Lecture#2

Data Structures

Dr. Abu Nowshed Chy

Department of Computer Science and Engineering

University of Chittagong

January 05, 2025

Faculty Profile

Practice Problem

No#1: Estimate the worst-case time complexity of the following code snippet.

```
sum = 0;
for i=0 to n
      for j=0 to n
            for k=0 to n
                  sum++;
print(sum);
```

Practice Problem

No#2: Estimate the worst-case time complexity of the following pseudocode.

Algorithm 2.4: (Linear Search) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = O.

1. [Initialize] Set K := 1 and LOC := 0.
2. Repeat Steps 3 and 4 while LOC = 0 and K ≤ N.
3. If ITEM = DATA[K], then: Set LOC: = K.
4. Set K := K + 1. [Increments counter.]
  [End of Step 2 loop.]
5. [Successful?]
  If LOC = 0, then:
    Write: ITEM is not in the array DATA.
  Else:
    Write: LOC is the location of ITEM.
  [End of If structure.]
6. Exit.

String Matching

Basic Terminologies

- Each programming language contains a character set that is used to communicate with the computer. This usually indicates the following:
  - Alphabet: A,B,C,D…..,Z
  - Digits: 0,1,2,3,4,5,6,7,8,9
  - Characters: +, -, /, *, ^, &, %, =
- A finite sequence of 0 or more characters is called a string.
- The string with zero characters is called the empty string or null string.

Storing Strings

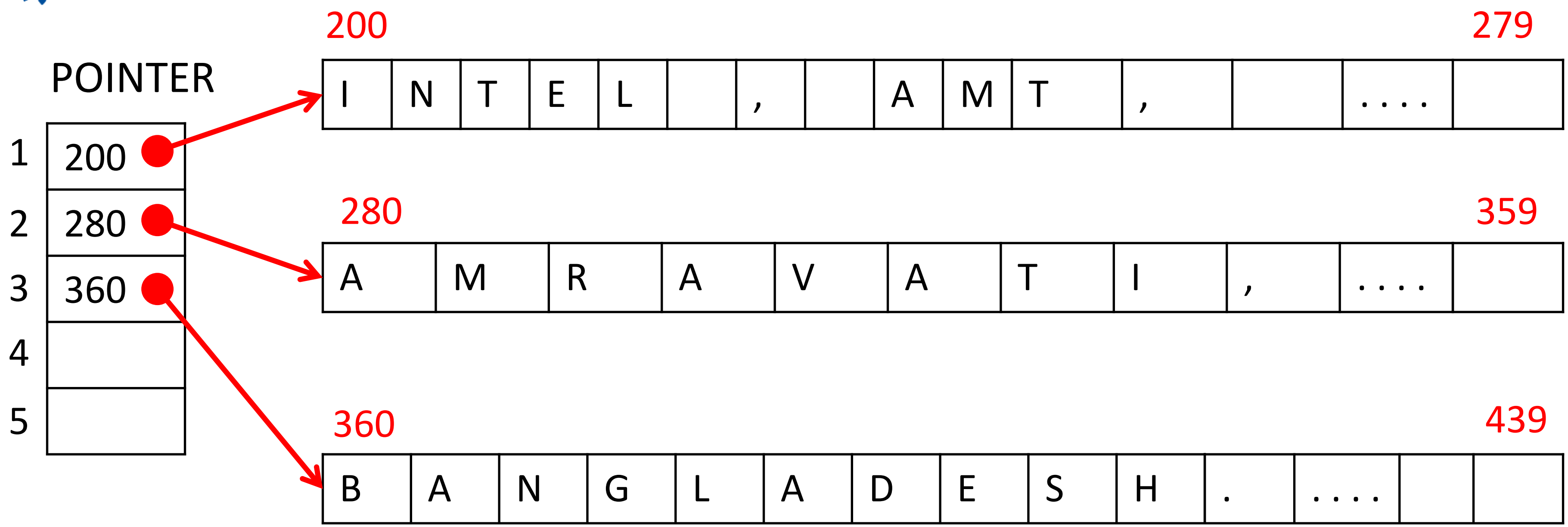
Strings are stored in there types of structures

- Fixed-length structure
- Variable-length structure
- Linked Structure

Storing Strings (Fixed-length)

- In this storage each line is considered as record, where all record have same length i.e contain same number of character.
- Advantage: Ease of accessing and updating data.
- Disadvantage:
  - Time is wasted reading an entire record if most of storage consist of inessential blank space.
  - Certain records may require more space than available.
  - When correction consist of more or fewer characters than the original text, changing a misspelled word requires the entire record be changed.

Storing Strings (Fixed-length)



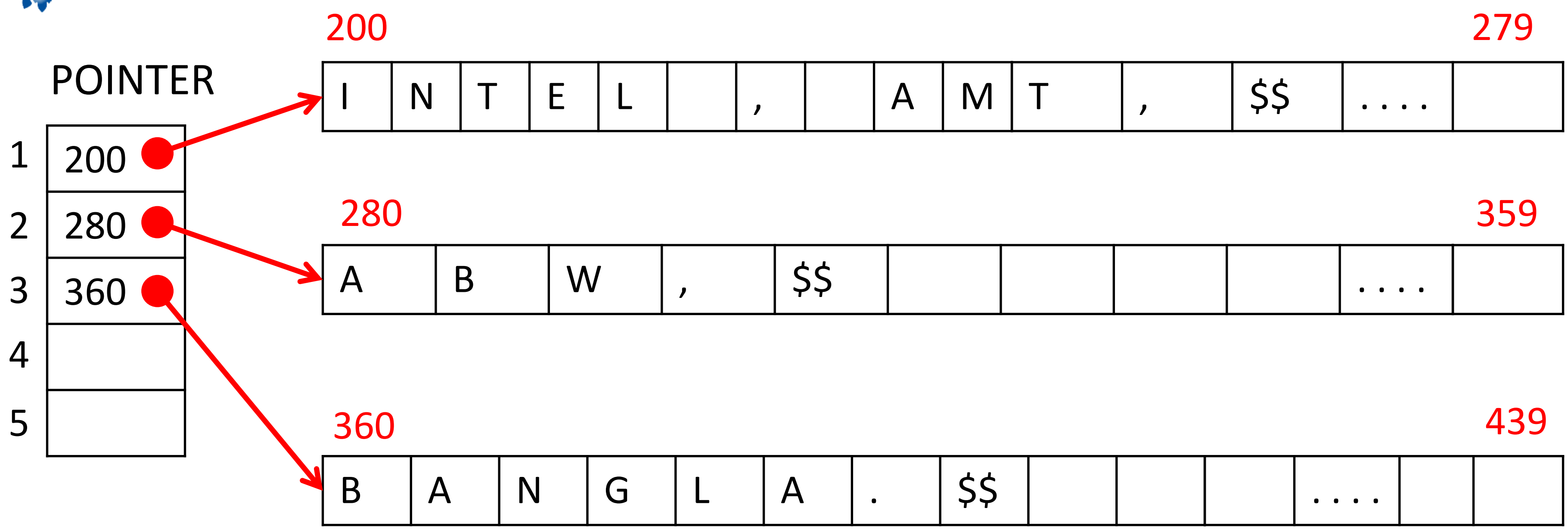
Suppose input contains sting as,
‘INTEL , AMT,
‘AMRAVATI ,’
‘BANGLADESH.’

Storing Strings (Variable length)

The storage of variable length strings in memory cells with fixed length can be done in two general ways.

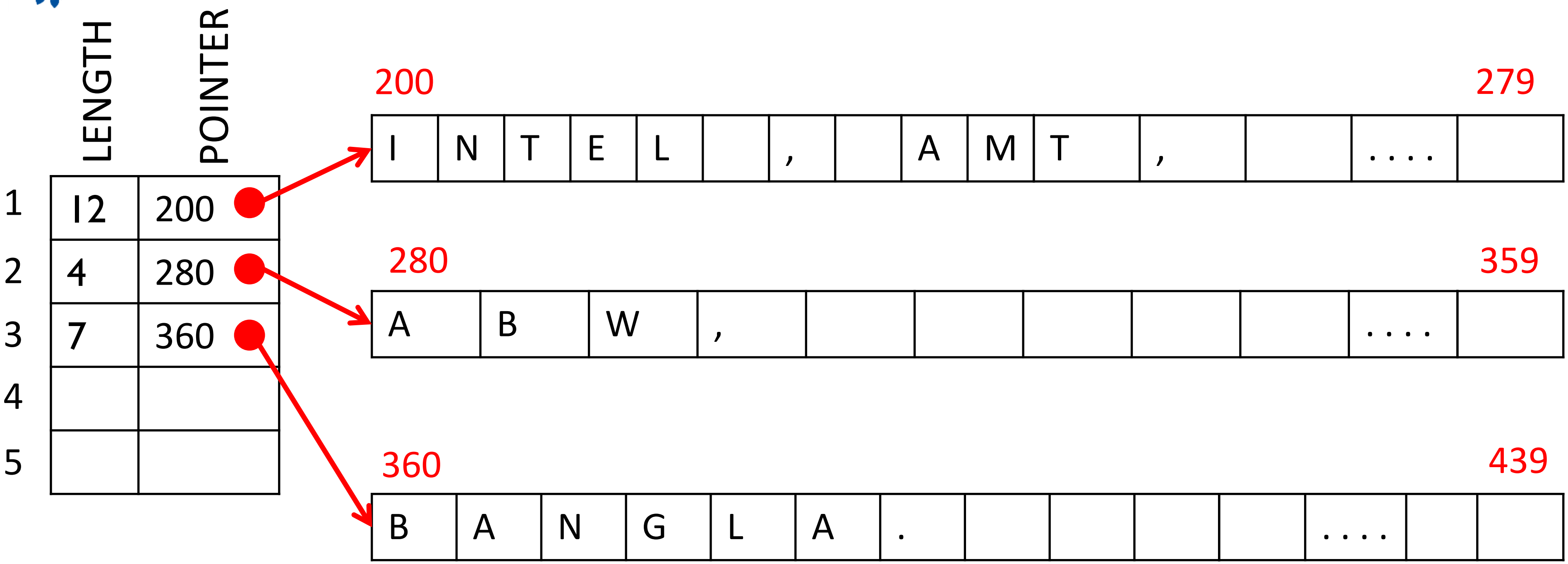
- One can use a marker, such as two dollar signs ($$), to signal the end of the string.
- One can list the length of the string as an additional item in the pointer array.

Storing Strings (Variable length)



Suppose input contains sting as,
‘INTEL , AMT,
‘ABW,
‘BANGLA.’

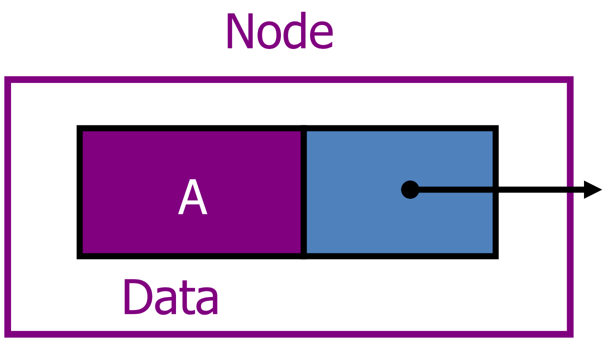
Storing Strings (Variable length)

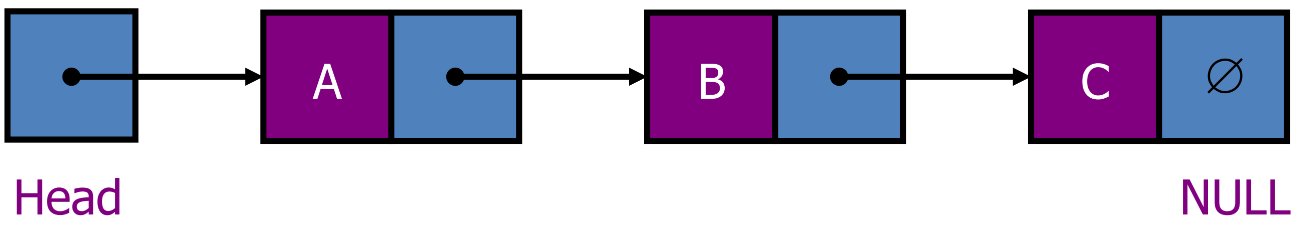


Suppose input contains sting as,
‘INTEL , AMT,
‘ABW,
‘BANGLA.’

Storing Strings (Linked Structure)

- A *linked list* is a series of connected *nodes*
- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- *Head*: pointer to the first node
- The last node points to NULL



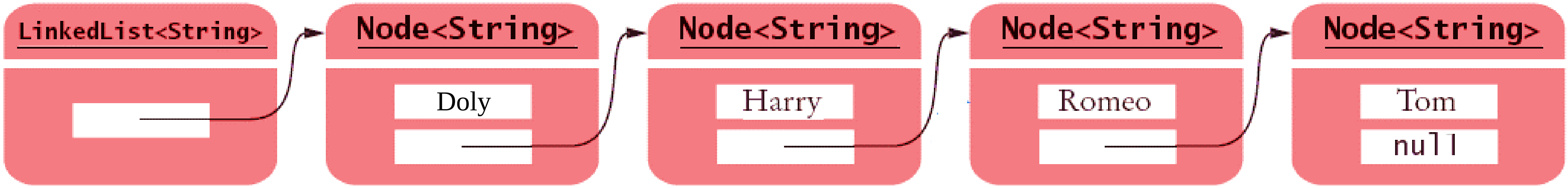


Storing Strings (Linked Structure)

A linked list is a data structure which can change during execution.

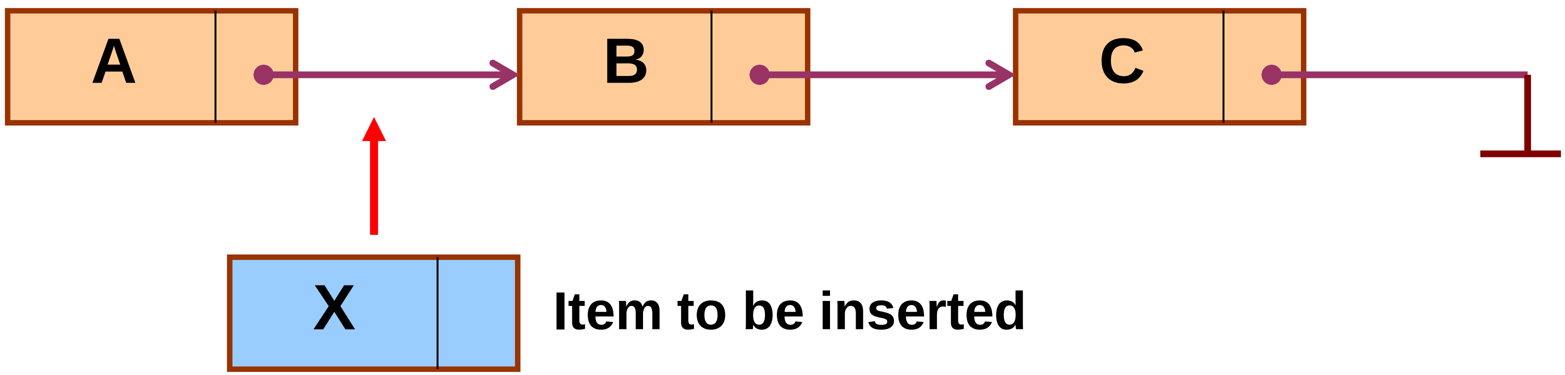
- Successive elements are connected by pointers.
- Last element points to NULL.
- It can grow or shrink in size during execution of a program.
- It can be made just as long as required.
- It does not waste memory space.

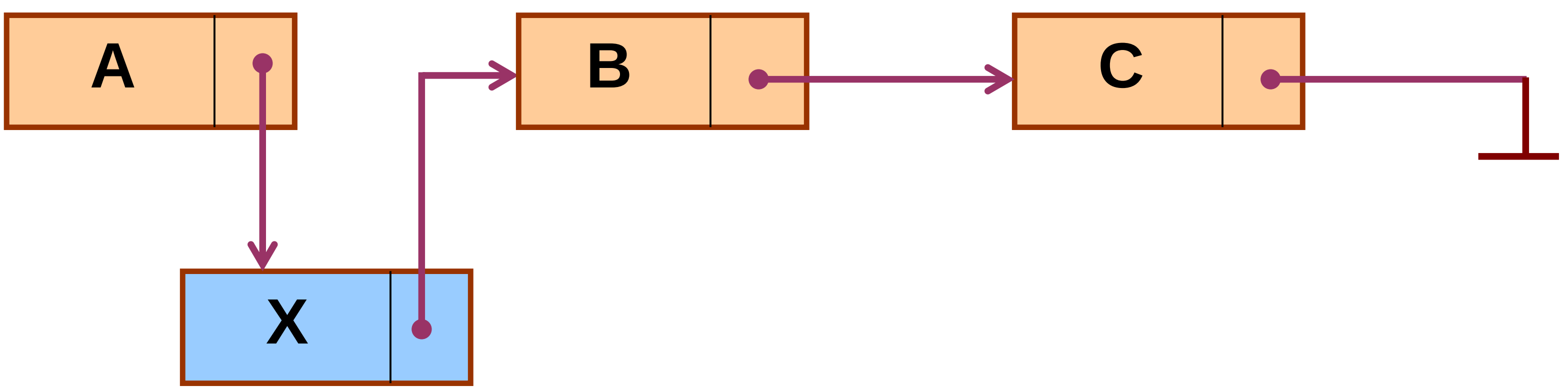
Storing Strings (Linked Structure)



Storing Strings (Linked Structure)

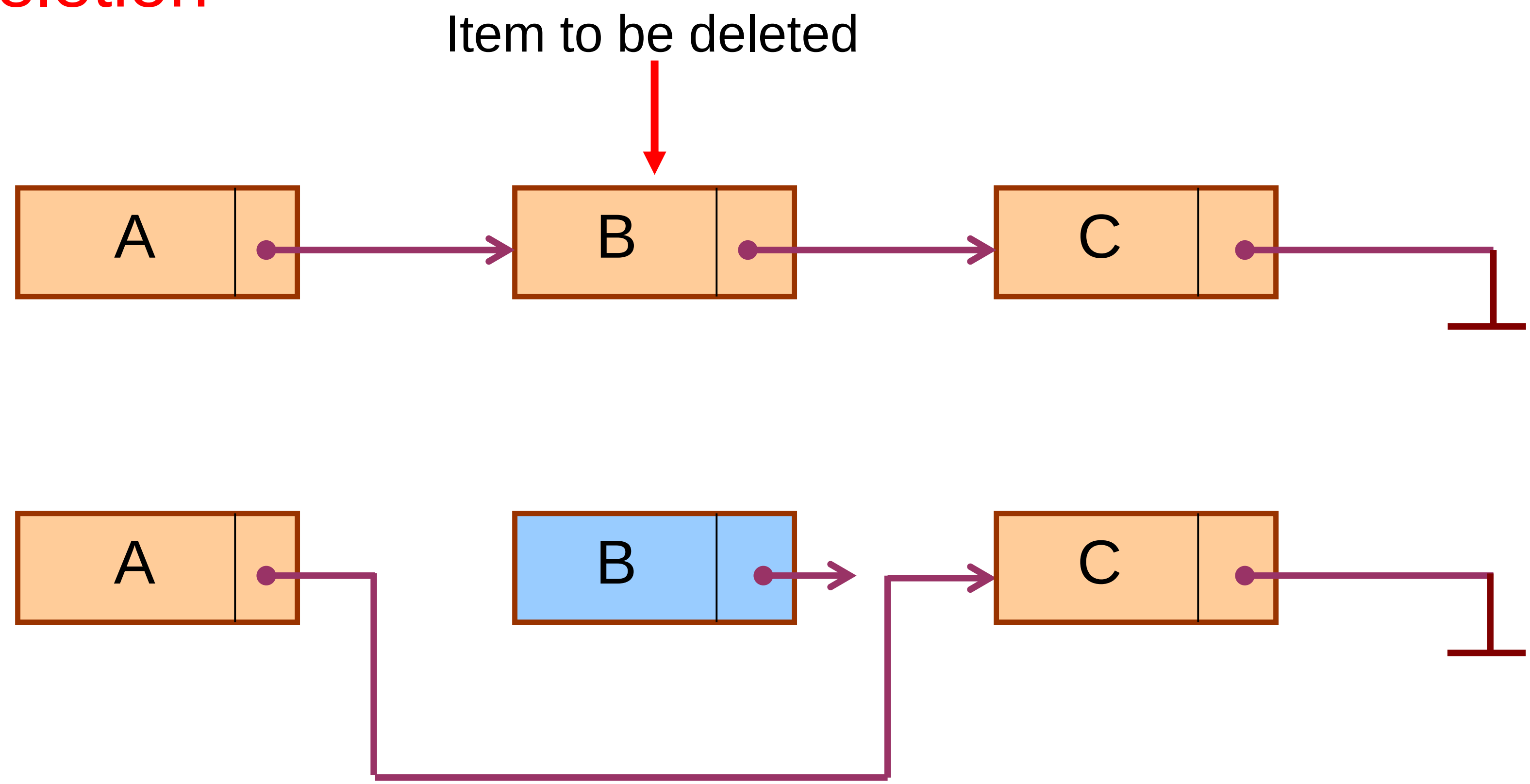
Insertion or Update





Storing Strings (Linked Structure)

Deletion



Character Data Type

Constant: Many languages denotes string constant by placing the string in either single or double quotation mark.

Static character variable is that whose length is defined before the program is executed and cannot change throughout the program.

Semistatic variable is that in which length may vary during the execution of the program as long as the length does not exceed a maximum value determined by the program before the program is executed.

Dynamic character variable we mean a variable whose length can change during the execution of program.

String Operations

There are some operations to manipulate the string data:

1. Length
2. Substring
3. Indexing
4. Concatenation
5. Insertion
6. Deletion
7. Replacement

String Operations (Length)

The number of character in string is called its length

LENGTH(string)

1 2 3 4 5 6 7 8 9
LENGTH('COMPUTER ') = 9

String Operations (Substring)

Accessing a substring from a given string requires three pieces of information:

1. The name of string or the string itself
2. The position of the first character of the substring in the given string
3. The length of the substring

SUBSTRING(String , Initial , length)

String Operations (Substring)

SUBSTRING(String , Initial , length)

1 2 3 4 5 6 7 8 9 . . . .
SUBSTRING('T O B E O R N O T T O B E', 4, 7)

SUBSTRING('THE END', 4, 4)

String Operations (Indexing)

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

INDEX(text , pattern)

INDEX('He is wearing glasses', 'ear')= 8

String Operations (Indexing)

INDEX(text , pattern)

Suppose T contains the text

'HIS FATHER IS THE PROFESSOR'

Then,

INDEX(T, 'THE'), INDEX(T, 'THEN') and INDEX(T, '□THE□')

String Operations (Concatenation)

Let S1, and S2 be string then concatenation of S1 and S2 is denoted by S1 // S2 is the string consisting of the character of S1 followed by the character S2.

S1 = ‘MARK’

S2 = ‘TWIN’

S1//S2 = ‘MARKTWIN”

String Operations (Insertion)

Insertion means inserting a string in the middle of a string.

INSERT (string, position, string)

INSERT(‘ABCDEF’, 3 , ‘XYZ’) = ‘ABXYZCDEF’

String Operations (Deletion)

Deletion means deleting a string from a string.

DELETE (string, position, length)

DELETE (‘ABCDEFG’, 4, 2) = ‘ABCFG’

DELETE (‘ABCDEFG’, 2, 4) = ‘AFG’

DELETE (‘ABCDEFG’, 0, 2) = ‘ABCDEFG’

String Operations (Replacement)

Suppose in a given text T we want to replace the first occurrence of a pattern P1 by a pattern P2. we will denote this operation by

REPLACE (string, pattern1, pattern2)

REPLACE('ABXYEFGH', 'XY' , 'CD') = 'ABCDEFGH'

String Operations (Replacement)

REPLACE (text, pattern1, pattern2)

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

Lab Task#1

Write a program that implements all the string related functions discussed.

String Pattern Matching

Pattern Matching Algorithm

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

1. First Pattern Matching Algorithm
   - The first pattern matching algorithm is the obvious one in which we compare a given pattern P with each of the substrings of T, moving from left to right, until we get a match.

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.

Pattern Matching Algorithm

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

1. First Pattern Matching Algorithm

   - The first pattern matching algorithm is the obvious one in which we compare a given pattern P with each of the substrings of T, moving from left to right, until we get a match.

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.

First Pattern Matching Algorithm

Algorithm 3.3: (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.

Worst Case Time Complexity:

First Pattern Matching Algorithm

T = ababababab P = abc

Length, S = 10 Length, R = 3

MAX = S-R+1=10-3+1=8

1. aba = abc	5. aba = abc
2. bab = abc	6. bab = abc
3. aba = abc	7. aba = abc
4. bab = abc	8. bab = abc

Second Pattern Matching Algorithm

Consider the pattern P = ababab. Construct the table and the corresponding labeled directed graph used in the “fast,” or second pattern matching algorithm.

The initial substrings of P are:

$Q_0 = \Lambda$,

$Q_1 = a$,

$Q_2 = ab$,

$Q_3 = aba$,

$Q_4 = abab$,

$Q_5 = ababa$,

$Q_6 = ababab \rightarrow P$

Second Pattern Matching Algorithm

The initial substrings of P are:

$Q_0 = \Lambda$, $Q_1 = a$, $Q_2 = ab$, $Q_3 = aba$,
$Q_4 = abab$, $Q_5 = ababa$, $Q_6 = ababab \rightarrow P$

The function f giving the entries in the table are as follows:

$f(\Lambda, a) = a$ | $f(\Lambda, b) = \Lambda$
$f(a, a) = a$ | $f(a, b) = ab$
$f(ab, a) = aba$ | $f(ab, b) = \Lambda$
$f(aba, a) = a$ | $f(aba, b) = abab$
$f(abab, a) = ababa$ | $f(abab, b) = \Lambda$
$f(ababa, a) = a$ | $f(ababa, b) = P$

Second Pattern Matching Algorithm

$Q_0 = \Lambda$, $Q_1 = a$, $Q_2 = ab$, $Q_3 = aba$,

$Q_4 = abab$, $Q_5 = ababa$, $Q_6 = ababab \rightarrow P$

$f(\Lambda, a) = a$ $f(\Lambda, b) = \Lambda$

$f(a, a) = a$ $f(a, b) = ab$

$f(ab, a) = aba$ $f(ab, b) = \Lambda$

$f(aba, a) = a$ $f(aba, b) = abab$

$f(abab, a) = ababa$ $f(abab, b) = \Lambda$

$f(ababa, a) = a$ $f(ababa, b) = P$

Table

	a	b
Q_0	Q_1	Q_0
Q_1	Q_1	Q_2
Q_2	Q_3	Q_0
Q_3	Q_1	Q_4
Q_4	Q_5	Q_0
Q_5	Q_1	P

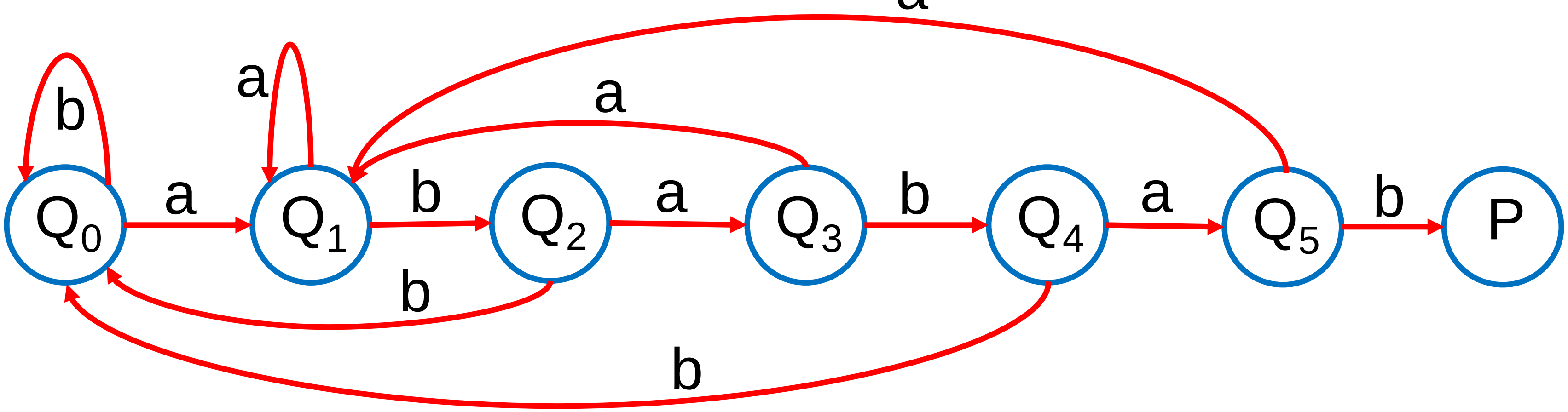
Second Pattern Matching Algorithm

Table

	a	b
Q_0	Q_1	Q_0
Q_1	Q_1	Q_2
Q_2	Q_3	Q_0
Q_3	Q_1	Q_4
Q_4	Q_5	Q_0
Q_5	Q_1	P

Pattern P = ababab

Labeled Directed Graph



Second Pattern Matching Algorithm

Consider the pattern P = aaabb. Construct the table and the corresponding labeled directed graph used in the "fast," or second pattern matching algorithm.

The initial substrings of P are:

$Q_0 = \Lambda$,

$Q_1 = a$,

$Q_2 = aa = a^2$,

$Q_3 = aaa = a^3$,

$Q_4 = aaab = a^3b$,

$Q_5 = aaabb = a^3b^2 \rightarrow P$

Second Pattern Matching Algorithm

$Q_0 = \Lambda$, $Q_1 = a$, $Q_2 = aa = a^2$,

$Q_3 = aaa = a^3$, $Q_4 = aaab = a^3b$, $Q_5 = aaabb = a^3b^2 \rightarrow P$

$f(\Lambda, a) =$ $f(\Lambda, b) =$

$f(a, a) =$ $f(a, b) =$

$f(a^2, a) =$ $f(a^2, b) =$

$f(a^3, a) =$ $f(a^3, b) =$

$f(a^3b, a) =$ $f(a^3b, b) =$

Table

	a	b
Q_0	Q_1	Q_0
Q_1	Q_2	Q_0
Q_2	Q_3	Q_0
Q_3	Q_3	Q_4
Q_4	Q_1	P

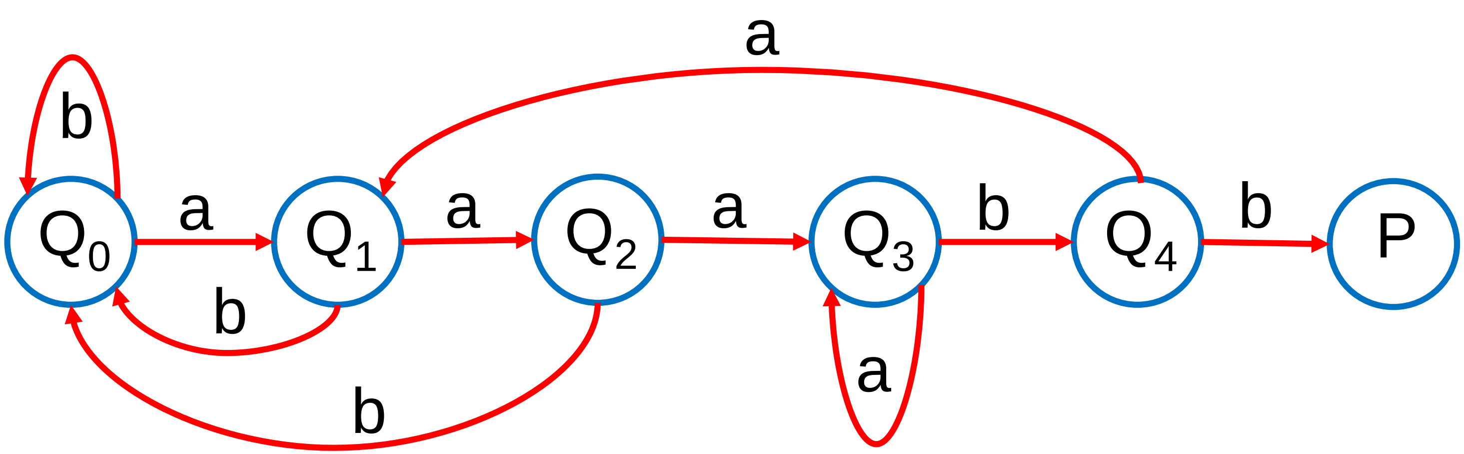
Second Pattern Matching Algorithm

Table

	a	b
Q_0	Q_1	Q_0
Q_1	Q_2	Q_0
Q_2	Q_3	Q_0
Q_3	Q_3	Q_4
Q_4	Q_1	P

Pattern P = aaabb

Labeled Directed Graph



Second Pattern Matching Algorithm

Consider the pattern P = aaba. Construct the table and the corresponding labeled directed graph used in the "fast," or second pattern matching algorithm.

The initial substrings of P are:

$Q0 = \Lambda$,

$Q1 = a$,

$Q2 = aa = a^2$,

$Q3 = aab = a^2b$,

$Q4 = aaba = a^2ba \rightarrow P$

Second Pattern Matching Algorithm

$Q0 = \Lambda$, $Q1 = a$, $Q2 = aa = a^2$,

$Q3 = aab = a^2b$, $Q4 = aaba = a^2ba$ ➔ P

$f(\Lambda, a) =$ $f(\Lambda, b) =$

$f(a, a) =$ $f(a, b) =$

$f(a^2, a) =$ $f(a^2, b) =$

$f(a^2b, a) =$ $f(a^2b, b) =$

Table

	a	b
Q_0	Q_1	Q_0
Q_1	Q_2	Q_0
Q_2	Q_2	Q_3
Q_3	P	Q_0

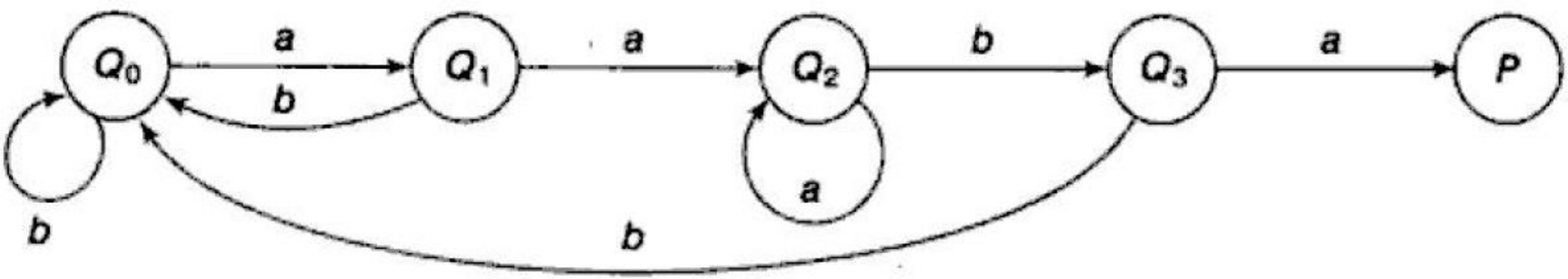
Second Pattern Matching Algorithm

Table

	a	b
Q_0	Q_1	Q_0
Q_1	Q_2	Q_0
Q_2	Q_2	Q_3
Q_3	P	Q_0

Pattern P = aaba

Labeled Directed Graph



Second Pattern Matching Algorithm

Algorithm 3.4: (Pattern Matching). The pattern matching table $F(Q_1, T)$ of a pattern P is in memory, and the input is an N-character string $T = T_1T_2 \ldots T_N$. This algorithm finds the INDEX of P in T.

1. [Initialize.] Set $K := 1$ and $S_1 = Q_0$

2. Repeat Steps 3 to 5 while $S_K \neq P$ and $K \leq N$.

3. Read T_K.

4. Set $S_{K+l} := F(S_K, T_K)$. [Finds next state.]

5. Set $K := K + 1$. [Updates counter.]

[End of Step 2 loop.]

6. [Successful?]

If $S_K = P$, then:

INDEX = K − LENGTH(P).

Else:

INDEX = 0.

[End of If structure.]

7. Exit

Worst Case Time Complexity:


Thank You!