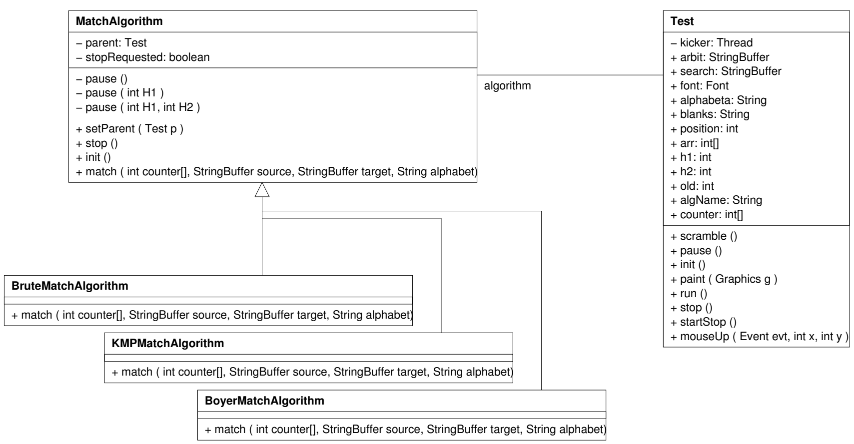
Recap

- String matching problem.
- Brute force solution
- Good points / Bad points about it.
- Better solution? Try using automata.

Today:

- Brute-force implementation.
- How do we use automata? String-matching automaton.
- Examples + analysis.

Brute force Implementation




```
class BruteMatchAlgorithm extends MatchAlgorithm{

    void match(int counter[],         // ??
               StringBuffer T,        // text
               StringBuffer P,        // pattern
               String alphabet        // not used by BruteMatch
               ) throws Exception{

        counter[0] = 0;
        int n = T.length();  // length of the input text
        int m = P.length();  // length of the pattern

        // finding all the matchings of P in T
        for ( int s = 0 ; s <= n - m ; s++ ) {
            if(stopRequested)
                return;
            counter[1] = 0;
            // checking equality of P and T[s + 0..s + m - 1]
            for ( int j = 0 ; j < m ; j++ )
                if ((T.toString()).charAt(s + j) != (P.toString()).charAt(j))
                    counter[1] = 1;
            counter[0] = (s*6)\%900;
            pause(1,1);
            counter[1]=0;
        }
    }
}
```


String-matching automaton, Example

Let's consider again the pattern $P \equiv \mathsf{abc}$, and let's define the string-matching automaton in this case (for simplicity let's assume the alphabet $\mathcal{S} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$).

- $Q = \{0, 1, 2, 3\}$, the automaton will have four states.
- The initial state is (always) $q_0 = 0$.
- The only accepting state is (always) m.

Transition function

The only step that needs some care is the definition of the transition function. We can represent it as a table with rows indexed by characters and columns indexed by states.

Let's start with $\delta(0,\mathsf{a})$. By definition of σ_P this is "the length of the longest prefix of P that is a suffix of $P_0\mathsf{a}(\equiv \mathsf{a})$". Therefore $\delta(0,\mathsf{a}) = 1$. Next comes $\delta(1,\mathsf{a})$. This is "the length of the longest prefix of P that is a suffix of $P_1\mathsf{a}(\equiv \mathsf{aa})$". Again $\delta(1,\mathsf{a}) = 1$. Iterating this process one can get δ's full definition (reported in the following table).

	0	1	2	3
a	1	1	1	1
b	0	2	0	0
c	0	0	3	0

Simulation

Let $T =$ aababcabcbb.

T	a	a	b	a	b	c	a	b	c	b	b	
q	0	1	1	2	1	2	3	1	2	3	0	0
output							3			6		

Time complexity analysis ... cheating!

- The simple loop structure of FINITE-AUTOMATON-MATCHING implies that its running time is $O(|T|)$.
- However, this does not include the time to compute the transition function δ: we will look at this later!
- Correctness? Not easy, brace yourself!

  Let's start by understanding what correctness means.

Main Result

> For each $i \leq n$, the value of q after the ith iteration of the main for loop in FINITE-AUTOMATON-MATCHING is $\sigma_P(T_i)$, i.e. the length of the longest prefix of the pattern P that is a suffix of T_i.

By definition of σ_P, $\sigma_P(T_i) = m$ if and only if P is a suffix of T_i, i.e. a matching has just occurred, therefore the result "says" that the process returns all the valid shifts of the given pattern.

Suffix-function inequality

$\sigma(xa) \leq \sigma(x) + 1$, for any string x and character a.

Case 1. If $\sigma(xa) = 0$, then the result trivially holds, because σ is a positive function.

Case 2. Otherwise,

$P_{\sigma(xa)}$ is a suffix of xa, by definition of σ.

Furthermore $P_{\sigma(xa)-1}$ must be a suffix of x (we just drop the end of both strings).

But then $\sigma(x)$ is the largest k such that P_k is a suffix of x, then $\sigma(xa) - 1$ must be at most $\sigma(x)$.

Suffix-function recursion lemma

$\sigma(xa) = \sigma(P_{\sigma(x)}a)$, for any string x and character a.

To prove this one shows that

$$\sigma(xa) \leq \sigma(P_{\sigma(x)}a) \text{ and } \sigma(xa) \geq \sigma(P_{\sigma(x)}a).$$

1. $P_{\sigma(x)}$ is a suffix of x (by definition of σ).
2. $P_{\sigma(x)}a$ is a suffix of xa (just add the same character to both strings),
3. ... and, obviously, so is $P_{\sigma(xa)}$!
4. We thus have two suffixes of xa,
$$P_{\sigma(xa)} \text{ and } P_{\sigma(x)}a$$
and, by the previous Lemma $\sigma(xa) \leq \sigma(x) + 1 = |P_{\sigma(x)}a|$. Hence it must be that $P_{\sigma(xa)}$ is a suffix of $P_{\sigma(x)}a$.
5. Therefore $\sigma(xa) \leq \sigma(P_{\sigma(x)}a)$.

The opposite inequality is proved similarly (see Cormen page 920).

Proof of Main Result

By induction on i. If $i = 0$, then $T_0 = \varepsilon$ and the theorem holds. Else we assume $\sigma(T_i)$ is the value of q after the ith iteration and prove that q will be set to $\sigma(T_{i+1})$ the next time around.

To simplify notations let $q = \sigma(T_i)$ and $a = T[i+1]$.

The next value of q will be $\delta(q, a)$ (just look at the code!).

By definition of δ the value above is equal to $\sigma(P_q a)$, which is $\sigma(P_{\sigma(T_i)}a)$ by definition of q.

Now we use the recursion lemma,

$$\sigma(P_{\sigma(T_i)}a) = \sigma(T_i a)$$

and we are done (!) since $T_i a = T_{i+1}$.

Computing the transition function

The following procedure computes the transition function δ from a given pattern P and alphabet $\mathcal{A}$.

COMPUTE-TRANSITION-FUNCTION (P, $\mathcal{A}$)
$m \leftarrow$ length(P)
for $q \leftarrow 0$ **to** m
for each $a \in \mathcal{A}$
$k \leftarrow \min(m+1, q+2)$
repeat $k \leftarrow k - 1$ **until** P_k is a suffix of $P_q a$
$\delta(q, a) \leftarrow k$

The running time is $O(m^3|\Sigma|)$ ... why?

Complexity improvable to $\Theta(m|\Sigma|)$, which is best possible.

The picture so far

- Defined the String Matching problem.
- Defined, implemented and seen examples of the brute-force algorithm. Time complexity $\Theta((n-m)m)$.
- Defined and seen examples of an alternative approach based on automata theory. Time complexity $O(n + m|\Sigma|)$.

So? Can we actually solve pattern matching in linear time?