Grid Decomposition: Closest Pair

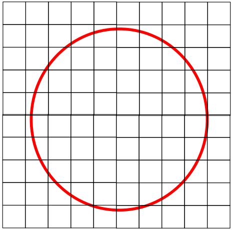
Yufei Tao

CSE Dept
Chinese University of Hong Kong

Lemma (Packing Lemma): Impose a regular grid on $\mathbb{R}^d$ where every cell is a box with side length s on each dimension. Any ball with radius r can overlap with no more than

$$\left(1 + \left\lceil \frac{2r}{s} \right\rceil\right)^d$$

cells.

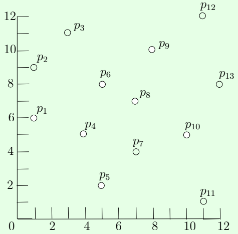


When d and r/s are constants, the number of overlapping cells is $O(1)$.

The packing lemma is surprisingly useful for solving computational geometry problems. Today, we will demonstrate an application of the lemma on the **closest pair** problem.

Let P be a set of points $\mathbb{R}^d$. The objective of the **closest pair problem** is to return a pair of distinct points $p, q \in P$ with the smallest Euclidean distance to each other.

Example:



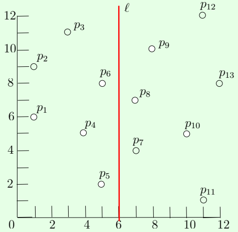
The answer is (p_6, p_8).

We will present an algorithm to solve the closest pair problem in $O(n \log n)$ expected time.

We will focus on 2D. Divide P evenly using a vertical line ℓ. Let P_1 (or P_2) be the set of points on the left (or right) of ℓ. Recursively find the closest pairs in P_1 and P_2, respectively.

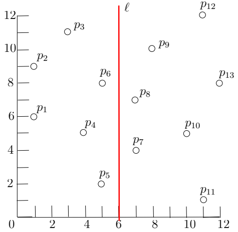
r_1 = the distance of the closest pair in P_1
r_2 = the distance of the closest pair in P_2.
Define $r = \min\{r_1, r_2\}$.

Example:



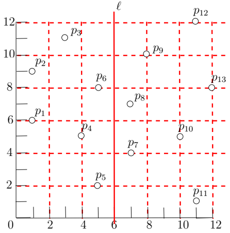
The closest pair of P_1 is (p_2, p_3) and that of P_2 is (p_7, p_8). Hence, $r_1 = \sqrt{8}$, $r_2 = 3$, and $r = \min\{r_1, r_2\} = \sqrt{8}$.

Next, we consider the **cross pairs** (p_1, p_2) where $p_1 \in P_1$ and $p_2 \in P_2$.



> **Observation:** We can focus on only the cross pairs within distance r.

Impose a grid G where (i) each cell is an axis-parallel square with side length $r/\sqrt{2}$, and (ii) ℓ is a line in the grid.

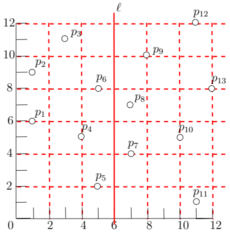


Each point p can be covered by at most 4 cells.

For each cell c, denote by $c(P)$ the set of points in P covered by c.

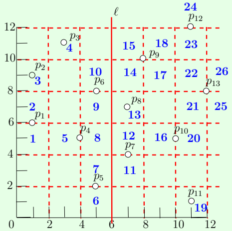
Observation: For every c, $|c(P)| \leq 2$.

The diagonal of c has length r. Convince yourself that c covering more than 2 points would contradict the definition of r.



Group the points by the cells they belong. A cell is **non-empty** if it covers at least one point. There can be at most $4n$ non-empty cells.

Example:



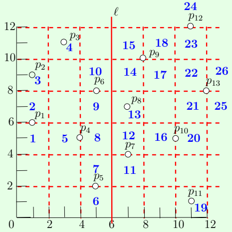
The non-empty cells are marked with numbers.

For each cell c, create a linked list containing the points in $c(P)$. This can be done in $O(n)$ expected time by hashing.

Two cells c_1 and c_2 are ***r*-neighbors** if their minimum distance is at most r.

Observation: A cell can have $O(1)$ r-neighbor cells (Packing Lemma).

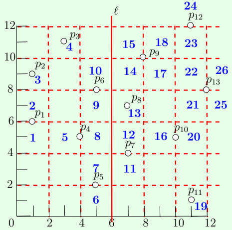
Example:



The r-neighbors of cell 12 are cells 5, 6, 7, 8, 9, 10, 11, 13, 14, 16, 17, 19, 20, 21, and 22.

It suffices to consider non-empty cells c_1 and c_2 such that (i) c_1 (resp., c_2) is on the left (resp., c_2) of ℓ, and (ii) they are r-neighbors.

Example:



We need to consider the cell pair (5, 11), but not (5, 15).

The above discussion motivates the following algorithm for finding the closest cross pair within distance r:

1. **for** every non-empty cell c_1 on the left of ℓ
2. **for** every r-neighbor cell c_2 of c_1 on the right of ℓ
3. calculate the distance of each pair of points $(p_1, p_2) \in c_1(P) \times c_2(P)$
4. **return** the closest one among all the pairs inspected at Line 3 within distance r.

Think: How to implement the algorithm in $O(n)$ time?

Let $f(n)$ be the expected running time of our algorithm on n points. It follows that

$$f(n) \quad \leq \quad 2 \cdot f(n/2) + O(n)$$

while $f(n) = O(1)$ for $n \leq 2$. The recurrence solves to $f(n) = O(n \log n)$.