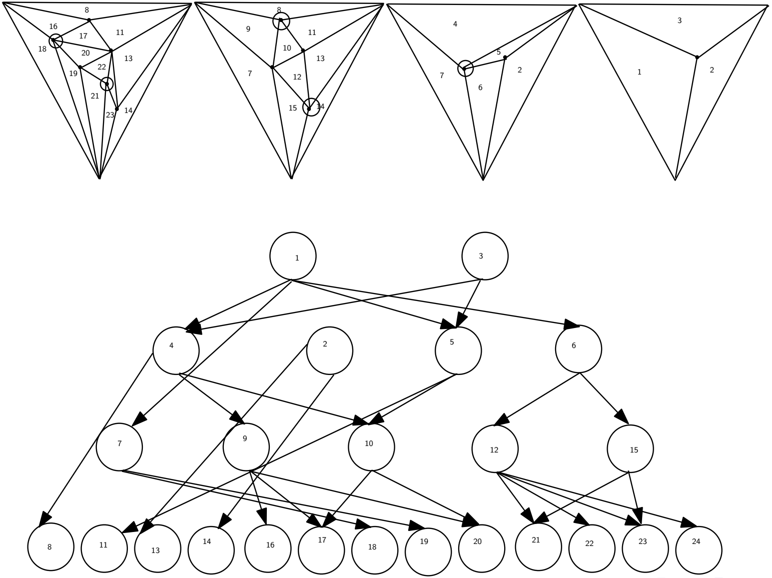




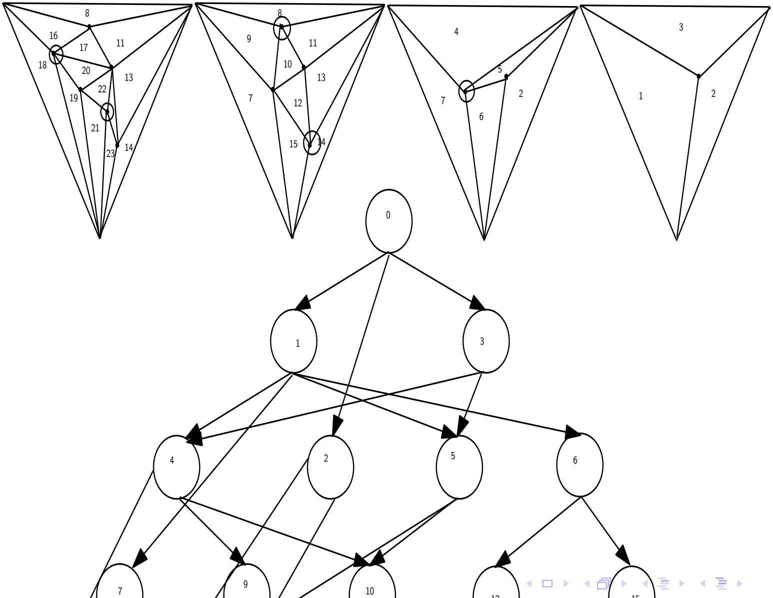
Planar point location by triangulation refinement



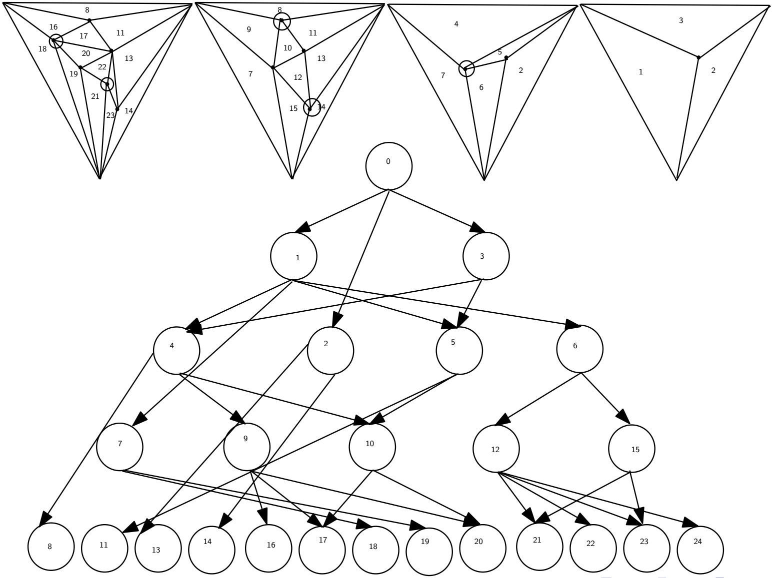
Planar point location by triangulation refinement



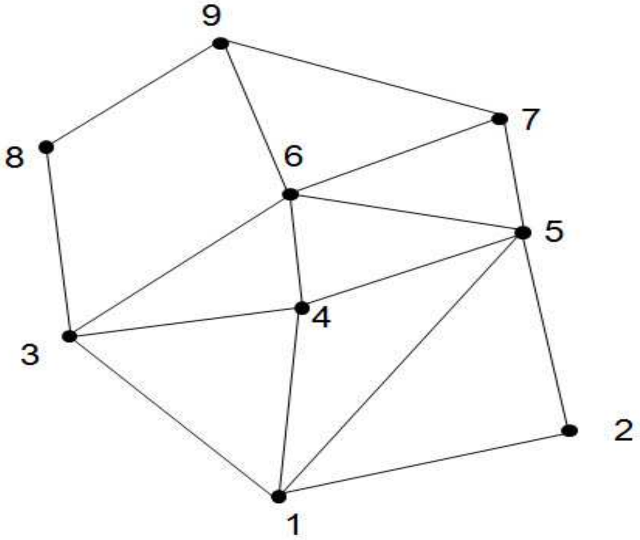
Planar point location by triangulation refinement



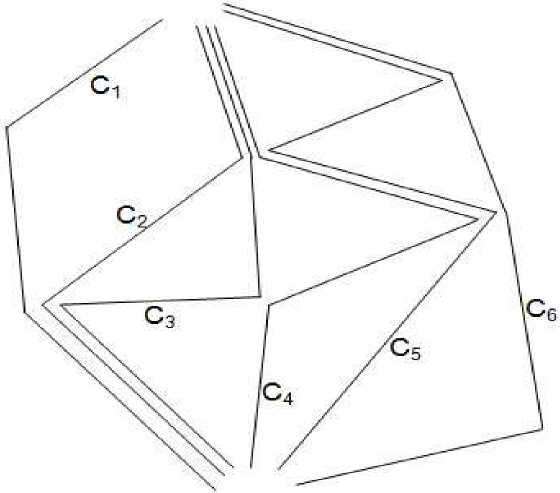
Planar point location by triangulation refinement



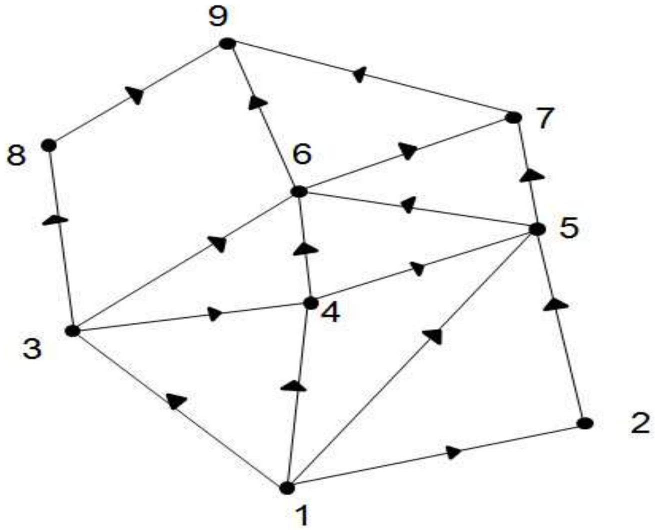
Planar point location using monotone chains



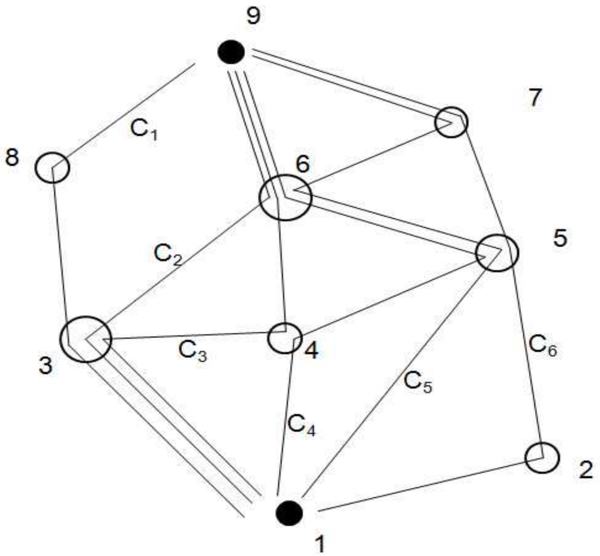
Planar point location using monotone chains



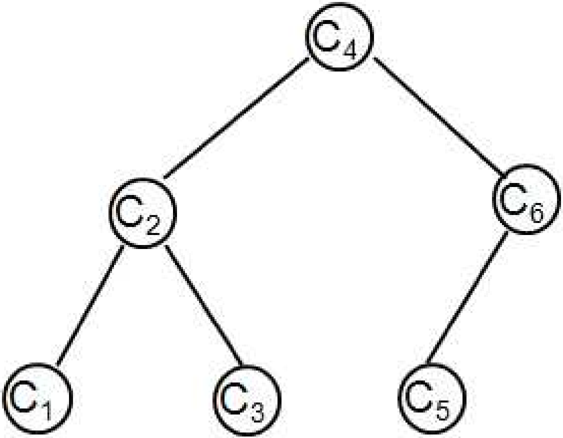
Planar point location using monotone chains



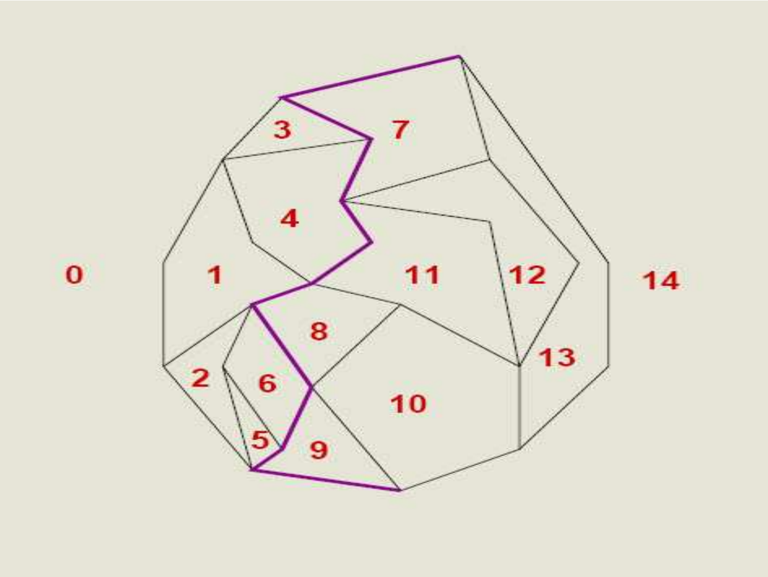
Planar point location using monotone chains



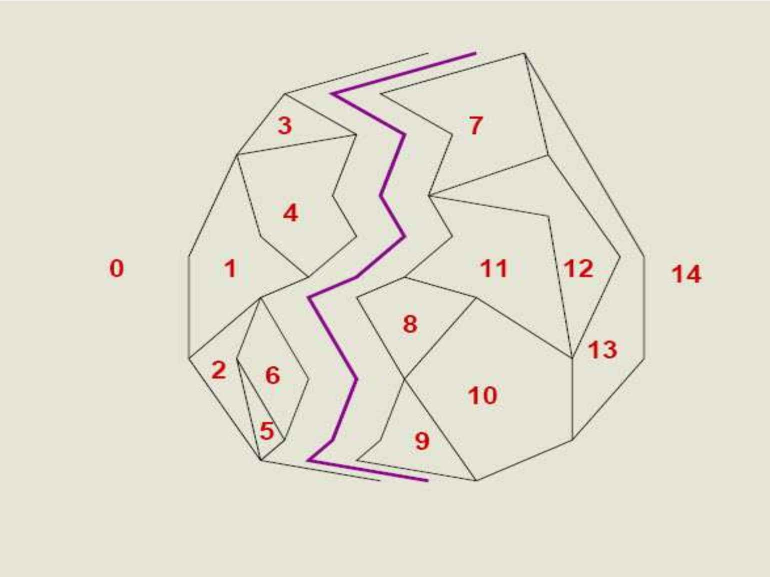
Planar point location using monotone chains



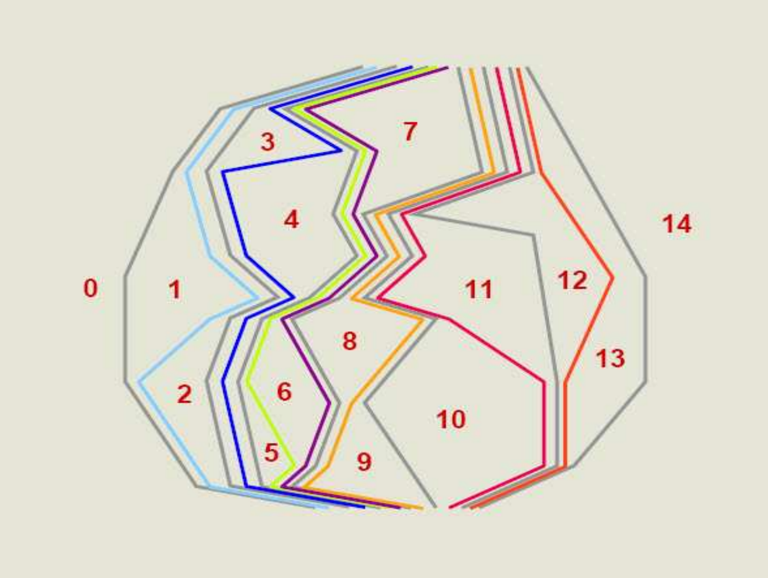
Planar point location using monotone chains



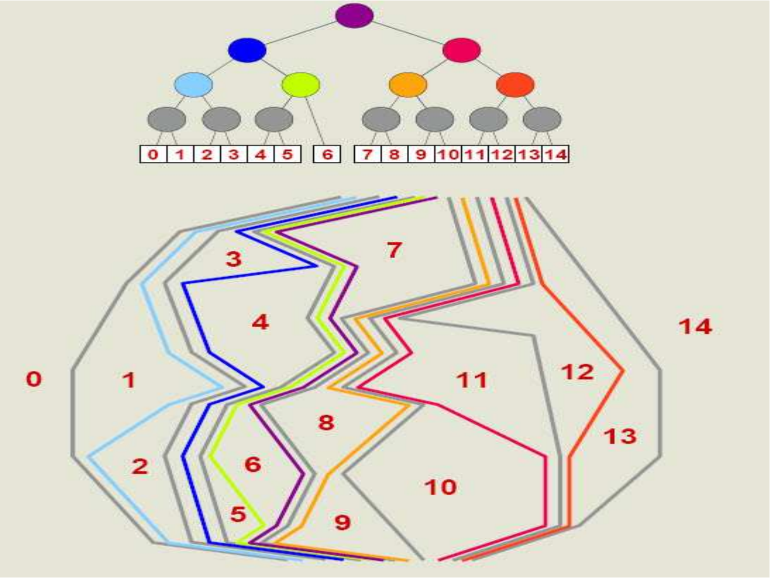
Planar point location using monotone chains



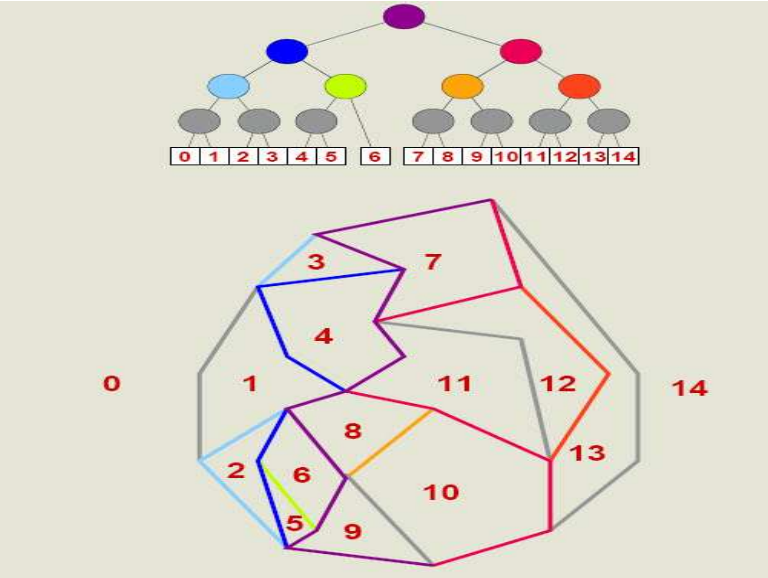
Planar point location using monotone chains



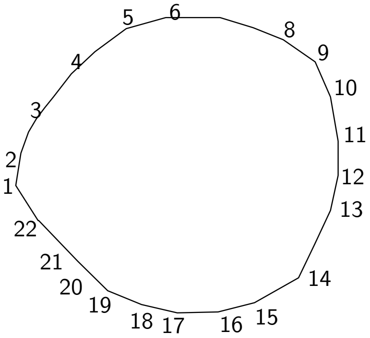
Planar point location using monotone chains

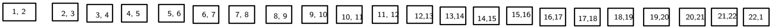


Planar point location using monotone chains

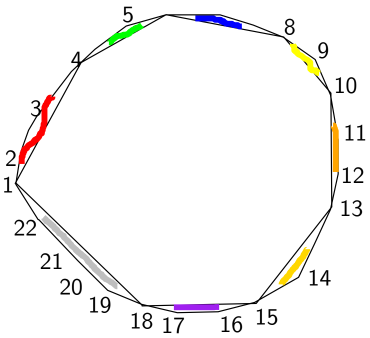


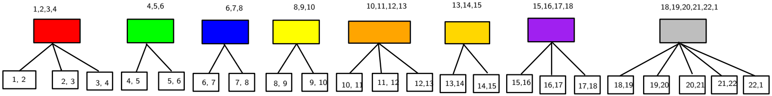
Representing a convex object layer by layer



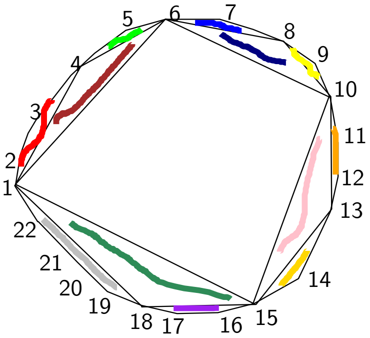


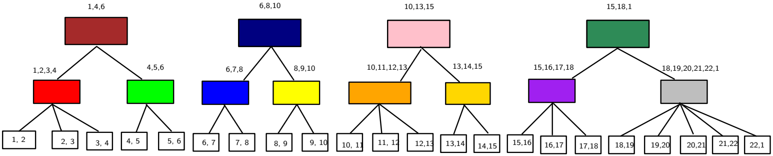
Second layer

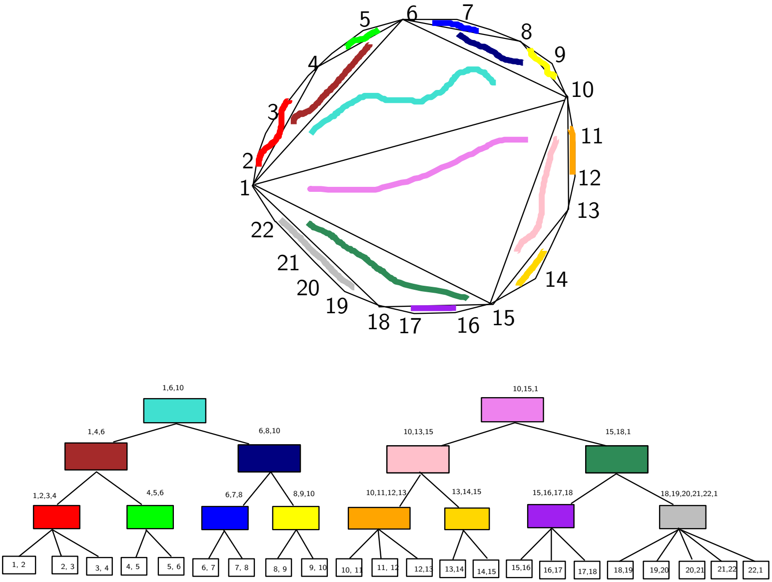




Third layer

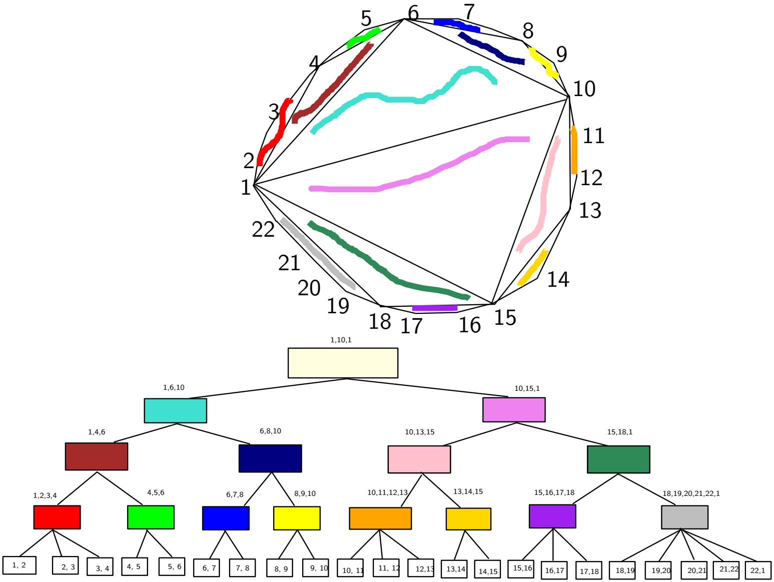

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22


1,4,6
6,8,10
10,13,15
15,18,1
1,2,3,4
4,5,6
6,7,8
8,9,10
10,11,12,13
13,14,15
15,16,17,18
18,19,20,21,22,1
1, 2
2, 3
3, 4
4, 5
5, 6
6, 7
7, 8
8, 9
9, 10
10, 11
11, 12
12,13
13,14
14,15
15,16
16,17
17,18
18,19
19,20
20,21
21,22
22,1



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
1,6,10
1,4,6
6,8,10
1,2,3,4
4,5,6
6,7,8
8,9,10
1, 2
2, 3
3, 4
4, 5
5, 6
6, 7
7, 8
8, 9
9, 10
10,15,1
10,13,15
15,18,1
10,11,12,13
13,14,15
15,16,17,18
18,19,20,21,22,1
10, 11
11, 12
12,13
13,14
14,15
15,16
16,17
17,18
18,19
19,20
20,21
21,22
22,1

Point inclusion and Line intersection queries



- Mark de Berg, Otfried Schwarzkopf, Marc van Kreveld and Mark Overmars, Computational Geometry: Algorithms and Applications, Springer.
- S. K. Ghosh, Visibility Algorithms in the Plane, Cambridge University Press, Cambridge, UK, 2007.
- Kurt Mehlhorn, Data Structures and Algorithms, Vol. 3, Springer.
- F. P. Preparata and M. I. Shamos, Computational Geometry: An Introduction, New York, NY, Springer-Verlag, 1985.