Pattern matching

Katalin Friedl
BME SZIT
friedl@cs.bme.hu

February 6, 2017

Basic problem: A given string (pattern) is sought in a given text. For example, it could be the task of a word processor, or just a pattern is sought in a file, or given piece of genetic code is looked for.

1 Basic concepts, notations

Let Σ be a finite set of symbols, the *alphabet*. It could be for example the English alphabet, numerical digits, bits, or A, G, C, T in genetics. Elements of Σ are called letters or *characters*. Finite sequences of elements of Σ are called *words*.

The collection of all words over Σ is denoted by Σ^*. The *pattern* we search for is a word of length $m > 1$, that is a sequence (array) $M[1]M[2]\dots M[m]$, where $M[i] \in \Sigma$. Similarly the *text* is a word of length $n \geq m$, that is a sequence $S[1]S[2]\dots S[n]$, where $S[i] \in \Sigma$. Pattern M is said to *occur in text* S *with shift* k ($k \geq 0$), if $M[j] = S[k+j]$ holds for all $j = 1, \dots, m$, or in short $M = S[k{+}1..k{+}m]$. Pattern M *occur* in text S if there exists an $0 \leq k \leq n-m$, such that M *occurs in text* S *with shift* k. In other words, the word S can be decomposed into three parts $S = UMV$, where the lengths of U or V could be 0. (We have $|U| = k$ and $|V| = n - k - m$.)

Several natural questions arise.

- Does pattern M occur in text S?
- Where is the first occurrence of M in S?
- Where does M occur in text S? (We want every occurrence of M.)

There are many known algorithms for each type of questions. The best to be used depends on many things, such as

- size of alphabet Σ;
- sizes of n and m;
- whether a given pattern is sought once or many times in different texts;
- whether many different patterns are sought in the same text.

Here we study only two methods. For further procedures check the book Christian Charras and Thierry Lecroq: Exact String Matching Algorithms (`http://www-igm.univ-mlv.fr/~lecroq/string`).

2 Simple matching

This is the "brute force" idea. For each $k = 0, 1, 2, \ldots, n - m$ it checks whether the pattern occurs with shift k in the text. One such test consists of comparisons $M[j] \stackrel{?}{=} S[k + j]$ for $j = 1, 2, \ldots, m$, where in case of non-equality the test continues with shift $k + 1$. It is clear that this procedure finds all occurrences of the pattern, or if we are only interested in the first one, or just the existence, then the procedure can be stopped at the first positive test.

Number of comparisons

At most m comparisons are made for each value of k, in total $(n - m + 1) \cdot m = O(nm)$

Example 1 *If S consists of only characters* `a` *and M consists of $m - 1$ of characters* `a` *and one* `b` *at the end, then this algorithm uses indeed $(n-m+1) \cdot m$ comparisons.*

Example 2 *If S consists of only characters* `a` *and M consists of only characters* `b`*, then each matching test requires only one comparison, that is $n - m + 1$ comparisons are enough in total*

3 Quick Search

The idea is that after checking the matching with shift k, the character $S[k + m + 1]$ will certainly occur in the next matching test, so only such shifts are worth trying where this character matches the character of the pattern shifted there. If M is shifted j further to the right (that is we check M with shift $k + j$, then $M[m + 1 - j]$ is checked against $S[k + m + 1]$, and this shift is worth checking only if they agree.

Preprocessing – the jump function

A *jump function* U is determined based on the pattern in this preprocessing phase of the algorithm. U is given by an array indexed by elements of alphabet Σ, $U[x]$ is the smallest i such that x stands in M on the i^{th} position counting backwards from the end of the pattern. Furthermore, let $U[x] = m + 1$, if x does not occur in pattern M. Array U can be filled up effectively:

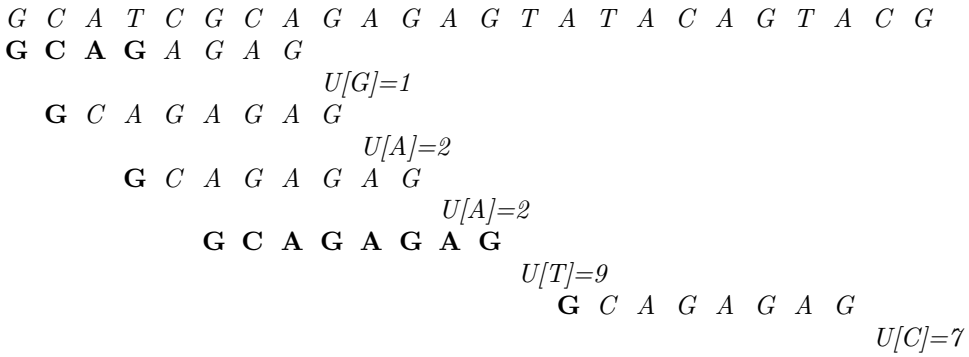
1. initialize all elements of U to value $m + 1$
2. For $i = 1, 2, \ldots, m$ let $U[M[i]] = m + 1 - i$

Example 3 *Let the pattern be sequence GCAGAGAG, and the alphabet be* $\Sigma = \{A, C, G, T\}$. *Then* $m = 8$, *and* $U[A] = 2$, $U[C] = 7$, $U[G] = 1$, $U[T] = 9$.

The algorithm

Start with shift $k = 0$. When a new shift is to be checked, then instead of $k+1$, let the next shift be $k + U[S[k + m + 1]]$.

Example 4 (continued) *Let the text S be GCATCGCAGAGAGTATACAGTACG* ($n = 24$)



5 actual shifts instead of the possible 17.
The number of comparisons is: $4 + 1 + 1 + 8 + 1 = 15$

The algorithm works correctly, because every time we jump to a new shift so that any smaller shift would be clearly bad, since $S[k + m + 1]$ would be positioned against a non-matching character in M.

Number of comparisons

The preprocessing takes $O(m + |\Sigma|)$ time, which is $O(m)$, since the size of the alphabet is constant.
We cannot state any better upper bound for the number of comparisons in the algorithm than $O(nm)$. So theoretically this method is not better (it is even worse, since we have to do the preprocessing), however in practice it works very

well. For exam pl, Simple search uses always at least $\Omega(n)$ comparisons, while there are cases of Quick search when $O(n/(m+1))$ is enough. (Why?)

Remark 1 *There are algorithms that use information collected from already matched characters in order to decrease the number of unsuccessful shifts. In fact, Quick search (Sunday, 1990) is obtained as a simplified version of such a more general method (Boyer–Moore-algorithm, 1977) and became popular.*

Remark 2 *There is an algorithm (Knuth-Morris-Pratt, 1977), whose running time is linear, i.e. $O(n+m)$. The details can be found in the book mentioned above.*

4 Solution using deterministic finite automaton

A natural tool is using finite automata (state machines). In the preprocessing phase a DFA is created based on the pattern, and using that matchings with the text can be determined in $O(n)$ running time. Construction of the automaton and its storage costs $O(m|\Sigma|)$. An example of this occurred in Digital technologies, and a version in Programming. However, we'd better define Deterministic Finite automata first...