Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Midterm Prep: Divide and Conquer

Given a set of points $P = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$, a point $(x_i, y_i) \in P$ is Pareto-optimal if there does not exist any $j \neq i$ such that such that $x_j > x_i$ and $y_j > y_i$. In other words, there is no point in P above and to the right of (x_i, y_i). Design a $O(n \log n)$-time divide-and-conquer algorithm that given P, outputs all Pareto-optimal points in P.

(Hint: Split the array by x-coordinate. Show that all points returned by one of the two recursive calls is Pareto-optimal, and that you can get rid of all non-Pareto-optimal points in the other recursive call in linear time).

Solution: Let L be the left half of the points when sorted by x-coordinate, and R be the right half. Recurse on L and R, let L', R' be the sets of Pareto-optimal points returned. Every point in R' is Pareto-optimal, since all points in L have smaller x-coordinates and can't violate Pareto-optimality of points in R'. For each point in L', it's Pareto-optimal iff its y-coordinate is larger than y_{max}, the largest y-coordinate in R. We can compute y_{max} in a linear scan, and then remove all points in L' with a smaller y-coordinate. We then return the union of L', R'.

This runs in $T(n) = 2T(n/2) + O(n) = O(n \log n)$ time.

2 Midterm Prep: FFT

(a) Cubing the 9^{th} roots of unity gives the 3^{rd} roots of unity. Next to each of the third roots below, write down the corresponding 9^{th} roots which cube to it. The first has been filled for you. *We will use ω_9 to represent the primitive 9^{th} root of unity, and ω_3 to represent the primitive 3^{rd} root.*

$\omega_3^0 : \omega_9^0,\quad ,$

$\omega_3^1 : \quad ,\quad ,$

$\omega_3^2 : \quad ,\quad ,$

(b) You want to run FFT on a degree-8 polynomial, but you don't like having to pad it with 0s to make the (degree+1) a power of 2. Instead, you realize that 9 is a power of 3, and you decide to work directly with 9th roots of unity and use the fact proven in part (a). Say that your polynomial looks like $P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_8 x^8$. Describe a way to split $P(x)$ into three pieces (instead of two) so that you can make an FFT-like divide-and-conquer algorithm.

(c) What is the runtime of FFT when we divide the polynomial into three pieces instead of two?

Solution:

(a) $\omega_3^0 : \omega_9^0, \omega_9^3, \omega_9^6$
$\omega_3^1 : \omega_9^1, \omega_9^4, \omega_9^7$
$\omega_3^2 : \omega_9^2, \omega_9^5, \omega_9^8$

(b) Let $P(x) = P_1(x^3) + xP_2(x^3) + x^2 P_3(x^3)$
where $P_1(x^3) = a_0 + a_3 x^3 + a_6 x^6$.
and $P_2(x^3) = a_1 + a_4 x^3 + a_7 x^6$.
and $P_3(x^3) = a_2 + a_5 x^3 + a_8 x^6$.

(c) We have the recurrence $T(n) = 3 * T(n/3) + O(n) = O(n \log n)$. So splitting up FFT into three pieces instead of two doesn't affect the runtime asymptotically.

3 Midterm Prep: DFS

Suppose we just ran DFS on a directed (not necessarily strongly connected) graph G starting from vertex r, and have the pre-visit and post-visit numbers $pre(v), post(v)$ for every vertex. We now delete vertex r and all edges adjacent to it to get a new graph G'. Given *just* the arrays $pre(v), post(v)$, describe how to modify them to arrive at new arrays $pre'(v), post'(v)$ such that $pre'(v), post'(v)$ are a valid pre-visit and post-visit ordering for some DFS of G'.

Solution: For all v such that $pre(r) < pre(v) < post(v) < post(r)$, set $pre'(v) = pre(v) - 1, post'(v) = post(v) - 1$. For all other v in G', $pre'(v) = pre(v) - 2, post'(v) = post(v) - 2$.

One valid DFS on G' is: Run DFS, whenever we need to pick a new vertex to explore from, or whenever we choose a neighbor of the "current vertex" to explore, choose the unvisited vertex with the smallest value of $pre(v)$. This will visit all vertices in G' in the same order as the DFS on G. For example, notice that vertices adjacent to r have lower previsit numbers than vertices that can't be reached from r. So this DFS on G' will explore the vertices reachable from r in G first, and then vertices not reachable from r, just like the DFS on G.

For vertices with $pre(r) < pre(v) < post(v) < post(r)$, their pre/post-visit number decreases by 1 in this DFS since we no long pre-visit r before them. For all other vertices, their pre/post-visit number decreases by 2 since we no long pre-visit or post-visit r before them.

4 Midterm Prep: Shortest Paths

You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights, and there is a special node $v_0 \in V$. Give an efficient algorithm that computes the length of the shortest path from s to t that passes through v_0 for all pairs s, t. Your algorithm should take $O(|V|^2 + |E| \log |V|)$ time.

Solution: The length of the shortest path from s to t that passes through v_0 is the same as the length of the shortest path from s to v_0 plus the length of the shortest path from v_0 to t.

We compute the shortest path length from v_0 to all vertices t using Dijkstra's. Next, we reverse all edges in G, to get G^R, and then compute the shortest path length from v_0 to all vertices in G^R. The shortest path length from v_0 to s in G^R is the same as the shortest path length from s to v_0 in G. These calls to Dijkstra's take $O((|V| + |E|) \log |V|)$ time.

Now, we can combine the results of the two calls to Dijkstra's to write down the output in $O(|V|^2)$ time.