7주차 1차시 스트링 매칭의 이해

【학습목표】

1. 직선적 매칭과 라빈카프 알고리즘을 이해할 수 있다.
2. KMP 알고리즘을 이해할 수 있다.

학습내용1 : 직선적 매칭

* 스트링 매칭 문제란?
주어진 텍스트에서 주어진 패턴이 어디에 나타나지를 알아내는 문제이다. 예를 들어서 어떤 문서에서 특정 단어를 찾을 때 이것이 스트링 매칭이 되는데, 여기서 문서를 텍스트라 하며 찾아낼 특정단어를 패턴(Pattern)이라 한다.
텍스트 T[1..n]에서 패턴 P[1..m]과 일치하는 모든 부분을 찾아내는 문제이다.
스트링 매칭에는 직선적, 라빈-카프, KMP, 보이어무어(BM)알고리즘 등이 있다.

① 스트링매칭
- 주어진 텍스트에서 주어진 패턴이 어기에 나타나는지를 알아내는 문제
- 알파벳 Σ
- 텍스트 T[0..n-1], 크기 n
- 패턴 P[0,,m-1], 크기 m

② 스트링
- 문자가 연속적으로 나열된 것.
- 스트링 x의 길이 : |x|

③ 공 스트링 ε
- 길이 0인 스트링

④ 스트링 접속 xy
- 스트링 x뒤에 y를 붙인 것.
- x=wy
    - w : x의 접두부. w⊏x
    - y : x의 접미부. y⊐x
- y⊐x 이면 ya⊐xa 이고 그 역도 성립한다.
- 추이적 관계 : u⊏v, v⊏w → u⊏w

1. 직선적 알고리즘

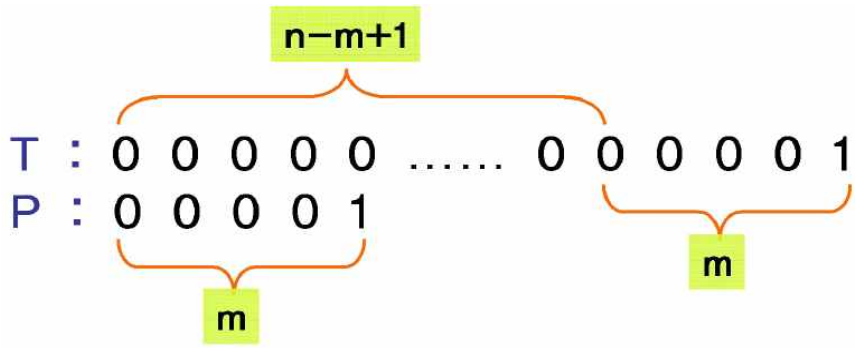
텍스트의 매 위치에서 패턴 일치가 발생하는 지를 조사하는 방법
그러면 실제로 간단한 예를 통하여 스트링 매칭을 하는 문제를 생각해 보자. 다음과 같이 텍스트 T와 패턴 P가 주어졌다고 하자.

텍스트 T: abgoodfathgoodkhfobj
패턴 P: good

T에서 P가 나타나는 부분을 찾는 직선적인 방법은 하나, 하나씩 차례로 비교해 나가는 것이다. 즉 T[0]와 P[0]로부터 하나씩 비교해 나가는데 이를 불일치가 발생할 때까지 반복한다. 위 예의 경우 처음부터 불일치가 발생한다. 패턴을 다시 오른쪽으로 한번 이동시켜 패턴의 처음부터 비교를 다시 반복한다. 예를 들어 아래 abgoodfathgoodkhfobj 에서 good를 찾을 때 [표 4-1]과 같이 찾는다.

```
텍스트 T : abgoodfathgoodkhfobj
   패턴 P: good
           good
            good 패턴의 위치2에서 발견
                 .
                 .
                    good 패턴의 위치10에서 발견
```

* 최악의 경우



O(m(n-m+1))=O(mn)

직선적 스트링 매칭 방법의 시간 복잡도는 O(mn)이며 좀 더 정확하게는 O(m(n-m+1))이 된다.

학습내용2 : 라빈카프 알고리즘

1. 라빈카프 알고리즘의 개요

라빈-카프 알고리즘은 최악의 시간복잡도는 여전히 O(m(n-m+1))이지만 평균적으로는 매우 빠른 알고리즘으로서 스트링을 숫자 값으로 바꾸어, 즉 해시(hash)값을 계산하여 매칭 하는 알고리즘이다.

※ 주요정리

◎ 라빈-카프 매칭법의 주요 아이디어는 스트링을 $|\Sigma|$=d진법의 숫자로 바꾸어 이 숫자를 비교하려는 방법 이다.

◎ 라빈-카프 방법에서 mod연산이 사용되는 이유는 숫자가 너무 커져서 오버플로우가 발생하는 것을 피하기 위함이다.

◎ 라빈-카프 방법의 시간복잡도를 따져보면 최악의 경우는 O(m(n-m+1))이지만 기대치는 O(n+m)이 된다,

◎ 접두부(prefix)와 접미부(suffix)를 설명하면 간단히 말해서 스트링의 앞부분과 뒤 부분을 말한다.

◎ 길이 0의 공 스트링을 ε으로 나타내기로 하고 스트링 x의 길이를 $|x|$로 나타낸다. 또 두 스트링 x, y의 접속, 즉 x 뒤에 y를 붙인 것을 xy로 표기한다.
w, x, y를 스트링이라 할 때, $x = wy$라면 w는 x의 접두부(prefix)라고 하며 $w \sqsubset x$로 표기 하기로 한다. 또한 y는 x의 접미부(suffix)라고 하며 $y \sqsupset x$로 나타낸다. 예를 들어 $ab \sqsubset goodfa$, $ab \sqsupset goodfa$로 표기한다.

* [보충학습] 호너의 방법

만약 패턴이 31413 이라면 텍스트와 패턴의 각위치의 숫자는 실제로는 문자 이다. 즉 31413라고 분류된다. 이것을 10진수로 표현하기 위해선 호너의 방법을 쓸 수 있다.

$$p = P[0]10^{m-1} + P[1]10^{m-2} + \cdots + P[m-1]10^0$$
$$= P[m-1] + 10(P[m-2] + 10(P[m-3] + \cdots + 10(P[1] + 10P[0])\cdots)$$

p=31413
$=3*10^4 + 1 * 10^3 + 4*10^2 + 1 * 10^1 + 3 *10^0$
$=3 + 10(1 +10(4 +10(1 +10 * 3))$
와 같이 계산하여 10진수로 바꿀 수 있다

라빈-카프 알고리즘을 간단한 예로 설명을 하려 한다.
간단한 라빈-카프 알고리즘의 실행 예

텍스트 T :

2	7	3	1	2	7

패턴 P :

1	2	7

T0

2	7	3

T1

7	3	1

T2

3	1	2

T3

1	2	7

해시함수의 나머지법을 이용하여 나머지를 구한다.(통상 소수를 선택한다)

패턴 P 의 나머지 127 mod 7 = 1
T0 의 나머지 273 mod 7 = 0
T1 의 나머지 731 mod 7 = 3
T2 의 나머지 312 mod 7 = 4
T3 의 나머지 127 mod 7 = 1

패턴 P 의 나머지1 과 텍스트 T3 의 나머지1 이 같음으로 일치하는 문자열을 찾게 된다.

학습내용3 : KMP 알고리즘

1. KMP 알고리즘의 개요

kmp 알고리즘은 Knuth-Morris-Pratt 세명의 이름을 따서 만든 알고리즘으로 이 알고리즘의 목적은 최저의 시간으로 문자열 속의 패턴을 찾아내는 것이다.
KMP 알고리즘은 패턴의 최대 접미부와 접두부 쌍을 구하여 불일치 발생시 텍스트의 앞부분으로 돌아감이 없이 패턴 매칭을 하는 방법을 사용한다. 그러므로 이미 비교한 텍스트의 앞부분을 다시 비교하는 일이 없게 된다. 패턴을 전처리하여 접미부와 최대로 일치하는 진접두부를 구하고 이를 이용하여 최대 접두부 테이블 SP에 만들어 놓는다.
텍스트의 위치와 패턴의 위치를 비교한 결과 불일치가 발생하였다면 다음에 비교할 위치는 일치하지 않는 부분의 왼쪽 부분을 가지고 경계를 찾는다. 경계를 찾지 못하면 문자열과 패턴의 틀린 부분으로 이동한다.
직선적 방법이나 라빈-카프의 방법 모두 최악의 시간복잡도는 $O(mn)$이 되지만 KMP 알고리즘의 시간 복잡도는 $O(m+n)$이 된다.

보충학습

접두부(Prefix)라 하고 접미부(Suffix)라 한다.

접두부 는 문자열의 머리 부분이고 접미부는 꼬리 부분이 된다.

접두부		접미부
C		A
C		AA
CAA		CAA
CAAC		ACAA
CAACA		CACAA
CAACAC		ACACAA
CAACACA		AACACAA

[표 4-5] 접두부 와 접미부

그림은 접두부 접미부 그리고 경계를 설명하기 위해 CAACACAA 라는 문자를 설명하는 그림이다. 그림에서 연회색이 접두부 이고 머리부분을 뜻하며 연회색 부분은 모두 접두부가 될 수 있다. 연분홍색 부분은 접미부로 꼬리부분을 뜻하는데 연분홍색 부분은 모두 접미부가 될 수 있다.

CAACACAA 문자열의 전체는 접두부 접미부 모두가 될 수 있기 때문에 둘 다 아니다.

C	A	A	C	A	C	A	A
접두부			경	계	접미부		

[표 4-6] 접두부와 접미부 표시

위의 표에서 가운데 CA를 접두부와 접미부의 경계라 한다.

※ kmp 알고리즘은 다음과 같다

1. 비교할 패턴을 문자열의 첫 번째에 위치시킨다.
2. 먼저 비교할 패턴을 문자열의 비교할 대상과 위치시키고 비교를 한다.
3. 패턴의 왼쪽에서 오른쪽으로 비교를 하면서 일치하지 않는 부분을 찾는다.
   일치 하는 문자열을 찾으면 성공 종료 한다.
4. 일치하지 않는 부분의 왼쪽 부분을 가지고 경계를 찾는다.
   경계를 찾지 못하면 문자열과 패턴의 틀린 부분만큼 이동한다.
5. 경계를 찾으면 접미부와 접두부가 나오는데 경계가 1개면 상관이 없지만 2개 이상도 나올 수 있다. 여기서 선택하는 경계는 경계의 최대가 되었을 경우를 설정해 주는 것이다,
그리고 최대경계일 경우 접두부 시작 위치에서 접미부 위치로 이동을 시켜준다.
6. 이동을 하면 2번으로 이동하여 반복적으로 수행 한다.

2. KMP 알고리즘의 예

텍스트 T : CAACAACAC
패턴 P : CAACAC

C	A	A	C	A	A	C	A	C

C	A	A	C	A	C

C	A	A	C	A	A	C	A	C
접두부			접미부					

접미부의 시작 위치로 이동 시킨다

C	A	A	C	A	A	C	A	C

	C	A	A	C	A	C

이동

일치 하는 문자열을 찾는데 성공

3. kmp 알고리즘 실행 예

※ SP 테이블 만들기 (이동 경로 테이블)
CAACACAA를 가지고 예를 들어보면

일치 접두부의 길이	0	1	2	3	4	5	6	7	8
문자열	C	A	A	C	A	C	A	A	
최대 경계너비					1				

일치 접두부의 길이 : 일치 하는 부분
(만약 일치 접두부 길이가 4라면 CAAC 까지만 일치하고 4가 위치한 A가 다르다는 뜻이다.)
문자열 : 패턴을 의미한다.
최대 경계너비 : 일치하는 부분을 찾고 그 곳에서 접두부와 접미부를 찾았을때 경계가 가장 최대가 되었을 경우 접두부와 접미부의 길이 이다.

예를 들어보면 CAAC 일때

접두부	경계	접미부
C	AA	C
CA		AC

위 표에서 경계가 최대가 AA인 경우이다. 이 경우 접두부와 접미부의 길이를 최대 경계너비에 설정해주면 C하나가 됨으로 1이 된다.
이동하는 공식 : 이동길이 = 일치 접두의 길이 - 최대 경계 너비 길이
예를 들어 일치 접두부 길이가 4라면 경계너비는 1이 됨으로 3만큼 이동 하게 되는 것을 의미한다.

테이블을 만들고 최대 경계너비만 설정해주면 된다.

일치 접두부의 길이	0	1	2	3	4	5	6	7	8
문자열	C	A	A	C	A	C	A	A	
최대 경계너비									

① 일치하는 접두부의 길이가 0인 경우 (문자열과 패턴이 0번째 즉 처음부터 틀렸을 때)
- 이 경우에는 1칸만 이동을 해야 한다.
- 이동경로 = 일치 접두부 길이 - 최대 경계너비 (0 -1 =-1) 이 된다.

② 일치 접두부의 길이 (1~3)까지 진행함에 있어 경계가 나오지 않는다. 그래서 0으로 설정

일치 접두부의 길이	0	1	2	3	4	5	6	7	8
문자열	C	A	A	C	A	C	A	A	
최대 경계너비	-1	0	0	0					

③ 일치 접두부의 길이 4

- 이 경우 CAAC에서 경계를 " C AA C"가 됨으로 경계의 최대 길이가 AA 이기 때문에 접두부와 접미부는 B가 된다. 그래서 B의 길이인 1이 된다.

일치 접두부의 길이	0	1	2	3	4	5	6	7	8
문자열	C	A	A	C	A	C	A	A	
최대 경계너비	-1	0	0	0	1				

④ 일치 접두부 길이가 5인 경우 "CAACA"가 됨으로 경계는 "CA A CA"에서 A가 된다. 즉 접두부와 접미부 길이가 CA인 2가 된다.

일치 접두부의 길이	0	1	2	3	4	5	6	7	8
문자열	C	A	A	C	A	B	A	A	
최대 경계너비	-1	0	0	0	1	2			

⑤ 일치 접두부의 길이(6~7)까지 진행함에 있어 경계가 나오지 않는다. 그러므로 0이 된다.

일치 접두부의 길이	0	1	2	3	4	5	6	7	8
문자열	C	A	A	C	A	C	A	A	
최대 경계너비	-1	0	0	0	1	2	0	0	

⑥ 일치 접두부의 길이 8 경우 (문자열과 패턴이 모두 일치할 때)

- 모두 일치 한다고 해서 패턴만큼 모두 이동하면 안 된다.

_ CAACACAACACAA 문자열이 있을때

CAACACAA 패턴이 모두 일치 한다고 해서

- CAACACAACACAA

--------->CAACACAA 패턴 길이만큼 이동하면

CAACACAACACAA

CAACACAA 처럼 중간에 일치하는 부분을 지나칠 수 있다.

그래서

CAACACAA 를 "CAA CA CAA"로 하면 경계가 CA가 되고 접두부와 접미부가 "CAA"인 3이 된다.

일치 접두부의 길이	0	1	2	3	4	5	6	7	8
문자열	C	A	A	C	A	C	A	A	
최대 경계너비	-1	0	0	0	1	2	0	0	3

실제 예제를 가지고 확인해보자

텍스트 T : CACBACACAACACAA
패턴 P: CAACACAA

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

일때

①먼저 찾으려는 문자열이 CAACACAA이다.
테이블 공식 이동경로 = 일치 접두부길이 - 최대 경계너비

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

② 첫 위치에 설정하고 비교를 하면 3번째 A가 틀리게 된다.
일치 접두부의 길이 CA 인 2가 되고 최대 경계너비는 경계가 없음으로 0이 된다.
그래서 2-0 은 2가 됨으로 2칸 이동 하게 된다.

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

2칸이동

③ 다시 비교 2번째 A가 틀림을 알 수 있다.
이동경로 = 일치 접두부길이 - 최대 경계너비 에 의해서 (1-0) 이 됨으로 1칸 이동

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

1칸이동

④ 다시 비교 하면 첫 번째 부터 다름을 알 수 있다.

이동경로 = 일치 접두부길이 - 최대 경계너비 에 의해서 (0-(-1))이 됨으로 1칸 이동

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

1칸이동

⑤ 다시비교를 하니 3번째 A가 틀림을 알 수 있다.

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

이동경로 = 일치 접두부길이 - 최대 경계너비 에 의해서 (2-0)이 됨으로 2칸 이동하게 된다.

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

⑥ 다시 비교 하니 모두 일치 하나 중복의 경우를 생각하여 이동경로 공식에 대입
이동경로 = 일치 접두부길이 - 최대 경계너비 에 의해서 (8-3)이 됨으로 5칸 이동

C	A	C	B	A	C	A	C	A	A	C	A	C	A	A

C	A	A	C	A	C	A	A

위와 같이 이동하여 빠르게 매칭을 찾게 된다.

4. 스트링 매칭의 성능비교

스트링 매칭	수행시간	
	평 균	최 악
직선적 알고리즘	O(nm)	
라빈-카프 알고리즘	O(n+ m)	○(m(n-m+ 1))
KMP 알고리즘	O(n+ m)	
보이어-무어 알고리즘	O((n-m+ 1)m+ |∑|)	

▶ 라빈-카프알고리즘
최악의 실행시간 ○(m(n-m+1))
- 스트링을 숫자값으로 바꾸어 계산한다.
- 해시값을 계산하여 매치하는 알고리즘을 찾는다.
- 텍스트의 해시값과 패턴의 해시 값을 비교한다.
- 해시값이 커지는 것을 방지하기 위해서 Module 연산을 한다.

▶ KMP알고리즘
- 시간복잡도는 ○(m+n)
- 패턴을 진처리하여 진접두부를 구하여, 최대 접두부 테이블 SP에 만들어 놓는다.
- 최대접두부 테이블 계산 시간 ○(m) ,매칭○(n) 이다.

【학습정리】

1. 스트링 매칭
- 스트링 매칭이란 텍스트 T[1..n]에서 패턴 P[1..m]과 일치하는 모든 부분을 찾아내는 문제
- x=wy일 때 w는 x의 접두부, y는 x의 접미부

2. 직선적 알고리즘
- 가장 간단한 방법으로, 텍스트의 매 위치에서 패턴 일치가 발생하는지를 조사하는 방법
- 최악의 경우는 텍스트의 매 위치에서 패턴의 모든 문자를 비교하는 경우 예: T=00000...000001, P=00001
- 시간복잡도: O(m(n-m+1)) → O(mn)

3. 라빈-카프 알고리즘
- 스트링을 |Σ|=d진법의 숫자로 바꾸어, 즉 해시값을 계산하여 이를 비교하는 방법
- 스트링을 숫자로 바꾸는 과정에서 숫자가 너무 커져서 오버플로가 발생하는 것을 피하기 위해서 mod 연산을 사용.
- 최악의 실행시간: O(m(n-m+1) ← 패턴의 해시값과 텍스트의 해시값이 모두 일치하여 다시 일일이 문자를 비교해야 하는 경우.
- 최선의 경우: O(m+n) ← 패턴의 해시값과 텍스트의 해시값이 모두 불일치하는 경우
- 실제로 대부분의 경우 해시값이 불일치하는 경우가 많기 때문에 실행시간의 기대치는 O(m+n)

4. KMP 알고리즘
- 패턴의 최대 접미부-접두부 쌍을 구하여 불일치 발생시 텍스트의 앞부분으로 돌아감이 없이 패턴 매칭을 하는 방법.
- KMP 알고리즘의 경우 이미 비교한 텍스트의 앞부분을 다시 비교하는 일이 없다.
- 패턴을 전처리하여 접미부와 최대로 일치하는 진접두부를 구하여 이를 최대 접두부 테이블 SP에 만들어 놓는다.
- 텍스트의 위치 i와 패턴의 위치 j를 비교한 결과 불일치가 발생하였다면 다음에 비교할 위치는 T[i]와 P[SP[j-1]+1]이다.
- 시간복잡도 O(m+n) ← SP 계산 O(m), 매칭 O(n)