


Advanced 양성 과정

**문자열**

# 차례

 해싱(Hashing)


 문자열

 패턴 매칭

- ✓ 고지식한 패턴 검색 알고리즘
- ✓ KMP 알고리즘
- ✓ 보이어-무어 알고리즘


**해싱(HASHING)**

# 문제제시 : 파일 이름으로 바로 찾기

 대부분의 파일시스템들은 하나의 디렉토리에 존재가능한 파일의 수에 제한이 없다. 따라서, 다수의 디렉토리마다 대량의 파일들이 존재할 수 있다.


 아래와 같은 작업들이 빈번하게 수행되는 경우를 생각해보자.

- ✓ 디렉토리 내에 존재하는 파일들을 나열하기
- ✓ 디렉토리 경로를 따라가기
- ✓ 특정 파일의 존재 유무를 판별하기

 성능 저하의 원인이 될 것이다.

 해결책?

# 해싱 (Hashing)

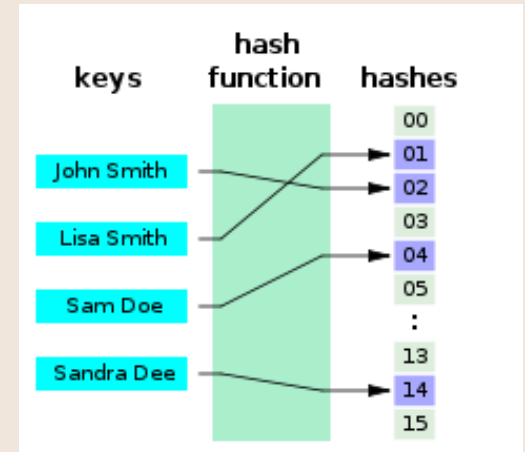
 특정 항목을 검색하고자 할 때, 탐색키를 이용한 산술적 연산을 이용해 키가 있는 위치를 계산하여 바로 찾아가는 방법

## 해시 함수(hash function)

✓ 탐색키를 항목의 위치로 변환하는 함수

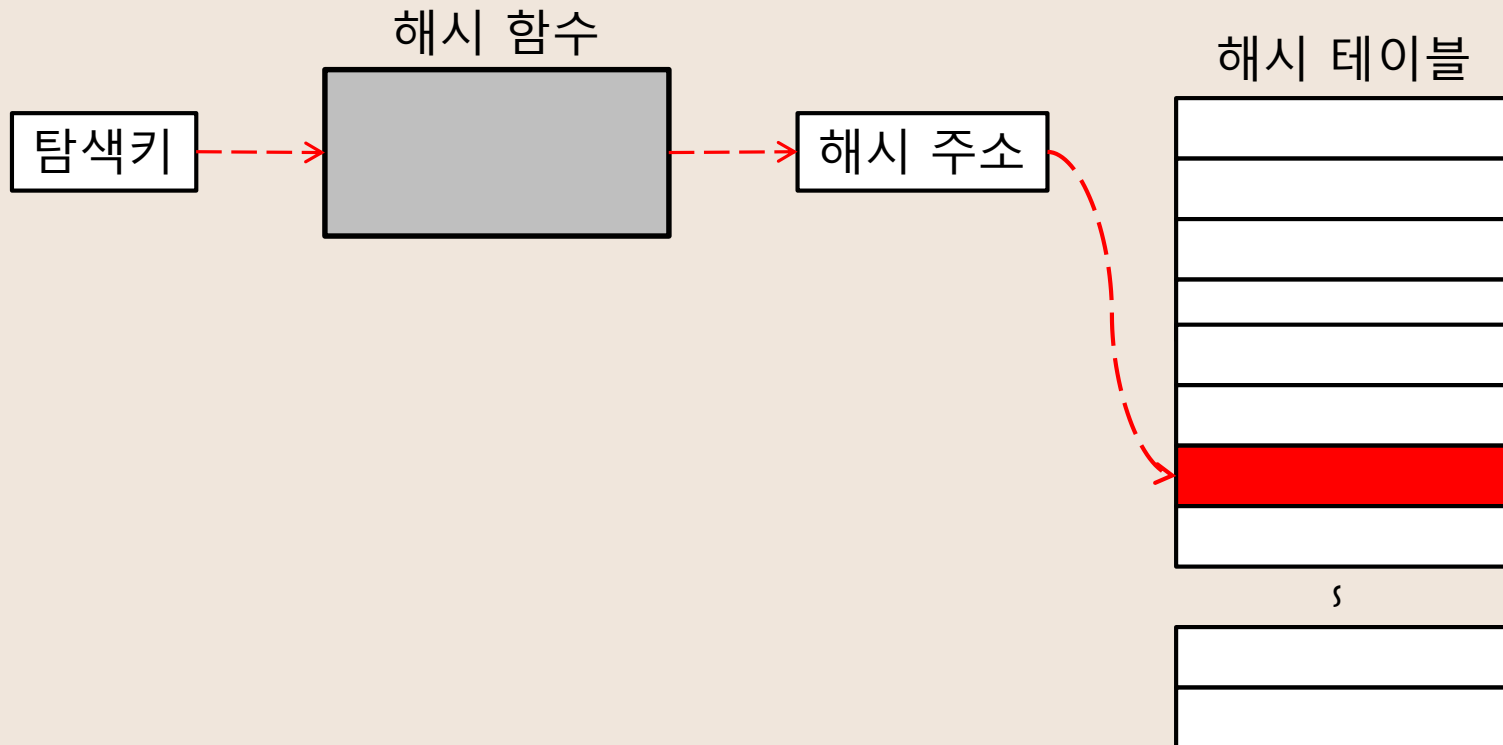
## 해시 테이블(hash table)

✓ 해싱 함수에 의해 반환된 주소의 위치에 항목을 저장한 표



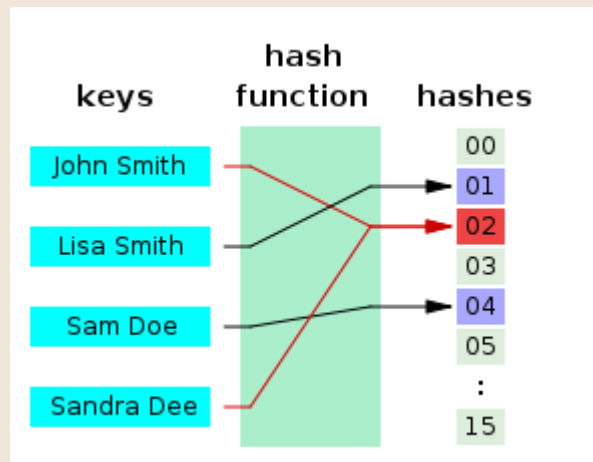
## 해시 검색 과정

- ✓ 해시 함수에 탐색키를 입력하여 주소를 구하고  
구한 주소에 해당하는 해시 테이블로 이동
- ✓ 해당 주소에 원하는 항목이 있으면 검색 성공, 없으면 실패



## 충돌 (Collision)

- ✓ 서로 다른 탐색키를 해시 함수에 적용하였는데, 반환된 해시 주소는 동일한 경우
- ✓ 해시 함수가 아무리 해시 주소를 공평하게 분배한다고 해도, 해시 테이블에 저장되는 자료의 수가 증가하면서 충돌은 불가피하다고 할 수 있다.



충돌: 서로 다른 이름이  
동일한 해시 주소를 가리키는 경우

## 충돌에 대한 해결방법

- ✓ 개방 주소법(Open addressing)
- ✓ 체이닝(chaining)

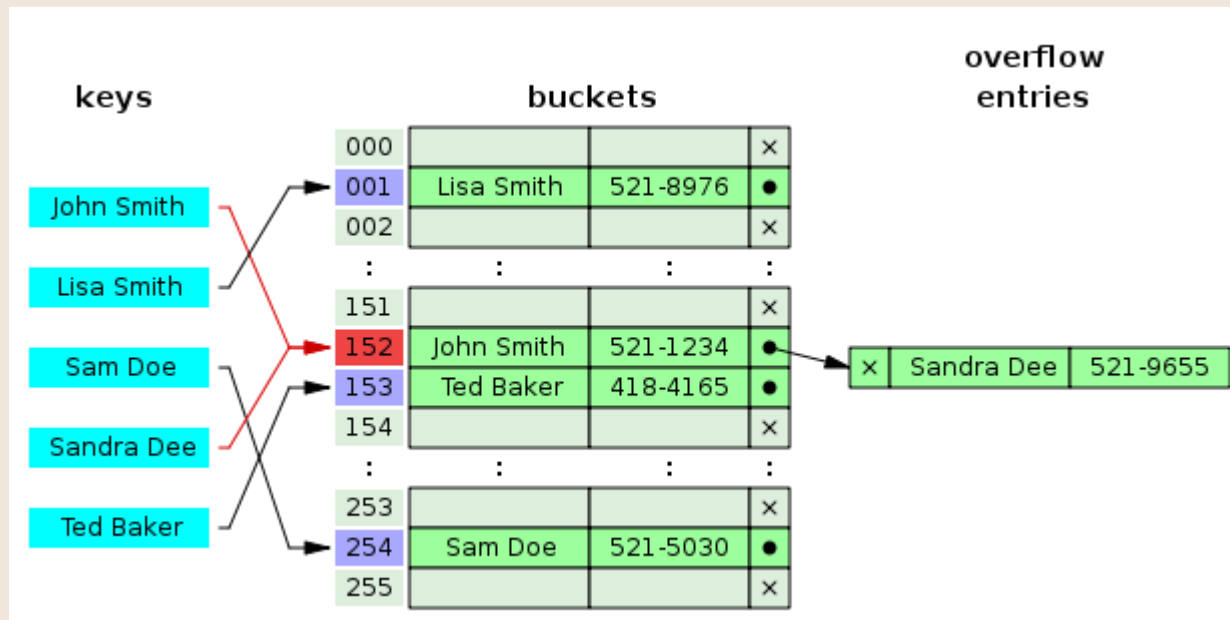
## 체이닝

- ✓ 해시 테이블의 구조를 변경하여 각 버킷에 하나 이상의 키 값을 가지는 자료가 저장될 수 있도록 하는 방법
- ✓ 하나의 버킷에 여러 개의 키값을 저장하도록 하기 위해 연결 리스트를 활용함



## 예: 그림

- ✓ John Smith와 Sandra Dee의 해시 주소 간 충돌이 발생한다.
- ✓ John Smith가 저장된 버킷에 연결하여 Sandra Dee를 저장하기 위한 리스트 노드를 생성하고 값을 저장한다

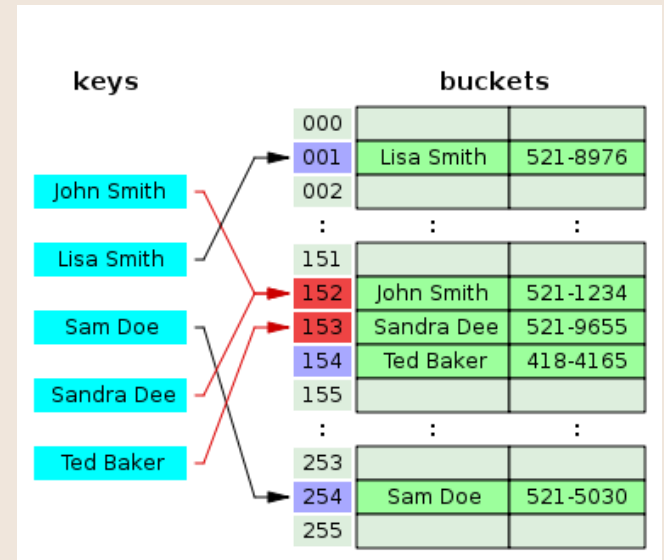


## ✎ 개방 주소법 (Open Addressing)

- ✓ 해시 함수로 구한 주소에 빈 공간이 없어 충돌이 발생하면, 그 다음 공간에 빈 공간이 있는지 조사한다.
  - ✦ 빈 공간이 있으면, 탐색키에 대한 항목을 저장한다.
  - ✦ 빈 공간이 없으면, 공간이 나올때까지 탐색을 반복한다.

### ✓ 예: 오른쪽 그림

- ✦ John Smith와 Sandra Dee가 충돌한다.
- ✦ 이에 Sandra Dee가 다음 공간에 저장된다.
- ✦ Ted Baker는 원래 153에 저장되어야 했으나, 뒤로 밀려 154에 저장된다.



💡 주어진 문제를 풀  
어보세요.

# Earth Day Word Search

Try to find all of the hidden Earth Day words in the word puzzle below.  
Remember, words can be diagonal, vertical, horizontal, frontward or backwards.

F S U A R B L L I K O T J N V  
J T Q O J C U N V I R P K N J  
K N Y X E E R G I E U O V U T  
W A T E R E C Y C L E L L C N  
V L L B U X O A T T R L C O E  
I P L S O I L D G E G U I H M  
J G E C F I I H T E I T C T N  
X N Z O G J I T A E A I R Z O  
W O W T P Z I R E V N O P M R  
G L O B A L W A R M I N G K I  
H F T L X J E E E T M T X Y V  
M H N Y N N S T D N A F N E N  
G X Q D Q N L O U H L V B B E  
R C I N O X X S C G S O U T H  
Z B M C N D D W E W X T R U K

Earth Day  
Environment  
Conservation

Reduce  
Reuse  
Recycle

Air  
Soil  
Water

People  
Plants  
Animals

Litter  
Pollution  
Global Warming

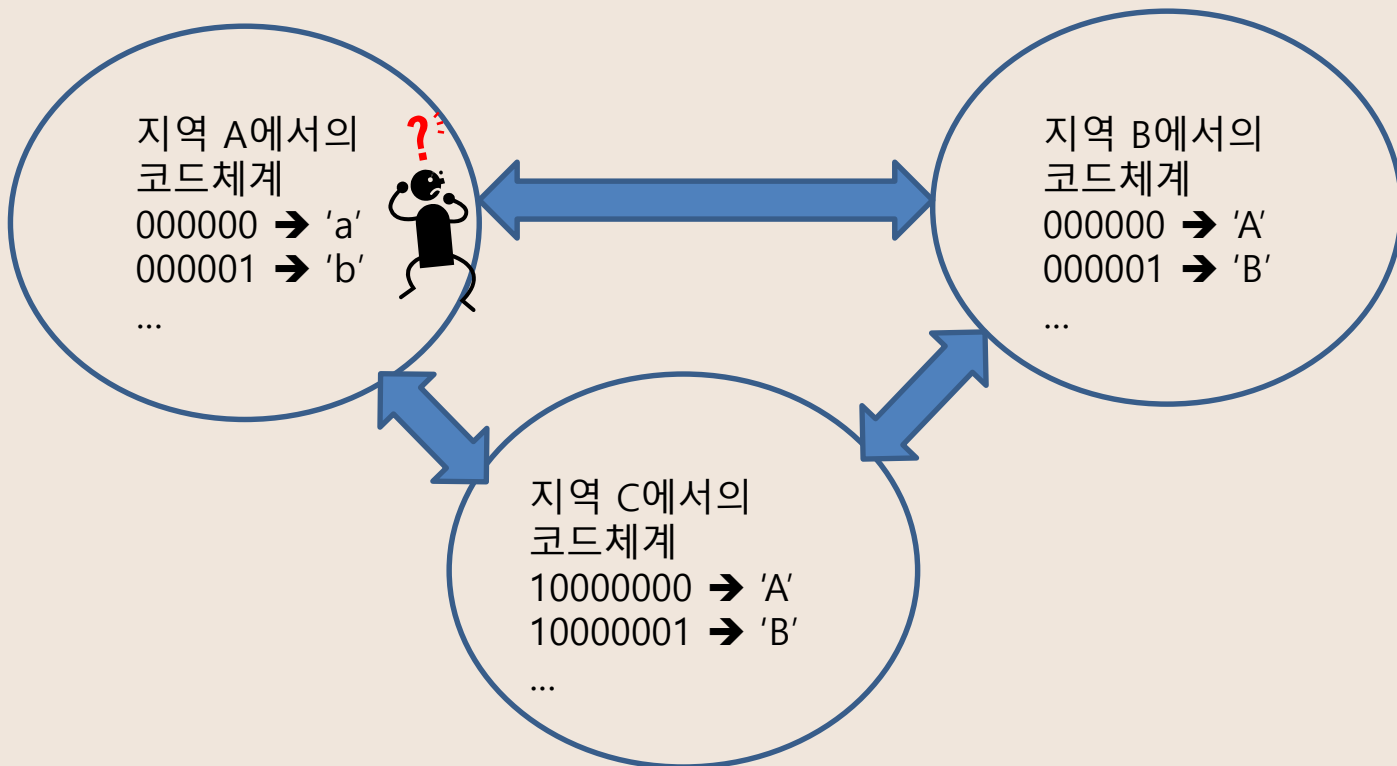
**문자열(String)**

# 문자의 표현

## 컴퓨터에서의 문자표현

- ✓ 글자 A를 메모리에 저장하는 방법에 대해서 생각해보자
- ✓ 물론 칼로 A라는 글자를 새기는 방식은 아닐 것이다. 메모리는 숫자만을 저장할 수 있기 때문에 A라는 글자의 모양 그대로 비트맵으로 저장하는 방법을 사용하지 않는 한(이 방법은 메모리 낭비가 심하다) 각 문자에 대해서 대응되는 숫자를 정해 놓고 이것을 메모리에 저장하는 방법이 사용될 것이다.
- ✓ 영어가 대소문자 합쳐서 52 이므로 6(64가지)비트면 모두 표현할 수 있다. 이를 코드체계라고 한다.
  - ★ 000000 → 'a', 000001 → 'b'

- ✓ 그런데 네트워크가 발전되기 전 미국의 각 지역 별로 코드체계를 정해 놓고 사용했지만
- ✓ 네트워크(인터넷 : 인터넷은 미국에서 발전했다)이 발전하면서 서로간에 정보를 주고 받을 때 정보를 달리 해석한다는 문제가 생겼다.




- ✎ 그래서 혼동을 피하기 위해 표준안을 만들기로 했다.
- ✎ 바로 이러한 목적으로 1967년, 미국에서 ASCII(American Standard Code for Information Interchange)라는 문자 인코딩 표준이 제정되었다.
- ✎ ASCII는 7bit 인코딩으로 128문자를 표현하며 33개의 출력 불가능한 제어 문자들과 공백을 비롯한 95개의 출력 가능한 문자들로 이루어져 있다.

## 출력 가능 아스키 문자 (32 ~126)

ASCII		ASCII		ASCII		ASCII		ASCII		ASCII	
32		48	<b>0</b>	64	<b>@</b>	80	<b>P</b>	96	<b>`</b>	112	<b>p</b>
33	<b>!</b>	49	<b>1</b>	65	<b>A</b>	81	<b>Q</b>	97	<b>a</b>	113	<b>q</b>
34	<b>"</b>	50	<b>2</b>	66	<b>B</b>	82	<b>R</b>	98	<b>b</b>	114	<b>r</b>
35	<b>#</b>	51	<b>3</b>	67	<b>C</b>	83	<b>S</b>	99	<b>c</b>	115	<b>s</b>
36	<b>\$</b>	52	<b>4</b>	68	<b>D</b>	84	<b>T</b>	100	<b>d</b>	116	<b>t</b>
37	<b>%</b>	53	<b>5</b>	69	<b>E</b>	85	<b>U</b>	101	<b>e</b>	117	<b>u</b>
38	<b>&amp;</b>	54	<b>6</b>	70	<b>F</b>	86	<b>V</b>	102	<b>f</b>	118	<b>v</b>
39	<b>'</b>	55	<b>7</b>	71	<b>G</b>	87	<b>W</b>	103	<b>g</b>	119	<b>w</b>
40	<b>(</b>	56	<b>8</b>	72	<b>H</b>	88	<b>X</b>	104	<b>h</b>	120	<b>x</b>
41	<b>)</b>	57	<b>9</b>	73	<b>I</b>	89	<b>Y</b>	105	<b>i</b>	121	<b>y</b>
42	<b>*</b>	58	<b>:</b>	74	<b>J</b>	90	<b>Z</b>	106	<b>j</b>	122	<b>z</b>
43	<b>+</b>	59	<b>;</b>	75	<b>K</b>	91	<b>[</b>	107	<b>k</b>	123	<b>{</b>
44	<b>,</b>	60	<b>&lt;</b>	76	<b>L</b>	92	<b>\</b>	108	<b>l</b>	124	<b> </b>
45	<b>-</b>	61	<b>=</b>	77	<b>M</b>	93	<b>]</b>	109	<b>m</b>	125	<b>}</b>
46	<b>.</b>	62	<b>&gt;</b>	78	<b>N</b>	94	<b>^</b>	110	<b>n</b>	126	<b>~</b>
47	<b>/</b>	63	<b>?</b>	79	<b>O</b>	95	<b>_</b>	111	<b>o</b>		



 **확장 아스키는 표준 문자 이외의 악센트 문자, 도형 문자, 특수 문자, 특수 기호 등 부가적인 문자를 128개 추가할 수 있게 하는 부호이다.**

- ✓ 표준 아스키는 7bit를 사용하여 문자를 표현하는 데 비해 확장 아스키는 1B 내의 8bit를 모두 사용함으로써 추가적인 문자를 표현할 수 있다.
- ✓ 컴퓨터 생산자와 소프트웨어 개발자가 여러 가지 다양한 문자에 할당할 수 있도록 하고 있다. 이렇게 할당된 확장 부호는 표준 아스키와 같이 서로 다른 프로그램이나 컴퓨터 사이에 교환되지 못한다.
- ✓ 그러므로 표준 아스키는 마이크로컴퓨터 하드웨어 및 소프트웨어 사이에서 세계적으로 통용되는 데 비해, 확장 아스키는 프로그램이나 컴퓨터 또는 프린터가 그것을 해독할 수 있도록 설계되어 있어야만 올바르게 해독될 수 있다.

## 확장 아스키 예

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	€	160	A0	á	192	C0	Ł	224	E0	α
129	81	ü	161	A1	í	193	C1	ł	225	E1	β
130	82	é	162	A2	ó	194	C2	Ť	226	E2	Γ
131	83	â	163	A3	ú	195	C3	ţ	227	E3	π
132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
134	86	å	166	A6	*	198	C6	‡	230	E6	μ
135	87	ç	167	A7	°	199	C7	‡	231	E7	ι
136	88	ê	168	A8	¿	200	C8	℔	232	E8	Φ
137	89	ë	169	A9	ƒ	201	C9	℥	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	℥	234	EA	Ω
139	8B	ÿ	171	AB	½	203	CB	¥	235	EB	Ø
140	8C	î	172	AC	¼	204	CC	℥	236	EC	∞
141	8D	ï	173	AD	¡	205	CD	=	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	℥	238	EE	ε
143	8F	Å	175	AF	»	207	CF	±	239	EF	∩
144	90	É	176	B0	☐	208	DO	℥	240	FO	≡
145	91	æ	177	B1	☐	209	D1	¥	241	F1	±
146	92	Æ	178	B2	☐	210	D2	π	242	F2	≥
147	93	ô	179	B3		211	D3	℔	243	F3	≤
148	94	ö	180	B4	†	212	D4	℔	244	F4	[
149	95	ò	181	B5	‡	213	D5	℥	245	F5	]
150	96	û	182	B6	‡	214	D6	℥	246	F6	÷
151	97	ù	183	B7	π	215	D7	℥	247	F7	≈
152	98	ÿ	184	B8	‡	216	D8	‡	248	F8	•
153	99	Ö	185	B9	‡	217	D9	℥	249	F9	•
154	9A	Ü	186	BA		218	DA	℥	250	FA	·
155	9B	ó	187	BB	π	219	DB	■	251	FB	√
156	9C	£	188	BC	π	220	DC	■	252	FC	∞
157	9D	¥	189	BD	π	221	DD	■	253	FD	∞
158	9E	℔	190	BE	℥	222	DE	■	254	FE	■
159	9F	f	191	BF	℥	223	DF	■	255	FF	□

✎ 오늘날 대부분의 컴퓨터는 문자를 읽고 쓰는데 ASCII 형식을 사용한다.

✎ 그런데 컴퓨터가 발전하면서 미국 뿐 아니라 각 나라에서도 컴퓨터가 발전했으며

✎ 각 국가들은 자국의 문자를 표현하기 위하여 코드체계를 만들어서 사용하게 되었다.

✓ 우리나라도 아주 오래된 얘기지만 한글 코드체계를 만들어 사용했고 조합형, 완성형 두 종류를 가지고 있었다.

- ✎ 인터넷이 전 세계로 발전하면서 ASCII를 만들었을 때의 문제와 같은 문제가 국가간에 정보를 주고 받을 때 발생했다.
- ✎ 자국의 코드체계를 타 국가가 가지고 있지 않으면 정보를 잘못 해석할 수 밖에 없었다.
- ✎ 그래서 다국어 처리를 위해 표준을 마련했다 이를 유니코드라고 한다.



## 유니코드의 일부

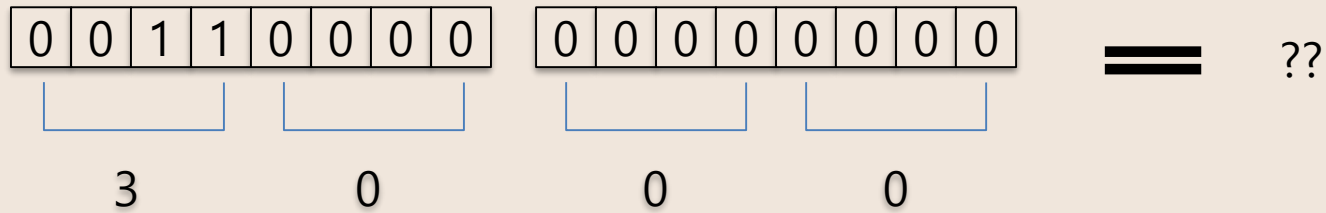
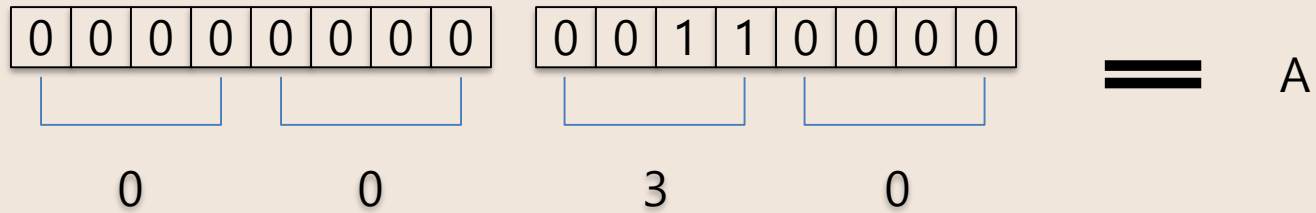
	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074

	B30	B31	B32	B33	B34	B35	B36	B37
0	대 B300	데미 B310	단 B320	닷 B330	델 B340	덱 B350	덜 B360	테 B370
1	덱 B301	데미 B311	닷 B321	당 B331	덱 B341	덜 B351	덜 B361	텍 B371
2	덱 B302	데미 B312	당 B322	닷 B332	덱 B342	덜 B352	덜 B362	텍 B372
3	덱 B303	데미 B313	단 B323	닷 B333	덱 B343	덜 B353	덜 B363	텍 B373
4	덴 B304	데미 B314	달 B324	닥 B334	덱 B344	더 B354	덜 B364	덴 B374

## 유니코드도 다시 Character Set으로 분류된다.

- ✓ UCS-2(Universal Character Set 2)
- ✓ UCS-4(Universal Character Set 4)
- ✓ 유니코드를 저장하는 변수의 크기를 정의
- ✓ 그러나, 바이트 순서에 대해서 표준화하지 못했음.
- ✓ 다시 말해 파일을 인식 시 이 파일이 UCS-2, UCS-4인지 인식하고 각 경우를 구분해서 모두 다르게 구현해야 하는 문제 발생
- ✓ 그래서 유니 코드의 적당한 외부 인코딩이 필요하게 되었다.

## big-endian, little-endian




 **유니코드 인코딩(UTF:Unicode Transformation Format)**

 **UTF-8(in web)**

✓ MIN: 8bit, MAX: 32bit(1 Byte \* 4)

 **UTF-16(in windows, java)**

✓ MIN: 16bit, MAX: 32bit(2 Byte \* 2)

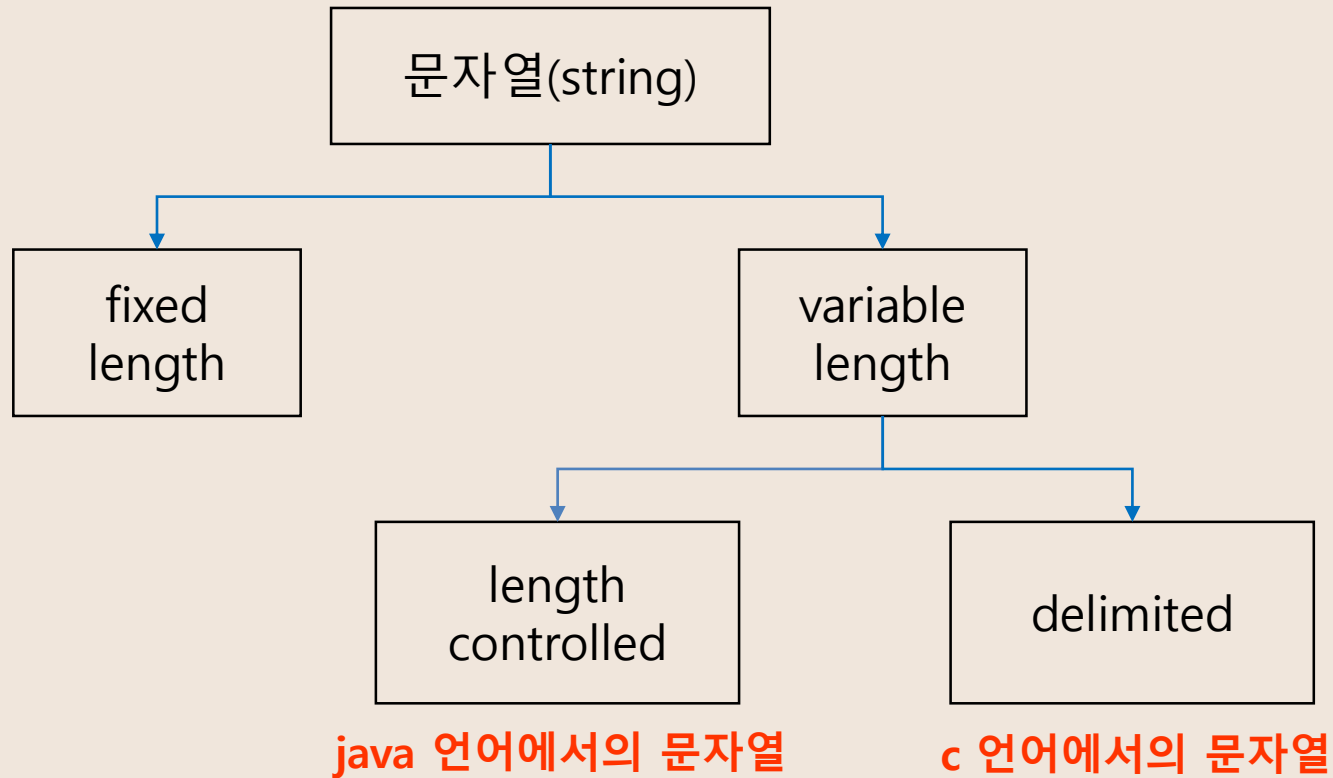
 **UTF-32(in unix)**

✓ MIN: 32bit, MAX: 32bit(4 Byte \* 1)



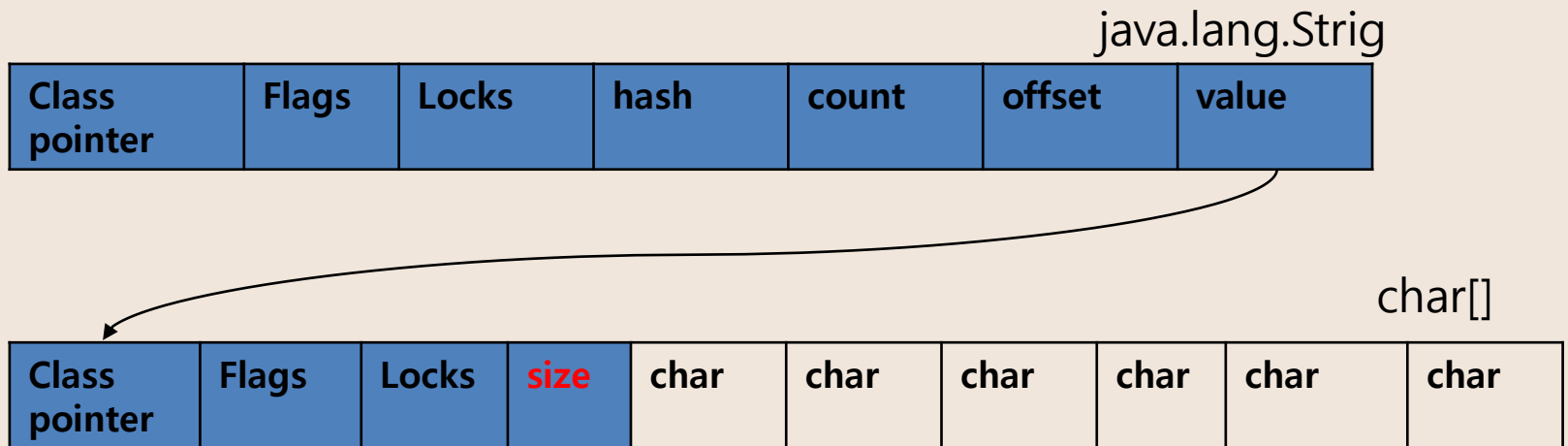
# 문자열

## 문자열의 분류



## 🖋 java에서 String 클래스에 대한 메모리 배치 예

- ✓ 그림에서 보이듯, java.lang.String 클래스에는 기본적인 객체 메타 데이터 외에도 네 가지 필드들이 포함되어 있는데, hash값(hash), 문자열의 길이(count), 문자열 데이터의 시작점(offset), 그리고 실제 문자열 배열에 대한 참조(value)이다.



## C언어에서 문자열 처리

- ✓ 문자열은 문자들의 배열 형태로 구현된 응용 자료형
- ✓ 문자배열에 문자열을 저장할 때는 항상 마지막에 끝을 표시하는 널문자 (' 0')를 넣어줘야 한다.

★ `char ary[]={ 'a', 'b', 'c', ' 0' }; // 또는 char ary[]="abc";`

- ✓ 문자열 처리에 필요한 연산을 함수 형태로 제공한다.

★ `strlen(), strcpy(), strcmp(),...`

## Java[객체지향 언어]에서의 문자열 처리

- ✓ 문자열 데이터를 저장, 처리해주는 클래스를 제공한다.

- ✓ String클래스를 사용한다.

  - ✦ `String str="abc";` //또는 `String str = new String("abc")`

- ✓ 문자열 처리에 필요한 연산을 연산자, 메소드 형태로 제공한다.

  - ✦ `+`, `length()`, `replace()`, `split()`, `substring()`,...

  - ✦ 보다 풍부한 연산을 제공한다.

## C와 Java의 문자열 처리의 기본적인 차이점

- ✓ c는 아스키 코드로 저장한다.
- ✓ java는 유니코드(UTF16, 2byte)로 저장한다.

C

```
char * name = "홍길동";  
int count = strlen(name);  
printf("%d", count);
```

6이 출력된다.

Java

```
String name = "홍길동";  
System.out.println(name.length());
```

3이 출력된다.

# 문자열 복사

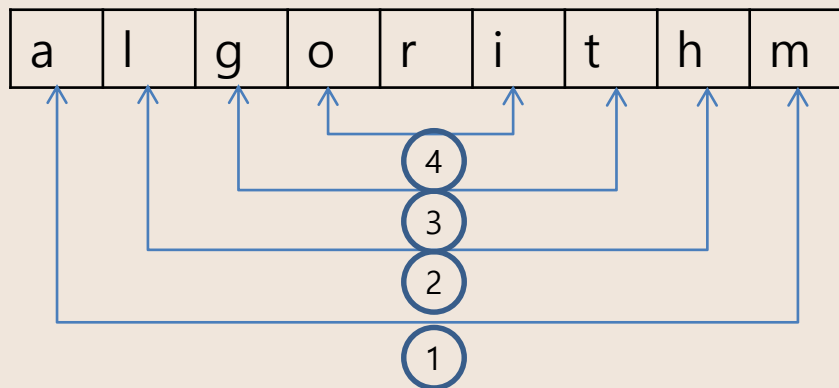
- ✎ 소스로부터 한 글자씩 읽어서 타겟에 한 글자씩 복사한다.
- ✎ 문자열의 끝( '\0' )까지 반복한다. 타겟에 문자열 끝을 표시한다.

C

```
void user_strcpy(char *des, char *src)
{
    while(*src!='\0'){
        *des = *src;
        src++;
        des++;
    }
    *des='\0';
}
```

# 문자열 뒤집기

- ✎ 자기 문자열에서 뒤집는 방법이 있고 새로운 빈 문자열을 만들어 소스의 뒤에서부터 읽어서 타겟에 쓰는 방법이 있겠다.
- ✎ 자기 문자열을 이용할 경우는 swap을 위한 임시 변수가 필요하며 반복 수행을 문자열 길이의 반만을 수행해야 한다.




문자열 길이 9  
 $9 / 2 = 4.5$   
4회 반복

# 연습문제1

- ✎ java에서는 StringBuffer 클래스의 reverse() 메소드를 이용하면 된다.
- ✎ c에서는 앞의 알고리즘 대로 구현해야 한다.
- ✎ 구현 해 봅시다.



# 문자열 비교

 c strcmp() 함수를 제공한다.

 java에서는 equals() 메소드를 제공한다.

✓ 문자열 비교에서 == 연산은 메모리 참조가 같은지를 묻는 것이다. 차이점을 이해하자.

 다음은 c로 구현한 strcmp()의 예이다.

C

```
int my_strcmp(const char *str1, const char *str2)
{
    int i = 0;
    while(str1[i] != '\0')
    {
        if(str1[i] != str2[i]) break;
        i++;
    }
    return (str1[i] - str2[i]);
}
```

# 문자열 숫자를 정수로 변환하기

- ✎ c 언어에서는 `atoi( )` 함수를 제공한다. 역 함수로는 `itoa( )`가 있다.
- ✎ java에서는 숫자 클래스의 `parse` 메소드를 제공한다.
  - ✓ 예 : `Integer.parseInt(String)`
  - ✓ 역함수로는 `toString( )` 메소드를 제공한다.

## atoi()


C

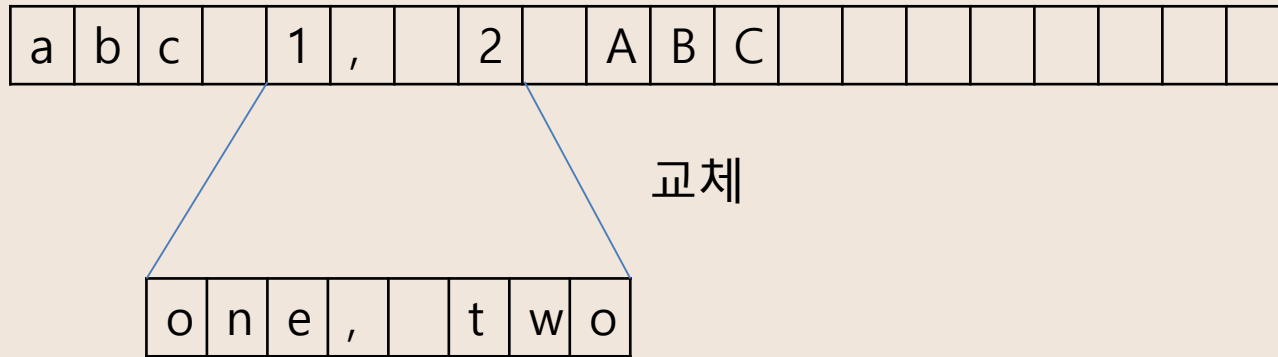
```
int atoi(const char *string)
{
    int value = 0, digit, c;

    while ((c = *string++) != '\0') {
        if (c >= '0' && c <= '9')
            digit = c - '0';
        else
            break;

        value = (value * 10) + digit;
    }
    return value;
}
```

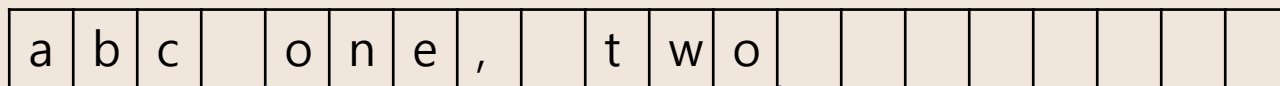
# 문자열 교체하기

 다음 예에서 문자열 내에서 “1, 2” 라는 문자열을  
“one, two” 로 변경해 보자. (교체될 문자열의 저장 공간은 충분히 크다고 가정)

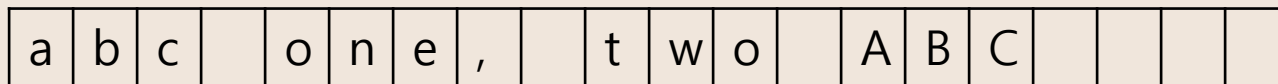
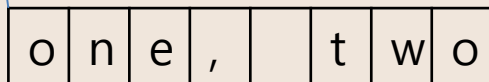




복사




교체



복사



# 연습문제2

 itoa()를 구현해 봅시다.

- ✓ 양의 정수를 입력 받아 문자열로 변환하는 함수
  - ✓ 입력 값 : 변환할 정수 값, 변환된 문자열을 저장할 문자배열
  - ✓ 반환 값 : 없음
- 
- ✓ 음수를 변환할 때는 어떤 고려 사항이 필요한가요?

**패턴 매칭**

# 패턴매칭

## 패턴 매칭에 사용되는 알고리즘 들

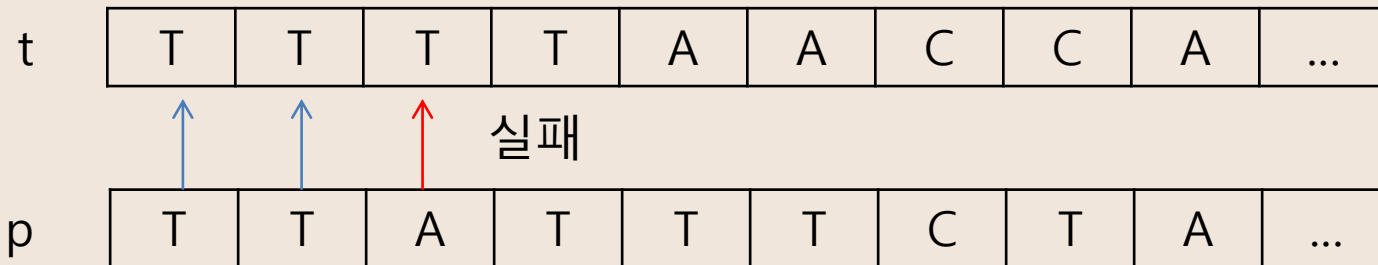
- ✓ 고지식한 패턴 검색 알고리즘
- ✓ 카프-라빈 알고리즘
- ✓ KMP 알고리즘
- ✓ 보이어-무어 알고리즘



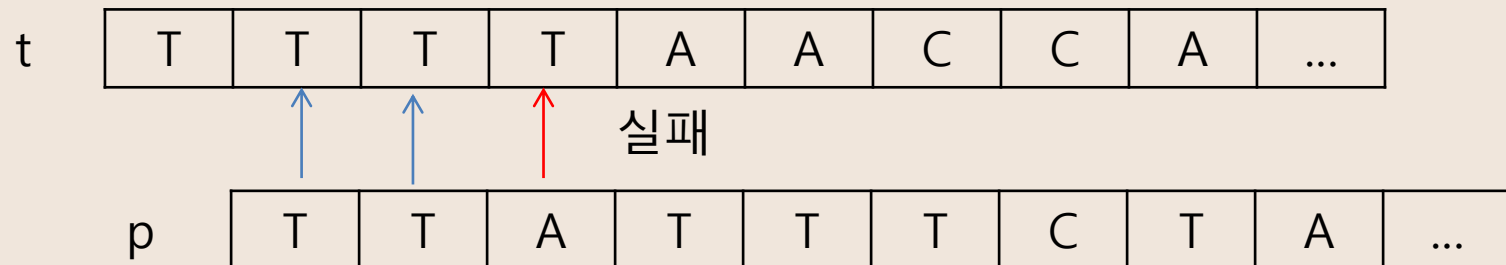
**고지식한 알고리즘**

## ✎ 고지식한 알고리즘(Brute Force)

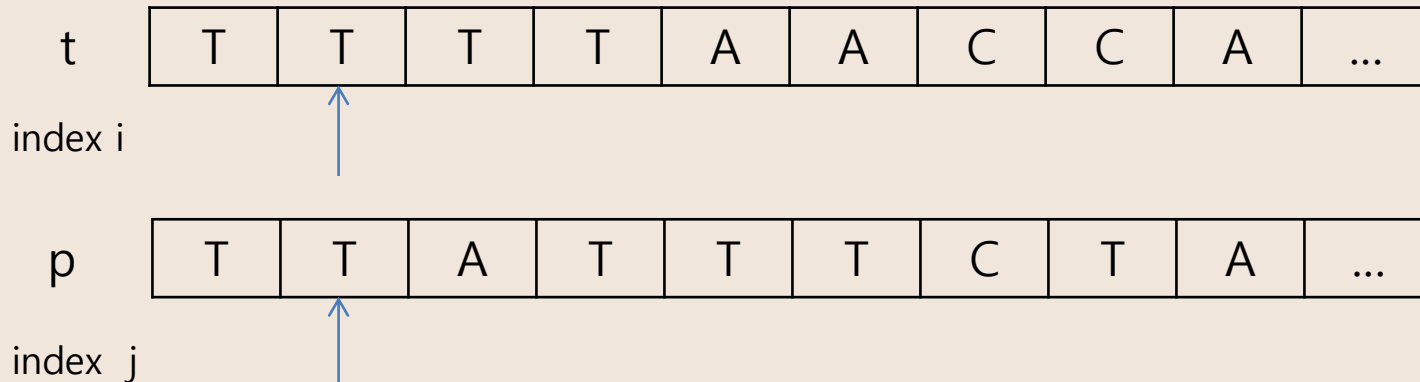
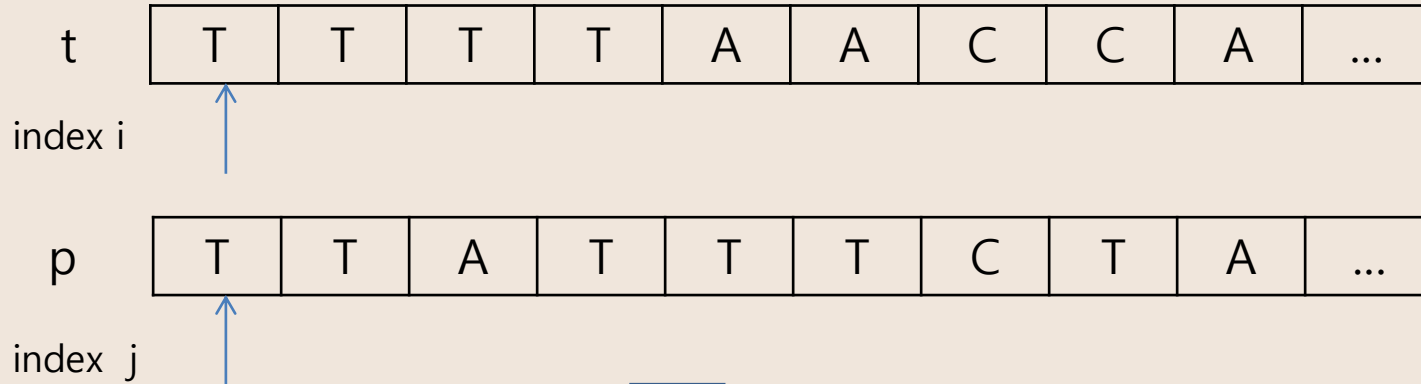
- ✓ 본문 문자열을 처음부터 끝까지 차례대로 순회하면서 패턴 내의 문자들을 일일이 비교하는 방식으로 동작

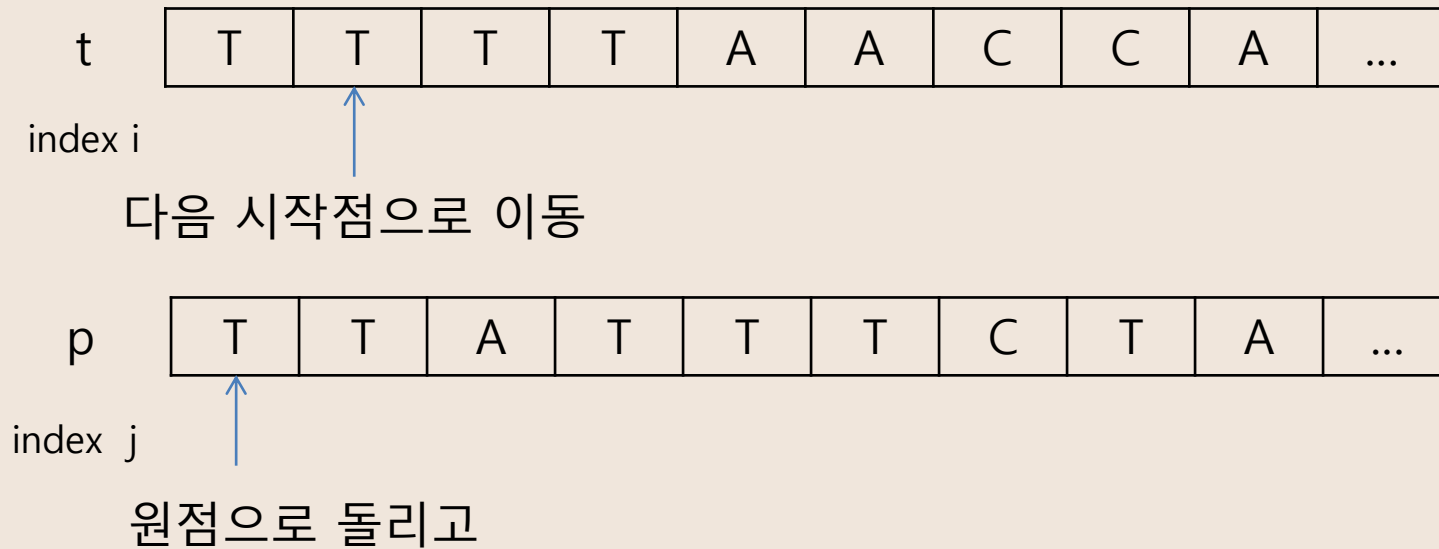
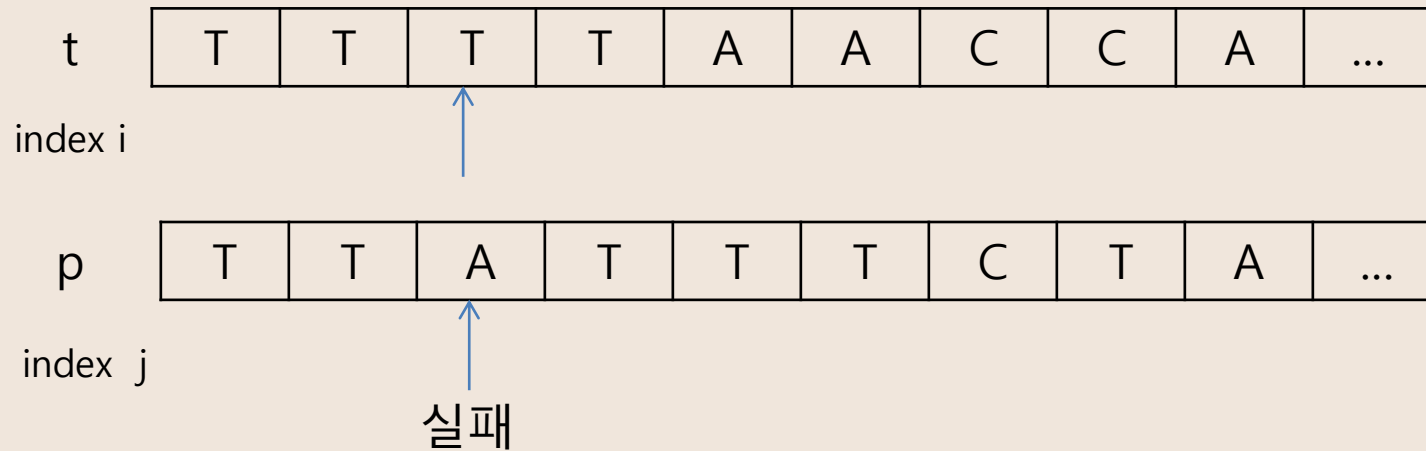


한 칸 이동, 비교



## 알고리즘 설명





## 고지식한 패턴 검색 알고리즘

```
// p[] : 찾을 패턴  
// t[] : 전체 텍스트  
// M : 찾을 패턴의 길이;  
// N : 전체 텍스트의 길이;  
// i : t의 인덱스
```

```
BruteForce(p[], t[])  
  i ← 0, j ← 0  
  while(j < M and i < N) do {  
    if (t[i] ≠ p[j]) then {  
      i ← i - j;  
      j ← -1;  
    }  
    i ← i + 1, j ← j + 1  
  }  
  if (j = M) then return i - M;  
  else return i;  
end BruteForce()
```

## 고지식한 패턴 검색 알고리즘의 시간 복잡도

- ✓ 최악의 경우 시간 복잡도는 텍스트의 모든 위치에서 패턴을 비교해야 하므로  $O(MM)$ 이 됨
- ✓ 예에서는 최악의 경우 약  $10,000 * 80 = 800,000$  번의 비교가 일어난다.
- ✓ 비교횟수를 줄일 수 있는 방법은 없는가?

# KMP 알고리즘

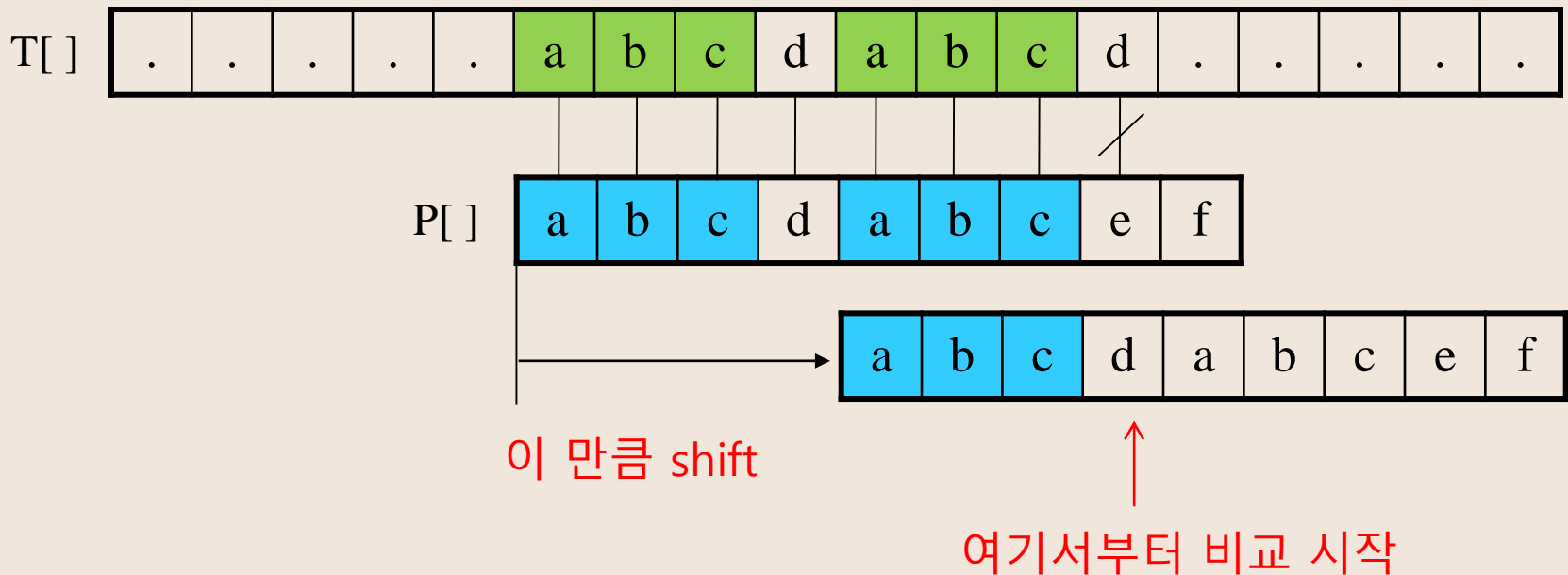
# KMP 알고리즘

- ✎ 불일치가 발생한 텍스트 스트링의 앞 부분에 어떤 문자가 있는지를 미리 알고 있으므로, 불일치가 발생한 앞 부분에 대하여 다시 비교하지 않고 매칭을 수행
- ✎ 패턴을 전처리하여 배열 `next[M]`을 구해서 잘못된 시작을 최소화함
  - ✓ `next[M]` : 불일치가 발생했을 경우 이동할 다음 위치
- ✎ 시간 복잡도 :  $O(M+N)$



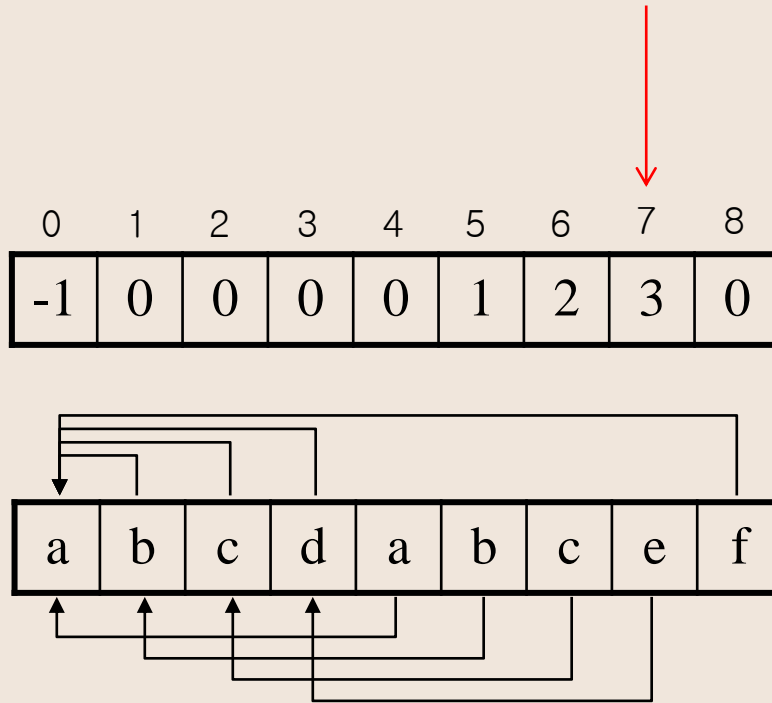
## 아이디어 설명

- ✓ 텍스트에서 abcdabc까지는 매치되고, e에서 실패한 상황 패턴의 맨 앞의 abc와 실패 직전의 abc는 동일함을 이용할 수 있다
- ✓ 실패한 텍스트 문자와 P[4]를 비교한다



## 매칭이 실패했을 때 돌아갈 곳을 계산한다.




앞의 예에서 e와 매칭이 실패 했고 이때 돌아갈 곳의 계산 값은 3로 문자 d의 위치를 의미한다.



패턴의 각 위치에 대해  
매칭에 실패했을 때  
돌아갈 곳을 준비해 둔다

**보이어-무어 알고리즘**

# 보이어-무어 알고리즘

-  오른쪽에서 왼쪽으로 비교
-  대부분의 상용 소프트웨어에서 채택하고 있는 알고리즘
-  보이어-무어 알고리즘은 패턴에 오른쪽 끝에 있는 문자가 불일치 하고 이 문자가 패턴 내에 존재하지 않는 경우, 이동 거리는 무려 패턴의 길이 만큼이 된다.

T[ ] 

.	.	.	.	.	.	.	.	b	w	a	t	e	r	.	.
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

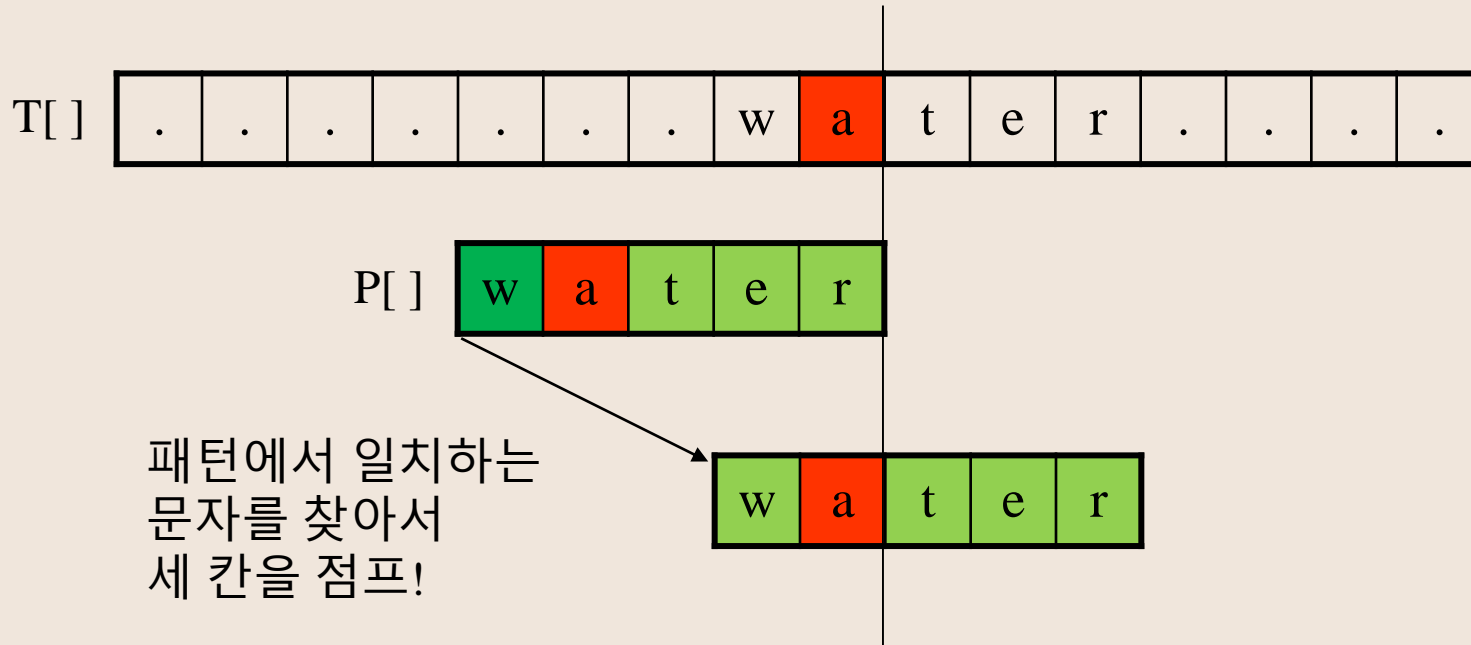
P[ ] 

w	a	t	e	r
---	---	---	---	---

다섯칸 한꺼번에 점프!

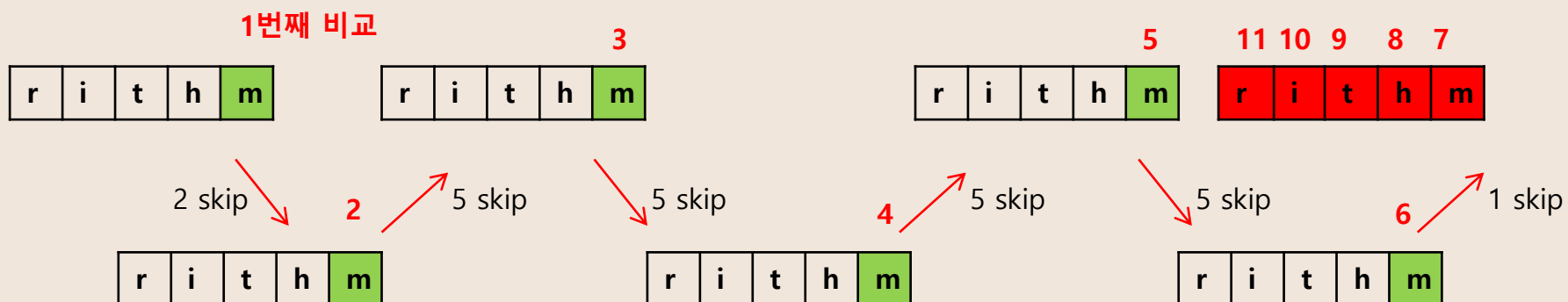
w	a	t	e	r
---	---	---	---	---

✎ 오른쪽 끝에 있는 문자가 불일치 하고 이 문자가 패턴 내에 존재할 경우



## ✍ 보이어-무어 알고리즘을 이용한 예

a p a t t e r n m a t c h i n g a l g o r i t h m






✓ rithm 문자열의 skip 배열

m	h	t	i	r	다른 모든 문자
5	1	2	3	4	5

## 문자열 매칭 알고리즘 비교

- ✓ 찾고자 하는 문자열 패턴의 길이  $m$ , 총 문자열 길이  $n$
- ✓ 고지식한 패턴 검색 알고리즘 : 수행시간  $O(mn)$
- ✓ 카프-라빈 알고리즘 : 수행시간  $\Theta(n)$
- ✓ KMP 알고리즘 : 수행시간  $\Theta(n)$
- ✓ 보이어-무어 알고리즘
  - ✦ 앞의 두 매칭 알고리즘들의 공통점 텍스트 문자열의 문자를 적어도 한번씩 훑는다는 것이다. 따라서 최선의 경우에도  $\Omega(n)$
  - ✦ 보이어-무어 알고리즘은 텍스트 문자를 다 보지 않아도 된다
  - ✦ 발상의 전환: 패턴의 오른쪽부터 비교한다
  - ✦ 최악의 경우 수행시간:  $\Theta(mn)$
  - ✦ 입력에 따라 다르지만 일반적으로  $\Theta(n)$ 보다 시간이 덜 든다

## 연습문제3

-  고지식한 방법을 이용하여 패턴을 찾아 봅시다.
-  임의의 본문 문자열과 찾을 패턴 문자열을 만듭니다.
-  결과 값으로 찾은 위치 값을 결과로 출력합니다.