Code Jam 2017 - Round 1A

Analysis: Alphabet Cake

Alphabet Cake: Analysis

A brute force cell filling Small approach

In the Small dataset, the cake can consist of at most 12 units. A viable brute-force strategy is to try all ways of filling in every blank cell with each letter already appearing on the cake. If *L* letters are already present, then we have *L* possibilities for each of the remaining 12-*L* cells. Then, the number of combinations to try is $L^{(12-L)}$, which is a lot less than 12^{12}. In this case, since *L* can range from 1 to 12, inclusive, the maximum value that this can take is 5^7 = 78125, which is a pretty small number for a computer.

One way to check a cake is to first make one pass through the cake, going left-to-right, top-to-bottom, and find each letter's upper leftmost position, lower rightmost position, and frequency. Then, for each letter, check that the rectangle defined by the upper leftmost position and lower rightmost position contains only that letter, and that its area equals the frequency count.

A brute force rectangle-extending Small approach

Another possible strategy is to start at a letter, draw a bounding box around it that includes only `?`s, move on to another letter, and so on. In the Small dataset, there are relatively few choices for how to draw these bounding boxes. As long as you use other pre-existing letters to prune the possible bounding boxes for each letter, and you carefully check for overlaps and unused cells, an exhaustive search should be fast enough.

There are non-exhaustive strategies similar to this that have the potential to fail, though. Consider this algorithm: for each letter, start with a 1-by-1 bounding box containing only that letter. Stretch the box as far as it will go to the top and bottom without hitting other letters, then stretch the box as far as it will go to the left and right without hitting other letters. Depending on the order in which you handle the letters, this can fail. For example, consider this case:

```
A?B
C??
??D
?EF
```

If the algorithm proceeds in left-to-right, top-to-bottom order, it will fill up the grid as follows, leaving an unfilled `?`:

```
AAB
C?B
CDD
CEF
```

A recursive Large approach

Fittingly enough for a problem about cake, there is a simple divide-and-conquer strategy for the Large dataset. If the cake has only one letter on it, we can fill it up with that letter. Otherwise, we can make a horizontal or vertical cut that divides the cake into two sub-cakes, such that each part has at least one letter on it; this creates two more instances of the same problem. It is always possible to do this if there are at least two letters on the cake; one surefire way is to make a dividing cut that includes the right border of the leftmost of a pair of letters, or, if they are in the same column, the bottom border of the topmost of the pair.

A greedy Large approach

There is a simple non-recursive approach as well. First, within each row, we can extend each existing letter into all cells to the right of that letter, until we reach another existing letter or the edge of the cake. Then, we can extend the leftmost existing letter (if any) into all cells to the left of that letter.

At this point, any rows that had no existing letters are still empty. We can make a pass through the grid from the second row to the bottom row, replacing each empty row with the row above it. Then we can make one more pass from the next-to-last row up to the top row, replacing each empty row with the row below it.

It is not difficult to prove that it is impossible for this strategy to produce a non-rectangular region for a letter, to produce disjoint regions for a letter, or to leave any cell unfilled.