Closest Pair of Points

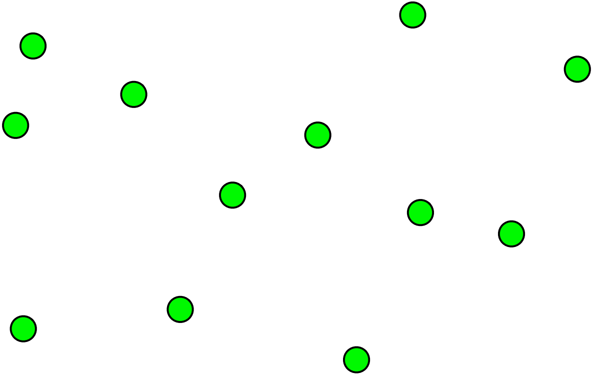
Slides by Carl Kingsford

Feb. 22, 2013

Section 5.4

Finding closest pair of points

Problem Given a set of points $\{p_1, \ldots, p_n\}$ find the pair of points $\{p_i, p_j\}$ that are closest together.



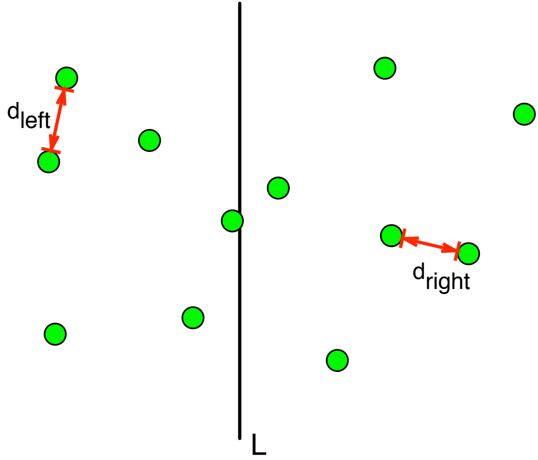
Goal

- Brute force gives an $O(n^2)$ algorithm: just check every pair of points.
- Can we do it faster? Seems like no: don't we have to check every pair?
- In fact, we can find the closest pair in $O(n \log n)$ time.
- What's a reasonable first step?

Divide

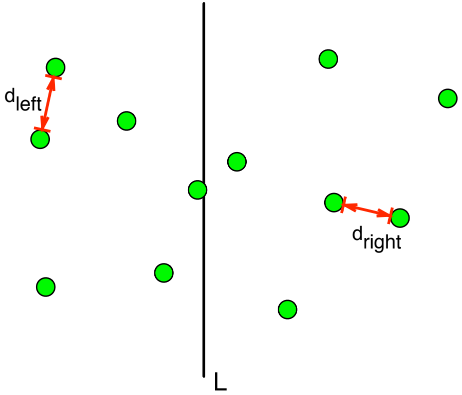
Split the points with line L so that half the points are on each side.

Recursively find the pair of points closest in each half.



Merge: the hard case

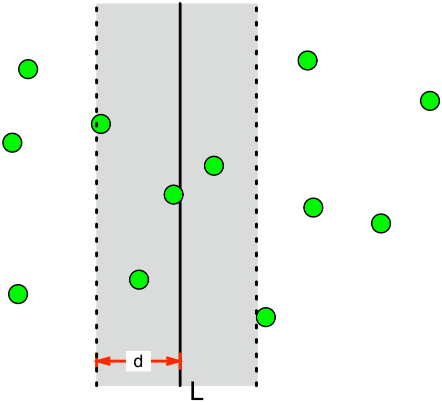
Let $d = \min\{d_{\text{left}}, d_{\text{right}}\}$.



- d would be the answer, except maybe L split a close pair!

Region Near L

If there is a pair $\{p_i, p_j\}$ with $\mathrm{dist}(p_i, p_j) < d$ that is split by the line, then both p_i and p_j must be within distance d of L.



Let S_y be an array of the points in that region, sorted by decreasing y-coordinate value.

Slab Might Contain All Points

- Let S_y be an array of the points in that region, sorted by decreasing y-coordinate value.
- S_y might contain all the points, so we can't just check every pair inside it.

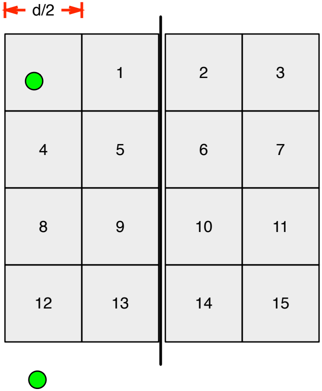
Theorem. *Suppose* $S_y = p_1, \ldots, p_m$. *If* $dist(p_i, p_j) < d$ *then* $j - i \leq 15$.

In other words, if two points in S_y are close enough in the plane, they are close in the array S_y.

Proof, 1

Divide the region up into squares with sides of length $d/2$:

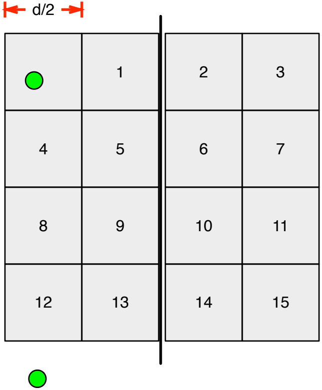
How many points in each box?



Proof, 1

Divide the region up into squares with sides of length $d/2$:

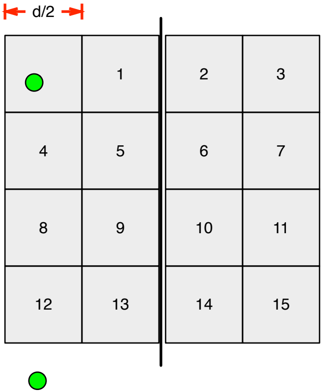
How many points in each box?



At most 1 because each box is completely contained in one half and no two points in a half are closer than d.

Proof, 2

Suppose 2 points are separated by > 15 indices.



- Then, at least 3 full rows separate them (the packing shown is the smallest possible).
- But the height of 3 rows is $> 3d/2$, which is $> d$.
- So the two points are father than d apart.

Linear Time Merge

Therefore, we can scan S_y for pairs of points separated by $< d$ in linear time.

```
ClosestPair(Px, Py):
   if |Px| == 2: return dist(Px[1],Px[2])    // base

   d1 = ClosestPair(FirstHalf(Px,Py))     // divide
   d2 = ClosestPair(SecondHalf(Px,Py))
   d = min(d1,d2)

   Sy = points in Py within d of L     // merge
   For i = 1,...,|Sy|:
       For j = 1,...,15:
           d = min( dist(Sy[i], Sy[j]), d )
   Return d
```

Total Running Time

- Divide set of points in half each time: $O(\log n)$ depth recursion
- Merge takes $O(n)$ time.
- Recurrence: $T(n) \leq 2T(n/2) + cn$
- Same as MergeSort $\implies O(n \log n)$ time.

Major topics covered so far

- Minimum spanning tree (Prim's, Kruskal, Reverse-Delete, Cycle Property, Cut Property)
- Heaps
- Heapsort
- Union-find (two implementations)
- Big-O notation
- BFS
- DFS
- Topological sort (two algorithms)
- Bipartite testing
- Shortest paths in graphs: Dijkstra's, Bellman-Ford
- A*
- Traveling salesman
- Counting inversions
- Mergesort
- Closest pair of points

Algorithm design techniques covered

- Tree growing (Examples: Prim's MST, Dijkstra's)
- BFS, DFS (Examples: topological sort, bipartite testing)
- Data structure-based (Example: Heapsort)
- Heuristic search with A* (Examples: TSP, geometric shortest paths)
- Divide and conquer (Examples: Mergesort, counting inversions, closest points)