Preview

- **String**
  - String is a simple array of chars (characters)
  - String is one-dimensional array of char
  - The null character '\0' must be included at the end of String
  - Various built-in functions are provided for String function

String Operation

- **Length: LENGTH (string)**
  e.g.- LENGTH('Mark Zuckerberg')= 15

- **Substring: SUBSTRING(string, initial, length)**
  e.g.- SUBSTRING('Impossible is a word found in coward's dictionary',0,20) = Impossible is a word

- **Indexing: INDEX(string, pattern)**
  e.g.- INDEX('He is wearing glasses', 'ear')= 7

- **Concatenation: String1//String2**
  e.g.- 'To be or not to be'// ', this is the question.'= To be or not to be, this is the question

String Operation

- **Word Processing-**

  Insertion: INSERT(string, position, string)
  e.g.- INSERT('ABCDEIJKL',5,'FGH')=
  ABCDEFGHIJKL

  Deletion: DELETE(string, position, length)
  e.g.- DELETE('ABCDEFG', 4, 2)= ABCDG

String Operation

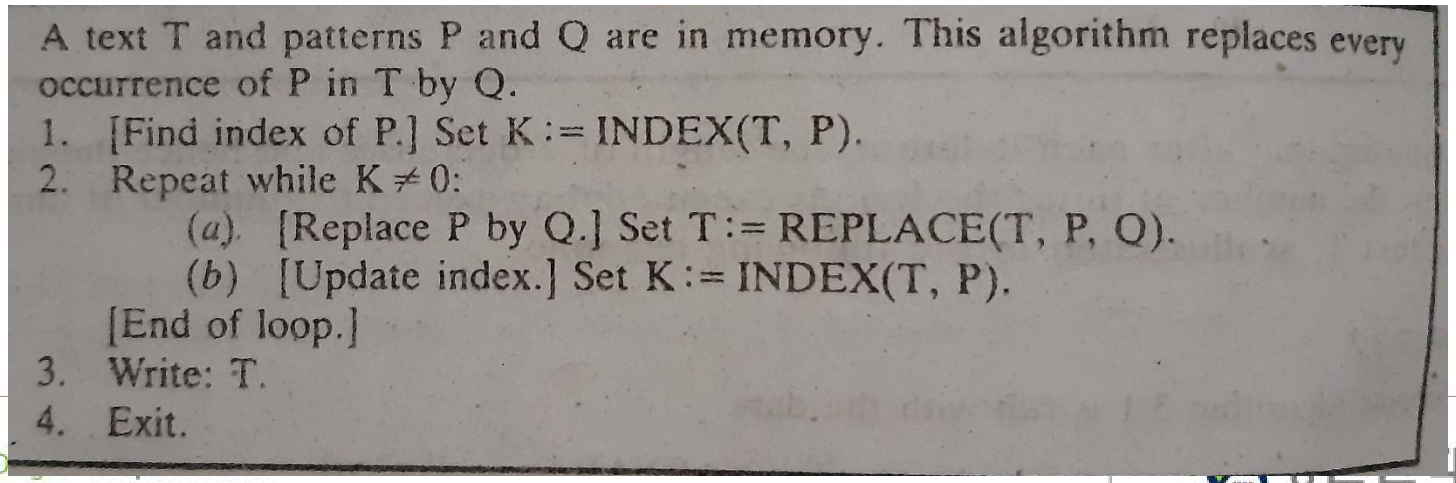
- **Replacement: REPLACE(string, pattern1, pattern2)**

  e.g.- REPLACE('XABYABZ', 'AB', 'c')= XCYABZ

REPLACE function can be executed be using the following three steps-

1. K:= INDEX(string, P1)
2. T:= DELETE(string, K, LENGTH(P1))
3. INSERT(T, K, P1)

- So, the algorithm is-



A text T and patterns P and Q are in memory. This algorithm replaces every occurrence of P in T by Q.

1. [Find index of P.] Set K := INDEX(T, P).
2. Repeat while K ≠ 0:
   (a) [Replace P by Q.] Set T := REPLACE(T, P, Q).
   (b) [Update index.] Set K := INDEX(T, P).
   [End of loop.]
3. Write: T.
4. Exit.

String Operation

– Pattern Matching:

Pattern matching is the problem of deciding whether or not a given string pattern P appears in a text.

Widely used in word processing.

– So, a basic algorithm is-

(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 ≤ MAX:
3. Repeat for L = 1 to R: [Tests each character of P.]
   If P[L] ≠ 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.

Deletion: DELETE(string, position, length)

Algorithm for deletion -

A text T and a pattern P are in memory. This algorithm deletes every occurrence of P in T.

1. [Find index of P.] Set K := INDEX(T, P).
2. Repeat while K ≠ 0:
   (*a*) [Delete P from T.]
   Set T := DELETE(T, INDEX(T, P), LENGTH(P))
   (*b*) [Update index.] Set K := INDEX(T, P).
   [End of loop.]
3. Write: T.
4. Exit.

Deletion example-

a) Suppose Algorithm 3.1 is run with the data

$$T = XABYABZ, \qquad P = AB$$

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.

Suppose Algorithm 3.1 is run with the data

$$T = XAAABBBY, \qquad P = AB$$

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.