String Matching: Rabin-Karp

String Matching

In this tour we are looking for occurrences of the given white pattern in the yellow text. Here, the alphabet consists of the decimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Thus, the pattern can be considered as a (often very long) decadic number.

Using Step button, the text can be "read", i.e., shifted left. A sequence of pink boxes that will be called a *text section* is a copy of a section of the text that is located above the pattern. When Step has been clicked and the text is moving, and there is no direct correspondence of the pattern boxes to the text boxes and hence the text section boxes are empty. When the text stops, the part of the text that corresponds to the pattern is copied to the text section again.

When the characters in the text section matches the text in the pattern (i.e., an occurrence of the pattern in the text has been found), an alert "Matching" is displayed.

Text Section Update

The previous scene assumed that after the text stopped its leftward moving during a read phase, characters from the text are copied again to (now empty) boxes of the text section. The present scene shows more efficient way to update the text section boxes (click Phase to go through the following steps):

- As the text moves left, the text section moves together with it, and on the right the text section is completed by a new box containing the digit '0'.
- The text character that appears above the new text section box is moved to that box.
- The leftmost box of the text section is erased.

These operations modify the text section so that it again agrees with the boxes of the text that are above it (and above the pattern).

Assume now that our computer is able to perform directly arithmetic operations with decadic numbers that have as many digits as is the length of the

pattern and the text section[1]. In such a case, the previous three operations could be described in the language of arithmetics as follows:

- The shift of the text section to the left and its completion by '0' is a *multiplication by 10*.
- Replacing the rightmost '0' of the text section by a digit of the text is *adding the digit* to the (number representing) the text section.
- Removing the digit d that is the leftmost digit of the text section is *subtracting* $d \cdot 10^{\ell}$ from the text section, where ℓ is the length of the shorter form of the text section (which is equal to the length of the pattern).

A new occurrence of the pattern in the text is found if and only if the number represented by the pattern is *equal* to the number represented by the text section in the moment when the text does not move.

Since the pattern can be longer than the limit given by the hardware of the computer, the algorithm in the present form has strict limits of applicability. However, we will show how to modify it to accept any pattern length.

Text Section Building

This scene shows how to build the initial form of the text section, using arithmetic operations described in the previous scene. The text starts more right, just right of the rightmost box of the pattern. We can imagine that missing text boxes above the pattern are in fact present with the value '0', and hence all boxes of the text section contain '0' as well. After as many moves of the text left as it is the length of the pattern, the initial value of the text section is obtained. Note that during these operations, the leftmost box of the text section is '0' all the time, so we do not need to subtract anything when removing that box.

The role of Phase and Step buttons is changed in this scene to make this part of the computation faster: a click to Phase executes one round of myltiplying by 10, adding a digit and subtracting the leftmost box value, while after a click to Step all preparation of the initial value of the text section is executed in a continuous sequence of operations.

Modular Algorithm

The key idea of Rabin-Karp algorithm comes in the present scene. If, for long patterns, we can't perform easily arithmetics with sufficient precision, let us compare the text section and the pattern *modulo* some number m, the value of which is given by the line labeled "Modulo" in the control bar.

[1] E.g., in C++, the type **unsigned long long int** can store integers in the range 0 to 18,446,744,073,709,551,615, i.e., all 19-digit decadic numbers and some 20-digit ones, but, at this moment, we assume that this limitation does not apply.

We are not evaluating the text section and the pattern as (very large) numbers; instead, we want to know their remainder modulo m. The remainders are shown in the boxes right of the text section and the pattern, respectively.

The update of the text section, using a modular variant of the procedure of the previous scenes, is easy. Before matching we evaluate the modular remainder of the pattern and the number $S = 10^\ell \mod m$, where ℓ is the length of the pattern (note that 10^ℓ is the number containing a long sequence of 0's that appear below the pattern during the removal of the leftmost box of the text section): Then we proceed as follows:

- The shift of the text to the left is done by multiplying the modular remainder of the text section by 10 and taking the remainder modulo m.
- Replacing the rightmost '0' of the text section by a digit d is adding d to the modular remainder of the text section and taking the remainder modulo m.
- Removing the leftmost digit d of the text section is subtracting $d \cdot S$ from the modular remainder of the text section and taking the remainder modulo m.

The modular remainders of the text section and the pattern can be seen in the respective rectangles right to the text section and the pattern, resp.

Of course, if the modular remainders of the text section and the pattern are different, than it is clear that the pattern is *not* equal to the text section and therefore it does not match the text at that moment.

However, if the modular remainders are equal, the text section and the pattern *need not be equal*. In this case, the (un)equality must be checked box by box, which is animated in the screen.

The modulo m must be such that all numbers we obtain can be represented by a computer. Note that in one point of the algorithm, a modular remainder is multiplied by 10 - and should not overflow. If, e.g., we assume that our numbers are standard signed 4-byte integer, then we need $10m \leq 2^{31} - 1$, so any m below 200 million can be used.

The advantage of the algorithm is that steps with unequal modular remainders are very fast - just 3 simple modular operations. If the modulo m is quite large (within the limits given by the computer hardware), it is usually quite unlikely that a false alarm - modular equality of unequal text section and pattern - occurs.

In the pattern matches the text, the remainders are equal as well, and the equality text of the text section and the pattern has to go through their whole length, which is not a very efficient operation. However, if the number of occurrences of the pattern in the text is not large, the slow-down of the algorithm is not serious.

Thus, if the text does not contain too many occurrences of the pattern and the modulo m is sufficiently large and well chosen, Rabin-Karp algorithm is very fast yet very simple method of string matching.

Complete Algorithm

The last scene is the previous scene of the modular computation completed by computing of the initial value of the text section, again using the modular approach.