Programming Assignment I : Closest-Points problem

Tamal Mondal

September 11, 2021

1 Problem Statement

We have N coordinates in two dimensional plane as $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$. The problem is to find the pair of points which are closest to each other or in other words the Euclidean distance between the two points is minimum. If there are two points with same coordinate then the distance will be 0(zero) between them and that pair of points will be answer. We need to output the closest pair of points with the distance between them.

2 Divide and conquer algorithm

In the naive way we can calculate the distance between every pair of points and find the minimum, which will be of $O(n^2)$ time complexity where n is the total number of points. In this divide and conquer approach, we divide the problem in smaller sub-problems and based on the results of the sub-problems, we will see that we don't need to calculate distance between every pair of points and that's how we improve the time complexity.

The high level idea of the algorithm is to first sort the points by X coordinate and divide them into two equal halves, in this way we get a *mid* point. Now we recursively find the closest pair of points in the left and right half. Let's say the minimum distance that we get from left and right half is d.

Now the closest pair can be in such a way that one point lies on the left and another in the right half. So we need to find the closest pair which is crossing the *mid*, if there is any. Notice that we only need to check the points whose X coordinate lies between $(X_{mid} - d)$ and $(X_{mid} + d)$ as we don't need to check the points which are apart from each other by more than d units. We will call this region as *strip* of $2d$ width.

Now to calculate the closest pair on the strip, even though number of points should be quite less in this region, in the worst case it can still take $O(n^2)$ time using brute force. To optimize this we will use some observations. We will see that if the points on the strip are sorted by Y coordinate, to check the closest point for any point say (x, y), we only need to check next 7 points(which means next 7 points in upward direction). In this way we can compute closest pair on strip in $O(n)$. As we need the points sorted by Y coordinate as well, we do this in the beginning and pass it as a parameter to the recursive function.

Finally the minimum distance among the distances in left half, right half and on the strip will be our answer. Algorithm 1 is the pseudo code of the main divide and conquer procedure to find closest pair of points and Algorithm 2 is the pseudo code to find the closest pair of points on the strip.

3 Proof of correctness

3.1 Correctness of the algorithm for finding closest points on strip

Here we will discuss the correctness of the procedure that takes the points on the strip(Let's say P_y) sorted by Y coordinate and the minimum distance found on the left and right half(let's say it's d), and it gives the closest pair which is on the strip having distance less than d.

First thing that we need to check if just looking on this strip of $2d$ width around the *mid* point is enough to find a closest pair that crosses the center. We will call the left region as L, right region as R and the middle point

Algorithm 1 : Divide and conquer algorithm to find the closest points

procedure FINDCLOSESTPAIROFPOINTS($P_x, P_y, low, high$)
    // Base case
    if $high - low + 1 <= 3$ **then**
        return $(minimumDistance, point1, point2)$ found by brute force
    end if

    // Get the mid point and find solution on the left and right recursively
    $mid = (low + high)/2$
    P_y^{left} = Points on the left of $P_x[mid]$, from P_y
    P_y^{right} = Points on the right of $P_x[mid]$, from P_y
    $(minDistance_{left}, point1_{left}, point2_{left}) = findClosestPairOfPoints(P_x, P_y^{left}, low, mid)$
    $(minDistance_{right}, point1_{right}, point2_{right}) = findClosestPairOfPoints(P_x, P_y^{right}, mid + 1, high)$

    // Get the minimum on the strip
    $minDistance = min(minDistance_{left}, minDistance_{right})$
    P_y^{strip} = Points on the strip, from P_y
    $(minDistance_{strip}, point1_{strip}, point2_{strip}) = closestPointsOnStrip(P_y^{strip}, minDistance)$

    // Return the minimum from left, right or strip
    if $minDistance_{left} <= minDistance_{right}$ and $minDistance_{left} <= minDistance_{strip}$ **then**
        return $(minDistance_{left}, point1_{left}, point2_{left})$
    else if $minDistance_{right} <= minDistance_{left}$ and $minDistance_{right} <= minDistance_{strip}$ **then**
        return $(minDistance_{right}, point1_{right}, point2_{right})$
    else
        return $(minDistance_{strip}, point1_{strip}, point2_{strip})$
    end if
end procedure

Algorithm 2 : Algorithm to find closest pair of points on the strip

procedure CLOSESTPOINTSONSTRIP($P_y, minimumDistance$)
    for $i = 0\,to\,(P_y.length - 2)$ **do**
        $j = i + 1$
        while $j < P_y.length \quad and \quad (P_y[j].y - P_y[i].y) < minimumDistance$ **do**
            $distance = getEuclideanDistance(P_y[i], P_y[j])$
            if $distance < minimumDistance$ **then**
                $minimumDistance = distance$
                $point1 = P_y[i]$
                $point2 = P_y[j]$
            end if
            $j = j + 1$
        end while
    end for
    return $(minimumDistance, point1, point2)$
end procedure

by which we divided the region as (x_{mid}, y_{mid}). Now if (x_l, y_l) and (x_r, y_r) are the two points in the region L and R respectively such that distance between them is closest and lesser than d, then we can write:

$$x_{mid} - x_l <= x_r - x_l < d$$

similarly,

$$x_r - x_{mid} <= x_r - x_l < d$$

which means that we only need to check in the d units to the left and d units to the right or in other words in the strip of $2d$ width.

Now we will focus in finding the closest pair on this strip. It's quite easy to think that in the worst case this strip can have all the points on it and we need to find closest pair among those, which is eventually the problem we are trying to solve. But we have the amazing observation that the distribution of points is quite sparse in nature on the strip and we can quantify it to some extent. To understand this fact, let's divide the strip into $d \times d$ squares in such a way that every square belongs to only in the L or R region and shares a side with imaginary middle line. Figure 1 depicts this.

As we know that the minimum distance we have found on the L and R region is d, every $d \times d$ square will have maximum 4 points and that has to be on the 4 corners as they need to maintain minimum d distance between them(notice that the 4 points will also be either in L or R region). So now if we consider any arbitrary point on the strip for which we need to find another point which is closest, we can just look into 4, $d \times d$ squares around it or only 15 points(not 16, as the point for which we are checking is also present in that region). Here it's important to understand that the reason of having points sorted by Y(let's say P_y) helps us as we can just iterate through the list and check next 7 points(basically 2, $d \times d$ squares in the upward direction) to find the closest and this way the whole computation becomes of $O(n)$ where n is the number of points on strip. This proves that we will definitely find the closest pair of points on strip if there is any by just checking next 7 points for every point on the strip.

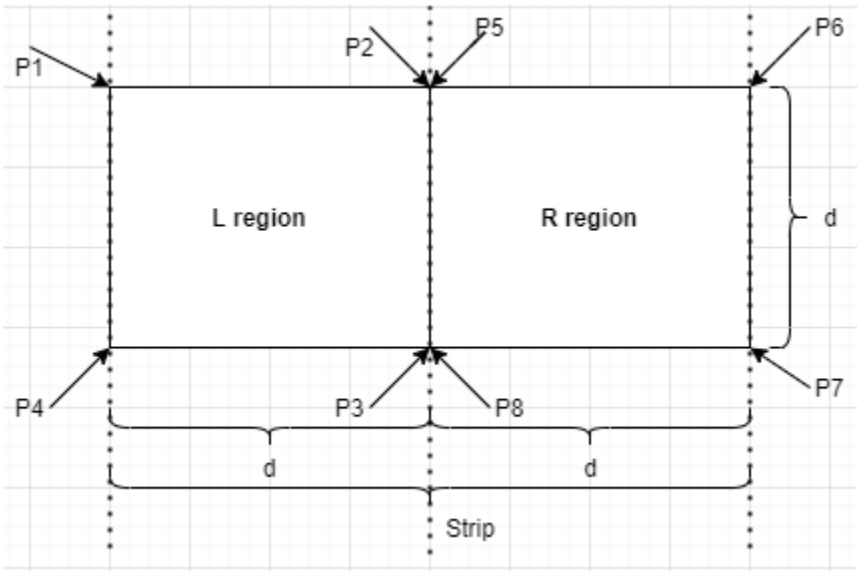

Figure 1: alignment of 8 points in the 2, $d \times d$ square

3.2 Correctness of the algorithm for the main recursive algorithm

The correctness of the main divide and conquer algorithm we will prove by the method of induction.

Base case: When number of points(n) = 1, 2 or 3, we solve the problem by brute force to find the closest pair of points.

Induction step: Now let's consider our algorithm gives closest pair of points correctly when number of points are less then n. Now let's say we call the procedure with n points. So as per algorithm, we will divide the n points in $n/2$ points in each half. As per induction hypothesis, by calling the same procedure with $n/2$ points will surely give us closest pair of points in the left and right half. If the minimum distance found is d, then we find all the points on the strip(of $2d$ width), sorted by Y coordinate. We pass these points to the procedure that finds the closest pair of points on the strip, which we have already proved as correct. As we get closest pair of points on left half, right half and on the strip, we just return the minimum out of those, and that would be the closest pair of points among all n points.

4 Time and space complexity analysis

Following is the recurrence relation of the divide and conquer algorithm discussed above:

$$T(n) = \begin{cases} c_1, & \text{if } n <= 3, \text{ where } c_1 \text{ is constant} \\ 2T(\frac{n}{2}) + c_2.n, & \text{otherwise, where } c_2 \text{ is constant} \end{cases} \tag{1}$$

Solving the above recurrence we can see that the time complexity is $O(nlogn)$. If we use merge sort for sorting the coordinates by X and Y, it will be of $O(nlogn)$ and as it's outside the recursion step, it does not effect the asymptotics.

As we need separate arrays to store all the coordinates in the sorted order by X and Y, space complexity is of $O(n)$ clearly.