CS345 : Design and Analysis of Algorithms Spring 2014

Lecture 2: Closest Pair Problem

We revisit the algorithm paradigm of *Divide and Conquer*. You might have seen simple algorithmic problems based on this paradigm in your previous course - merge sort, quick sort, product of two n bit numbers, counting inversions. We shall now consider some advanced problems which are solved using divide and conquer paradigm. The first problem is the following geometric problem.

Problem: Compute the closest pair of points from a given set P of n points in x-y plane.

For sake of simplicity, we shall just compute the distance between the closest pair of point. Let $x(p)$ and $y(p)$ denote respectively the x-coordinate and y-coordinate of a point $p \in P$. There is a trivial $O(n^2)$ time algorithm for this problem. We shall now design an $O(n \log n)$ time algorithm for this problem based on divide and conquer strategy. The algorithm uses the following tools.

Tools needed

1. **A geometric fact** :
   In a unit square, if there are more than 4 points, then there must be at least 2 points at distance less than 1.

   As a simple corollary, if there is a unit square has some points such that the closest pair among them is at distance at least 1, then there can be at most 4 points in the square.

2. **The purpose of data structures**:
   If an algorithm requires execution a single operation multiple times on a given data, it makes sense to organize the data into a suitable data structure such that each operation can be performed efficiently.

An $O(n \log^2 n)$ time algorithm

In an interactive manner, through a question-answer session, we designed Algorithm 1 (see next page) for computing distance between the closest pair of points. Please see Figure 1 for some notations used in the algorithm.

If $T(n)$ denote the time complexity of the algorithm. The following recurrence captures the asymptotic behavior of $T(n)$.

$$T(n) = cn \log n + 2T(n/2)$$

The solution of this recurrence is $T(n) = O(n \log^2 n)$.

There are two components of the algorithm that contribute to $O(n \log n)$ term in the recurrence. First is the line 14 which sorts L'. Second component is the line 16 where we follow a procedure similar to the binary-search of $y(q)$ in sorted array L' to compute the points (at most 8) in δ-rectangle of a point $q \in R'$. In order to accomplish these tasks in $O(n)$ time, and hence achieve $O(n \log n)$ time complexity for the problem, we get inspiration from merge-sort (ponder over it).

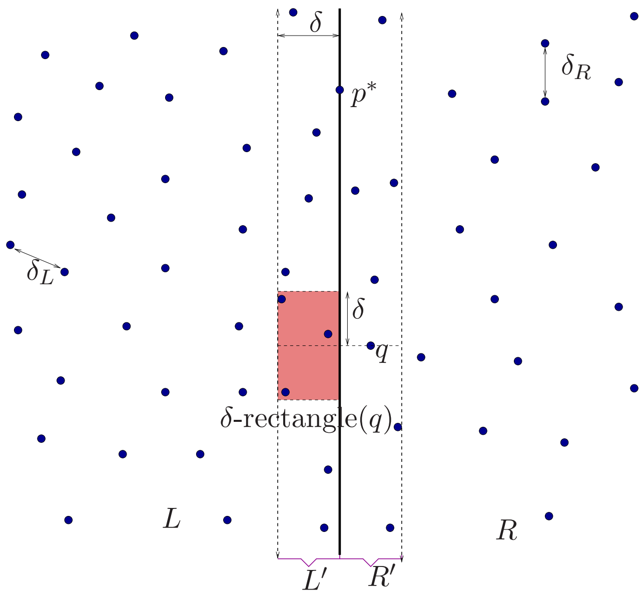

Figure 1: Initially $\delta = \min(\delta_L, \delta_R)$. Note that δ-rectangle(q) will have ≤ 8 points from L'.

Algorithm 1: Closest-pair(P)

---

if $|P| = 1$ **then**
    return ∞
else
    if $|P| = 2$ **then**
        return the distance between the 2 points in P
    else
        $p^* \leftarrow x$-median(P);
        $(L, R) \leftarrow$ Split-according-to-x-coordinates(P, p^*);
        $\delta_L \leftarrow$ Closest-pair(L);
        $\delta_R \leftarrow$ Closest-pair(R);
        $\delta \leftarrow \min(\delta_L, \delta_R)$;
        $L' \leftarrow \{p \in L | x(p) \geq x(p^*) - \delta\}$;
        $R' \leftarrow \{p \in R | x(p) \leq x(p^*) + \delta\}$;
        Sort L' in increasing order of y-coordinate;
        foreach *each point* $q \in R'$ **do**
            Compute all-points of L' present in δ-rectangle of q;
            Compute distance from q to each of at most 8 points in δ-rectangle of q;
            update δ if needed;
        end
        return δ;
    end
end

Improving the time complexity to $O(n \log n)$

Here is the sketch of the $O(n \log n)$ time algorithm for computing distance of the closest pair of points from P.

Algorithm 2: Closest-pair(P)

```
if |P| = 1 then
    return (∞, P)
else
    if |P| = 2 then
        ...;
    else
        p* ← x-median(P);
        (L, R) ← Split-according-to-x-coordinates(P, p*);
        (δ_L, L) ← Closest-pair(L);
        (δ_R, R) ← Closest-pair(R);
        δ ← min(δ_L, δ_R);
        L' ← {p ∈ L | x(p) ≥ x(p*) − δ};
        R' ← {p ∈ R | x(p) ≤ x(p*) + δ};
        // Note that L' and R' are sorted here;
        while L' ≠ ∅ and R' ≠ ∅ do
            ... ;
            ... ;
            ... ;
        end
        P ← y-merge(L, R);
        return (δ, P);
    end
end
```

As a homework, do the following exercises.

Exercise 1: Fill all the details of Algorithm 2. All it requires is your knowledge of merge sort and understanding of Algorithm 1.

Exercise 2: Some student pointed out that the algorithm may be simplified by first sorting P along x-axis and keeping it in an array A and sorting P along y-axis and keeping it in another array B. Convince yourself that sorting P along y-axis initially is not helpful in the algorithm.

Exercise 3: How will you extend the algorithm to compute closest-pair of n points in 3-dimensional space ?

Remark: As far as deterministic algorithms are concerned, $O(n \log n)$ is the best they can achieve for the closest-pair problem. However, there exists an equally simple but randomized algorithm for this problem that takes expected (average) $O(n)$ time.