طراحی الگوریتم

۱۹ آذر ۹۸

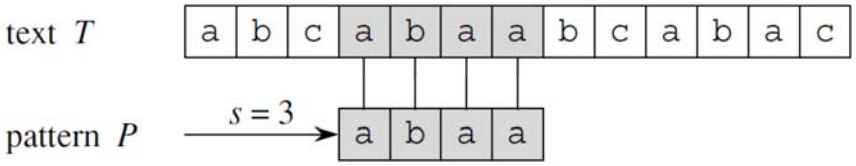
ملکی مجد

Topic	Reference
Recursion and Backtracking	Ch.1 and Ch.2 JeffE
Dynamic Programming	Ch.3 JeffE and Ch.15 CLRS
Greedy Algorithms	Ch.4 JeffE and Ch.16 CLRS
Amortized Analysis	Ch.17 CLRS
Elementary Graph algorithms	Ch.6 JeffE and Ch.22 CLRS
Minimum Spanning Trees	Ch.7 JeffE and Ch.23 CLRS
Single-Source Shortest Paths	Ch.8 JeffE and Ch.24 CLRS
All-Pairs Shortest Paths	Ch.9 JeffE and Ch.25 CLRS
Maximum Flow	Ch.10 JeffE and Ch.26 CLRS
String Matching	Ch.32 CLRS
NP-Completeness	Ch.34 CLRS

String matching – تطابق رشته ها

- Finding all occurrences of a pattern in a text
  - Application: text editing, DNA pattern

Formal definition of string marching



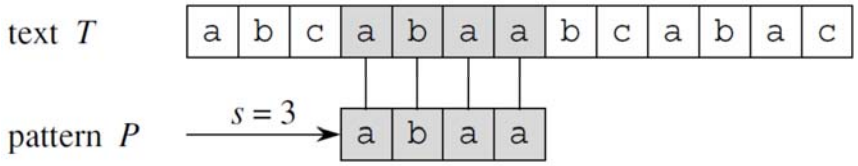
- The text is an array $\boldsymbol{T}\,[\boldsymbol{1}\,..\boldsymbol{n}]$ of length n
- That the pattern is an array $\boldsymbol{P}[\boldsymbol{1}\,..\boldsymbol{m}]$ of length $m \leq n$
- Elements of P and T are characters drawn from a finite alphabet Σ
  - For example $\Sigma = \{0,1\}$ or $\Sigma = \{a, b, \ldots, z\}$.
  - Character arrays P and T are often called ***strings*** of characters

Pattern *P* ***occurs with shift s*** in text *T:*(*P* ***occurs beginning at position s*** **+ 1** in *T)*

If $0 \leq s \leq n - m$ and $T\,[s + 1\,..\,s + m] = P[1\,..\,m]$

(that is, if $T\,[s + j\,] = P[\,j\,]$, for $1 \leq j \leq m$).

Formal definition of string marching (2)



- If P occurs with shift s in T, we call s a ***valid shift***
  - Otherwise, we call s an ***invalid shift***.

- The string-matching problem is

  the **problem of finding all valid shifts** with which a given pattern P occurs in a given text T

Some string-matching algorithms:

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

The preprocessing time is based on the pattern

This course covers the algorithms of Naïve, Radin-Karp and Finite automaton

Notation and terminology

- Σ^*: the set of all finite-length strings formed using characters from the alphabet Σ
- ε **:** zero-length ***empty string***
- $|x|$ **:** length of a string x
- xy *:* ***Concatenation*** of two strings *x* and *y*
- $w \sqsubset x$ **:** w is a ***prefix*** of a string x *(*$x = wy$ for some string $y \in \Sigma^*$*.)*
  - ab ⊏ abcca
- $w \sqsupset x$ **:** w is a ***suffix*** of a string x *(*$x = yw$ for some string $y \in \Sigma^*$*.)*
  - cca ⊐ abcca

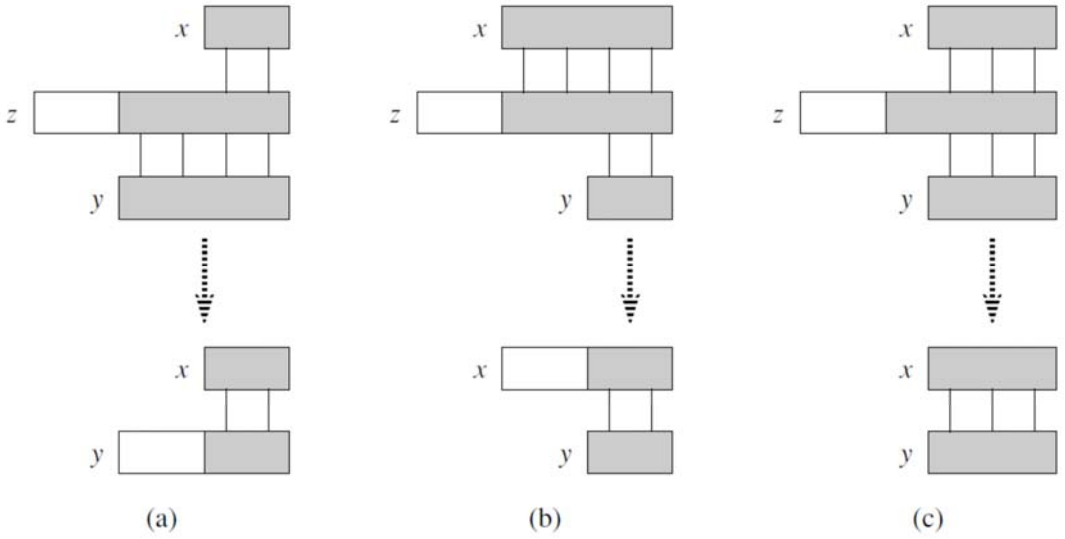
The empty string ε is both a suffix and a prefix of every string.

x ⊐ *y* if and only if *xa* ⊐ *ya*.

⊏ and ⊐ are transitive relations

Overlapping-suffix lemma

Suppose that x, y, and z are strings such that $x \sqsupset z$ and $y \sqsupset z$. If $|x| \leq |y|$, then $x \sqsupset y$. If $|x| \geq |y|$, then $y \sqsupset x$. If $|x| = |y|$, then $x = y$.



Another notation

The **string-matching problem** :

Finding all shifts *s* in the range $0 \leq s \leq n - m$ such that $P \sqsupset T_{s+m}$.

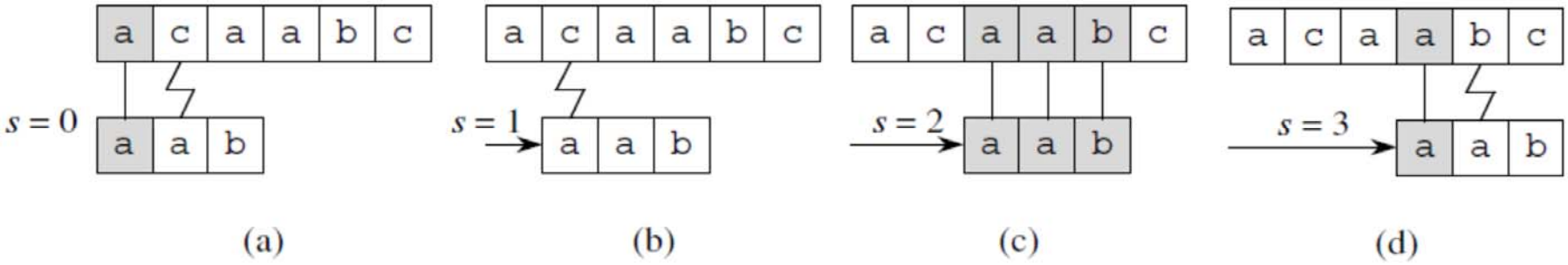
- denote the *k*-character prefix $P[1 \,..\, k]$ of the pattern $P[1 \,..\, m]$ by P_k

The naive string-matching algorithm

```
NAIVE-STRING-MATCHER(T, P)
1  n ← length[T]
2  m ← length[P]
3  for s ← 0 to n − m
4       do if P[1..m] = T[s + 1..s + m]
5            then print "Pattern occurs with shift" s
```

takes time $O((n - m + 1)m)$, and this bound is tight in the worst case.

Example : text a^n and pattern a^m



The operation of the naive string matcher for the pattern $P = aab$ and the text $T = acaabc$. We can imagine the pattern P as a *template* that we slide next to the text.

Think together

Suppose that all characters in the pattern P are different.

Show how to accelerate NAIVE-STRING-MATCHER to run in time $O(n)$ on an n-character text T .

The Rabin-Karp algorithm

- The Rabin-Karp algorithm uses $\Theta(m)$ preprocessing time,
- Its worst-case running time is $\Theta((n-m+1)m)$.
- Its average-case running time is better

- **use of elementary number-theoretic notions**
  - E.g., equivalence of two numbers modulo a third number

Assumption

- For now, let assume

$$\Sigma = \{0, 1, 2, \ldots, 9\},$$

(each character is a decimal digit.)

- We can view a string of *k* consecutive characters as representing a **length-*k* decimal number**

- let *p* denote its corresponding decimal value of the pattern *P*[1 . .*m*]
- let t_s denote its corresponding decimal value of substring *T* [*s* + 1 . . *s* + *m*],
- ***s* is a valid shift if and only if $t_s = p$**

compute p *(and similarly* t_0*)* in time $\Theta(m)$ using Horner's rule:

$p = P[m] + 10\,(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1])\cdots))$

compute $\boldsymbol{p}$ ***and*** $\boldsymbol{t_s}$ in time $\boldsymbol{\Theta(m)}$

and **all the** $\boldsymbol{t_s}$ values in a total of $\boldsymbol{\Theta(n-m+1)}$ time

t_{s+1} can be computed from t_s in constant time

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]$$

we could determine all valid shifts s in time $\Theta(m) + \Theta(n-m+1) = \boldsymbol{\Theta(n)}$
by comparing p with each of the t_s 's (if the numbers are not very large!)

p and t_s may be too large

- a simple cure for this problem :

  Compute p and the t_s 's modulo a suitable modulus q.

we can compute p modulo q in $\Theta(m)$ time and all the t_s 's modulo q in $\Theta(n - m + 1)$ time

- The modulus q is typically chosen as a prime such that $10q$ just fits within one computer word, which allows all the necessary computations to be performed with single-precision arithmetic.

solution of working modulo q

- We can thus use the test $t_s \equiv p \ (mod \ q)$ as a fast heuristic test to rule out invalid shifts *s*.

- Any shift *s* for which $t_s \equiv p \ (mod \ q)$ must be tested further to see if *s* is really valid or we just have a ***spurious hit***.

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n ← length[T]
 2  m ← length[P]
 3  h ← d^(m−1) mod q
 4  p ← 0
 5  t_0 ← 0
 6  for i ← 1 to m                          ▷ Preprocessing.
 7       do p ← (dp + P[i]) mod q
 8          t_0 ← (dt_0 + T[i]) mod q
 9  for s ← 0 to n − m                      ▷ Matching.
10       do if p = t_s
11             then if P[1..m] = T[s + 1..s + m]
12                  then print "Pattern occurs with shift" s
13          if s < n − m
14             then t_{s+1} ← (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

Time complexity?

Rabin Karp in practice:

- takes $\Theta(m)$ preprocessing time, and

  its matching time is $\Theta((n - m + 1)m)$ in the worst case

- In many applications, we expect few valid shifts (perhaps some constant *c* of them); in these applications, the expected matching time of the algorithm is only

  $O((n - m + 1) + cm) = O(n + m)$, plus the time required to process spurious hits.

Hashing instead of module q

- a random mapping from $\Sigma^*$to $\boldsymbol{Z_q}$
- We can then expect that the number of spurious hits is $O(n/q)$
- the expected matching time taken by the Rabin-Karp algorithm is

  O(n) + O(m(v + n/q)) , (v is the number of valid shifts)
- expected matching time is $O(n)$ if $v = O(1)$ *and* we choose $q \geq m$

Think together

Show how to extend the Rabin-Karp method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters. (The pattern may be shifted vertically and horizontally, but it may not be rotated.)