String Matching

CLRS Chapter 32

- Definition of string matching
- Naive string-matching algorithm
- String matching algorithms – an overview
- Rabin-Karp algorithm
- Finite automata
- Linear time matching using finite automata
- (Knuth-Morris-Pratt algorithm)

Martin Zachariasen, DIKU

May 18, 2009

String-matching problem

Given:

- Text $T[1\,..\,n]$
- Pattern $P[1\,..\,m]$, where $m \leq n$

Characters of text and pattern are drawn from a common finite alphabet Σ: $T \in \Sigma^*$ and $P \in \Sigma^*$.

Find:
All occurrences of pattern P in T, that is, all valid shifts s, where $0 \leq s \leq n - m$, such that

$$T[s+1\,..\,s+m] = P[1\,..\,m]$$

or

$$T[s+j] = P[j], \quad j = 1, \,..\,, m$$

Naive string-matching algorithm

Iterate through all shifts s, and for each of these check if the shift is valid: $T[s+j] = P[j],\ \ j = 1, .., m$.

Clearly takes time $\Theta((n-m+1)m)$, or $\Theta(n^2)$ if $m = \lfloor n/2 \rfloor$.

More clever algorithms use information obtained when checking one value of s in the following iteration(s).

String-matching algorithms — an overview

Divide running time into preprocessing and matching time.

Preprocessing: Setup some data structure based on pattern P.

Matching: Perform actual matching by comparing characters from T with P and precomputed data structure.

String-matching algorithms considered:

Algorithm	Preprocessing time	Matching time
Naive	0	$\Theta((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$\Theta((n-m+1)m)$
Finite automation	$O(m\lvert\Sigma\rvert)$	$\Theta(n)$
(Knuth-Morris-Pratt)	$\Theta(m)$	$\Theta(n)$

Note: Rabin-Karp uses $O(n)$ expected matching time.

Rabin-Karp algorithm

Consider (sub)strings as numbers. Characters in a string correspond to digits in a number written in radix-d notation (where $d = |\Sigma|$).

Numerical value p corresponding to pattern $P[1\,..\,m]$:

$$p = P[1]d^{m-1} + P[2]d^{m-2} + \ldots + P[m-1]d + P[m]$$

or by using Horner's rule:

$$p = P[m]+d(P[m-1]+d(P[m-2]+\ldots+d(P[2]+dP[1])\ldots))$$

Let t_s correspond to the decimal value of $T[s+1\,..\,s+m]$.

Main observation: Valid shift s is obtained if and only if $p = t_s$.

Fast computation of text string numbers

Assume that we have computed $t_0, \ldots, t_s$.

Question: How can we compute t_{s+1} efficiently?

Answer: Just need to drop the most significant digit from t_s and append the least significant digit from t_{s+1}.

Let $h = d^{m-1}$. Then we have:

$$t_{s+1} = d(t_s - T[s+1]h) + T[s+m+1]$$

Thus given t_s we can compute t_{s+1} in constant time — assuming that arithmetic operations take constant time.

Reducing the size of decimal numbers

Problem: Numbers p and t_s cannot be computed or compared in constant time!

Solution: Compute all numbers modulo some (small) number q.

Basic facts on modulus computations:

- Remainder/residue of a division: The number

  $$r = a \bmod q$$

  is the remainder of the integer division a/q, or the unique number $0 \leq r < q$ such that $a = kq + r$ (where k is the result of the integer division).

- Equivalence classes modulo q: For two integers a and b we have that

  $$a \equiv b \pmod{q}$$

  if and only if there exists some number k such that

  $$a - b = kq$$

Properties of modified algorithm

New main observation: When

$$p \equiv t_s \pmod{q}$$

then we either have a valid shift s or a so-called spurious hit.

Need to check every such hit explicitly. Takes $O(m)$ time for each hit.

The expected number of spurious hits is $O(n/q)$. If v is the number of valid shifts, the expected matching time is

$$O(n) + O(m(v + n/q))$$

which is $O(n)$ if v is a constant and $q \geq m$.

Finite automata

We may build a finite automaton that recognizes pattern P. More precisely, the automaton should recognize all strings x such that P is a suffix of x: $P \sqsupset x$.

Why? A shift s is valid if and only if $P \sqsupset T[1 \mathinner{.\,.} m+s]$.

Note that the automaton is built in the preprocessing phase and uses only the pattern P as input.

Some notation on finite automata:

- Q is the set of states (where $q_0 \in Q$ is the start state),
- $A \subseteq Q$ is the set of accepting states,
- δ is the transition function,
- ϕ is the (implicit) final-state function: $\phi(x)$ is the state that the automation ends in after scanning string x.

Building a string-matching automation

Need to define the so-called suffix function σ which maps any string $x \in \Sigma^*$ to the set $\{0, 1, \ldots, m\}$ according to

$$\sigma(x) = \max\{k : P_k \sqsupset x\}$$

where P_k is the prefix $P[1..k]$.

The finite automation will have the following properties:

- Set of states $Q = \{0, 1, \ldots, m\}$, start state $q_0 = 0$, and only accepting state $A = \{m\}$.
- Transition function: $\delta(q, a) = \sigma(P_q a)$
- Invariant maintained while reading the text T is

  $$\phi(T_i) = \sigma(T_i)$$

  where T_i is the prefix $T[1..i]$. The state number should be equal to the length of the longest prefix of P that is a suffix of T_i.

Correctness of finite automation

Need to prove that the machine is in state $\sigma(T_i)$ after scanning character $T[i]$.

Proceed in two steps:

1. (Lemma 32.3) For any string T_i and character a, if $q = \sigma(T_i)$, then
$$\sigma(T_i a) = \sigma(P_q a)$$

2. (Theorem 32.4) For all $i = 0, 1, \ldots, n$, we have
$$\phi(T_i) = \sigma(T_i)$$

Proof of property 1

$$\boxed{\begin{array}{c}\text{For any string } T_i \text{ and character } a\text{, if } q = \sigma(T_i)\text{, then} \\ \sigma(T_i a) = \sigma(P_q a)\end{array}}$$

We cannot have $\sigma(T_i a) > q + 1$ since this would imply that $\sigma(T_i) > q$.

However, if $P_{q+1} \sqsupset T_i a$ then $\sigma(T_i a) = q + 1$.

Since $q = \sigma(T_i)$ we have $P_q \sqsupset T_i$. Thus computing $\sigma(T_i a)$ is the same as computing $\sigma(P_q a)$ since $\sigma(T_i a) \leq q + 1$.

Proof of property 2

For all $i = 0, 1, \ldots, n$, we have

$$\phi(T_i) = \sigma(T_i)$$

Proof by induction on i.

Basis: $\phi(T_0) = 0 = \sigma(T_0)$, since T_0 is the empty string.

Inductive step: Assume that $\phi(T_i) = \sigma(T_i)$. Would like to prove that $\phi(T_{i+1}) = \sigma(T_{i+1})$.

Let $\phi(T_i) = q$. By induction we have $\sigma(T_i) = q$, and hence by property 1 that $\sigma(T_i a) = \sigma(P_q a)$ for any character a.

Let $a = T[i+1]$. Now we have

$$\begin{aligned}
\phi(T_{i+1}) &= \phi(T_i a) && \text{[by the definition of } a\text{]} \\
&= \delta(\phi(T_i), a) && \text{[by the definition of } \phi\text{]} \\
&= \delta(q, a) && \text{[by the definition of } q\text{]} \\
&= \sigma(P_q a) && \text{[by the definition of } \delta\text{]} \\
&= \sigma(T_i a) && \text{[as argued above]} \\
&= \sigma(T_{i+1}) && \text{[by the definition of } T_{i+1}\text{]}
\end{aligned}$$

Computing the transition function

May use a straight-forward $O(m^3|\Sigma|)$ time algorithm:

For each state $q \in Q$ and character $a \in \Sigma$ find the maximum $k \in \{0, 1, \ldots, m\}$ such that

$$P_k \sqsupset P_q a$$

The result is defined to be the value of $\delta(q, a)$.

There are $m + 1$ states, $|\Sigma|$ characters, at most $m + 1$ possible values of k and at most m characters to check for the condition $P_k \sqsupset P_q a$.

Possible to devise an algorithm that runs in $O(m|\Sigma|)$ time.

(Knuth-Morris-Pratt algorithm)

Similar to finite automation, but avoids explicit computation of $\delta(q, a)$.

Only needs one auxiliary function $\pi[1 \,..\, m]$ that can be computed from P in $\Theta(m)$ time:

$$\pi[q] = \max\{k : k < q \text{ and } P_k \sqsupset P_q\}$$

We compute $\delta(q, a)$ iteratively by using the function π.

The amortized cost is $\Theta(m)$ for preprocessing and $\Theta(n)$ for matching.