



11주차

# 알고리즘 스터디

By. 선민기



# 목 차

○ 01 라빈카프 알고리즘

○ 부)1-1 롤링해시

○ 02 레벤슈타인 거리

○ 03 보이드무어

# 01

## 라빈카프 알고리즘

---

### 문자열 매칭 알고리즘

---

문자열 P에서 패턴 S를 탐색

### 롤링해시 사용

---

문자를 하나씩 비교하지 않고  
해싱 된 값을 사용



### $O(N)$ 의 시간복잡도

---

이전 단계에서 구한 해시 값을  
다음에도 사용

### 해시 충돌 처리 필요!!

---

같은 해시 값을 가지는  
데이터(해시 충돌)은 체이닝

## 부. 1-1

### 롤링해시

---

#### ○ 롤링해시?

예시로 설명합니다.

#### ○ 롤링해시 공식

$$H[i] = S[i] * 2^{M-1} + S[i+1] * 2^{M-2} + \dots + S[i+M-2] * 2^1 + S[i+M-1] * 2^0$$

#### ○ 예시1

$$\begin{aligned} ABC &= A \text{의 아스키코드} \times 2^2 + B \text{의 아스키코드} \times 2^1 + \dots \\ &= 65 \times 2^2 + 66 \times 2^1 + 67 \times 2^0 \\ &= 459 \end{aligned}$$

## 부. 1-1

### 롤링해시

---

#### ○ 예시2

$$\begin{aligned} ABC &= A\text{의 아스키코드} \times 2^2 + B\text{의 아스키코드} \times 2^1 + \dots \\ &= 65 \times 2^2 + 66 \times 2^1 + 67 \times 2^0 \\ &= 459 \end{aligned}$$

$$\begin{aligned} BCD &= 66 \times 2^2 + 67 \times 2^1 + 68 \times 2^0 \\ &= (ABC \text{ 해시} - A\text{의 아스키코드} \times 2^2) \times 2 + D\text{의 아스키코드} \times 2^0 \\ &= (459 - 260) \times 2 + 68 \\ &= 466 \end{aligned}$$

## 부. 1-1

### 롤링해시


#### 예시3


패턴 : "IC"


타겟 : "TWICE"

롤링 해시값 :  $73 * 2 + 67 * 1 = 213$

T	W	I	C	E
---	---	---	---	---


$$(84 * 2 + 87 * 1 = 255) \neq 213$$


$$((255 - 84 * 2) * 2 + 73 * 1 = 247) \neq 213$$


$$((247 - 87 * 2) * 2 + 67 * 1 = 213) == 213$$

## 02

### 레벤슈타인 거리

#### ○ 개요

두 문장이 차이를 수치로 나타낸 것,  
두 문장이 얼마나 유사한지를 값으로 나타내는 방법,  
문자를 삽입, 삭제, 치환해 다른 문자열로 변형에 최소 필요 횟수

#### 비교 대상

도라에모용

5글자



도라캡틴

4글자



# 둘이 얼마나 다른 걸까??

## 02

### 레벤슈타인 거리

#### 알고리즘 설명

1

문장1길이+1 X문장2길이+1인 2차원 배열 필요,  
배열[i][0] = 둘다 길이가 0인 문자열인 경우 => 0으로 세팅

2

배열[i][j] = 1 ~ 문자열 길이로 세팅,  
배열[j][0] = 1 ~ 문자열 길이로 세팅,

3

If 두 문자열의 길이가 같음 && 새로 추가된 문자가 동일:  
대각선의 값을 그대로 가져와서 입력  
Else:  
상, 좌, 대각선 중 가장 작은 값 + 1 입력





02

레벤슈타인 거리

예시



	NULL	도	라	애	모	옹
NULL	0	1	2	3	4	5
도	1	0	1	2	3	4
라	2	1	0	1	2	3
캡	3	2	1	1	2	3
틴	4	3	2	2	2	3

## 03

### 보이드 무어 알고리즘

---

#### 문자열 매칭 알고리즘

---

문자열 P에서 패턴 S를 탐색



#### 나쁜문자 탐색

---

불일치한 문자가 있는 위치까지  
그 문자와 일치하도록 점프

#### 착한 접미부 탐색

---

탐색범위가 역행하는 것을  
막음

#### $O(N)$ 의 시간복잡도

---

일반적으로 KMP 알고리즘보다  
높은 성능을 보임

# 03

## 보이드 무어 알고리즘

### ○ 나쁜문자?

패턴의 오른쪽 부터 탐색했을 때 타겟 문자열과 가장 처음으로 일치하지 않는 문자

	0	1	2	3	4	5	6	7	8	9	
T	a	b	a	a	b	a	b	a	c	b	a
P	c	a	<sup>#</sup> b	<sup>"</sup> a	<sup>"</sup> b						

### ○ 착한 접미부?

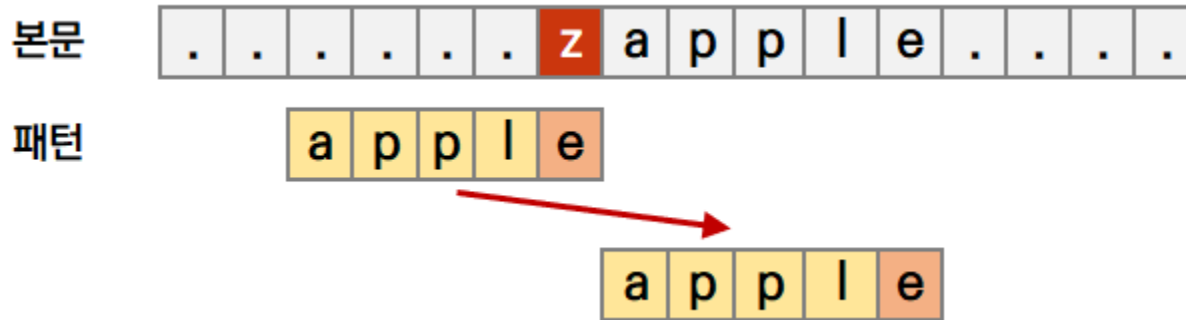
패턴의 맨 오른쪽 ~ 일치하지 않는 문자 까지의 문자열

	0	1	2	3	4	5	6	7	8	9	
T	a	b	a	a	b	a	b	a	c	b	a
P	c	a	b	a	b						
						c	a	b	a	b	

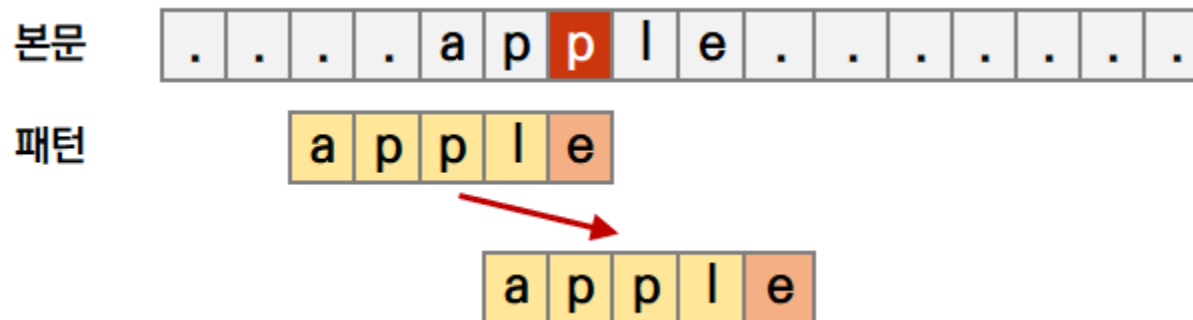
# 03

## 보이드 무어 알고리즘

### 예시 1



### 예시 2



# 03

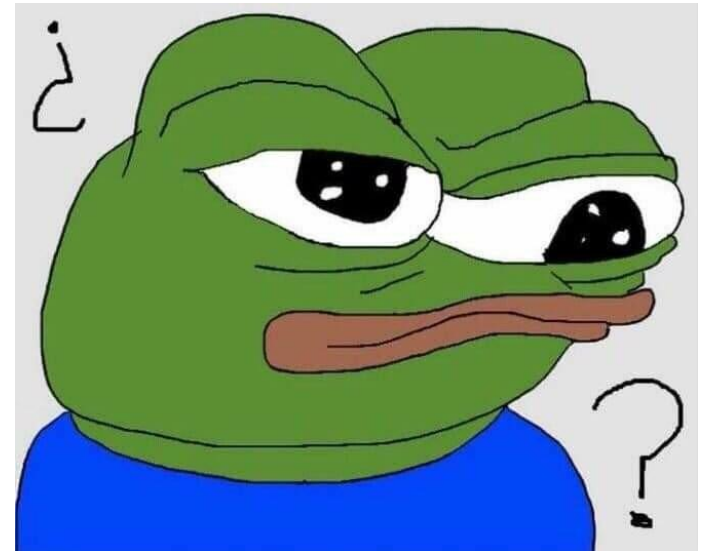
## 보이드 무어 알고리즘

### ○ 예외상황

T: 텍스트 P: 패턴

	0	1	2	3	4	5	6	7	8	9	
T	a	b	a	a	b	a	b	a	c	b	a
P			a	a	a	b					
	a	a	a	a	b						

왜 뒤로 가지??



착한 접미부를 써!!

## 03

### 보이드 무어 알고리즘

---

#### 예시 3

*T*: 텍스트 *P*: 패턴

	0	1	2	3	4	5	6	7	8	9	
<i>T</i>	a	b	a	a	b	a	b	a	c	b	a
<i>P</i>	c	a	b	a	b						
				c	a	b	a	b			

#### 예시 4

*T*: 텍스트 *P*: 패턴

	0	1	2	3	4	5	6	7	8	9	
<i>T</i>	a	b	c	a	b	a	b	a	c	b	a
<i>P</i>	c	b	a	a	b						
						c	b	a	a	b	

# 03

## 보이드 무어 알고리즘

### 예시 5

*T*: 텍스트 *P*: 패턴

	0	1	2	3	4	5	6	7	8	9		
<i>T</i>		a	a	b	a	b	b	a	b	c	b	a
<i>P</i>		a	b	b	a	b						
					a	b	b	a	b			





Q&A  
감사합니다