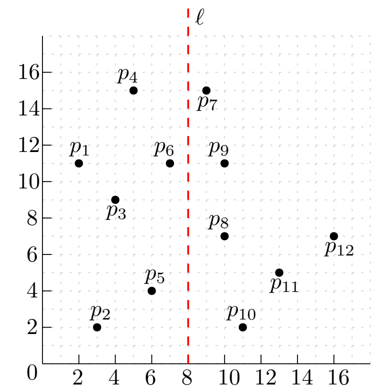
INFS 4205/7205: Exercise Set 7

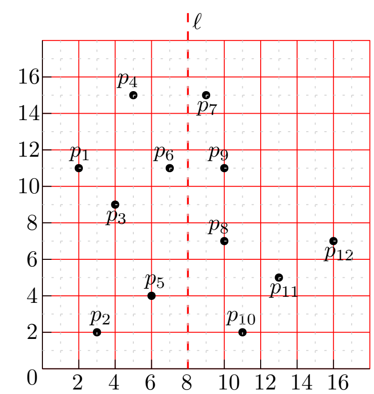
Prepared by Yufei Tao and Junhao Gan

Problem 1. Consider the set P of points as shown in the figure. Suppose that we run the closest pair algorithm on P. Recall that the algorithm first divides P in halves along the x-dimension using a vertical line ℓ (see the figure), recursively solves each half, and then builds a grid. Answer the following questions:



1. Draw the grid in the figure.
2. Consider the cell c_1 of the grid that covers point p_6. Recall that the algorithm needs to pair up c_1 with certain cells c_2 on the right of ℓ, in order to compute the distance of (p, q) for every pair of points p, q covered by c_1 and c_2, respectively. List the center coordiantes of all such cells c_2.

Solution.



The center coordinates of all such cells c_2 are: $(9, 7), (9, 11), (9, 15), (11, 7)$ and $(11, 11)$.

Problem 2. Let P be a set of points in $\mathbb{R}^d$. Give an $O(n \log n)$ expected time algorithm to find the 2nd closest pair of P. Formally, define $T = \{\{p, q\} \mid p, q \in P \wedge p \neq q\}$. The 2nd closest pair is the $\{p, q\} \in T$ that has the second smallest $dist(p, q)$ (i.e., Euclidean distance between p, q).

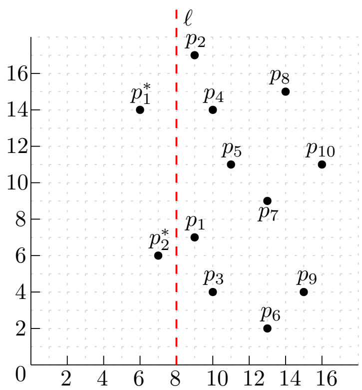
For instance, in the example dataset Problem 1, the 2nd closest pair is (p_6, p_9) (note that the first closest pair is (p_1, p_3)).

Solution. First find the closet pair (p_1, p_2). Then, remove p_1 from P, and find the closest pair (p'_1, p'_2) of the remaining points. Now, put back p_1, but remove p_2 from P, and find the closest pair (p''_1, p''_2) of the remaining points. The second closest pair must be either (p'_1, p'_2) or (p''_1, p''_2)

Problem 3. Let ℓ be a vertical line. Let p be a point on the left of ℓ, and P be a set of points on the right of ℓ. Define r as the distance of the closest pair of P. We throw away from P all the points whose distances to ℓ are greater than r. Define P' to be the set of remaining points in P.

For p, we define its *r-bounded nearest neighbor* (NN) as the point q in P that is closest to p, among all the points whose distances to p are at most r (if no such points exist, then p has no r-nearest neighbor).

For example, in the figure below, the closest pair in $P = \{p_1, \ldots, p_{10}\}$ is (p_5, p_7) whose distance is $2\sqrt{2}$. Thus, $r = 2\sqrt{2}$ and $P' = \{p_1, p_2, p_3, p_4\}$. If $p = p_1^*$, then p has no r-bounded NNs, while if $p = p_2^*$, the r-bounded NN of p is p_1.



Consider the following approach of finding the r-bounded NN of p. First, sort $P' \cup \{p\}$ by y-coordinate. Then, identify the position of p in the sorted list. Inspect the 20 points before and after p, respectively (namely, in total 40 points are inspected). Prove that the r-bounded NN (if exists) must be among those 40 points.

Proof. Impose an arbitrary grid in the data space such that: (i) each cell is an axis-paralleled square with side length $r/\sqrt{2}$, and (ii) ℓ is a line in the grid. In the class, we showed that: (i) each cell covers at most 2 points of P, and (ii) the cell containing p (on the left of ℓ) has at most 10 r-neighbors on the right of ℓ. Thus, at most 20 points can possibly be within distance r from p. Furthermore, in the sorted list of $P' \cup \{p\}$ by y-coordinate, all these (at most) 20 points must be stored consecutively around the position of p in the list. This completes the proof.

Problem 4. Let ℓ be a vertical line. Let P_1 be a set of points on the left of ℓ, and P_2 be a set of points on the right of ℓ. Define r_1 (or r_2) as the distance of the closest pair in P_1 (or P_2, resp.), and $r = \min\{r_1, r_2\}$. Suppose that P_1 and P_2 have been sorted by y-coordinate. Give an $O(n)$ time (where $n = |P_1| + |P_2|$) algorithm to find, for each $p_1 \in P_1$, its r-bounded NN in P_2.

Solution. Scan P_1 (or P_2) to obtain a sorted list P'_1 (or P'_2, resp.) containing only the points of P_1 (or P_2, resp.) whose distances to ℓ are at most r. Merge P'_1 and P'_2 into one list P', sorted by y-coordinate. The cost so far is $O(n)$.

Now scan P'. At any moment, keep the last 20 points seen from P'_1: call it the *P'_1-buffer*. Similarly a *P'_2-buffer* defined in the same way. Every time a point in P'_1 is encountered, calculate its distances to the points in the P_2'-buffer, and decide its r-bounded NN accordingly. Every time a point $p_2 \in P_2'$ is encountered, calculate its distance to each point p_1 in the P_1'-buffer. If p_2 is closer to p_1 than the r-bounded NN of p_1, update the r-bounded NN to p_2. In this way, each point is processed in $O(1)$ time. The total time is therefore $O(n)$.

Problem 5. Let P be a set of points in $\mathbb{R}^2$. Give an algorithm to find the closest pair of P in $O(n \log n)$ *worst case* time.

Solution. The algorithm is the same as the one taught in the class, except that we apply the solution in Problem 4 to find the "crossing" closest pair. A bit of care is used to maintain the sorted lists, in order to avoid repeated sorting.

- Sort P into separately by x- and y-coordinate, respectively. This gives two sorted lists $L_x(P)$ and $L_y(P)$ (each point duplicated twice, once in each list).
- Divide P into two equal halves P_1 and P_2 by a vertical line ℓ. Partition $L_x(P)$ into $L_x(P_1), L_x(P_2)$ and $L_y(P)$ into $L_y(P_1), L_y(P_2)$. The meanings of $L_x(P_1), L_x(P_2)$, $L_y(P_1)$, $L_y(P_2)$ follow those of $L_x(P)$ and $L_y(P)$.
- Recursively find the closest pairs in P_1 and P_2, respectively. Define r_1 (or r_2) as the distance of the closest pair in P_1 (or P_2, resp.), and $r = \min\{r_1, r_2\}$.
- For each point p of P_1, compute the r-bounded NN q in P_2 with respect to ℓ by utilizing the sorted lists $L_y(P_1)$ and $L_y(P_2)$. Pick the closest one among all such pairs (p, q) as the crossing closest pair.
- Return the best among the three pairs: the one reported in P_1, the one reported in P_2 and the crossing closest pair.

By the algorithm in Problem 4, the crossing closest pair between P_1 and P_2 can be computed in $O(|P_1| + |P_2|)$ worst case time. The total running time is therefore bounded by $O(n \log n)$ worst case time.