The closest pair problem:
A plane sweep algorithm

Michiel Smid*

November 8, 2003

Let $S$ be a set of $n$ points in the plane. We want to compute a closest pair in $S$, i.e., two distinct points $P$ and $Q$ in $S$ such that

$$d(P, Q) = \min \{d(p, q) : p, q \in S, p \neq q\}.$$ 

Here, $d(p, q)$ denotes the Euclidean distance between the points $p$ and $q$,

$$d(p, q) = \left( (p_x - q_x)^2 + (p_y - q_y)^2 \right)^{1/2}.$$ 

We will solve this problem using the plane sweep paradigm. Hence, we move (sweep) a vertical line $SL$, the sweep line, from left to right over the points of $S$. During the sweep, we maintain the invariant that we have computed a closest pair among all points to the left of $SL$. Once the sweep line has visited the rightmost point, the invariant implies that we have found a closest pair in the entire set $S$.

During the algorithm, we maintain two data structures. The $Y$-structure contains information that is needed to update the closest pair each time $SL$ hits at a point of $S$. Observe that if $SL$ hits at a point of $S$, this $Y$-structure will change, i.e., it has to be updated. The positions at which the $Y$-structure changes are maintained in the $X$-structure.

The main problem is to find out how the $X$- and $Y$-structures look like. Here are the two main observations. Let $p$ be a point of $S$, let $S'$ be the set of all points of $S$ that are to the left of $p$, and let $\delta$ be the minimum distance in the set $S'$. Assume the sweep line hits at point $p$. At this moment, we know

---

*School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6. E-mail: michiel@scs.carleton.ca.
the value of $\delta$ (because of the invariant). In order to maintain the invariant, we have to compute the minimum distance in the set $S' \cup \{p\}$. We can do this by assigning
\[
\delta := \min(\delta, d(p, S')).
\] (1)

**Observation 1** In order to execute (1), we do not have to consider points of $S'$ whose $x$-coordinates are less than or equal to $p_x - \delta$.

Let $S''$ be the set of all points of $S'$ whose $x$-coordinates are larger than $p_x - \delta$. (Of course, these $x$-coordinates are at most equal to $p_x$.) Then Observation 1 says that we only have to consider points of $S''$.

**Observation 2** In order to execute (1), we only have to consider points of $S'$ whose $y$-coordinates are between $p_y - \delta$ and $p_y + \delta$. Moreover, there are at most six such points. (The last claim follows from the fact that all pairs of points of $S'$ have distance at least $\delta$.)

Now we can describe the $X$- and $Y$-structures. The $X$-structure is an array $A[1..n]$ containing the points of $S$ sorted by their $x$-coordinates, whereas the $Y$-structure is a balanced binary search tree containing the points of $S''$ sorted by their $y$-coordinates.

More precisely, if the sweep line $SL$ is the vertical line through point $p$ of $S$, then we have (refer to Figure 1)

1. a variable $r$ whose value is the position in the $X$-structure where point $p$ is stored, i.e., $A[r] = p$,

2. a variable $\delta$ whose value is the minimum distance among all points to the left of $SL$, i.e., the minimum distance among the points in $A[1..r-1]$,

3. a variable $\ell$ whose value is the index of the leftmost point in the $X$-structure whose $x$-coordinate is larger than $p_x - \delta$, i.e.,
\[
\ell = \min\{i : (A[i])_x > p_x - \delta\}
\]
(hence, $S'' = A[\ell..r-1]$),

4. a $Y$-structure, implemented as a balanced binary search tree, storing the points of $A[\ell..r-1]$ sorted by their $y$-coordinates. (By Observation 1, only these points are of interest to us, whereas by Observation 2, we have to be able to search these points by $y$-coordinate.)
The plane sweep algorithm for computing the closest pair in the set $S$ is given in Figure 2. I hope it is clear that this algorithm correctly solves the closest pair problem for any point set $S$. There remains one problem to be solved: how do we implement line $(\ast)$? We have to search in the $Y$-structure for all points having a $y$-coordinate between $p_y - \delta$ and $p_y + \delta$. By Observation 2, there can be at most six such points. Therefore, we do the following: We search in $Y$ for the six successors of $p_y - \delta$, i.e., the six points that are immediately above the point $(p_x, p_y - \delta)$. These six points surely include all points $q$ in the $Y$-structure for which $p_y - \delta < q_y < p_y + \delta$. Observe that in a balanced binary search tree, one successor can be found in $O(\log n)$ time.

We now have completely specified the algorithm. Let us consider the running time. The initialization takes $O(n \log n)$ time: It takes $O(n \log n)$ time to sort the points; the rest takes $O(1)$ time. (Observe that after the first while-loop, the value of $\ell$ is at most three.) Consider the main while-loop, in which $r$ runs from 3 to $n$. In one iteration, we need $O(\log n)$ time to search for the six points $q$, update $\delta$, and insert $p$ into $Y$. The inner while-loop may take much time, because we may have to delete a large number of points.
**Algorithm** \textit{fast\_closest\_pair}(S)  
\((\ast S \text{ is a set of } n \text{ points in the plane } \ast)\)  
sort the points from left to right, and store them in an array \(A[1..n]\);  
\(\delta := d(A[1], A[2]); \ r := 3; \ p := A[r]; \)  
\(\ell := 1; \)  
while \((A[\ell])_x \leq p_x - \delta\)  
do \(\ell := \ell + 1\)  
endwhile;  
is\(\text{initialize an empty balanced binary search tree } Y;\)  
for \(i := \ell \text{ to } r - 1\)  
do insert \(A[i]\) into \(Y\)  
endfor;  
\((\ast \text{ the initialization is now complete } \ast)\)  
while \(r \leq n\)  
do for each point \(q\) in \(Y\) such that \(p_y - \delta < q_y < p_y + \delta\) \((\ast)\)  
do \(\delta := \min(\delta, d(p, q))\)  
endfor;  
is\(\text{insert } p \text{ into } Y;\)  
if \(r < n\)  
then \(p := A[r + 1];\)  
\(\text{while } (A[\ell])_x \leq p_x - \delta\)  
do delete \(A[\ell]\) from \(Y;\)  
\(\ell := \ell + 1\)  
endwhile  
endif;  
endwhile;  
\(r := r + 1\)  
endwhile;  
return \(\delta\)

Figure 2: The plane sweep closest pair algorithm.
from $Y$. Observe, however, that each point can be deleted from $Y$ only once. Moreover, one such deletion takes $O(\log n)$ time. Therefore, the entire main while-loop takes $O(n \log n)$ time. We have proved the following result.

**Theorem 1** Algorithm fast closest pair$(S)$ computes the closest pair in a set of $n$ points in the plane in $O(n \log n)$ time.

**Exercise 1** Try to generalize this algorithm to points in three dimensions. What are the difficulties that you encounter?

We now consider a very simple variant of algorithm fast closest pair$(S)$. Its running time is $\Theta(n^2)$ in the worst case, but for random inputs, it will be quite fast. Moreover, it is very easy to implement.

We only maintain the array $A[1..n]$ and the variables $\delta$, $\ell$ and $r$. (That is, there is no $Y$-structure!) During one iteration of the main while-loop, we compute the distance from $p$ to all points in $A[\ell..r-1]$. This algorithm is still correct, because these points include those having a $y$-coordinate between $p_y - \delta$ and $p_y + \delta$. The pseudocode is given in Figure 3.

**Exercise 2** Prove that the worst-case running time of the new algorithm closest pair$(S)$ is $\Theta(n^2)$.

**Exercise 3** Implement algorithm closest pair$(S)$ in your favorite programming language. In order to save square root operations, compute $\delta^2$ instead of $\delta$. Test your implementation on random inputs for different values of $n$. Count how many times line (** ) is executed, and try to express this number as a function of $n$. This number is quadratic in $n$ in the worst case, but for random inputs, it should be much smaller. In algorithm fast closest pair$(S)$, the corresponding line is executed a linear number of times.
Algorithm closest_pair(S)
(* S is a set of n points in the plane *)
sort the points from left to right, and store them in an array A[1..n];
\( \ell := 1 \);
while \( (A[\ell])_x \leq p_x - \delta \)
do \( \ell := \ell + 1 \)
endwhile;
(* the initialization is now complete *)
while \( r \leq n \)
do for \( i := \ell \) to \( r - 1 \)
do \( \delta := \min(\delta, d(p, A[i])) \) (**)
endfor;
if \( r < n \)
then \( p := A[r + 1] \);
while \( (A[\ell])_x \leq p_x - \delta \)
do \( \ell := \ell + 1 \)
endwhile
endif;
r := r + 1
endwhile;
return \( \delta \)

Figure 3: A simple variant of the plane sweep closest pair algorithm. This one has a high worst-case running time, but will be fast on random inputs.