CMPSCI 311: Introduction to Algorithms

Lecture 11: Wrapping up Material for First Exam

Akshay Krishnamurthy and Andrew McGregor

University of Massachusetts

Last Compiled: October 13, 2016

Minimum Distance Algorithm

- Divide points P with a vertical line into P_L and P_R where
$$|P_L| = |P_R| = n/2$$
- Recursively find minimum distance within P_L and P_R:
$$\delta_L = \min_{p,q \in P_L : p \neq q} d(p,q)$$
$$\delta_R = \min_{p,q \in P_R : p \neq q} d(p,q)$$
- Compute $\delta_M = \min_{p \in P_L, q \in P_R} d(p,q)$ and return
$$\min(\delta_L, \delta_R, \delta_M)$$
- We'll show who to do Step 3 in $O(n)$ time, which gives
$$T(n) \leq 2T(n/2) + O(n) \Longrightarrow T(n) = O(n \log n)$$

Making Step 3 Efficient

- Need to find $\min(\delta_L, \delta_R, \delta_M)$ where $\delta_M = \min_{p \in P_L, q \in P_R} d(p,q)$
- Suppose that the dividing line is $x = m$ and $\delta = \min(\delta_L, \delta_R)$
- Once we know δ, only need $O(n)$ comparisons to find $\min(\delta, \delta_M)$
  - Only compare $(p_1, p_2) \in P_L, (q_1, q_2) \in P_R$ if
$$m - \delta < p_1 \leq q_1 < m + \delta \quad \text{and} \quad |p_2 - q_2| < \delta$$
  - Each point $p \in P_L$ only gets compared with $O(1)$ points in P_R
- Need to identify the relevant comparisons in $O(n)$ time
  - Make two copies of points sorted by each coordinate
  - Ensure both lists are passed to each recursion sorted pause
  - Given sorted lists, it's easy to find the relevant points

Divide and Conquer Algorithms Recap

- Given a problem of an input of size n,
  - We generate (multiple) smaller instances of the problem
  - We solve each of these smaller instances
  - We use the solutions of the small instances to solve the original problem.
- Suppose that the first and third steps can be performed in $O(n^\alpha)$ time. If there are a smaller instances generated, each of size $n/2$, then the running time $T(n)$ of the algorithm satisfies the recurrence.
$$T(n) \leq aT(n/2) + O(n^\alpha)$$

Divide and Conquer: Recurrences

- Suppose $T(n) \leq aT(n/2) + n^\alpha$ and $T(1) \leq 1$. Then:
$$T(n) = \begin{cases} O(n^\alpha) & \text{if } \alpha > \log_2 a \\ O(n^{\log_2 a}) & \text{if } \alpha < \log_2 a \\ O(n^\alpha \log n) & \text{if } \alpha = \log_2 a \end{cases}$$
- If you forget this formula just apply the "unrolling method":
$$\begin{aligned} T(n) &\leq aT(n/2) + n^\alpha \\ &\leq a(aT(n/4) + (n/2)^\alpha) + n^\alpha \\ &\leq a(a(aT(n/8) + (n/4)^\alpha) + (n/2)^\alpha) + n^\alpha \\ &\leq \ldots \end{aligned}$$
- Some example recurrence: $T(n) \leq T(n/2) + 1$ and $T(n) \leq 4T(n/2) + n$
- Another a divide and conquer example: counting inversions.

Greedy Algorithms

- Greedy algorithms are "short sighted" algorithms that take each step based on what looks good in the short term.
  - **Example:** Kruskal's Algorithm adds lightest edge that doesn't complete a cycle when building an MST.
  - **Example:** When maximizing the number of non-overlapping TV shows we always added the show that finished earliest out of the remaining shows.
- Things to note:
  - If a greedy algorithm requires first sorting the input, remember to include the running time of sorting in your overall analysis.
  - Usually greedy algorithms can easily be shown to be poly-time but often extra work is required to get the most efficient implementation, e.g., union-find data structure.
  - Correctness proofs can be tricky: saw proofs by contradiction and induction, rearrangement arguments, some graph theory.
- Another example: minimizing number of coins when giving someone change.

Graph Algorithms: BFS and DFS Trees

- BFS trees with root r:
  - Partitions the nodes into levels $L_0 = \{r\}, L_1, L_2, L_3 \ldots$ where L_i consists of all neighbors of nodes in L_{i-1} that aren't already in $L_0 \cup L_1 \cup \ldots \cup L_{i-1}$.
  - If $v \in L_i$ the length of the shortest path in the original graph between r and v is i.
  - For any edge (u, v) in the original graph, u and v are in the same level or adjacent levels.
- DFS trees with root r:
  - For any edge (u, v) in the original graph, u is an ancestor of v in the tree or vice versa.
- Can be used to find the connected components of a graph and test whether the graph is bipartite.
- A directed graph is acyclic is there is no directed cycle. There is no directed cycle iff there is a topological ordering. Can find a topological order using the fact that a DAG has a node with no incoming edges.

Asymptotic Analysis

Given two positive functions $f(n)$ and $g(n)$:

- $f(n) = O(g(n))$ iff $f(n)/g(n)$ tends to some constant $c \geq 0$ as $n \to \infty$
- $f(n) = \Omega((g(n))$ iff $g(n)/f(n)$ tends to some constant $c \geq 0$ as $n \to \infty$
- $f(n) = \Theta((g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega((g(n))$.