

CSE231: Data Structures

Lecture 07

by

Mehedi Hasan Bijoy

WORD/TEXT PROCESSING

The operations usually associated with word processing are the following

- **Insertion:** Inserting a string in the middle of the text
- **Deletion:** Deleting a string from the text
- **Replacement:** Replacing one string in the text by another

WORD/TEXT 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

$\text{INSERT}(\text{text}, \text{position}, \text{string})$

For example,

$\text{INSERT}('ABCDEFGH', 3, 'XYZ') = 'ABXYZCDEFGH'$

$\text{INSERT}('ABCDEFGH', 6, 'XYZ') = 'ABCDEXYZFGH'$

WORD/TEXT PROCESSING

```
char* INSERT(char* S1,int K,char*S2)
{
    char RESULT[80];
    strcpy(RESULT,SUBSTR(S1,1,K-1));
    strcat(RESULT,S2);
    strcat(RESULT,SUBSTR(S1,K,strlen(S1)-K+1));
    return(RESULT);
}
```

WORD/TEXT PROCESSING

Deletion

Suppose in a given text T we want to delete the substring which begins at 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 '

WORD/TEXT PROCESSING

```
char* DELETE(char* S1,int K,int L)
{
    char RESULT[80];
    strcpy(RESULT,SUBSTR(S1,1,K-1));
    strcat(RESULT,SUBSTR(S1,K+L,strlen(S1)-K-L+1));
    return(RESULT);
}
```

WORD/TEXT PROCESSING

(a) Suppose $T = \text{'ABCDEFGH'}$ and $P = \text{'CD'}$. Then $\text{INDEX}(T, P) = 3$ and $\text{LENGTH}(P) = 2$. Hence

$\text{DELETE}(\text{'ABCDEFGH'}, 3, 2) = \text{'ABEFGH'}$

(b) Suppose $T = \text{'ABCDEFGH'}$ and $P = \text{'DC'}$. Then $\text{INDEX}(T, P) = 0$ and $\text{LENGTH}(P) = 2$. Hence, by the "zero case,"

$\text{DELETE}(\text{'ABCDEFGH'}, 0, 2) = \text{'ABCDEFGH'}$

as expected.

WORD/TEXT PROCESSING

Suppose after reading into the computer a text T and a pattern P , we want to delete every occurrence of the pattern P in the text T . This can be accomplished by repeatedly applying

$\text{DELETE}(T, \text{INDEX}(T, P), \text{LENGTH}(P))$

until $\text{INDEX}(T, P) = 0$ (i.e., until P does not appear in T). An algorithm which accomplishes this follows.

WORD/TEXT PROCESSING

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 $K := \text{INDEX}(T, P)$.
2. Repeat while $K \neq 0$:
 - (a) [Delete P from T .]
Set $T := \text{DELETE}(T, \text{INDEX}(T, P), \text{LENGTH}(P))$
 - (b) [Update index.] Set $K := \text{INDEX}(T, P)$.[End of loop.]
3. Write : T .
4. Exit.

WORD/TEXT 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

$$\text{REPLACE}(\text{'XABYABZ'}, \text{'AB'}, \text{'C'}) = \text{'XCYABZ'}$$
$$\text{REPLACE}(\text{'XABYABZ'}, \text{'BA'}, \text{'C'}) = \text{'XABYABZ'}$$

WORD/TEXT PROCESSING

Replacement function can be executed by using the following three steps:

```
K := INDEX(T, P1)  
T := DELETE(T, K, LENGTH(P1))  
INSERT(T, K, P2)
```

PATTERN MATCHING ALGORITHMS

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

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

$a^2b^3ab^2$ for $aabbbabb$ and $(cd)^3$ for $cdcdcd$

PATTERN MATCHING ALGORITHMS

- **First Pattern Matching Algorithm**

It 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.

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

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char P[80] = {"bab"};
    char T[80] = {"aabbbabb"};
    int R, S, K, L, MAX, INDEX;
    R = strlen(P);
    S = strlen(T);
    K = 0;
    MAX = S - R;
```

```
    while(K <= MAX){
        for(L = 0; L < R; L++){
            if(P[L] != T[K+L]){
                break;
            }
        }
        if(L == R){
            INDEX = K;
            break;
        }else{
            K = K + 1;
        }
    }
    if (K > MAX){
        INDEX = -1;
    }
    cout << "P = " << P << endl;
    cout << "T = " << T << endl;
    if(INDEX != -1){
        cout << "\nIndex of P in T is " << INDEX << endl ;
    }else{
        cout << "\nP does not exist in T" << endl;
    }
    return 0;
}
```

PATTERN MATCHING ALGORITHMS

- **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 ALGORITHMS

$P = \text{aaba}$

Inputs: a, b

States: $Q_0 = \text{“”}$

$Q_1 = a$

$Q_2 = aa$

$Q_3 = aab$

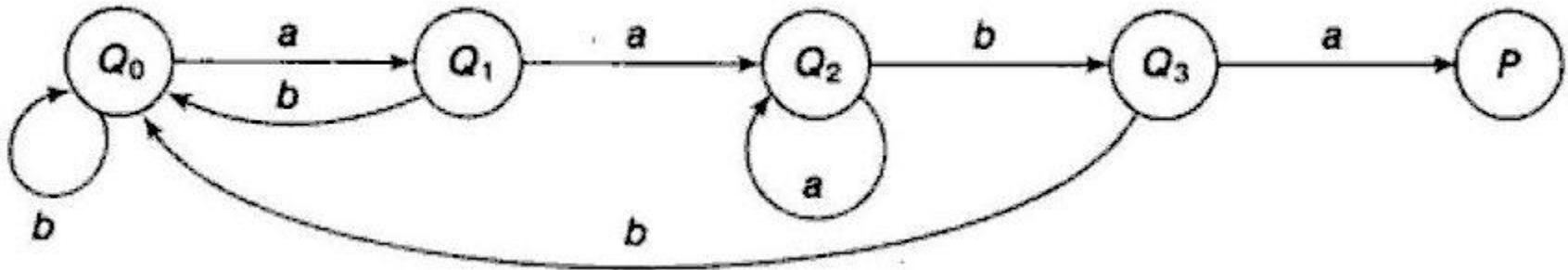
$Q_4 = \text{aaba} = P$

PATTERN MATCHING ALGORITHMS

	a	b
Q0	Q1	Q0
Q1	Q2	Q0
Q2	Q2	Q3
Q3	P	Q0

PATTERN MATCHING ALGORITHMS

	a	b
Q0	Q1	Q0
Q1	Q2	Q0
Q2	Q2	Q3
Q3	P	Q0



```

#include <iostream>
#include <cstring>
using namespace std;

char F(char, char);
int state[4][2];

int main()
{
    char P[80] = {"aaba"};
    char T[80] = {"abcaabaca"};
    int N, K, S, I, INDEX;

    state[0][0] = 1; state[0][1] = 0;
    state[1][0] = 2; state[1][1] = 0;
    state[2][0] = 2; state[2][1] = 3;
    state[3][0] = -1; state[3][1] = 0;

    N = strlen(T), K = 0, S = 0;

```

```

    while(K<N && S != -1){
        if(T[K] == 'a'){ I = 0; }
        if(T[K] == 'b'){ I = 1; }
        if(T[K] == 'x'){ I = 2; }
        S = F(S, I);
        K = K+1;
    }

    if(S == -1){ INDEX = K-strlen(P); }
    else{ INDEX = -1; }

    cout << "P = " << P << endl;
    cout << "T = " << T << endl;

    if(INDEX != -1){
endl;        Cout << "\nIndex of P in T is " << INDEX <<
    }else{
        cout << "\nP does not exist in T\n";
    }

    return 0;
}

char F(char SK, char TK){
    return state[SK][TK];
}

```

Thank You!