International Islamic University Chittagong
Department of Computer Science & Engineering
Autumn - 2022

Course Code: CSE-2321
Course Title: Data Structures

Mohammed Shamsul Alam
Professor, Dept. of CSE, IIUC

Lecture – 6

String

String

- A finite sequence S of zero or more characters is called a **String**.
- The string with zero character is called the **empty string** or **null string**.
- The number of characters in a string is called its **length**.
- Specific string will be denoted by enclosing their character in **single quotation mark**. For example:

  ‘THE END’ ‘TO BE OR NOT TO BE’ ‘’ ‘123‘

  are strings with lengths 7, 18, 0 and 3.
- Let S1 and S2 be the strings. The string consisting of the character of S1 followed by the characters of S2 is called the **concatenation** of S1 and S2 . It will be denoted by S1 || S2 . For example,

  S1 = ‘THE’ S2 = ‘END then

  S1 || S2 = ‘THEEND’

  S1 || ‘□‘ || S2 = ‘THE END’ Here ‘□‘ means blank space
- The length of S1 || S2 is equal to the sum of lengths of the strings S1 and S2.

String

- A string Y is called a **substring** of a string S if there exits strings X and Z such that S = X || Y || Z.
- If X is an empty string, then Y is called *initial substring* of S, and if Z is an empty string then Y is called *terminal substring* of S.
- For example

  ‘BE OR NOT’ is a substring of ‘TO BE OR NOT TO BE’

  ‘THE’ is an initial substring of ‘THE END’

  ‘END’ is a terminal substring of ‘THE END’
- If Y is a substring of S then the length of Y cannot exceed the length of S.

String Operations

Length

The number of characters in a string is called its **length**. We will write

LENGTH (string)

for the length of a given string. For example,

LENGTH (‘Computer’) = 8

LENGTH (‘0’) = 1

Concatenation

Let S1 and S2 be strings. The concatenation of S1 and S2 is denoted by **S1 || S2** , is the string consisting of the characters of S1 followed by the characters of S2 . For example,

Suppose S1 = ‘Kazi‘ and S2 = ‘Nazrul’ then

S1 || S2 = ‘KaziNazrul’

S1 || ‘□‘ || S2 = ‘Kazi Nazrul’

String Operations

Substring

Accessing a substring from a given string requires 3 pieces of information.

i) The name of the string or the string itself

ii) The position of the first character of the substring in the given string

iii) The length of the substring or position of the last character of the substring

We call this operation SUBSTRING. Specifically, we write

SUBSTRING (String, initial, length)

to denote the substring of a string S beginning in a position K and having a length L or **SUBSTRING (S, K, L)**.

For example,

SUBSTRING (‘TO BE OR NOT TO BE‘, 4, 7) = ‘BE OR N’

SUBSTRING (‘THE END’, 4, 4) = ‘□END’

String Operations

Indexing

Indexing also called *pattern matching,* refers to finding the position where a string pattern P first appears in a given string text T.

We call this operation INDEX and write as

INDEX (text, pattern)

- If the pattern **P** does not appear in text **T** then INDEX is assigned the value 0.
- The arguments text and pattern can be either string constants or string variables.

For example,

T = ‘HIS FATHER IS THE PROFESSOR‘

Then

INDEX (T, ‘THE‘) = 7

INDEX (T,’THEN‘) = 0

INDEX (T, ‘□THE‘) = 14

Word Processing

Insertion

Suppose in a given text T we want to insert a string S so that S begins in position K. We denote this operation by

INSERT(text, position, string)

For example,

INSERT ('ABCDEFG', 3, 'XYZ') = 'ABXYZCDEFG'
INSERT ('ABCDEFG', 6, 'XYZ') = 'ABCDEXYZFG'

Deletion

Suppose in a given text T we want to delete the substring which begins in position K and has length L. We denote this operation by

DELETE(text, position, length)

For example,

DELETE(' ABCDEFG ', 4, 2) = ' ABCFG '
DELETE(' ABCDEFG ', 2, 4) = ' AFG '

We assume that nothing is deleted if position K = 0. Thus

DELETE(' ABCDEFG ', 0, 2) = ' ABCDEFG '

Word Processing

(a) Suppose Algorithm 3.1 is run with the data

T = XABYABZ, P = AB

Then the loop in the algorithm will be executed twice. During the first execution, the first occurrence of AB in T is deleted, with the result that T = XYABZ. During the second execution, the remaining occurrence of AB in T is deleted, so that T = XYZ. Accordingly, XYZ is the output.

(b) Suppose Algorithm 3.1 is run with the data

T = XAAABBBY, P = AB

Observe that the pattern AB occurs only once in T but the loop in the algorithm will be executed three times. Specifically, after AB is deleted the first time from T we have T = XAABBY, and hence AB appears again in T. After AB is deleted a second time from T, we see that T = XABY and AB still occurs in T. Finally, after AB is deleted a third time from T, we have T = XY and AB does not appear in T, and thus INDEX(T, P) = 0. Hence XY is the output.

Word Processing

Replacement

Suppose in a given text T we want to replace the first occurrence of a pattern P_1 by a pattern P_2. We will denote this operation by

$$\text{REPLACE}(\text{text}, \text{pattern}_1, \text{pattern}_2)$$

For example

REPLACE('XABYABZ', 'AB , 'C') = 'XCYABZ'
REPLACE('XABYABZ' , 'BA', 'C') = 'XABYABZ'

In the second case, the pattern BA does not occur, and hence there is no change.

Replacement [Every occurrence of the pattern P in T by Q]

T = XABYABZ,   P = AB,   Q = C

Then the loop in the algorithm will be executed twice. During the first execution, the first occurrence of AB in T is replaced by C to yield T = XCYABZ. During the second execution, the remaining AB in T is replaced by C to yield T = XCYCZ. Hence XCYCZ is the output.

Word Processing

Replacement **[Every occurrence of the pattern P in T by Q]**

(b) Suppose Algorithm 3.2 is run with the data

$$T = XAY, \quad P = A, \quad Q = AB$$

Then the algorithm will never terminate. The reason for this is that P will always occur in the text T, no matter how many times the loop is executed. Specifically,

$T = XABY$ at the end of the first execution of the loop
$T = XAB^2Y$ at the end of the second execution of the loop
....
$T = XAB^nY$ at the end of the nth execution of the loop
(The infinite loop arises here since P is a substring of Q.)

NB:

suppose the length of Q is smaller than the length of P. Then the length of T after each replacement decreases. This guarantees that in this special case where Q is smaller than P the algorithm must terminate.

Pattern Matching Algorithm

Pattern matching is the problem of deciding whether or not a given string pattern P appears in a string text T. We assume that the length of P does not exceed the length of T. This section discusses two pattern matching algorithms. We also discuss the complexity of the algorithms so we can compare their efficiencies.

Remark: During the discussion of pattern matching algorithms, characters are sometimes denoted by lowercase letters (a, b, c, ...) and exponents may be used to denote repetition; e.g.,

$$a^2b^3ab^2 \text{ for } aabbbabb \quad \text{and} \quad (cd)^3 \text{ for } cdcdcd$$

In addition, the empty string may be denoted by Λ, the Greek letter lambda, and the concatenation of strings X and Y may be denoted by $X \cdot Y$ or, simply, XY.

Pattern Matching Algorithm

(Pattern Matching) P and T are strings with lengths R and S, respectively, and are stored as arrays with one character per element. This algorithm finds the INDEX of P in T.

1. [Initialize.] Set $K := 1$ and $MAX := S - R + 1$.
2. Repeat Steps 3 to 5 while $K \le MAX$:
3. Repeat for $L := 1$ to R: [Tests each character of P.]
   If $P[L] \ne T[K + L - 1]$, then: Go to Step 5.
   [End of inner loop.]
4. [Success.] Set $INDEX = K$, and Exit.
5. Set $K := K + 1$.
   [End of Step 2 outer loop.]
6. [Failure.] Set $INDEX = 0$.
7. Exit.

Pattern Matching Algorithm

Pattern: abaa; *searched string*: ababbaabaaab

```
ababbaabaaab                  ...........

abaa_______   step 1          ABA#         mismatch: 4th letter
_abaa______   step 2          _#...        mismatch: 1st letter
__abaa_____   step 3          __AB#.       mismatch: 3rd letter
___abaa____   step 4          ___#...      mismatch: 1st letter
____abaa___   step 5          ____#...     mismatch: 1st letter
_____abaa__   step 6          _____A#..    mismatch: 2nd letter
______abaa_   step 7          ______ABAA   success
```

Pattern Matching Algorithm

```
for ( j = 0; j <= n - m; j++ ) {
   for ( i = 0; i < m && x[i] == y[i + j]; i++ );
   if ( i >= m ) return j;
}
```

Main features of this easy (but slow) $O(nm)$ algorithm:

- No preprocessing phase
- Only constant extra space needed
- Always shifts the window by exactly 1 position to the right
- Comparisons can be done in any order
- mn expected text characters comparisons

Pattern Matching Algorithm

Single Pattern Algorithms (Summary)

Notation:

m – the length (size) of the pattern; n – the length of the searched text

String search algorithm	**Time complexity for**	
	preprocessing	matching
Naïve	0 (none)	$\Theta(n \cdot m)$
Rabin-Karp	$\Theta(m)$	avg $\Theta(n+m)$ worst $\Theta(n \cdot m)$
Finite state automaton	$\Theta(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Boyer-Moore	$\Theta(m+\|\Sigma\|)$	$\Omega(n/m)$, $O(n)$
Bit based (approximate)	$\Theta(m+\|\Sigma\|)$	$\Theta(n)$

See `http://www-igm.univ-mlv.fr/~lecroq/string` for some animations of these and many other string algorithms