Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 03

Wednesday January 11, 2017
based on notes by Denis Pankratov

Divide & Conquer (continue)

Closest Pair of Points in Euclidean 2D Space

Definition: $p_1, p_2 \in \mathbb{R}^2, p_1 = (x_1, y_2), p_2 = (x_2, y_2), d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Input: array A of n points in 2D, $n \geq 2$
Ouptut: (p_1, p_2) so that $p_1 < p_2$ appears in $A, d(p_1, p_2) = \min\{d(A[i], A[j]) | i \neq j\}$
Measure of Interest: # of operations on real #'s

```
def ClosestPairBruteForce(A, n):
  ans = float('inf')
  p_1 = p_2 = None

  for i = 1 to n:
    for j = i + 1 to n:
      if d(A[i], A[j]) < ans
        ans = d(A[i], A[j])
        p_1 = A[i]
        p_2 = A[j]

  return (p_1, p_2)
```

Running Time: $\Theta(n^2)$

What if we use Divide & Conquer?

1. Use vertical line l to split A into two arrays of roughly equal size
2. (p_1, p_2) - solution to the LHS
   (p_3, p_4) - solution to the RHS
3. Still have to consider pairs with one endpoint $\in L$, another $\in R$
4. Doing step (3) trivially involves checking $n/2 \cdot n/2$ pairs
5. Leads to a recurrence $T(n) = 2T(n/2) + \mathcal{O}(n^2)$, where $T(n)$ is the # of real operations on inputs of length n and $\mathcal{O}(n^2)$ is a bottleneck. Using the Master Theorem (*see last lecture*), this evaluates to $\mathcal{O}(n^2)$.

Approach: Splitting Points

$$\delta = \min(d(p_1, p_2), d(p_3, p_4))$$

Look at 2δ-bound around l and let B be an array of points within the bound sorted by the y-coordinate.

Claim: If $\exists p', p'' \in B$ so that $d(p', p'') < \delta$ then p', p'' appear within 8 indices of each other in B.

Now, consider p' and rectangle R.
Claim: $\exists$ at most 8 points from B in R
Proof: Suppose not $\exists 9$ points in R. According to the Pidgeonhole Principle (emphsee below), either $\exists 5$ points in R_1, or $\exists 5$ points in R_2. Without loss of generality, we may assume $\exists 5$ points in R_1.
Goal: $T(n) = 2T({}^n\!/_2 + \mathcal{O}(n)$, where the last part is the *implementation challenge*. To do this, we:
Keep X-copy of A sorted by x-coordinate.
Keep Y-copy of A sorted by y-coordinate.

```
def ClosestPair(A, n):
  X = copy of A sorted by x-coordinate
     (ties are broken by increasing y-coordinate)
  Y = copy of A sorted by y-coordinate

  /* check for duplicate points */
  for i= 1 to n - 1
     if X[i] &=& X[i + 1]
       return (X[i], X[i+1]) /* dist = 0 */

  return ClosestPairHelper(X, Y, n)
```

Hence, our goal is to make ClosestPairHelper have $T(n) = 2T({}^n\!/_2) + \mathcal{O}(n)$ (*see next lecture*).

Extra: Pidgeonhole Principle

Let $q_1, q_2, \ldots, q_n$ be positive integers. If

$$q_1 + q_2 + \cdots + q_n - n + 1$$

objects are distributed into n boxes, then either the first box contains at least q_1 objects, or the second box contains at least q_2 objects, $\ldots$, or the nth box contains at least q_n objects.

The simple form is obtained from this by taking $q_1 = q_2 = \cdots = q_n = 2$, which gives $n + 1$

objects. Taking $q_1 = q_2 = \cdots = q_n = r$ gives the more quantified version of the principle, namely:

Let n and r be positive integers. If $n(r-1)+1$ objects are distributed into n boxes, then at least one of the boxes contains r or more of the objects.

This can also be stated as, if k discrete objects are to be allocated to n containers, then at least one container must hold at least $\lceil k/n \rceil$ objects, where $\lceil x \rceil$ is the ceiling function, denoting the smallest integer larger than or equal to x. Similarly, at least one container must hold no more than $\lfloor k//n \rfloor$ objects, where $\lfloor x \rfloor$ is the floor function, denoting the largest integer smaller than or equal to x.

In *layman terms*, if you have 10 pidgeons and 9 pidgeonholes, then there must be *at least one* pidgeonhole with 2 or more pidgeons.