



FIND LONGEST MATCH STRING, BETWEEN TWO STRINGS

민경욱

1. 문제

- 두 개의 문자열을 비교하여 일치하는 가장 긴 패턴을 찾아서 화면에 출력하는 문자열 비교 프로그램을 만든다.
- `resultStr[2][SIZE*2]`에 저장하며 `resultStr[0]`과 `resultStr[1]`은 자리수를 맞춰야 한다.

2. 설계

- LCS알고리즘을 이용해 가장 긴 공통 수열을 찾을 수 있는 배열을 구성한뒤, 백트래킹을 이용해 찾는다.
- 백트래킹을 이용하면 값이 거꾸로 나오게 되므로, 이를 I, J인덱스와 함께 스택에 쌓아둔다.
- 스택에서 데이터를 pop하면서 resultStr[0]과 resultStr[1]에 데이터를 알맞게 위치 시킨다.

3. 알고리즘(LCS 배열 구성)

- 해당 문제 해결을 위해선 우선 가장 긴 공통 문자열을 찾아서 A와 B에서 해당 데이터에 맞춰 resultStr을 구성한다.
- 일치하는 가장 긴 패턴을 찾기 위해 LCS알고리즘을 응용한다.
- lcsArray[SIZE+1][SIZE+1]을 선언하고 lcsArray[0][j]와 lcsArray[i][0]은 모두 0으로 초기화 시킨다.

3. 알고리즘 (LCS 배열 구성) 계속

A = a, b, c, d, e, f

B = b, d, e, f, g, h 로 가정 하고

LCSArray 초기화 상태

		a	b	c	d	e	f
	0	0	0	0	0	0	0
b	0						
d	0						
e	0						
f	0						
g	0						
h	0						

3. 알고리즘 (LCS 배열 구성) 계속

- $lcsArray[i][j]$ 에서 $A[j-1]$ 의 문자와 $B[i-1]$ 의 문자가 일치하지 않다면, $lcsArray[i-1][j]$ 와 $lcsArray[i][j-1]$ 중 큰 수를 취한다.
- 만약 일치한다면, $lcsArray[i-1][j-1]$ 에서 +1을 더한 수를 취한다..

	-	a	b	c	d	e	f
-	0	0	0	0	0	0	0
b	0	0	1	1	1	1	1

3. 알고리즘 (LCS 배열 구성) 끝

완성된 LCSArray의 상태이며, 가장 큰 수가 최 하단에 위치하며 이 A, B에서 가장 긴 공통 수열의 길이는 4임을 알 수 있다.

	-	a	b	c	d	e	f
-	0	0	0	0	0	0	0
b	0	0	1	1	1	1	1
d	0	0	1	1	2	2	2
e	0	0	1	1	2	3	3
f	0	0	1	1	2	3	4
g	0	0	1	1	2	3	4
h	0	0	1	1	2	3	4

3. 알고리즘(LCS 백트래킹)

- 가장 긴 문자열을 구하기 위해 lcsArray를 백트래킹함.
- lcsArray[i][j]에서 A[j-1]과 B[i-1]의 문자열을 비교하고 일치하다면 스택에 i와 j를 넣고, i와 j를 하나씩 줄인다.
- 일치하지 않다면, lcsArray[i-1][j]과 lcsArray[i][j-1]중 숫자가 큰 쪽으로 이동한다.

3. 알고리즘 (LCS 백트래킹) 계속

최우측하단 부터 백트래킹을 시작한다.

$lcsArray[i][j]$ 에서 $A[j-1], B[i-1]$ 의 문자가 다르다면 좌측, 상측에서 큰 쪽으로 이동한다.

두 문자가 같다면 인덱스를 스택에 쌓고 좌측 대각선으로 이동한다.

	-	j	a	b	c	d	e	f
i	0	0	1	2	3	4	5	6
-	0	0	0	0	0	0	0	0
b	1	0	0	1	1	1	1	1
d	2	0	0	1	1	2	2	2
e	3	0	0	1	1	2	3	3
f	4	0	0	1	1	2	3	4
g	5	0	0	1	1	2	3	4
h	6	0	0	1	1	2	3	4

index	i	j
0	4	6

Stack의 상태

3. 알고리즘 (LCS 백트래킹) 계속

	-		a	b	c	d	e	f
	-	0	1	2	3	4	5	6
-	0	0	0	0	0	0	0	0
b	1	0	0	1	1	1	1	1
d	2	0	0	1	1	2	2	2
e	3	0	0	1	1	2	3	3
f	4	0	0	1	1	2	3	4
g	5	0	0	1	1	2	3	4
h	6	0	0	1	1	2	3	4

index	i	j
1	3	5
0	4	6

Stack의 상태

3. 알고리즘 (LCS 백트래킹) 계속

	-		a	b	c	d	e	f
-	0	0	0	0	0	0	0	0
b	1	0	0	1	1	1	1	1
d	2	0	0	1	1	2	2	2
e	3	0	0	1	1	2	3	3
f	4	0	0	1	1	2	3	4
g	5	0	0	1	1	2	3	4
h	6	0	0	1	1	2	3	4

index	i	j
2	2	4
1	3	5
0	4	6

Sack의 상태

3. 알고리즘 (LCS 백트래킹) 계속

A[3-1], B[1-1]의 문자가 다르므로 LCSArray[0][3]과 LCSArray[1][2]중 큰 쪽 LCSArray[1][2]로 이동한다.

	-		a	b	c	d	e	f
	-	0	1	2	3	4	5	6
-	0	0	0	0	0	0	0	0
b	1	0	0	1	1	1	1	1
d	2	0	0	1	1	2	2	2
e	3	0	0	1	1	2	3	3
f	4	0	0	1	1	2	3	4
g	5	0	0	1	1	2	3	4
h	6	0	0	1	1	2	3	4

index	i	j
2	2	4
1	3	5
0	4	6

Sack의 상태

3. 알고리즘 (LCS 백트래킹) 끝

위와 좌측이 모두 0이므로 백트래킹을 끝낸다.

	-		a	b	c	d	e	f
-	0	0	0	0	0	0	0	0
b	1	0	0	1	1	1	1	1
d	2	0	0	1	1	2	2	2
e	3	0	0	1	1	2	3	3
f	4	0	0	1	1	2	3	4
g	5	0	0	1	1	2	3	4
h	6	0	0	1	1	2	3	4

index	i	j
3	1	2
2	2	4
1	3	5
0	4	6

Sack의 상태

3. 알고리즘 (RESULT 구성)

index	i	j
3	1	2
2	2	4
1	3	5
0	4	6

Stack의 상태

iIndex = 0, jIndex = 0, firstK=0, secondK=0 부터 시작.

firstK, secondK 변수는 resultStr을 가르키기 위해 사용.

처음 Pop된 스택의 i-1은 0, j-1은 1이므로, A먼저 jIndex ~ stack.j 까지

resultStr에 누적 시키고 firstK를 증가..

firstK = 2가 된다.

	0	1	2	3	4	5
A[j]	a	b	c	d	e	f
B[i]	b	d	e	f	g	h

문자열의 상태

	0	1	2	3	4	...
Result[0]	a	b
Result[1]

3. 알고리즘 (RESULT 구성) 계속

index	i	j
3	1	2
2	2	4
1	3	5
0	4	6

Stack의 상태

	0	1	2	3	4	5
A[j]	a	b	c	d	e	f
B[i]	b	d	e	f	g	h

문자열의 상태

증가된 firstK까지 secondK와 iIndex를 증가시키며 B의 문자열을 넣는데,
길이를 맞추기 위해 ' '(공백)를 삽입한다.
firstK와 secondK는 둘다 2가 된다.

	0	1	2	3	4	...
Result[0]	a	b
Result[1]	공백	b

3. 알고리즘 (RESULT 구성) 계속

index	i	j
3	1	2
2	2	4
1	3	5
0	4	6

Stack의 상태

	0	1	2	3	4	5
A[j]	a	b	c	d	e	f
B[i]	b	d	e	f	g	h

문자열의 상태

다음 스택에서 pop된 데이터에서 jIndex-stack.j가 더 길기때문에 먼저 넣고 firstK를 증가시킨다.

	0	1	2	3	4	5	6	7
Result[0]	a	b	c	d				
Result[1]	공백	b						

3. 알고리즘 (RESULT 구성) 계속

index	i	j
3	1	2
2	2	4
1	3	5
0	4	6

Stack의 상태

	0	1	2	3	4	5
A[j]	a	b	c	d	e	f
B[i]	b	d	e	f	g	h

문자열의 상태

증가된 firstK까지 secondK를 증가시키며 B의 문자열을 넣는데, 길이를 맞추기 위해 ' '(공백)를 삽입한다.

	0	1	2	3	4	5	6	7
Result[0]	a	b	c	d				
Result[1]	공백	b	공백	d				

3. 알고리즘 (RESULT 구성) 계속

index	i	j
3	1	2
2	2	4
1	3	5
0	4	6

Stack의 상태

	0	1	2	3	4	5
A[j]	a	b	c	d	e	f
B[i]	b	d	e	f	g	h

문자열의 상태

ilindex – stack.i 와 jindex – stack.j가 같으므로 둘다 같이 resultStr에 삽입한다.

	0	1	2	3	4	5	6	7
Result[0]	a	b	c	d	e			
Result[1]	공백	b	공백	d	e			

3. 알고리즘 (RESULT 구성) 계속

index	i	j
3	1	2
2	2	4
1	3	5
0	4	6

Stack의 상태

	0	1	2	3	4	5
A[j]	a	b	c	d	e	f
B[i]	b	d	e	f	g	h

문자열의 상태

ilindex – stack.i 와 jindex – stack.j가 같으므로 둘다 같이 resultStr에 삽입하고, jindex가 문자열의 끝에 닿았으므로, B를 모두 삽입한뒤 resultStr[0]에는 길이에 맞게 공백을 넣은뒤 둘다 '\0'을 넣는다.

	0	1	2	3	4	5	6	7
Result[0]	a	b	c	d	e	f	공백	공백
Result[1]	공백	b	공백	d	e	f	g	h



Q & A