Analysis of Algorithms

String Matching

Basic String/Document Processing Algorithms

- String-driven information retrieval is at the core of many important computer applications, including:
  - Web searching and “surfing”: the Internet document formats HTML and XML are primarily text formats with added links to multimedia context
  - Searching for a certain DNA sequence in a genomic database, or searching for particular patterns in DNA sequences
- Document processing is one of the dominant functions of computers
  - editing, searching, transporting over the Internet, displaying, etc.

Text Documents

- From the perspective of algorithm design, documents can be viewed as simple **character strings**, that is, they can be abstracted as a sequence of characters.
- At the heart of algorithms for searching and processing text are methods for dealing with character strings
- *A* = “CGTAAACTGCTTTAATCAAACGC” DNA sequence
- *B* = “http://www.pucit.edu.pk”

The String Matching Problem

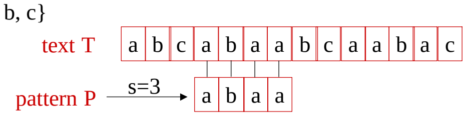
- Finding all occurrences of a pattern in a text
- We assume that
  - the text is an array *T*[1..*n*] of length *n* and the pattern is an array *P*[1..*m*] of length *m*.
  - the elements of *T* and *P* are characters drawn from a finite alphabet Σ
- Ex.: Σ = {0,1} or Σ = {a,b,…,z}
- The character arrays *P* and *T* are loosely called “strings of characters”

The String Matching Problem

Given: Two strings T[1..n] and P[1..m] over alphabet Σ.

Want to find all occurrences of P[1..m] “the pattern” in T[1..n] “the text”.

Example: Σ = {a, b, c}



Terminology:

- P occurs with shift s.
- P occurs beginning at position s+1.
- s is a valid shift.

Goal: Find all valid shifts with which a given pattern P occurs in a text T.

Notation and Terminology

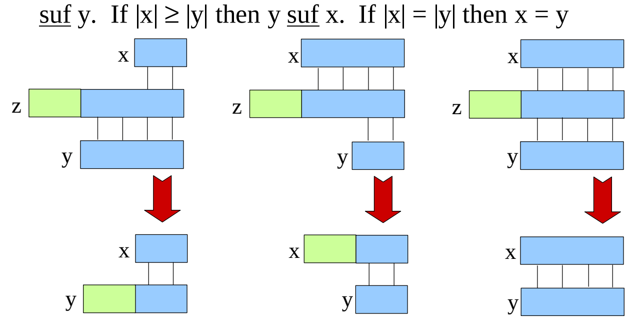
- An Alphabet Σ is a finite set of symbols
- Σ^* denotes the set of all finite-length strings over Σ.
  - The empty string is denoted ε
- The length of a string *x* is denoted |*x*|
- The concatenation of two strings *x* and *y* is denoted *xy*, and has length |*x*| +|*y*|
- We say that string *x* is a **prefix** of string *y*, denoted *x Ì y*, if *y* = *xw* for some string *w Î* Σ^*.

Notation and Terminology

- Ex: vzk Ì vzkavk, vzk Ì vzk
- We say that string *x* is a **sufix** of string *y*, denoted *x É y*, if *y* = *wx* for some string *w Î* Σ^*.
- avk *É* vzkavk, avk *É* avk

Lemma 32.1

Lemma 32.1: Suppose x suf z and y suf z. If |x| ≤ |y| then x suf y. If |x| ≥ |y| then y suf x. If |x| = |y| then x = y



Naïve Brute-Force Algorithm

```
Naïve(T, P)
    n := length[T];
    m := length[P];
    for s := 0 to n – m do
        if P[1..m] = T[s+1..s+m] then
            print “pattern occurs with shift s”
```


Example

| a | c | a | a | b | c |

s = 0 | a | a | b |

Example

| a | c | a | a | b | c |

s = 0 | a | a | b |

Example

| a | c | a | a | b | c |

s = 0 | a | a | b |

Example

| a | c | a | a | b | c |

s = 1 → | a | a | b |

Example

| a | c | a | a | b | c |

s = 1 → | a | a | b |

Example

| a | c | a | a | b | c |

s = 2 → | a | a | b |

Example

| a | c | a | a | b | c |

s = 2 → | a | a | b |

Example

a c a a b c

s = 2 → a a b

Example

a c a a b c

s = 2 → a a b

match!

Example

a c a a b c

s = 3 → a a b

Example

a c a a b c

s = 3 → a a b

Example

a c a a b c

s = 3 → a a b

Running time is Θ((n – m + 1)m).

String Matching with Finite Automata

- Many string matching algorithms build a finite automaton that scans *T* for all occurrences of *P*
- String matching automata are very efficient, since they examine each character only once
- Running time (excluding the time to build the automaton) is $\Theta(n)$
- The time to build the automaton can however be large, if Σ is large…

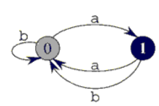
Finite Automata

- A finite automaton *M* is a 5-tuple (Q, $q0$, A, Σ, δ) where
- Q is a finite set of ***states***
- $q_0 \in Q$ is the ***start state***
- $A \subseteq Q$ is a distinguished set of ***accepting states***
- Σ is a finite ***input alphabet***
- δ is a function from $Q \times \Sigma$ into Q, called the ***transition function*** of *M*.

Example

state	input a	input b
0	1	0
1	0	0

(a)



(b)

Figure 32.6 A simple two-state finite automaton w … = 0, and
input alphabet $\Sigma = \{a, b\}$. **(a)** A tabular representatio … quivalent
state-transition diagram. State 1 is the only accept … ed edges
represent transitions. For example, the edge from sta … b) = 0.
This automaton accepts those strings that end in an … string x
is accepted if and only if $x = yz$, where $y = \varepsilon$ … here k is
odd. For example, the sequence of states this automa … the start
state) is ⟨0, 1, 0, 1, 0, 1⟩, and so it accepts this inpu … states is
⟨0, 1, 0, 0, 1, 0⟩, and so it rejects this input.

$Q = \{0,1\}$

$q_0 = 0$

$\Sigma = \{a, b\}$

$A = \{1\}$

Final State Function

- The FA begins with state q_0 and reads the characters of the input string one at a time.
- If the FA is in state q and reads input character c it makes a transition from state q to sate $\delta(q, c)$.
- When the current state $q \in A$, the machine M is said to have **accepted** the string read so far. (An input that is not accepted is said to be **rejected**.)
- function ϕ, called the **final state function** from Σ^* to Q such that $\phi(w)$ is the state M ends up after scanning the string w., FA ends up in after scanning the string w.
- Thus M accepts a string if and only if $\phi(w) \in A$.

Final State Function

- **Final state function** $\phi(w)$ - recursive definition
- $\phi(\varepsilon) = q_0$
- $\phi(wc) = \delta(\phi(w), c)$ for $w \in \Sigma^*$, $c \in \Sigma$

Suffix Function

- A string-matching automaton is constructed for pattern P, and then used to search the text string T.
- First, an auxiliary **suffix function** σ is defined for $P[1..m]$.
  - The suffix function is a mapping from Σ^* to $\{0, 1, \ldots, m\}$ such that $\sigma(x)$ is the length of the longest prefix of P that is a suffix of x.
  - $\sigma(x) = \max\ \{k : P_k \supset x\}\ (x \Rightarrow T_i\ )$

Example

- P= ab $\Rightarrow \sigma(\varepsilon) = 0$
- P= a**b** $\Rightarrow \sigma$(ccac**a**) =1
- P= **ab** $\Rightarrow \sigma$(cc**ab**) = 2

- Empty string $P_0 = \varepsilon$ is a suffix of every string.

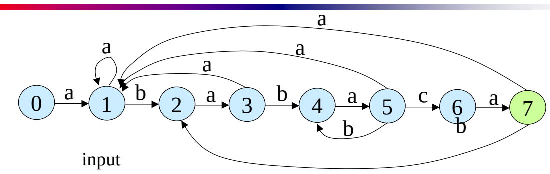
String Matching Automata

- The string-matching automaton constructed for pattern $P[1..m]$ is defined as follows:
  - The state set Q is $\{0, 1, \ldots, m\}$. The start state $q0$ is state 0 and state m is the only accepting state.
- The transition function δ, for any state q and character c, is defined by

$$\delta(q, c) = \sigma(P_q\ c)$$

  - That is, in order to compute the length of the longest suffix of $Ti\ c$ that is a prefix of P, we can compute the longest suffix of $P_q\ c$ that is a prefix of P.

Finite Automata Algorithm



input

state	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

i	--	1	2	3	4	5	6	7	8	9	10	11
T[i]	--	a	b	**a**	**b**	**a**	**b**	**a**	**c**	**a**	b	a
state ϕ(i)	0	1	2	3	4	5	4	5	6	**7**	2	3

Processing time takes Θ(n).
But have to first construct FA.
Main Issue: How to construct FA?

Finite-Automaton-Matcher

Finite-Automaton(T,δ,m)

$n \leftarrow$ *length*[T]

$q \leftarrow 0$

for $i \leftarrow 1$ **to** n

do $q \leftarrow \delta(q, T[i\ ])$

if $q = m$

then print "Pattern occurs with shift" s

- Running time is O(n)