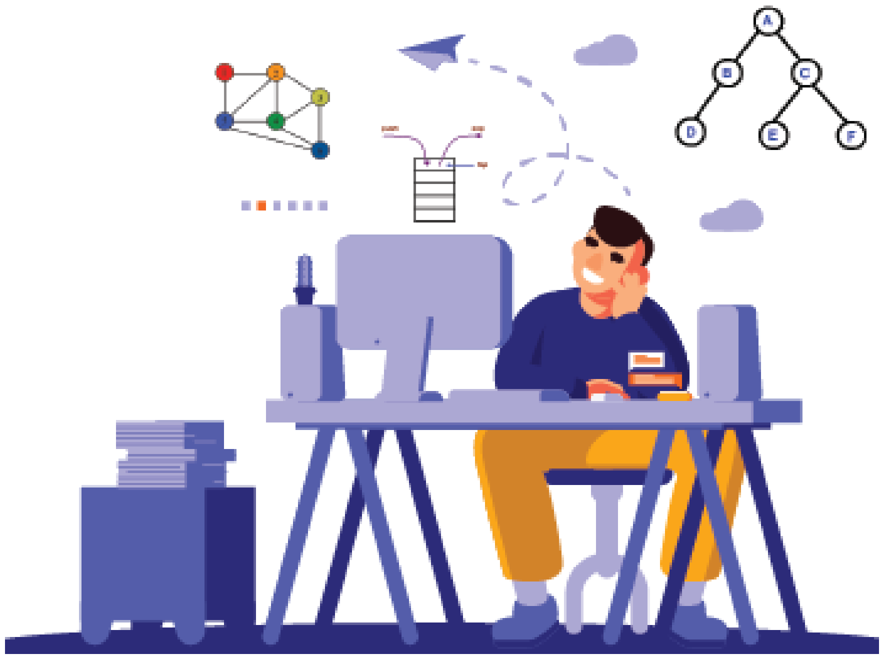
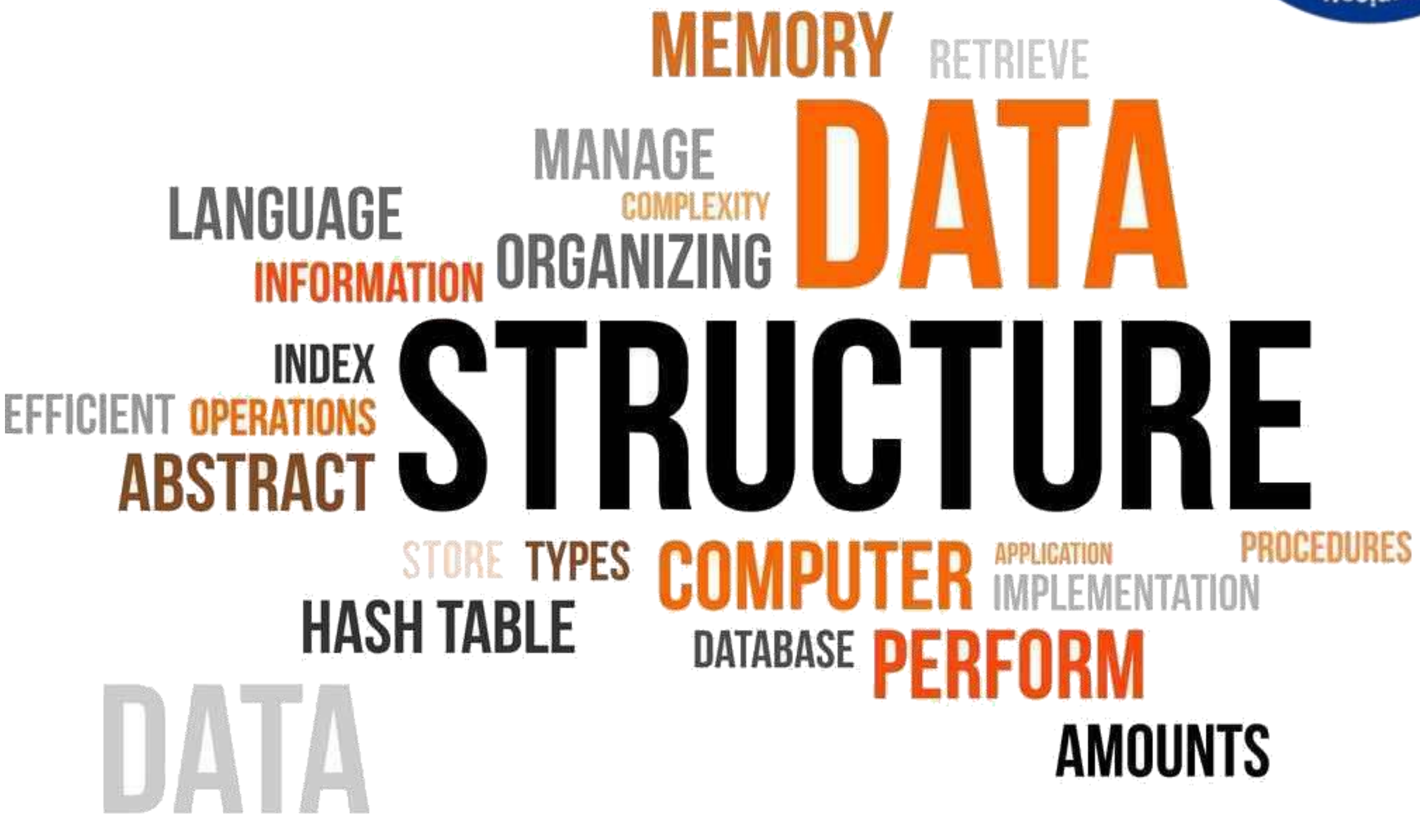


Data Structures

Course code: IT623



Dr. Rahul Mishra
Assistant Professor
DA-IICT, Gandhinagar

Lectures 12

String Processing…

PATTERN MATCHING ALGORITHM

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

"Here, we assume that the length of P does not exceed the length of T.

Notably,

→ Characters are sometimes denoted by lowercase letters (a, b, c, ...) and exponents may be used to denote repetitions.

$a^2 b^3 a b^2$ for aabbbabb

and

$(cd)^3$ for cdcdcd.

Additionally,

- The empty strings are denoted as Λ
- Concatenation X.Y.

First Pattern Matching Algorithm:-

Here, we compare a given pattern P with each of the substrings of T, moving from left to right, until we get a match.

$$\boxed{W_K = \text{SUBSTRING}(T, K, \text{LENGTH}(P))}$$

* W_K denote the substring of T having the same length as P and begining with the K th character of T.
* First, we compare P, character-by-character, with first substring, W_1
    * if all the characters are the same, then $P = W_1$ and "so P appears in T"
* IF, we find that some character of P is not the same as the corresponding character of Compare P with W_2. IF $P \neq W_2$ then we compare P with W_3 and so on.

Termination Criteria

15

1. The process stop when we find a match of P with some substring W_K and so P appears in T and INDEX (T, P) = K.

2. When we exhaust all the W_K's with no match and hence P does-not appear in T.

The maximum value MAX of the subscript K is equal to LENGTH (T) − LENGTH(P) + 1.

* For example,

Let us consider P is a 4-character string and that T is a 20-character string, and that $\boxed{\text{P and T}}$ appear in memory as linear arrays with one character per element.

P = P[1] P[2] P[3] P[4]

and

T = T[1] T[2] . . . T[19] T[20]

* P is compared with each of the following 4-character substrings of T: (16)

$\{ W_1 = T[1]T[2]T[3]T[4],\ W_2\, T[2]T[3]T[4]T[5],\ \ldots\ W_{17} = T[17]T[18]T[19]T[20] \}$

MAX = 20−4+1 = 17 such substrings of T.

(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.

Pattern Matching First Algorithm

* The Complexity of the pattern matching algorithm is measured by the number (17) of (C) comparison between characters of the text (T)

* To determine (C) we let (N_K) denote the number of comparisons that take place in the inner loop when P is compared with (W_K).

$$C = N_1 + N_2 + \cdots N_L$$

where (L) is the position L in T where P first appears or (L = MAX) if P does-not appear in (T).

This example computes C for some specific P and T where LENGTH(P) = 4 and LENGTH(T) = 20 and so $\boxed{MAX = 20 - 4 + 1.}$

(a) Suppose $\boxed{P = aaba}$ and $\boxed{T = cdcd\ldots cd = (cd)^{10}}$. Clearly P does-not occur in T. Also, for each of the $\boxed{17 \text{ cycles}}$, $\boxed{N_k = 1}$ since the first character of P does-not match W_K, Hence

$$\boxed{C = 1+1+1+\cdots+1 = 17}$$

(b) Let P = aaba and T = ababaaba ... Observe that P is a substring of T. Infact, $P = W_5$ and so $N_5 = 4$. Also, comparing P with $W_1 = abab$, we obtain that $N_1 = 2$, since the first letter do match; but comparing P with $W_2 = baba$ we obtain $N_2 = 1$. Similarly $N_3 = 2$ and $N_4 = 1$

$$\boxed{C = 2+1+2+1+4 = 10}$$

(c) Let $\boxed{P = aaab \text{ and } T = aa \cdots a = a^{20}}$. Here P does not appear in T. Also,

every $W_K = aaaa$; hence every $N_K = 4$

$$C = 4 + 4 + \cdots 4 = 17 \cdot 4 = 68$$

Let P is an r character string and T is an s character string, the data size for the algorithm is

$$\boxed{n = r + s}$$

The worst case occur when every character of P except the last matches every substring $\boxed{W_K}$, as in above example.

$$\boxed{C(n) = r(s - r + 1)}.$$

For fixed n, we have $\boxed{s = n - r}$, so that :- " $\boxed{C(n) = r\,(n - 2r + 1)}$ "

* The maximum value of $\boxed{C(r)}$ occurs when $\boxed{r = (n+1)/4}$. Accordingly, substituting this value for r in the formula for $\boxed{C(r)}$ yields:

$$\boxed{C(n) = \frac{(n+1)^2}{8} = O(n^2)}$$

How, we get $r = n+1/4$

$$\frac{dc}{dr} = ?$$

Please note down

* The complexity of the average case in any actual situation depends on certain probabilities which are usually unknown.

* When the character of P and T are randomly selected from some finite alphabet, the complexity of the average case is still not easy to analyze, but the complexity of the average case is still a factor of the worst-case.

* The complexity of this pattern matching algorithm is equal to $O(n^2)$.

Second Pattern Matching Algorithm :-

* The second pattern matching algorithm uses a table which is derived from a particular pattern P but is independent of the text T. For definiteness, suppose

$\Longrightarrow$ "$\boxed{P = aaba}$" pattern

* First we give the reason for the table entries and how they are used.

* Let $T = T_1 T_2 T_3 \ldots$, where T_i denotes the i^{th} character of T; and suppose the first two characters of T match those of P; i.e, suppose $T = aa\ldots$ Then T has one of the following three forms

(i) $T = aab\ldots$, (ii) $T = aaa\ldots$, (iii) $T = aax$

Where x is any character different from a or b.