Lecture 19: May 1, 2001

Computational Geometry

- Computational Model: arithmetic operations over reals
- Closest Pair
- Orthogonal Range Queries

Computation over reals

- We assume the input numbers (e.g., coordinates) are reals, arbitrary precision
- Therefore, we need to perform computation over reals

Why ?

- Avoid the discretization issues, e.g.:
  - What is a line in the discrete setting ?
  - Do two discrete lines intersect at only one point ?

Drawbacks:

- We have finite precision in real life, so sometimes we need to adapt the algorithms to finite precision
- If we allow arbitrary operations on real numbers, we get strange implications

A paradox: number compression

- Given: two reals $x, y \in (0, 1)$ such that $x = 0.x_1x_2\ldots$ and $y = 0.y_1y_2\ldots$
- Compressor: transform x, y into $z = 0.x_1y_1x_2y_2\ldots$
- Can compress many numbers into one !
- We are abusing the system ...

Solution

Allow only certain operations, e.g.

- Compute a (small degree) polynomial of the input coordinates, e.g., $x^2 + y^2$
- Compare result with 0, e.g., $x > 0$?
- Perhaps use floor/ceiling, e.g., $\lfloor x \rfloor$
- Try to do only reasonable things, e.g., in this lecture we only need to
  - check if $x_i > t$, where x_i is an input coordinate
  - compute distances between two points and compare their values

Closest Pair

- Given: $p_1 \ldots p_n \in \Re^2$
- Goal: find a pair p_i, p_j, $i \neq j$, such that

$$||p_i - p_j|| = \min_{i \neq j} ||p_i - p_j||$$

I.e., find the closest pair of points according to Euclidean distance.

Can do it easily in $O(n^2)$ time.
We will aim for $O(n \log n)$.

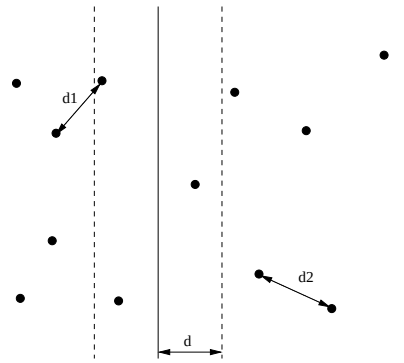
Divide and conquer

- Divide the points using a vertical line, such that the left set and the right set contain $n/2$ points each (i.e., find the median of x coordinates)
- Compute the closest pair for the left and right sets recursively
- Combine the results (the hard part)

Merge

Assume the length of the closest pair in the left set is d_1 (and d_2 for the right set).
Let $d = \min(d_1, d_2)$.

- Need to check only pairs which cross the median line
- We are interested only in pairs within distance $< d$
- $\Rightarrow$ Sufficient to look at points in the stripe around the median line



How to search the stripe ?

- Sort all points in the stripe by their y coordinates
- Let the resulting sequence be $q_1 \ldots q_k$
- For $i = 1$ to k
  - For $j = -7$ to 7, check the pair $q_i - q_{i+j}$

Why does it work ?

- Cannot have too many points within distance $< d$ from q_i, since otherwise the points would be so packed that some of them would have to be within $< d$ distance from each other, while the closest pair on each side has distance $\geq d$
- Argument as in the previous lecture ! (also in the book)

Running time

- Divide step takes $O(n)$ time
- Merge step takes $O(n \log n)$ due to sorting. However, we can sort only once at the beginning, so we get $O(n)$ time
- Then we get recursion $T(n) = 2T(n/2) + O(n)$, so $T(n) = n \log n$

Orthogonal range queries

- Given: a set of points $p_1 \ldots p_n$
- Goal: preprocess the points so that later, given a square $T = [x_l, x_r] \times [y_l, y_r]$, we can quickly check if there is $p_i \in T$
- In general, we can do more, e.g.
  - return the number of points falling into T
  - report all the points falling into T, etc.

  but we will keep things simple.

How to solve it ?

- Could get $O(n)$ time per query, but it is somewhat slow (e.g., compared to binary trees or hashing)
- Will aim for $O(\log n)$ query time

First idea: discretization

- Only the relative order between the coordinates (of points and query rectangle) is important, since

$$p = (p_x, p_y) \in [x_l, x_r] \times [y_l, y_r]$$

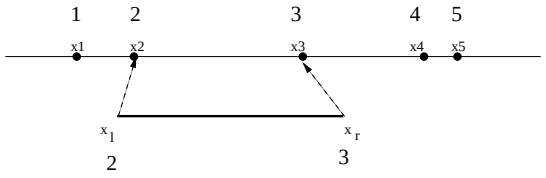
iff

$$x_l \leq p_x \leq x_r, y_l \leq p_y \leq y_r$$

- Therefore, if we replace the coordinates by other numbers, but preserve the ordering, nothing changes at all !

Implementation

- Sort all coordinates of points
- Replace each coordinate by its *rank*
- When a query T comes, round it as well (picture for x coordinates)



  - replace x_l by the rank of its successor in the point coordinate list
  - replace x_r by the rank of its predecessor
- Now all coordinates are integers from $\{1 \ldots n\}$, but the order did not change !

(to be continued)