182 Week 10 Discussion Notes

Colin Ni

May 22, 2025

This week for discussion section we will discuss three extra practice problems for the final.

Problem 1. Each tile in an $n \times n$ piece of land is either grassy or rocky. You want to build a square farm on the land, but you can only build on grassy tiles. Find the area of the largest farm you can build.

Solution. Let us model the land as a binary $n \times n$ matrix A, where $A[i, j]$ is zero if the tile at location (i, j) is rocky and one if it is grassy. We will find a largest square submatrix consisting of all ones. Square submatrices $A[i : i + s, j : j + s]$ can be parametrized by the location (i, j) of their top left corner and their side length s, and such a square submatrix consists of all ones if and only if

$$\text{sum}(A[i : i + s, j : j + s]) = s^2. \tag{1}$$

A brute force solution is to compute all sums of square matrices. There are n^2 locations (i, j) for the top left corner and $O(n)$ side lengths s, and it takes $O(n^2)$ time to compute the sum of each $s \times s$ square submatrix. Thus this takes $O(n^5)$ time overall.

We can use dynamic programming (or so-called 2D prefix sums) to determine the sum of any square submatrix in $O(1)$ time, given $O(n^2)$ preprocessing time. This reduces our runtime to $O(n^3)$ time. Let

$$C[i, j] = \text{sum}(A[: i, :, j])$$

be the cumulative sum of the array A. Since

$$C[i, j] = C[i - 1, j] + C[i, j - 1] - C[i - 1, j - 1],$$

we can compute C in $O(n^2)$ time, and this is what we referred to as the preprocessing time. Now we can compute the sum of any square submatrix in $O(1)$ time:

$$\text{sum}(A[i : i + s, j : j + t]) = C[i + s, j + t] - C[i, j + t] - C[i + s, j] + C[i, j].$$

We can do even better by binary searching the side length s. Consider the square matrices whose top left corner have location (i, j), and consider the array Q defined by

$$Q[s] = \text{sum}(A[i : i + s, j : j + s]).$$

Recalling (1), we want to find the maximal s such that $Q[s] = s^2$. Defining Q' to be the array defined by

$$Q'[s] = s^2 - Q[s],$$

we equivalently want to find the maximal s such that $Q'[s] = 0$. This array Q' is sorted! Indeed, $Q'[s]$ is the number of zeros in the square submatrix $A[i : i+s, j : j+s]$, and this is an increasing sequence. We can therefore binary search Q'. Let us discuss two implementation details. Since we want a maximal s such that $Q'[s] = 0$, we actually want to search for s such that $Q'[s] = 0$ and $Q'[s+1] > 0$, but this is easily doable. Moreover, as usual, since we can compute any element of $Q'[s]$ in $O(1)$ time, as usual we do not need to compute the entire array Q', so each binary search only takes $O(\log n)$ time, as expected. In summary, this reduces our runtime to $O(n^2 \log n)$. □

Remark. As a follow-up, what if your farm need not be square but instead must only be rectangular?

As a hint, recall that during discussion section week 5 (here are the notes), we described an algorithm that can do the following in $O(n)$ time: Given an $n \times n$ sorted array A, in the sense that $A[i+1, j] \geq A[i, j]$ and $A[i, j+1] \geq A[i, j]$ for all relevant i and j, determine whether an element x is in A. Using this to modify our solution, you should get an $O(n^3)$ time algorithm that solves the follow-up problem.

Problem 2. Suppose m men and w women tell a matchmaker (you) which of the people of the opposite gender they are willing to marry. Form as many marriages as possible.

Solution. Let us construct a network. Take m vertices for the men and w vertices for the women, and add a directed edge with capacity 1 from a man to a woman if both are willing to marry the other. Finally, add a source s and an edge with capacity 1 from s to each man, and add a target t and an edge with capacity 1 from each woman to t.

The problem is now equivalent to finding the maximum flow in this network. Recall that Edmonds-Karp runs in $O(|V||E|^2)$ time for any network. On the other hand, recall that for networks with integer capacities, Ford-Fulkerson runs in $O(|E|f)$ time, where f is the maximum flow in the network. Here we use Ford-Fulkerson to get a $O((m+w)^3)$ runtime, because there are at most $(m+w)^2$ edges and because the maximum flow is at most $\min(m, w) \leq m + w$. □

Problem 3. A cat and a mouse are playing a game on an undirected graph $G = (V, E)$ which has a vertex $x \in V$ made of cheese. They take turns moving, starting with the cat, and on each turn, they may choose to move to a neighboring vertex or remain still. The cat wins by catching the mouse, and the mouse wins by getting to the cheese x. If ever the position repeats, then the game is a draw. Assuming the cat and mouse both play optimally, determine the result of the game given the starting positions $s_{\text{cat}}, s_{\text{mouse}} \in V$.

Solution. We begin by constructing an auxiliary directed graph G'. Its vertices are the states of the game, which we describe as (v, w, t), where $v, w \in V$ are the locations of the cat and the mouse respectively and $t \in \{\text{cat} = -1, \text{mouse} = 1\}$ is the player whose turn it is to move. Its directed edges are the possible moves, which are of the form

$$(v, w, \text{cat}), (v', w, \text{mouse}) \quad \text{and} \quad (v, w, \text{mouse}), (v, w', \text{cat}),$$

where $(v, v'), (w, w') \in E$.

Let us take a moment to discuss some game theory common sense. If you have a winning position in a finite game, then you can guarantee a win in d moves for some minimal d. If it is your turn, then you can make a move that results in a state where you can win in $d-1$ moves, and if it is your opponent's turn, then any move your opponent makes will result in a state where you can win in $\leq d-1$ moves.

Let

$$f(v, w, t) \in \{\text{cat} = -1, \text{mouse} = 1, \text{draw} = 0\}$$

denote the outcome of the game from the state (v, w, t), and let $S_d \subset V$ denote the states of the game where one of the players can guarantee a victory in $\leq d$ moves. We claim that S_d consists of the vertices $(v, w, t) \in G'$ such that $f(v, w, t) \neq \text{draw}$ and (v, w, t) has distance at most d from a state (v', w', t') in S_0 such that $f(v, w, t) = f(v', w', t')$.

Optional proof: We proceed by induction. Observe that

$$f(v, v, t) = \text{cat} \quad \text{and} \quad f(x, w, t) = \text{mouse}$$

for all $v, w \in V$ and $t \in \{\text{cat}, \text{mouse}\}$, and these are the states in S_0. Now, suppose our claim holds for S_d, and let us consider S_{d+1}. Let $(v, w, t) \in S_{d+1}$ so that $f(v, w, t) \in \{t, -t\}$. If $f(v, w, t) = t$, then by our game theory common sense, there exists an edge from (v, w, t) to a vertex $(v', w', -t) \in S_d$ such that $f(v', w', -t) = t$. Thus since by induction $(v', w', -t)$ has distance $\leq d$ from S_0, it follows that (v, w, t) has distance $\leq d+1$ from S_0. On the other hand, if $f(v, w, t) = -t$, then by our game theory common sense, every neighboring vertex $(v', w', -t)$ of (v, w, t) is in S_d and is such that $f(v', w', -t) = -t$. By induction $(v', w', -t)$ has distance $\leq d$ from S_0, so again it follows that (v, w, t) has distance $\leq d+1$ from S_0.

We then have the following recurrence. Let $(v, w, t) \in S_{d+1}$. If $f(v', w', -t) = -t$ for all neighbors $(v', w', -t)$ in S_d of (v, w, t), then $f(v, w, t) = -t$. Otherwise if $f(v', w', -t) = t$ for some neighbor $(v', w', -t)$ in S_d, then $f(v, w, t) = t$. Otherwise $f(v, w, t) = \text{draw}$.

We use dynamic programming to compute this for all states (v, w, t). To order the subproblems, we simply BFS from S_0 because this visits all vertices in S_d before S_{d+1}.

Finally, let us discuss the runtime of this algorithm. Our auxilary graph G' has $2|V|^2$ vertices and $2|V||E|$ edges, and it can easily be constructed in $O(|V|^2 + |V||E|)$ time. The dynamic programming part is a BFS with constant

time computation per step, so it also takes $O(|V|^2 + |V||E|)$ time. It follows that the overall runtime is $O(|V|^2 + |V||E|)$. □