CLOSEST POINTS

• Closest Pair

• Nearest Neighbor
Closest Pair

Given a set \( P \) of \( N \) points, find \( p, q \in P \) such that the distance \( d(p, q) \) is minimum.

- Algorithms for determining the closest pair:
  - brute force \( O(N^2) \)
  - divide-and-conquer \( O(N \log N) \)
  - plane-sweep \( O(N \log N) \)
Brute Force Algorithm

Compute all the distances $d(p,q)$ and select the minimum distance.

\[
d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
\]

Time Complexity: $O(N^2)$
**Plane-Sweep Algorithm**

- Maybe we can avoid having to check the distance between every pair of points...

- Plane-sweep worked for segment intersection, maybe it can be useful here...

**Key observation**: if the closest pair of points to the left of the sweep line is distance $d$ apart, the next point encountered can’t be a closest pair with any point more than $d$ units to the left of the line.

closest point to the left of $p$ can only be in the red-shaded region
Stored Information

- Maintain the following information:
  - the closest pair \((a,b)\) found so far, and the distance \(d\) between them
  - ordered dictionary \(S\) of the points lying in a strip of width \(d\) to the left of the sweep line, using the y-coordinates as keys
Updating

- When the sweep line encounters a point p:
  - update the dictionary so it only contains points that might be a closest pair with p
  - remove all points r such that \( x(p) - x(r) > d \) from S
  - find the closest point q to p in S
  - if \( d(p, q) < d \) then update the current closest pair and distance
  - insert p into S
• **How to quickly find the closest point in the dictionary?**
  - could be $O(N)$ points in the dictionary...

  have $x$, $y$ spacing so that $y = d/(n-1)$

• **Good news**: not all of the points in the dictionary can improve $d$
  - only eligible points are in half circle of radius $d$ centered at $p$
Searching the Dictionary II

• But how to search in a half-circle?
  - a rectangle is almost a half-circle...
  - do a range search in the interval $[y(p)-d, y(p)+d]$
  - this will get all the points in the half-circle (and maybe some others)

• Use brute-force to check the distance to each point returned by the range query

• But isn’t that still a potentially large number of points?
  - actually, there are at most 6
  - **key observation**: all of the points in the dictionary are at least distance $d$ from each other
Putting It All Together

• sort points by x-coordinate and store in ordered sequence X

• maintain references to two positions in sequence
  - firstInStrip: the leftmost point in S
  - lastInStrip: the new point to be added to S

• at each step..
  
  // advance lastInStrip
  lastInStrip ← X.after(lastInStrip)

  // remove points that are no longer candidates from dictionary
  while x(point(firstInStrip)) < x(point(lastInStrip)) - d do
    S.remove(point(firstInStrip))
    firstInStrip ← X.after(firstInStrip)

  // update closest point information
  find point q closest to point(lastInStrip) in S
  if d(p,q) < d then
    update closest pair
    d ← d(p,q)

  // insert new point into dictionary
  S.insert(point(lastInStrip))
An Example

initial closest pair and dictionary

one point in rectangle but not half-circle; closest pair not updated
An Example Continued

one point in rectangle but not half-circle; closest pair not updated

one point in rectangle but not half-circle; closest pair not updated
Still Going...

two points in rectangle, one on border of half-circle; closest pair not updated

two points in rectangle and half-circle; closest pair updated to nearer of the two
Example Completed

nothing within rectangle; closest pair not updated

the final result, with closest pair shown
Running Time

• initial sort takes $O(N \log N)$ time

• each point is inserted and removed once from $S$
  - $S$ has at most $N$ elements, so each insertion/removal takes $O(\log N)$ time
  - total insertion/removal time is $O(N \log N)$

• dictionary is searched once each time a point is inserted into $S$
  - each range query takes $O(\log N + 6) = O(\log N)$ time
  - total time for range queries is $O(N \log N)$

• distance computations performed each time a point is inserted into $S$
  - at most 6 computations at each time
  - total time for distance computations is $O(N)$

Time Complexity: $O(N \log N)$

(definitely beats the brute force method!)
Nearest Neighbor

• Given a set S of sites, what is the closest site to point q?

I.e. which post office is closest?

• Brute force is only $O(N)$!
  - but if you repeat the query for k different points (using the same set of sites) the total time is $O(kN)$

• Could do something based on plane-sweep, but that takes $O(N \log N)$ time for each query...$O(kN \log N)$ for k queries

• There’s a better solution...
Voronoi Diagram

• $S = \{ s_1, s_2, ..., s_N \}$
  - set of points in the plane, called sites

• Voronoi cell of $s_i$:
  - $C(s_i) = \{ p : d(p,s_i) \leq d(p,s_j), \forall j \neq i \}$
  - that is, the region of the plane containing all of the points that are closer to $s_i$ than any other site $s_j$

• Voronoi diagram of $S$
  - subdivision of the plane into Voronoi cells
Constructing a Voronoi Diagram

• Construct the perpendicular bisectors $h_{ij}$ of each segment $(s_i, s_j)$

• Let $H_{ij}$ be the half-plane delimited by $h_{ij}$ and containing $s_i$
  - all the points $p$ in $H_{ij}$ are closer to $s_i$ than $s_j$

• Voronoi cell for $s_i$ is the intersection of the half-planes $H_{ij}$ for all sites $s_j$ ($j \neq i$)

• Voronoi diagram can be constructed in $O(N \log N)$ time
  - can use divide-and-conquer or plane-sweep technique
Fun Voronoi Facts

• Each Voronoi cell is convex

• A Voronoi cell is unbounded if and only if the site is on the convex hull

• If $s_j$ is the nearest neighbor of $s_i$, the Voronoi cells $C(s_i)$ and $C(s_j)$ touch
Applications

• Given the Voronoi diagram, a nearest neighbor query can be performed in $O(\log N)$ time
  - $k$ queries can be done in $O((N+k) \log N)$ time

• Other applications
  - all nearest neighbors: for every point $p \in P$, find its nearest neighbor $q$
  - closest pair
  - Delaunay triangulation
    - a triangulation is a division of the plane into a set of triangular regions
  - convex hull
  - not just limited to computational geometry...
    - model region of influence in archaeology, ecology, ...
Shameless Plug

• Want to know how to compute a Voronoi diagram in $O(N \log N)$ time?

• Want to know how to do a nearest-neighbor query in $O(\log N)$ time?

• Want to learn about other cool geometric algorithms?

Take CS252: Computational Geometry!