Kick Start 2017 - Round G

Analysis: Matrix Cutting

Matrix Cutting : Analysis

Small dataset

We have a 1-D array of size **M** which needs be broken down into **M** parts by making vertical cuts in an order of our choice, where each vertical cut yields a number of coins equal to the minimum value in the subarray at the time of the cut. Our objective is to make the cuts in an order that maximizes the total number of coins.

Since **M** can be no greater than 10 in this dataset, we can simply consider all possible orderings of the cuts. For each permutation, we can simulate the cuts and calculate the total number of coins, and then output the maximum total we find across all permutations. The complexity of this approach is O(**M**!), which is sufficient for the Small dataset.

However, we will have an easier time with the Large dataset if we come up with a better approach. One helpful observation is that once a cut is made, the problem has been reduced to two independent problems of the same type: one for the left subarray, and one for the right subarray. This strongly suggests a dynamic programming (DP) based solution.

We can define each subproblem in the DP as f(L, R): the answer for a subarray ranging from positions L to R, inclusive, in the original array. Our final answer is f(0, **M**-1). Now we need a recurrence relation, which we can develop by iterating over the position of the first cut we make, as in the following pseudocode:

```
f(L, R, A): // A is the original array

  ans = 0

  // Assuming first cut is made immediately to the right of cut_position
  for cut_position in L to R - 1, inclusive
    ans = max(ans , f(L, cut_position) + f(cut_position + 1, R))

  // we can calculate this in O(M).
  current_coins = minimum value in A over positions L to R, inclusive

  // For the current cut, we get the same number of coins no matter where we cut.
  return ans + current_coins
```

With memoization, the total complexity of this approach is O($\mathbf{M}^3$). Remember, the complexity of a DP approach is given by the number of possible distinct states times the cost of transitioning between states.

Large dataset

In the Large dataset, we have a 2-D matrix in which we can make horizontal cuts as well as vertical cuts. Since the total number of cuts could be up to 80, and those cuts could occur in many possible orders, our brute force method no longer works.

However, our efficient DP approach from the 1-D case can be extended to the 2-D case, by redefining our DP state to describe the answer for a submatrix instead of a subarray. So, we define f(L, R, P, Q) as the answer for a submatrix defined by the intersection of rows L through R, and columns P through Q (all limits inclusive). A recurrence relation can be derived by iterating over all possible horizontal and vertical cuts as the first cut we make. Pseudocode:

```
f(L, R, P, Q, A): // A is original matrix

  ans = 0

  // horizontal cuts
  for horz_cut = L to R - 1, inclusive
    ans = max(ans, f(L, horz_cut, P, Q) + f(horz_cut + 1, R, P, Q) + current_coins)
```

```
// vertical cuts
for vert_cut = P to Q - 1, inclusive
  ans = max(ans, f(L, R, P, vert_cut) + f(L, R, vert_cut + 1, Q) + current_coins)


// we need to calculate this in less than O(N + M), if we don't want this step
// to dominate the transition cost
current_coins = minimum value in current submatrix (defined by L, R, P, Q)

// For the current cut, we get the same number of coins no matter where we cut.
return ans + current_coins
```

The only remaining piece of the puzzle is: how do we calculate the minimum value in a submatrix efficiently, in time linear or better in the number of rows and columns of the submatrix? We can pre-calculate answer for all $O(\mathbf{N}^2\mathbf{M}^2)$ submatrices, which is easier if you fix the top-left corner of the submatrix and iterate over possible bottom-right corners. This can be done in $O(\mathbf{N}^2\mathbf{M}^2)$ overall.

The overall complexity of our DP approach is $O(\mathbf{N}^2\mathbf{M}^2(\mathbf{N} + \mathbf{M}))$, which is fast enough for the Large dataset.