Kick Start 2018 - Round D

Analysis: Funniest Word Search

Funniest Word Search: Analysis

Small dataset

As we approach the problem, we want to decide on what we can feasibly calculate. Since the upper bounds for **R** and **C** are reasonably low, we know that we can iterate through all combinations of (i1, j1, i2, j2) where (i1, j1) is the top left cell of a subgrid and (i2, j2) is the bottom right cell; however, recalculating the total length of words found for every subgrid will still take too long. Note that since the length of each valid word is 1 in the Small dataset, the total length from finding a single instance of a word is 4. One solution is to calculate prefix sums S(i, j) and then calculate totals[i1, j1, i2, j2] = S(i2, j2) - S(i1 - 1, j2) - S(i2, j1 - 1) + S(i1 - 1, j1 - 1).

The prefix sum solution is simpler, but we will examine a different algorithm that extends more readily to the Large dataset. For every i1, create an array column_sum[j] which, when we iterate through values for i2, will represent the total length of words found in the column bounded by (i1, j, i2, j). For every i2 ≥ i1, scan through row i2 to update column_sum[j] by adding 8 if (i2, j) is a word. After doing so, for every j1, start with totals[i1, j1, i2, j1] = column_sum[j1], and for every j2 > j1, update totals[i1, j1, i2, j2] = totals[i1, j1, i2, j2 - 1] + column_sum[j2]. As we calculate totals[], keep track of the best *fun size* and how many times it occurs. As a side-note, it is not necessary to store the entire totals[] array, since we only use one previous calculation. This algorithm takes O($\mathbf{R}^2\mathbf{C}^2$) time.

Large dataset

Knowing that all words are of length 1 is very beneficial in the Small dataset for quickly updating our column sums. Luckily, with a bit of precomputation, updating can be just as easy for the Large dataset. As long as our precomputations are not slower than O($\mathbf{R}^2\mathbf{C}^2$) time, we should not have any problems.

Before running our algorithm, we can add the reverse of every word to our dictionary, so we only need to scan for words going from left to right (now referred to as going right) and top to bottom (going down).

We can observe that the length of a word limits where it can begin. Using this property, we can reduce the number of words to search for at each location. As a result, we want to create a lookup table word_lookup[i] which has a list of words of length i at each respective index. Now let us use two more lookup tables right[i, j, k] and down[i, j, k] which will keep track of how many words start at cell (i, j) and have length ≤ k going right and down, respectively. We can populate the tables in the following manner: for every (i, j) start with length k = 1. While k is a valid length for a word going right starting at (i, j) with respect to the entire grid, check every word in word_lookup[k] to see if it can be found going right starting at (i, j). Record right[i, j, k] = (k * number of words found) + right[i, j, k - 1]. Increment k until it is not a valid length for a word starting at (i, j). Repeat for words going down instead of going right.

Now, we can use an algorithm which is very similar to the one we used on the Small dataset. Let us just focus on words going right. We will still use totals[i1, j1, i2, j2] for the same purpose. For every i1, have a table right_column_sums[j, k] which, when we iterate through values for i2, will represent the total length of words going right with length ≤ k which start at the column bounded by (i1, j, i2, j). For every i2 ≥ i1, scan through row i2 and update right_column_sums[j, k] = right_columns_sums[j, k] + right[i2, j, k] for all valid values of k (remember that a word must be short enough to start at column j). Then, we will work left to right. For every j2, with j2 starting at **C** - 1 and totals[i1, j2, i2, j2] = right_column_sums[j2, 1], and for every j1 < j2, update totals[i1, j1, i2, j2] = totals[i1, j1 + 1, i2, j2] + right_column_sums[j1, j2 - j1 + 1]. The process for words going down is analogous, but with the rows and columns switched, and starting our iteration on columns instead of rows. As totals[] is finalized for words going both right and down, keep track of the best *fun size* and how many times it occurs. It is also possible to not store the entirety of totals[] in the Large dataset solution and have a space complexity of $O((\mathbf{R} + \mathbf{C})^3)$ by solving for words going both right and down in the same loop, but doing so requires a slightly different precomputation step.

We still need to analyze the time complexity of the Large dataset solution. Creating word_lookup[] takes O(**W**) time. Creating right[] or down[] takes O(**R** Ã— **C** Ã— **W**) time. Creating all right_column_sums[] takes $O(\mathbf{R}^2\mathbf{C})$ time. Populating totals[] still has a longer runtime than any of our previously mentioned steps, resulting in our solution taking $O(\mathbf{R}^2\mathbf{C}^2)$ time.