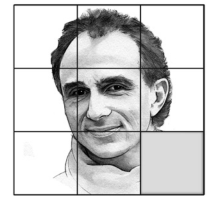

DOI:10.1145/3219818

Dennis Shasha

Upstart Puzzles
String Wars

SUPPOSE SOMEONE GIVES you two strings: X and Y. Your goal is to design a minimum-cost collection of smaller strings `coll(X|Y)` that match and cover every character of string X with order independence without matching any substring of string Y.

Let us first break down that last sentence:

The collection of strings in `coll(X|Y)` may have duplicates;

Matching and covering every character of string X means the strings in `coll(X|Y)` should tile string X without overlaps or gaps, and every tile should exactly match an underlying substring of X;

Not matching a substring of string Y means there should be no exact match of any string in `coll(X|Y)` to any substring of string Y; and

Order independence means no matter in which order the strings of `coll(X|Y)` is introduced and where they match, X will be tiled once the last string in `coll(X|Y)` is introduced.

When it satisfies all these properties, `coll(X|Y)` is called a "proper covering of X with respect to Y." The cost of `coll(X|Y)` is the sum of the squares of each element in the collection, including the duplicates.

Here is a simple example to get started. If string X is aaaaaaaaaa and Y is bbbbbbbbbbb, then `coll(X|Y)` consisting of 10 instances of "a" will be a proper covering. No instance of "a" will match any substring (letter, in this case) in Y. `Coll(X|Y)` is order-independent since the elements of `Coll(X|Y)` can be introduced in any order; all are just the single letter "a" after all. Further, the (total) cost is 10, because each "a" costs 1.

abaabaabaaba
bbabbbbaabba

A minimum-cost proper covering of the red string of characters with respect to the blue string is aba, aba, aba, aba for a cost of 4 × 9 = 36. A minimum-cost proper covering of the blue string with respect to the red string is abba, bbba, bbab for a cost of 3 × 16 = 48. The red string thus "beats" the blue string. Can you find a string that beats the red string?

Warm-Up 1. Continuing with this example, suppose X were abababababab and Y were aaaaaaaaaa. What would be a proper covering of X with respect to Y?

Solution to first warm-up. Five strings that are "ab" yielding a total cost of 20.

Warm-Up 2. Suppose X were abaababab and Y were bbababba. What would be a proper covering for X with respect to Y?

Solution to second warm-up. `coll(X|Y)` = {abaa, babab}. Breaking up either of these strings into shorter strings would entail some matches with Y.

Challenge. Given the scenario of the first warm-up, what would be a minimum-cost collection `coll(Y|X)` for Y that would cover Y with respect to X?

Solution. Note that five instances of "aa" would *not* be an order-independent cover of Y with respect to X. The reason is that, for example, one "aa" might match the second and third letters of Y, thus preventing a tiling, because no element would cover the first letter of Y. In fact, only `coll(Y|X)` = "aaaaaaaaaa" would work. That would have a cost of 10 × 10 = 100.

We see that an inexpensive order-independent covering of X may not work when elements of the covering might match Y. This brings up the possibility that an adversary—perhaps nature in the motivating use case of molecular biology—might create a Y that would greatly increase the cost of covering X.

String War Challenge. With respect to X = abaabaabaaba, the red string in the figure here, can you design a string Y of length 12 that can beat X? That is, we seek a Y such that the minimum-cost proper covering of Y with respect to X costs less than the minimum-cost proper covering of X with respect to Y.

Solution. Y = bbbbbabbbaba. `coll(Y|X)` = {bbbbba, bbbaba} having cost 36 + 36 = 72. `coll(X|Y)` = {abaabaabaaba} having cost 144.

String War Upstart. Given an X, can you always design a Y of the same length as X such that Y beats X? If so, design an algorithm to do so. Can you also design an algorithm to give a maximal difference in cost?

All are invited to submit their solutions to upstartpuzzles@cacm.acm.org; solutions to upstarts and discussion will be posted at http://cs.nyu.edu/cs/faculty/shasha/papers/cacmpuzzles.html

Dennis Shasha (dennisshasha@yahoo.com) is a professor of computer science in the Computer Science Department of the Courant Institute at New York University, New York, USA, as well as the chronicler of his good friend the omniheurist Dr. Ecco.