CSE 421

Divide and Conquer / Closest Pair of Points

Yin Tat Lee
Stuff

This Friday. A 20-min course assessment done by the college. I won’t be fired 😊 even if I got terrible review 😞. So, you can say ANYTHING. It is only used to help me to improve the teaching.

HW4 and HW5 is out. (You will like HW5.)

Midterm: Apr 30 (Monday)
It covers from Lecture 1 to Lecture 12 (Friday).

There is a midterm review for you to ask questions. I will be there and I will find an expert in grading (TA) to help me.
Removing weight Distinction Assumption

Suppose edge weights are not distinct, and Kruskal’s algorithm sorts edges so

\[ c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m} \]

Suppose Kruskal finds tree \( T \) of weight \( c(T) \), but the optimal solution \( T^* \) has cost \( c(T^*) < c(T) \).

Perturb each of the weights by a very small amount so that

\[ c'_{e_1} < c'_{e_2} < \cdots < c'_{e_m} \]

If the perturbation is small enough, \( c'(T^*) < c'(T) \).

However, this contradicts the correctness of Kruskal’s algorithm, since the algorithm will still find \( T \), and Kruskal’s algorithm is correct if all weighs are distinct.
Divide and Conquer Approach
Divide and Conquer

We reduce a problem to several subproblems. Typically, each sub-problem is at most a constant fraction of the size of the original problem.

Recursively solve each subproblem.
Merge the solutions.

Examples:
• Mergesort, Binary Search, Strassen’s Algorithm,
A Classical Example: Merge Sort

Split to $n/2$

sort recursively

merge
Why Balanced Partitioning?

An alternative "divide & conquer" algorithm:
• Split into n-1 and 1
• Sort each sub problem
• Merge them

Runtime

\[ T(n) = T(n - 1) + T(1) + n \]

Solution:

\[
T(n) = n + T(n - 1) + T(1) \\
= n + n - 1 + T(n - 2) \\
= n + n - 1 + n - 2 + T(n - 3) \\
= n + n - 1 + n - 2 + \cdots + 1 = O(n^2)
\]
Reinventing Mergesort

Suppose we've already invented Bubble-Sort, and we know it takes $n^2$

Try just one level of divide & conquer:

- Bubble-Sort (first $n/2$ elements)
- Bubble-Sort (last $n/2$ elements)

Merge results

Time: $2T(n/2) + n = n^2/2 + n \ll n^2$

Almost twice as fast!
Reinventing Mergesort

• “the more dividing and conquering, the better”
  • Two levels of D&C would be almost 4 times faster, 3 levels almost 8, etc., even though overhead is growing.
  • Best is usually full recursion down to a small constant size (balancing "work" vs "overhead").

  In the limit: you’ve just rediscovered mergesort!

• Even unbalanced partitioning is good, but less good
  • Bubble-sort improved with a 0.1/0.9 split:

    \[(.1n)^2 + (.9n)^2 + n = .82n^2 + n\]

    The 18% savings compounds significantly if you carry recursion to more levels, actually giving \(O(n \log n)\), but with a bigger constant.

• This is why Quicksort with random splitter is good – badly unbalanced splits are rare, and not instantly fatal.
Finding the Root of a Function
Finding the Root of a Function

Given a continuous function $f$ and two points $a < b$ such that
\[ f(a) \leq 0 \]
\[ f(b) \geq 0 \]
Find an approximate root of $f$ (a point $c$ where $f(c) = 0$).

$f$ has a root in $[a, b]$ by intermediate value theorem.

Note that roots of $f$ may be irrational.
So, we want to approximate the root with an arbitrary precision!
A Naive Approach

Suppose we want $\epsilon$ approximation to a root.

Divide $[a, b]$ into $n = \frac{b-a}{\epsilon}$ intervals. For each interval check $f(x) \leq 0, f(x + \epsilon) \geq 0$

This runs in time $O(n) = O\left(\frac{b-a}{\epsilon}\right)$

Can we do faster?
Divide & Conquer (Binary Search)

**Bisection** \((a, b, \varepsilon)\)

- If \((b - a) < \varepsilon\) then
  - return \(a\);
- Else
  - \(m \leftarrow (a + b) / 2\);
  - If \(f(m) \leq 0\) then
    - return Bisection\((c, b, \varepsilon)\);
  - Else
    - return Bisection\((a, c, \varepsilon)\);
Time Analysis

Let $n = \frac{a-b}{\epsilon}$ be the # of intervals and $c = (a + b)/2$

Always half of the intervals lie to the left and half lie to the right of c

So,

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

i.e., $T(n) = O(\log n) = O(\log(\frac{a-b}{\epsilon}))$

Shameless plug: How about $k$ dimension?
Yes, binary search still works for some functions. It takes $O(k^3 \log(\frac{k}{\epsilon}))$.
Fast Exponentiation
Fast Exponentiation

• **Power**($a, n$)

  **Input**: integer $n \geq 0$ and number $a$

  **Output**: $a^n$

• Obvious algorithm

  $n - 1$ multiplications

• Observation:

  if $n$ is even, then $a^n = a^{n/2} \cdot a^{n/2}$. 
Divide & Conquer (Repeated Squaring)

```c
Power(a, n) {
    if (n = 0)
        return 1
    else if (n is even)
        Compute x = Power(a, n/2)
        return x • x
    else
        Compute x = Power(a, (n - 1)/2)
        return x • x • a
}
```

Time (# of multiplications):
\[ T(n) \leq T(\lfloor n/2 \rfloor) + 2 \] for \( n \geq 1 \)
\[ T(0) = 0 \]

Solving it, we have
\[ T(n) \leq T(\lfloor n/2 \rfloor) + 2 \leq T(\lfloor n/4 \rfloor) + 2 + 2 \]
\[ \leq \cdots \leq T(1) + 2 + \cdots + 2 \leq 2 \log_2 n. \]

\[ \log_2(n) \] copies
Finding the Closest Pair of Points
Closest Pair of Points (general metric)

Given $n$ points and arbitrary distances between them, find the closest pair.

*Must* look at all $\binom{n}{2}$ pairwise distances, else any one you didn’t check might be the shortest. i.e., you have to read the whole input.
Closest Pair of Points (1-dimension)

Given $n$ points on the real line, find the closest pair, e.g., given 11, 2, 4, 19, 4.8, 7, 8.2, 16, 11.5, 13, 1 find the closest pair

Fact: Closest pair is adjacent in ordered list
So, first sort, then scan adjacent pairs.
Time $O(n \log n)$ to sort, if needed, Plus $O(n)$ to scan adjacent pairs

Key point: do not need to calculate distances between all pairs: exploit geometry + ordering
Closest Pair of Points (2-dimensions)

Given \( n \) points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.

Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force: Check all pairs of points in \( \Theta(n^2) \) time.

Assumption: No two points have same \( x \) coordinate.
Closest Pair of Points (2-dimensions)

No single direction along which one can sort points to guarantee success!
Divide & Conquer

**Divide**: draw vertical line $L$ with $\approx n/2$ points on each side.

**Conquer**: find closest pair on each side, recursively.

**Combine** to find closest pair overall

Return best solutions
Key Observation

Suppose $\delta$ is the minimum distance of all pairs in left/right of $L$. 
\[ \delta = \min(12, 21) = 12. \]

**Key Observation**: suffices to consider points within $\delta$ of line $L$.

Almost the one-D problem again: Sort points in $2\delta$-strip by their $y$ coordinate.
Almost 1D Problem

Partition each side of $L$ into $\frac{\delta}{2} \times \frac{\delta}{2}$ squares.

Claim: No two points lie in the same $\frac{\delta}{2} \times \frac{\delta}{2}$ box.

Proof: Such points would be within

$$\sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2} = \delta \sqrt{\frac{1}{2}} \approx 0.7\delta < \delta$$

Let $s_i$ have the $i^{th}$ smallest $y$-coordinate among points in the $2\delta$-width-strip.

Claim: If $|i - j| > 11$, then the distance between $s_i$ and $s_j$ is $> \delta$.

Proof: only 11 boxes within $\delta$ of $y(s_i)$. 
Closest Pair (2 dimension)

Closest-Pair(p_1, p_2, ..., p_n) {
    if(n ≤ 2) return |p_1 - p_2|

    Compute separation line L such that half the points are on one side and half on the other side.

    \[ \delta_1 = \text{Closest-Pair(left half)} \]
    \[ \delta_2 = \text{Closest-Pair(right half)} \]
    \[ \delta = \min(\delta_1, \delta_2) \]

    Delete all points further than \( \delta \) from separation line \( L \)

    Sort remaining points \( p[1]...p[m] \) by y-coordinate.

    for \( i = 1, 2, ..., m \)
        for \( k = 1, 2, ..., 11 \)
            if \( i + k ≤ m \)
                \[ \delta = \min(\delta, \text{distance}(p[i], p[i+k])) \];

    return \( \delta \).
}
Closest Pair Analysis

Let $D(n)$ be the number of pairwise distance calculations in the Closest-Pair Algorithm

$$D(n) \leq \begin{cases} 
1 & \text{if } n = 1 \\
2D\left(\frac{n}{2}\right) + 11n & \text{o.w.} \Rightarrow D(n) = O(n\log n)
\end{cases}$$

BUT, that’s only the number of distance calculations

What if we counted running time?

$$T(n) \leq \begin{cases} 
1 & \text{if } n = 1 \\
2T\left(\frac{n}{2}\right) + O(n \log n) & \text{o.w.} \Rightarrow D(n) = O(n\log^2 n)
\end{cases}$$
Closest Pair (2 dimension) Improved

Closest-Pair($p_1, p_2, \ldots, p_n$) {
    if($n \leq 2$) return $|p_1 - p_2|$

    Compute separation line $L$ such that half the points are on one side and half on the other side.

    $(\delta_1, p_1) = \text{Closest-Pair(left half)}$
    $(\delta_2, p_2) = \text{Closest-Pair(right half)}$
    $\delta = \min(\delta_1, \delta_2)$
    $p_{\text{sorted}} = \text{merge}(p_1, p_2)$ \hspace{1em} (merge sort it by $y$-coordinate)

    Let $q$ be points (ordered as $p_{\text{sorted}}$) that is $\delta$ from line $L$.

    for $i = 1, 2, \ldots, m$
        for $k = 1, 2, \ldots, 11$
            if $i + k \leq m$
                $\delta = \min(\delta, \text{distance}(q[i], q[i+k]));$

    return $\delta$ and $p_{\text{sorted}}$.
}

$T(n) \leq \begin{cases} 
    1 & \text{if } n = 1 \\
    2T\left(\frac{n}{2}\right) + O(n) & \text{o.w.}
\end{cases}$

$\Rightarrow D(n) = O(n \log n)$