182 Week 8 Discussion Notes

Colin Ni

May 21, 2025

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

Problem 1. There are n side-by-side mountains with heights $h_1, \dots, h_n$, and a storm brings an absurd amount of rain. Compute the amount of water trapped in the mountains. For example:

Solution. The amount of water trapped is the sum of the amount water trapped above each mountain $2, \dots, n-1$. For these values of i, let $\ell_i = \max(h_1, \dots, h_{i-1})$ and $r_i = \max(h_{i+1}, \dots, h_n)$ denote the height of the highest mountain to the left and right of mountain i, respectively. Then the amount of water trapped above mountain i is $\max(\min(\ell_i, r_i) - h_i, 0)$, so our answer is

$$\sum_{i=2}^{n-1} \max(\min(\ell_i, r_i) - h_i, 0).$$

The brute force way of computing this sum is $O(n^2)$ because computing ℓ_i and r_i take $O(n)$ time to compute. But we can compute all ℓ_i and r_i values in $O(n)$ time beforehand using the recurrences $\ell_{i+1} = \max(\ell_i, h_{i+1})$ and $r_{i-1} = \max(r_i, h_{i-1})$. Thus the total runtime is $O(n)$.

Here is code:

```
max_left = accumulate(height, max)
cmax_right = reversed(list(accumulate(reversed(height), max)))
return sum(
  max(min(cl, cr) - h, 0)
  for cl, cr, h in list(zip(cmax_left, cmax_right, height))[1: -1]
)
```

□

Problem 2. There are $n+2$ balloons with point values

$$v_0 = 1, \qquad v_1, \qquad \ldots, \qquad v_n, \qquad v_{n+1} = 1.$$

You may pop the balloons $1, \ldots, n$. When you pop a balloon, you get its point value multiplied by the point values of its neighboring balloons, and the balloon disappears. Determine the highest number of points you can get by popping the n available balloons.

For example, for $1, 3, 1, 5, 8, 1$, you can pop balloons $2, 3, 1, 4$ in that order to get

$$3 \cdot 1 \cdot 5 + 3 \cdot 5 \cdot 8 + 1 \cdot 3 \cdot 8 + 1 \cdot 8 \cdot 1 = 167$$

points, and this is optimal.

Solution. We use dynamic programming. Let $f(i, j)$ denote the highest number of points we can get by popping the balloons $i, \ldots, j-1$. To get a recurrence, the key idea is to consider the last balloon popped, say balloon k. When the last balloon is popped, it scores $v_{i-1}v_kv_j$ points. Moreover, the pops to the left of balloon k do not affect the pops to the right of balloon k. Thus the maximum we can get by popping balloons $i, \ldots, j-1$ given that we pop balloon k last is

$$f(i, k) + f(k+1, j) + v_{i-1}v_kv_j.$$

Our recurrence is therefore

$$f(i, j) = \max\{f(i, k) + f(k+1, j) + v_{i-1}v_kv_j \mid k = i, \ldots, j-1\}.$$

To order our subproblems, observe that the differences $k-i$ and $j-(k+1)$ are both smaller than $j-i$. Thus we start with the (i, j) pairs that have difference $d = 1$, and we proceed by increasing d.

Here is code:

```
f = [(n + 2) * [0] for _ in range(n + 2)]
for d in range(1, n + 1):
  for i in range(1, n + 2 - d):
    j = i + d
    f[i][j] = max(
      v[i - 1] * v[k] * v[j] + f[i][k] + f[k + 1][j]
      for k in range(i, j)
    )
return f[1][n + 1]
```

□

Problem 3. We say a sequence $a_1, \ldots, a_n$ of integers is calming if each number is in the open interval determined by the previous two numbers, *i.e.*

$$a_{i+1} \in (\min(a_{i-1}, a_i), \max(a_{i-1}, a_i))$$

for all $i = 2, \ldots, n-1$. Find a longest calming sequence in an $m \times m$ matrix A of integers, where you may move in any of the 8 cardinal directions.

Solution. Consider the directed graph G whose vertices are the pairs

$$((i_{\text{prev}}, j_{\text{prev}}), (i_{\text{curr}}, j_{\text{curr}}))$$

of locations in the matrix that differ by a cardinal direction and whose edges are the possible moves in a calming sequence, *i.e.* the pairs of vertices of the form

$$((i_{\text{prev}}, j_{\text{prev}}), (i_{\text{curr}}, j_{\text{curr}})), ((i_{\text{curr}}, j_{\text{curr}}), (i_{\text{next}}, j_{\text{next}}))$$

that satisfy

$$A_{i_{\text{next}}, j_{\text{next}}} \in (\min(A_{i_{\text{prev}}, j_{\text{prev}}}, A_{i_{\text{curr}}, j_{\text{curr}}}), \max(A_{i_{\text{prev}}, j_{\text{prev}}}, A_{i_{\text{curr}}, j_{\text{curr}}})).$$

This directed graph G is acyclic: no calming sequence can have a repeated number because the interval $(\min(a_{i-1}, a_i), \max(a_{i-1}, a_i))$ is open.

We have thus reduced the problem to finding a longest path in a DAG (V, E). To do this, let $v_1, \ldots, v_k$ be a topological sort of the vertices V, and let $f(i)$ denote the longest path using only the vertices $v_1, \ldots, v_i$. Then we have the recurrence

$$f(i+1) = 1 + \max\{f(j) \mid j \leq i \text{ such that } (v_j, v_{i+1}) \in E\}.$$

Let us analyze the runtime of this algorithm. Let $M = m^2$ be the number of entries in the matrix A. The directed graph G has $\leq 8M$ vertices and $\leq 64M$ edges, and each edge takes constant time to construct (*i.e.* checking the calming condition). Thus constructing G takes $O(M)$ time. Topological sorting G takes $O(M)$ time, and the dynamic programming computation takes $O(M)$ time. Thus our algorithm takes $O(M) = O(m^2)$ time overall. $\square$