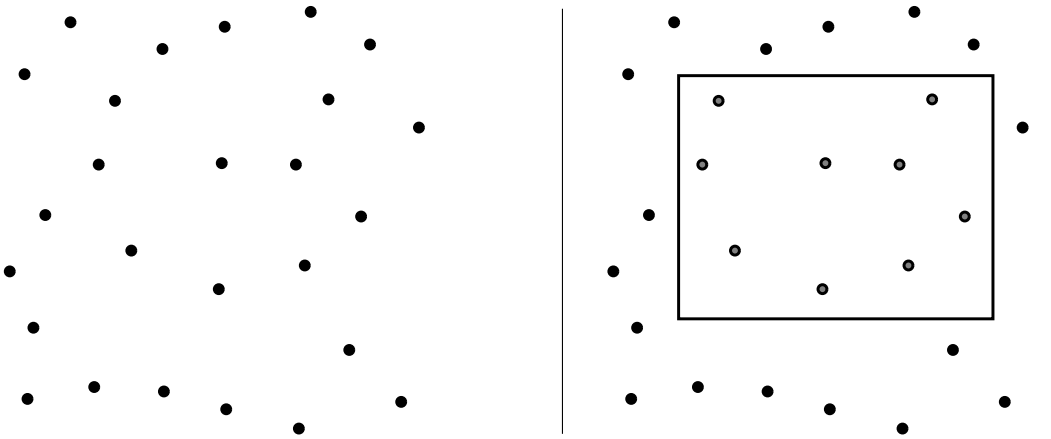

INPUT OUTPUT

17.6 Range Search

Input description: A set S of n points in E^d and a query region Q.

Problem description: What points in S lie within Q?

Discussion: Range-search problems arise in database and geographic information system (GIS) applications. Any data object with d numerical fields, such as person with height, weight, and income, can be modeled as a point in d-dimensional space. A *range query* describes a region in space and asks for all points or the number of points in the region. For example, asking for all people with income between \$0 and \$10,000, with height between 6'0" and 7'0", and weight between 50 and 140 lbs., defines a box containing people whose wallet and body are both thin.

The difficulty of range search depends on several factors:

- *How many range queries are you going to perform?* – The simplest approach to range search tests each of the n points one by one against the query polygon Q. This works just fine when the number of queries will be small. Algorithms to test whether a point is within a given polygon are presented in Section 17.7 (page 587).

- *What shape is your query polygon?* – The easiest regions to query against are *axis-parallel rectangles*, because the inside/outside test reduces to verifying whether each coordinate lies within a prescribed range. The input-output figure illustrates such an *orthogonal range query*.

When querying against a nonconvex polygon, it will pay to partition your polygon into convex pieces or (even better) triangles and then query the point set against each one of the pieces. This works because it is quick and easy to test whether a point lies inside a convex polygon. Algorithms for such convex decompositions are discussed in Section 17.11 (page 601).

- *How many dimensions?* – A general approach to range queries builds a kd-tree on the point set, as discussed in Section 12.6 (page 389). A depth-first traversal of the kd-tree is performed for the query, with each tree node expanded only if the associated rectangle intersects the query region. The entire tree might have to be traversed for sufficiently large or misaligned query regions, but in general kd-trees lead to an efficient solution. Algorithms with more efficient worst-case performance are known in two dimensions, but kd-trees are likely to work just fine in the plane. In higher dimensions, they provide the only viable solution to the problem.

- *Is your point set static, or might there be insertions/deletions?* – A clever practical approach to range search and many other geometric searching problems is based on Delaunay triangulations. Delaunay triangulation edges connect each point p to nearby points, including its nearest neighbor. To perform a range query, we start by using planar point location (see Section 17.7 (page 587)) to quickly identify a triangle within the region of interest. We then do a depth-first search around a vertex of this triangle, pruning the search whenever it visits a point too distant to have interesting undiscovered neighbors. This should be efficient, because the total number of points visited should be roughly proportional to the number within the query region.

  One nice thing about this approach is that it is relatively easy to employ "edge-flip" operations to fix up a Delaunay triangulation following a point insertion or deletion. See Section 17.3 (page 572) for more details.

- *Can I just count the number of points in a region, or do I have to identify them?* – For many applications, it suffices to count the number of points in a region instead of returning them. Harkening back to the introductory example, we may want to know whether there are more thin/poor people or rich/fat ones. The need to find the densest or emptiest region in space often arises, and this problem can be solved by counting range queries.

  A nice data structure for efficiently answering such aggregate range queries is based on the dominance ordering of the point set. A point x is said to *dominate* point y if y lies both below and to the left of x. Let $DOM(p)$ be a function that counts the number of points in S that are dominated by p. The number of points m in the orthogonal rectangle defined by $x_{\min} \leq x \leq x_{\max}$ and $y_{\min} \leq y \leq y_{\max}$ is given by

$$m = DOM(x_{\max}, y_{\max}) - DOM(x_{\max}, y_{\min}) - DOM(x_{\min}, y_{\max}) + DOM(x_{\min}, y_{\min})$$

The second additive term corrects for the points for the lower left-hand corner that have been subtracted away twice.

To answer arbitrary dominance queries efficiently, partition the space into n^2 rectangles by drawing a horizontal and vertical line through each of the n points. The set of dominated points is identical for each point in any rectangle, so the dominance count of the lower left-hand corner of each rectangle can be precomputed, stored, and reported for any query point within it. Range queries reduce to binary search and thus take $O(\lg n)$ time. Unfortunately, this data structure takes quadratic space. However, the same idea can be adapted to kd-trees to create a more space-efficient search structure.

Implementations: Both CGAL (*www.cgal.org*) and LEDA (see Section 19.1.1 (page 658)) use a dynamic Delaunay triangulation data structure to support circular, triangular, and orthogonal range queries. Both libraries also provide implementations of range tree data structures, which support orthogonal range queries in $O(k + \lg^2 n)$ time where n is the complexity of the subdivision and k is the number of points in the rectangular region.

ANN is a C++ library for both exact and approximate nearest neighbor searching in arbitrarily high dimensions. It performs well for searches over hundreds of thousands of points in up to about 20 dimensions. It supports fixed-radius, nearest-neighbor queries over all l_p distance norms, which can be used to approximate circular and orthogonal range queries under the l_2 and l_1 norms, respectively. *ANN* is available at *http://www.cs.umd.edu/∼mount/ANN/*.

Ranger is a tool for visualizing and experimenting with nearest-neighbor and orthogonal-range queries in high-dimensional data sets, using multidimensional search trees. Four different search data structures are supported by *Ranger*: naive kd-trees, median kd-trees, nonorthogonal kd-trees, and the vantage point tree. For each of these, *Ranger* supports queries in up to 25 dimensions under any Minkowski metric. It is available in the algorithm repository.

Notes: Good expositions on data structures with worst-case $O(\lg n + k)$ performance for orthogonal-range searching [Wil85] include [dBvKOS00, PS85]. Exposition on *kd*-trees for orthogonal range queries in two dimensions appear in [dBvKOS00, PS85]. Their worst-case performance can be very bad; [LW77] describes an instance in two dimensions requiring $O(\sqrt{n})$ time to report that a rectangle is empty.

The problem becomes considerably more difficult for nonorthogonal range queries, where the query region is not an axis-aligned rectangle. For half-plane intersection queries, $O(\lg n)$ time and linear space suffice [CGL85]; for range searching with simplex query regions (such as a triangle in the plane), lower bounds preclude efficient worst-case data structures. See Agrawal [Aga04] for a survey and discussion.

Related Problems: Kd-trees (see page 389), point location (see page 587).