

## 9주차

### 데이터의 표현

- 인코딩 및 압축
- 오류확인

		강의 주제	주 차
1부 컴퓨팅 사고력	컴퓨팅 사고력의 소개	IT 사회, 소프트웨어 세상, 컴퓨팅 사고력의 소개, 컴퓨팅 사고력의 개념	1
		컴퓨팅 사고력의 개념, 주위에서 볼 수 있는 컴퓨팅 사고력, 문제해결 방법	2
	문제해결 방법, 컴퓨터	문제해결 방법, 문제해결 과정 예, 문제해결을 위한 소프트웨어 설계 사상, 컴퓨터의 특징, 소프트웨어, 유한상태기계	3
2부 소프트웨어	알고리즘	알고리즘 소개, 알고리즘의 표현 방법, 의사코드, 흐름도	4
	프로그램	프로그램의 기능, 함수, 컴파일러	5
	파이썬	파이썬 소개 및 설치, 변수에 값 저장, 입력, 출력, 조건부 수행	6
		반복, 리스트, 함수, 출력 형식	7
3부 컴퓨팅 사고력 활용하기	데이터의 표현	이진수, 아스키코드, 오디오 데이터, 이미지 데이터, 자료구조	8
		인코딩 및 압축, 오류확인	9
	데이터의 저장과 검색	배열 및 연결 리스트, 선형검색, 이분검색, 색인순차검색, 해싱	10
		이진검색트리, 최대값 및 최소값 검색	11
	알고리즘설계	정렬, 분할정복 알고리즘, 탐욕적 알고리즘	12

인코딩 및 압축

Encoding and Compression



- 인코딩(encoding): 정보를 코드로 표현하는 것
- 디코딩(decoding): 코드로 표현된 것으로부터 정보를 추출하는 것
- 인코딩 시 가능한 저장공간을 줄이려고 함 – 압축(compression)



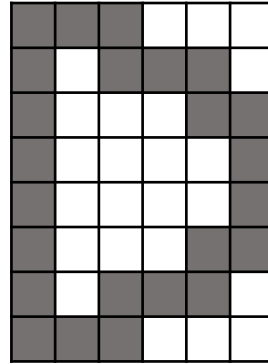
# 팩스(FAX)



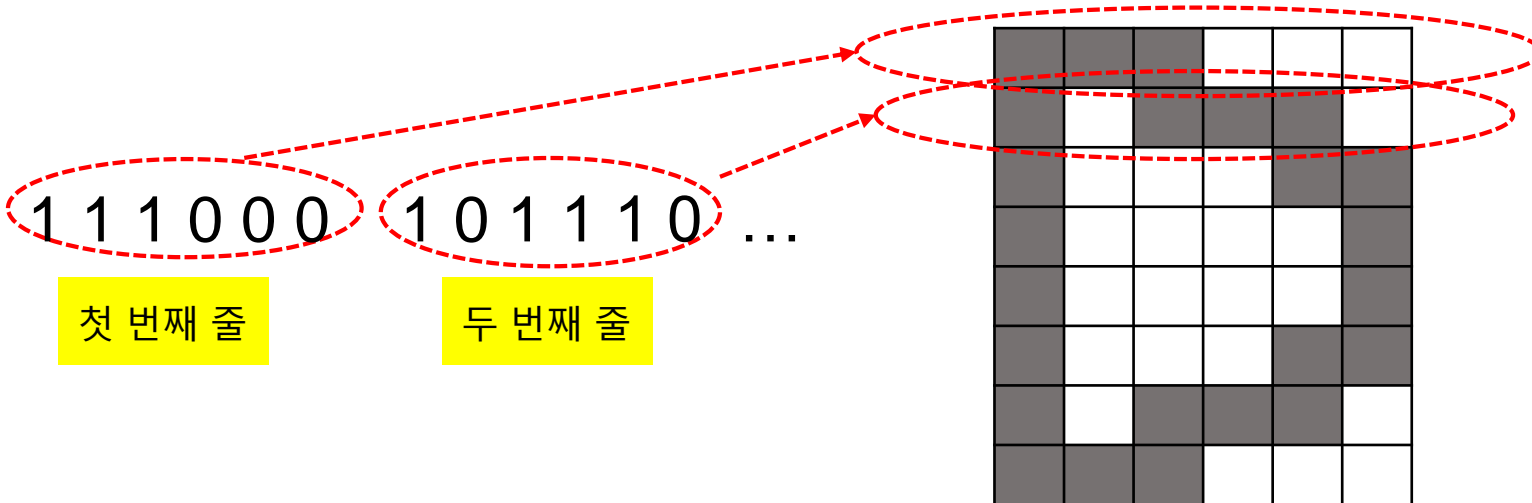
- 팩스(Fax, facsimile)
- 문서를 보내는 쪽의 팩스 기계는 문서를 읽어 들어서 이미지를 비트맵(bit map)으로 만든다.
- 비트맵은 격자형으로 이미지를 세분화해서, 각 격자 내의 명암을 0 또는 1로 표시한 것이다.
- 비트맵으로 변환된 데이터를 전화선을 통해 가청주파수톤(audio-frequency tone)으로 전송한다.
- 문서를 받는 쪽의 팩스 기계는 0과 1로만 표시된 데이터를 비트맵으로 복원하여 이미지를 생성하게 된다.



- D 라는 문자는 픽셀(pixel, picture element) 들의 격자(grid)로 표현



- D를 팩스로 전송할 때 다음의 이미지를 0 (white 표시) 또는 1 (black 표시)의 비트맵(bitmap)으로 전송하면 된다. - encoding



[problem] 흑백의 이미지를 인코딩(encoding)한 후 전송한다. 전송된 이미지를 디코딩(decoding)해서 이미지를 원본과 동일하게 (무손실, lossless) 재구성한다.

- 제한 조건
  - ✓ 압축한 후의 텍스트 크기가 압축 전 보다 줄어 들어야 한다.
  - ✓ 압축과 복원이 쉽게 이루어져야 한다.
  - ✓ 데이터의 손실이 없어야 한다.



## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

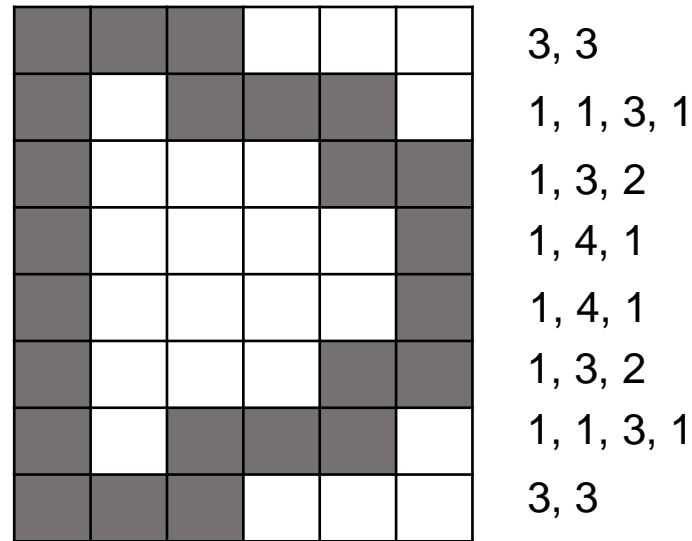
## ● 문제 해결 전략

- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용





- 그런데, 중복되는 데이터가 많다.
- 따라서 아래 그림 우측의 데이터만 전송하면 된다.
- 이러한 방식은 run-length 인코딩 방식이며, 데이터 압축방법 일종
- run : 반복하는 부분
- 각 라인의 첫 데이터는 black pixel의 개수. 다음은 white pixel의 개수, 다음은 black.....



- 수신하는 쪽에서는 수신하는 숫자를 해석하여 동일한 이미지를 생성할 수 있다.
- 각 줄에 있는 픽셀의 개수를 알고 있으므로, 연속적인 숫자를 줄 단위로 해석 가능

## Computational thinking

- 패턴 확인
  - ✓ 패턴을 찾는다.
  - ✓ 연속된 하얀색 픽셀, 검은색 픽셀의 개수를 파악
- 알고리즘 설계
  - ✓ 하얀색 픽셀, 검은색 픽셀의 개수를 전송하는 알고리즘
  - ✓ 수신 후 이미지 재구성 알고리즘 설계
- 데이터 표현
  - ✓ 전체 종이를 여러 개의 픽셀로 그리드 형성
  - ✓ 각 픽셀의 색깔(백, 흑)을 구분



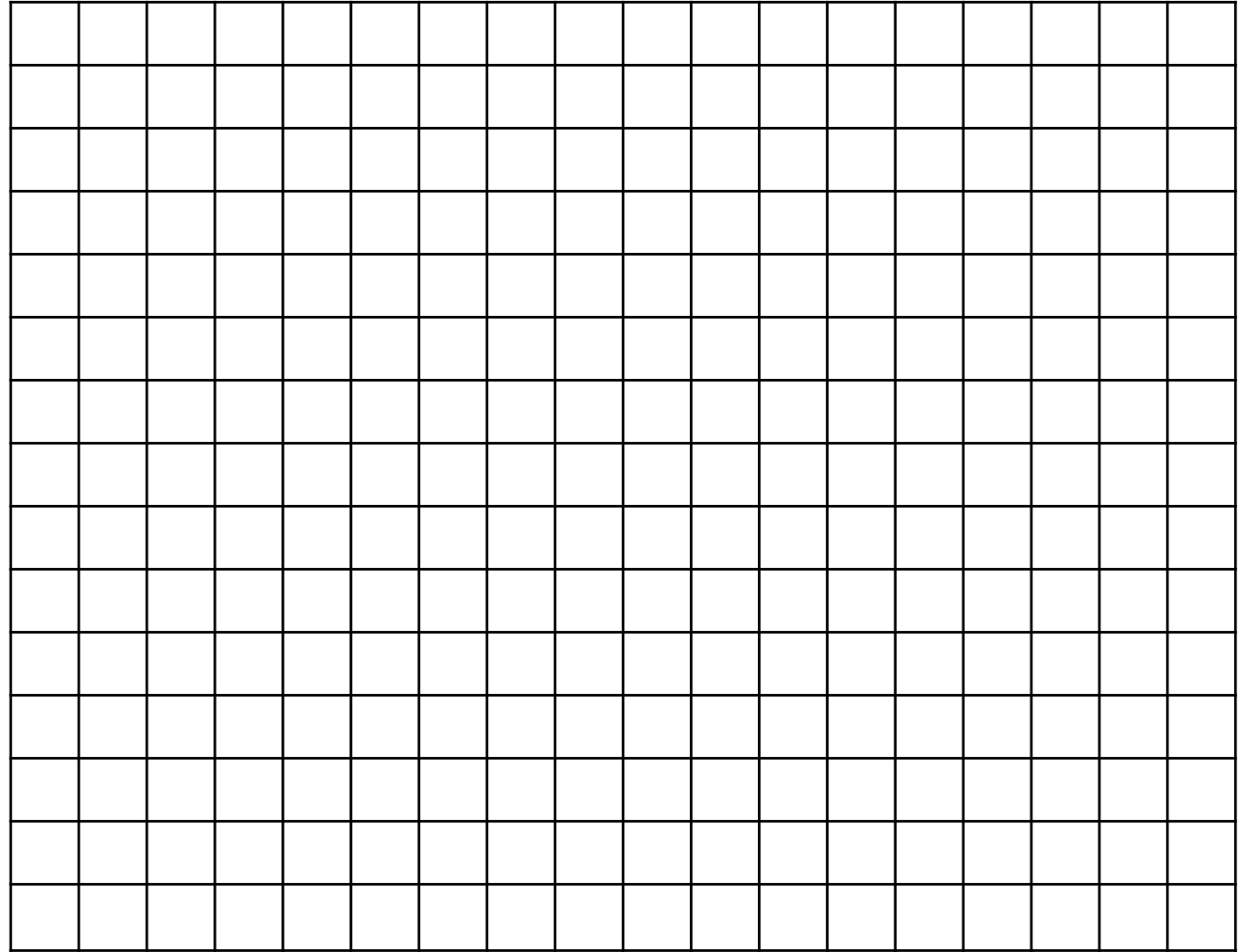
(연습문제) 다음을 run-length 인코딩 방식으로 인코딩하라

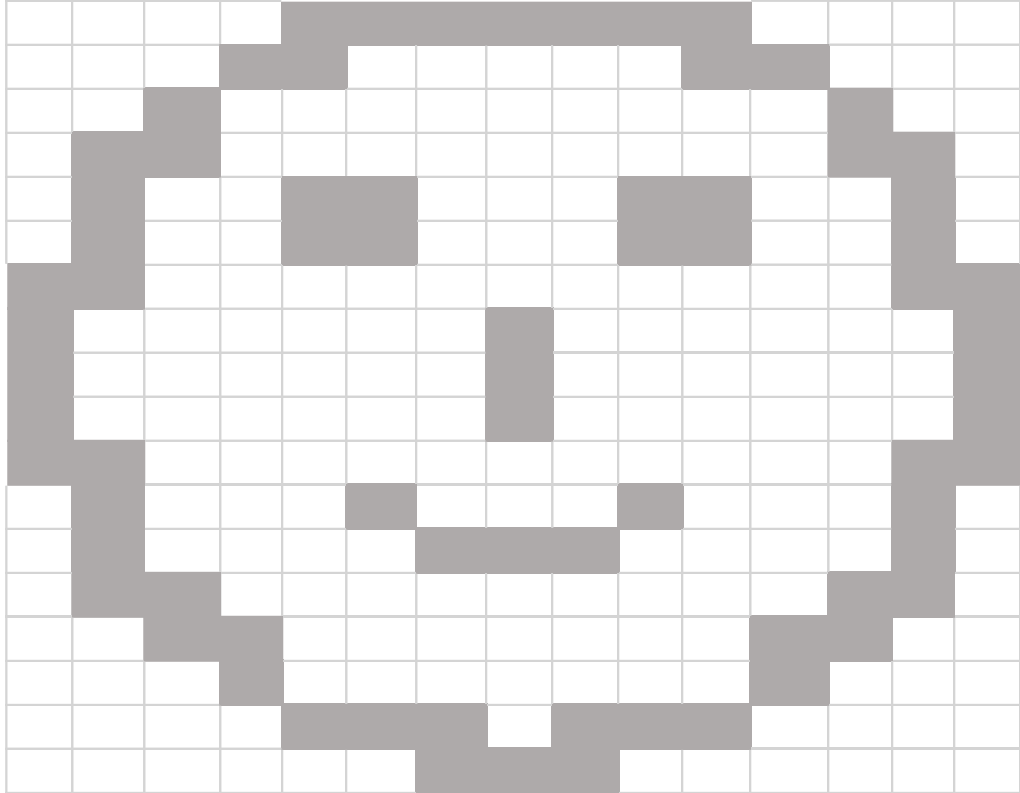
■	□	□	■	□	□	□
■	□	■	□	□	□	□
■	■	□	□	□	□	□
■	■	□	□	□	□	□
■	□	■	□	□	□	□
■	□	□	■	□	□	□
■	□	□	□	■	□	□



(연습문제) run-length 인코딩 방식의 다음 데이터를 수신하였다. 아래의 칸에 이미지를 그리시오. 처음 숫자가 white 픽셀의 개수. 한 줄은 15칸으로 구성.

4 7 4 3 2 5 2 3 2 1 9 1 2 1 2 9 2 1 1 1 2 2  
 3 2 2 1 1 1 1 2 2 3 2 2 1 1 0 2 11 2 0 1 6  
 1 6 1 0 1 6 1 6 1 0 1 6 1 6 1 0 2 1 1 2 1 1  
 3 1 3 1 3 1 1 1 1 4 3 4 1 1 1 2 9 2 1 2 2 7  
 2 2 3 1 7 1 3 4 3 1 3 4 6 3 6





Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure.

문서를 팩스로 송신할 때 이진수로 표현하기 위한 격자 모양



- 이것은 단순한 압축 방법이다
- 이 외에도 다양한 압축 알고리즘이 존재
- mpeg, jpeg, gif, png 등

# Run-Length 인코딩

- 하나의 문자는 여러 번 나타날 수 있다.
- 반복된 문자열을 다음의 두 가지 구성요소로 대체한다.
  - ✓반복된 문자
  - ✓반복 횟수
- bbbbb → b5
  - ✓ b : 반복된 문자
  - ✓ 5: b가 반복된 횟수





(ex)

- 원래 데이터

aaaakkkbbbbbbbggggggcccccc

- 인코딩된 텍스트

a4kkb8g5c6

✓ 여기서 k는 인코드되지 않았다. 이득이 없다.

kk → k2

✓ 압축률(compression ratio)은 10/25 or 0.4

압축한 후의 데이터의 크기가  
압축 전 보다 줄어 들어야 한다.

✓ 문자, 숫자 표현 비트수가 동일하다고 가정

- 원래의 텍스트 길이 = 25
- 인코딩된 후의 길이 = 10
- 압축률 = 10/25



(ex)

- 인코딩된 데이터

y8t4a5

- 원래 텍스트

yyyyyyyyttttaaaaa

✓ 동일한 데이터가 반복하는 형태는 일반적인 문장에서 나타나지 않는다.

✓ 이미지 압축 시에 효과적이다.



## Computational thinking

- run-length 인코딩은 어떤 경우에 효과적일까?
- 인코딩, 디코딩 알고리즘을 설계할 수 있는가?

# 텍스트 압축(Text compression): LZSS 인코딩

[problem] 다음의 텍스트를 압축한 후 무손실(lossless) 방식으로 복원할 수 있는 방법을 찾으시오.

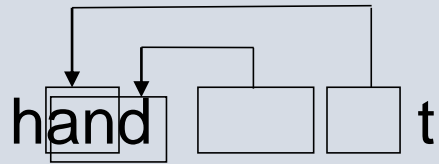
1. hand and ant
2. I love you and you love me
3. banana

- 제한 조건
  - ✓ 압축한 후의 text 크기가 압축 전 보다 줄어 들어야 한다.
  - ✓ 압축과 복원이 쉽게 이루어져야 한다.



아이디어: 이전에 나타난 동일한 문자열을 지시하는 것으로 문자열을 표시

hand and ant



hand (4,3) (8,2)t

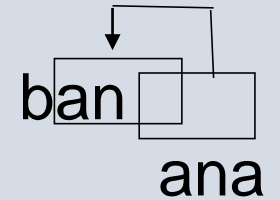


I love you and you love me



I love you and (8,3) (17,4) me

banana



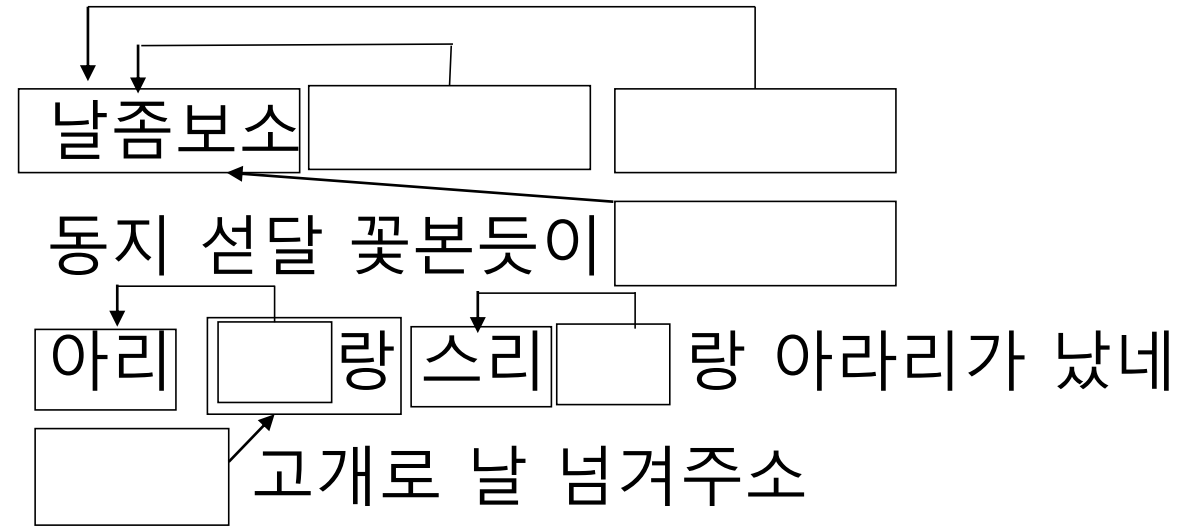
ban(2,3)

4칸 앞에서 3칸을 복사하라. d 다음 빈칸 한 개.



# 밀양아리랑

날좀보소 날좀보소 날좀보소  
동지 선달 꽃본듯이 날좀보소  
아리아리랑 스리스리랑 아라리가 났네  
아리랑 고개로 날 넘겨주소



여기서는 두 개의 문자까지만 전  
에 사용한 것을 확인했음

- 압축 전

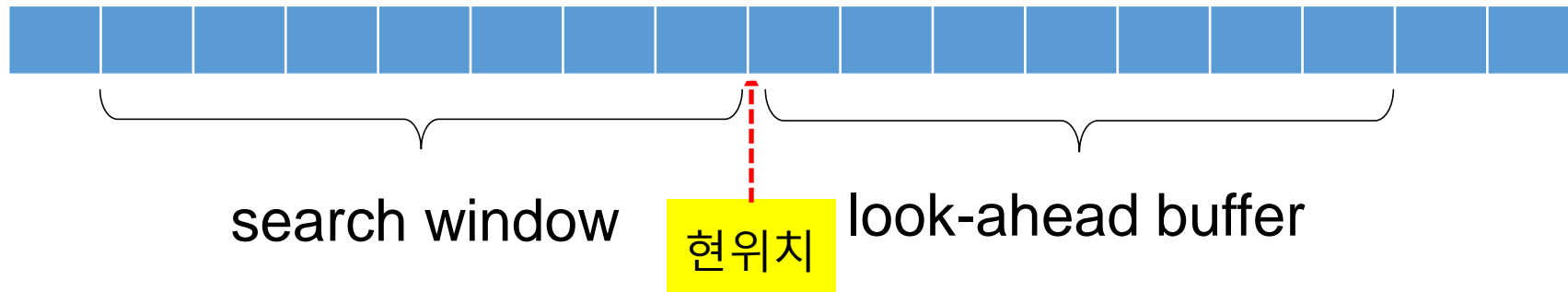
This is a sample. This is a sample. This is a sample. This is a sample.

- 압축 후

This (3,2) a sample. (18, 18) (36,36)

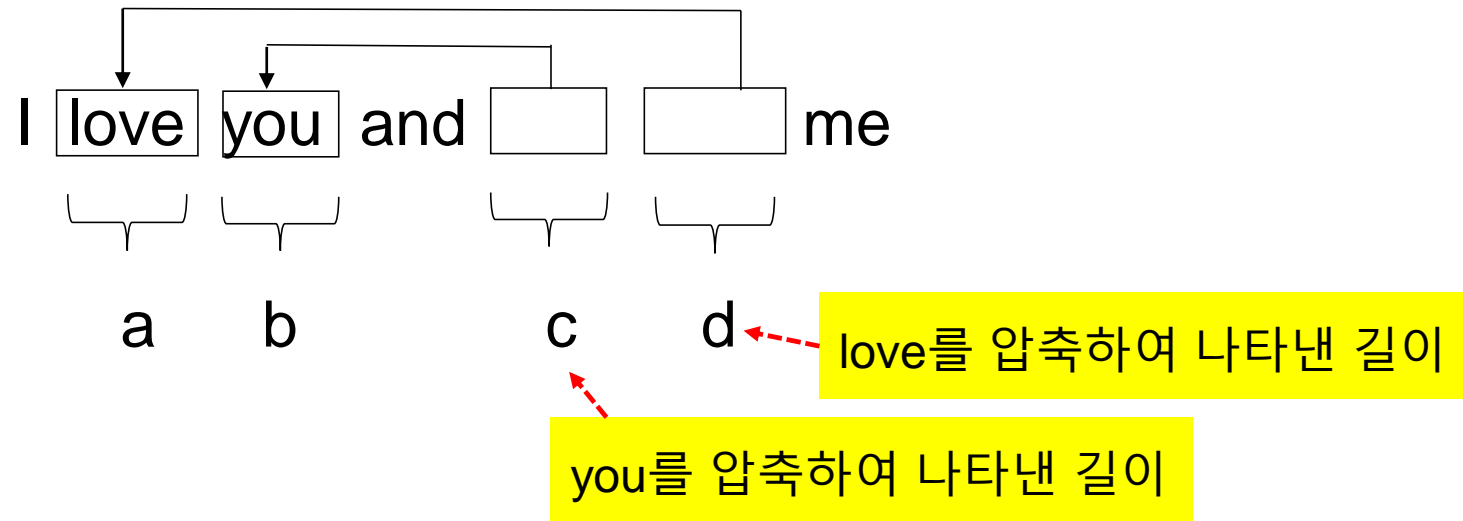
(앞으로 이동하여야 할 길이, 복사하여야 할 길이)

- 동일한 문자열을 앞부분에서 찾아 재활용
- 포인터로 문자열을 대체
- 일정한 크기의 search window 및 look-ahead buffer 대해 패턴을 찾는다.
  - ✓ search window: 이미 읽은 데이터에서 동일 패턴을 찾는다
  - ✓ look-ahead buffer : 앞으로 읽어 들여할 부분
  - ✓ 너무 멀리 있는 패턴을 찾으려면 시간 소요 – search window 크기 제한
  - ✓ 너무 긴 패턴을 찾으려면 시간 소요 – 패턴 길이 제한





- 하나의 패턴을 2개의 숫자로 표시
  - ✓ 일정한 크기 이상의 패턴에 대해서만 재활용을 해야 압축효과가 나타난다



- ✓ 압축한 길이가 원래 길이 보다 짧아야 한다.
- ✓ 따라서  $c < b$  and  $d < a$  이어야 한다.

다음 문장에 대해 화살표를 이용하여 압축을 표시하시오.

(2개 문자 이상의 문자열에 대해서)

1.    아리랑 아리랑 아라리요 아리랑 고개를 넘어간다.  
      나를 버리고 가시는 님은 십리도 못가서 발병난다.
  
2.    이런들 어떠하며 저런들 어떠하리  
      만수산 드렁칫이 얹혀진들 어떠하리  
      우리도 이같이 얹어져 백년까지 누리리라.

## Abraham Lincoln의 게티스버그 연설문 중(1863)

여기서는 세 개의 문자까지만 전에 사용한 것을 확인했음

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this



- 여기서 소개한 압축방법은 LZSS (Lempel-Ziv-Storer-Szymanski)(1982)
- LZSS는 LZ77(1977)의 개선된 형태
- LZ77은 Abraham Lempel과 Jacob Ziv가 개발한 무손실(lossless) 데이터 압축 알고리즘이다.
- LZ77로 부터 다양한 압축알고리즘이 개발되었다.
- LZW, LZSS, LZMA 등 압축알고리즘들의 기반이 되는 알고리즘이다.
- 이 알고리즘들은 GIF와 PNG의 기초개념이다.



## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

## ● 문제 해결 전략

- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용



## Computational thinking

- 패턴을 찾는다
  - ✓ 텍스트의 패턴을 찾는다 – 전략(look for an pattern)
  - ✓ 문자열에서 패턴을 찾는 것은 또 다른 문제이다.
- 알고리즘 설계
  - ✓ 인코딩 알고리즘과 인코딩된 정보를 디코딩할 수 있는 알고리즘을 만든다.
- 데이터 표현
  - ✓ 텍스트는 숫자로 표시된다.



# 후프만코드(Huffman Code)

[problem]

영문 알파벳으로 구성된 텍스트를 인코딩할 때 인코딩된 파일의 크기를 최소로 하는 인코딩 방법을 설계한다.

[제한요소]

- 최소 길이의 인코딩된 파일을 생성
- 무손실 인코딩



## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

## ● 문제 해결 전략

- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용





## [방법 1]

모든 알파벳에 같은 길이의 코드를 부여한다. 예를 들어 아스키코드 활용. 아스키코드는 8비트 사용.

## [문제점]

- 자주 나타나는 문자나 자주 나타나지 않는 문자가 모두 같은 길이의 코드를 갖게 된다.
- (예) AAAAABCCCC

A:00 B:01 C:11 이면 인코딩 후, 000000000000111111 총 18 비트 소요

## [방법 2]

- 자주 나타나는 문자는 짧은 코드, 자주 나타나지 않는 문자는 긴 코드 부여

- (예) AAAAABCCCC

B



A:0 B:01 C:1 이면 인코딩 후, 0000001111 총 10비트 소요

- 또 다른 문제점
  - ✓ 01 이 있는 경우, AC 인지 B 인지 구분할 수 없다.
  - ✓ 문자를 나타내는 영역을 구분하기 위해 구분자(delimiter)를 넣으면 인코딩 후 파일 크기가 증가
  - ✓ (예) 0/0/...0/01/1/1/1



### [방법 3]

- 코드의 길이가 가변적이면서, 구분자 없이 코드를 구분해 낼 수 있는 방법 필요
- 가변길이 코드(variable-length code)
- 전치(prefix) 코드

- 전치코드

- ✓ abc의 전치(prefix)는 a, ab, abc가 된다.
- ✓ 한 문자의 코드가 다른 문자의 코드의 앞부분이 될 수 없다.

(예) a:11, b:0, c:10 등 – 전치코드

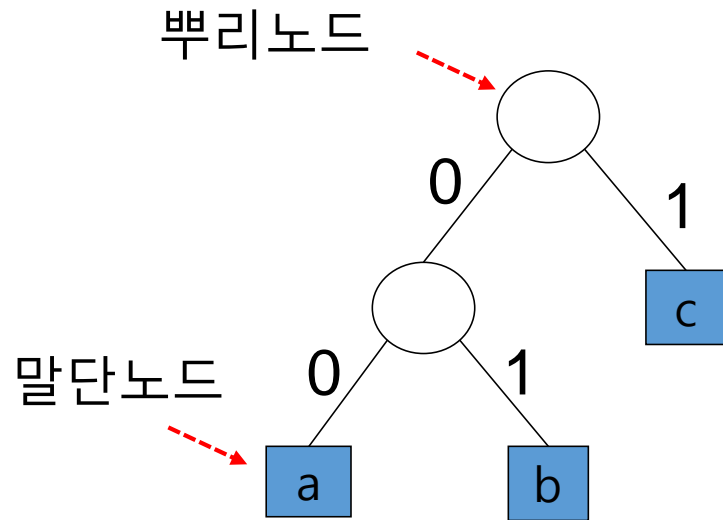
(예) 만일 a:01, b:0 이면 0을 어떻게 해석할 지 결정할 수 없다.

- ✓ 앞으로 읽을 비트를 확인하지 않아도 코드를 해석할 수 있음.
- ✓ 트리구조를 이용해서 코드 부여



## 코드 부여 방법

- 이진트리 (binary tree)이므로 최대 2개의 children을 갖는다.
- 각 노드에 연결된 에지 하나에 0(왼쪽 자식) 또는 1(오른쪽 자식)을 부여한다.
- 뿌리(root)에서 시작하여 말단노드(leaf node)로 이동하면서, 에지에 부여된 0,1을 연속적으로 읽는다.



a:00  
b:01  
c:1



(예) 파일 F = abbaca

전치코드 1: 010100110      9 비트  
전치코드 2: 10111110010      11 비트

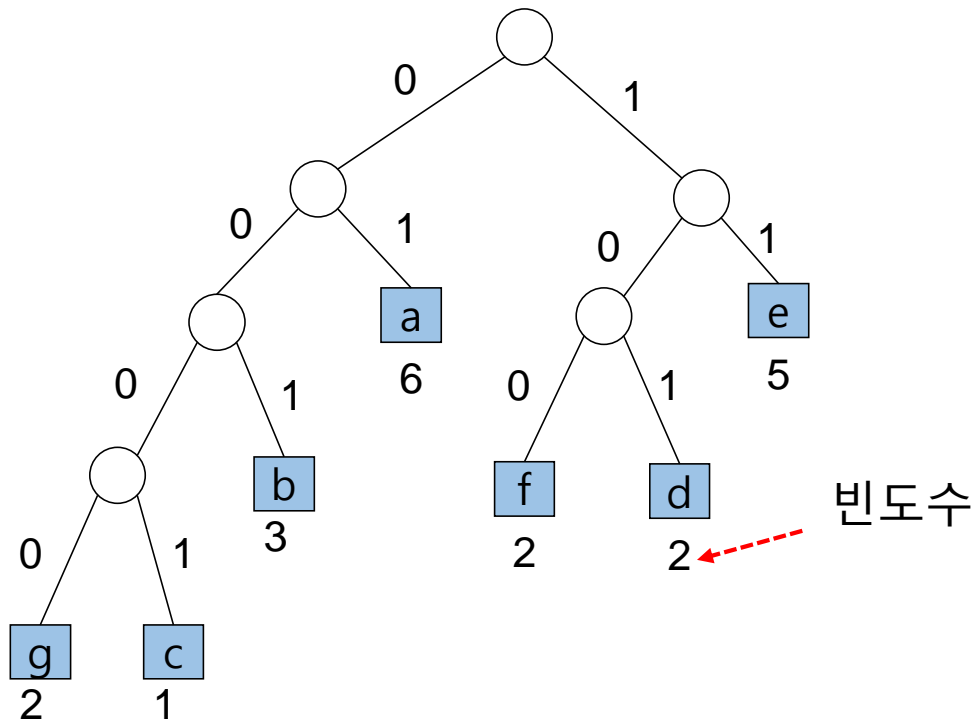
문자	전치 코드 1	전치 코드 2
a	0	10
b	10	11
c	11	0
총 비트 수	9	11

\* 전치코드 1이 파일 F에 대해서는 최적코드

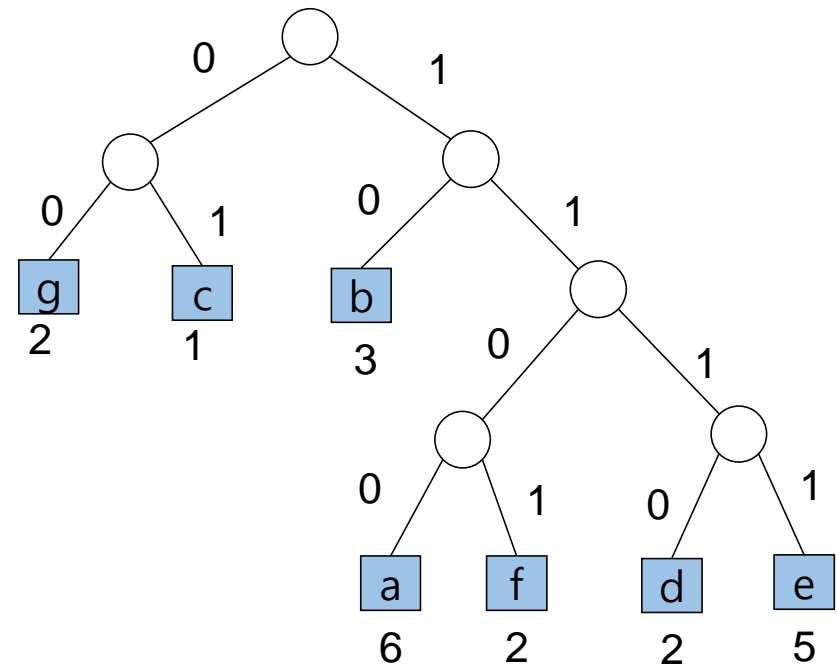
- 전치코드에 따라 최종 인코딩 된 파일의 크기가 달라진다.
- 최적 전치코드를 구하는 방법 필요



(예) 파일=cbbgaaaffdeebgaaadeeee



(A)



(B)

전치코드의 예. (A)는 후프만코드

## ● 후프만코드 생성 방법

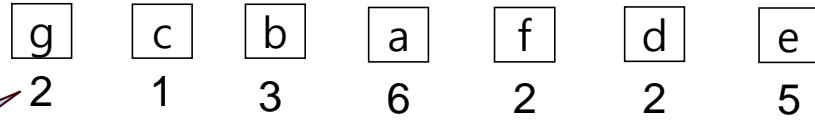
- (1) 인코딩하려는  $n$ 개의 데이터에 대해 빈도수를 표시하여  $n$ 개의 노드 생성
- (2) 두 노드의 빈도수의 합이 최소가 되는 노드를 찾는다.
- (3) 두 노드를 합병시켜서 이진트리로 만든다.
- (4) 모든 노드가 하나의 이진트리로 합쳐질 때까지 단계(2) (3)을 반복





(예) 파일=cbbgaaaffdeebgaaadeeee

[1]

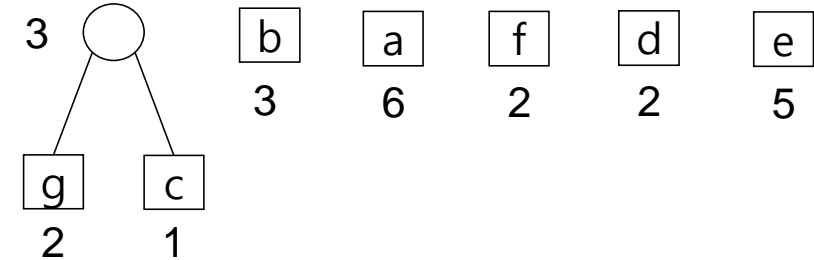


빈도수

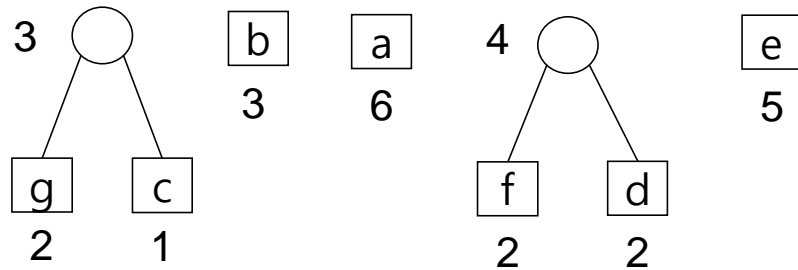
g는 2번 나타난다는 뜻

[2]

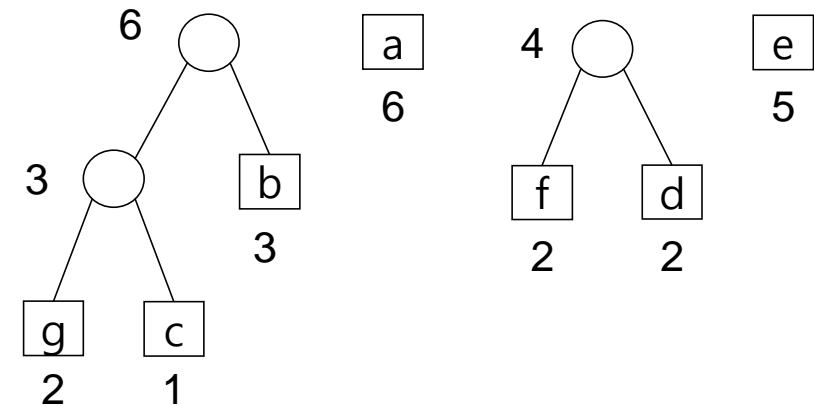
2+1=3이 두 수를  
합칠 때 가장 작은 경우



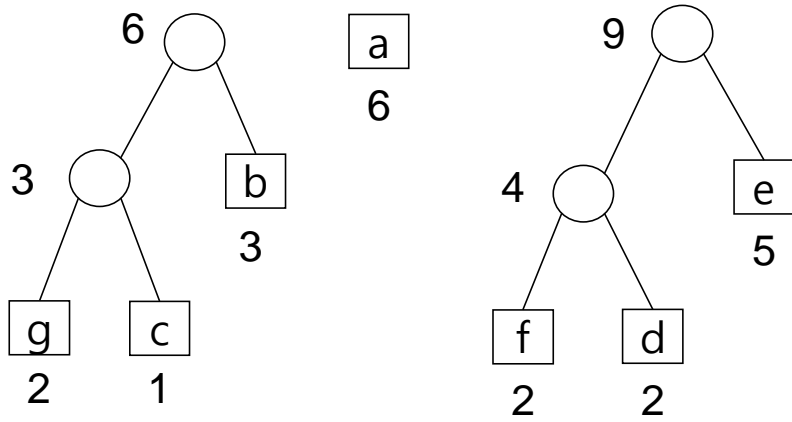
[3]



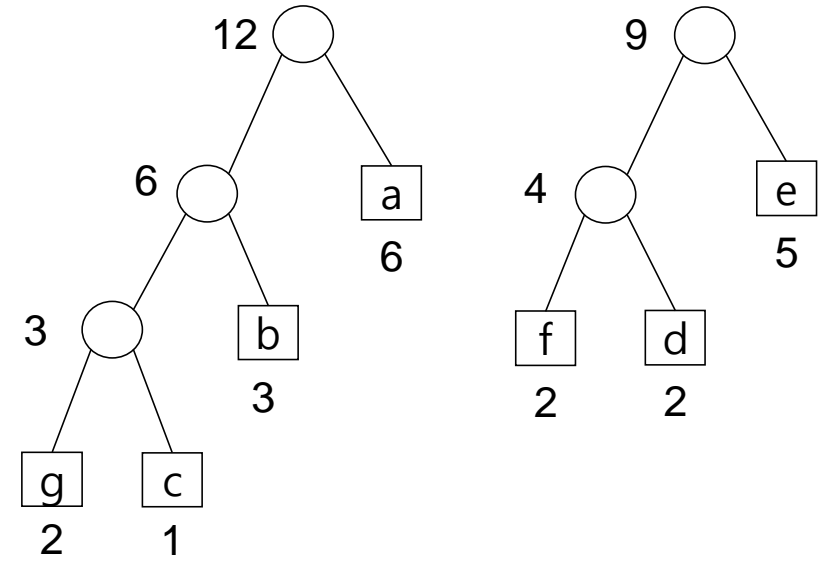
[4]



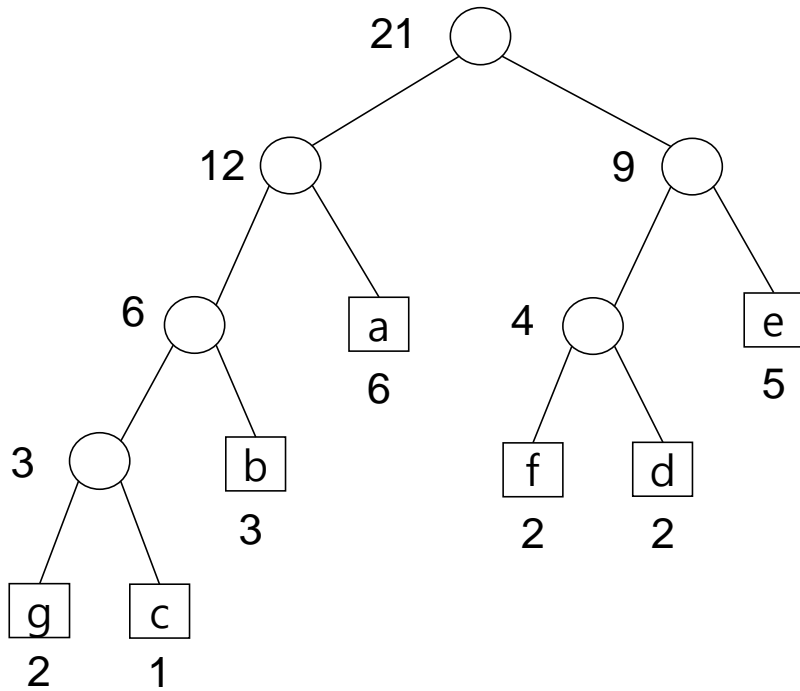
[5]



[6]



[7]

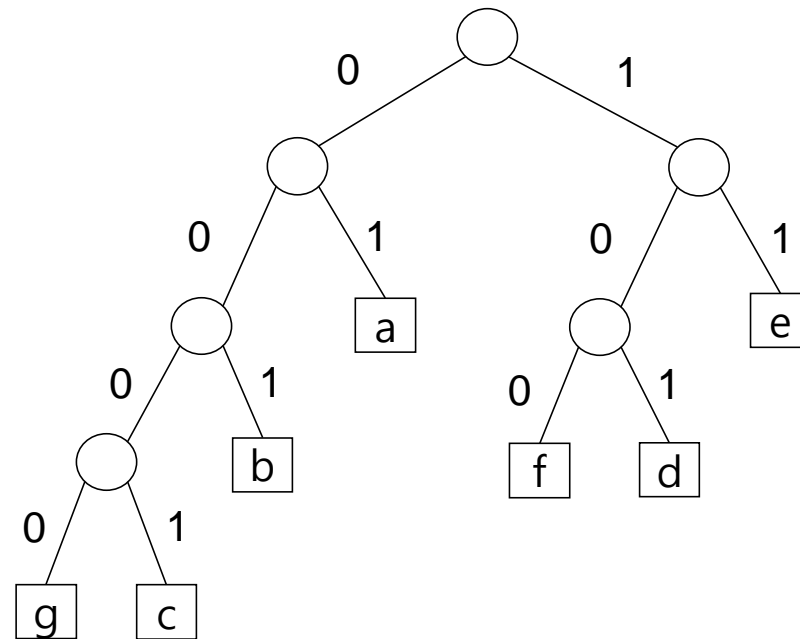


종료



왼쪽으로 이동 시 0 부여, 오른쪽으로 이동 시 1 부여

[8]



후프만코드



## 풀이과정을 재점검

- 자주 나타나는 문자는 나중에 트리에 합병되므로, 짧은 코드를 갖는다.
- 트리의 성질에 따라 코드는 전치코드다.
- 코드마다 크기가 다르므로 가변길이 코드이다.

[연습문제] 다음의 문자 빈도수가 있을 때 문자들의 후프만코드를 구성하라.

A: 1   B: 5   C: 2   D: 4   E: 7   F: 3



# 오류 확인

## Error Detection



- 송신자(sender)가 보낸 정보는 제3자의 악의적인 조작 또는 통신 오류에 의해 전송 중간 단계에서 변해서 수신자(receiver)에게 전달될 수 있다.



[problem]

- 수신자(receiver)는 수신자가 받은 정보가 송신자(sender)가 보낸 정보와 일치하는지 확신할 수 있는 방법이 필요하다.



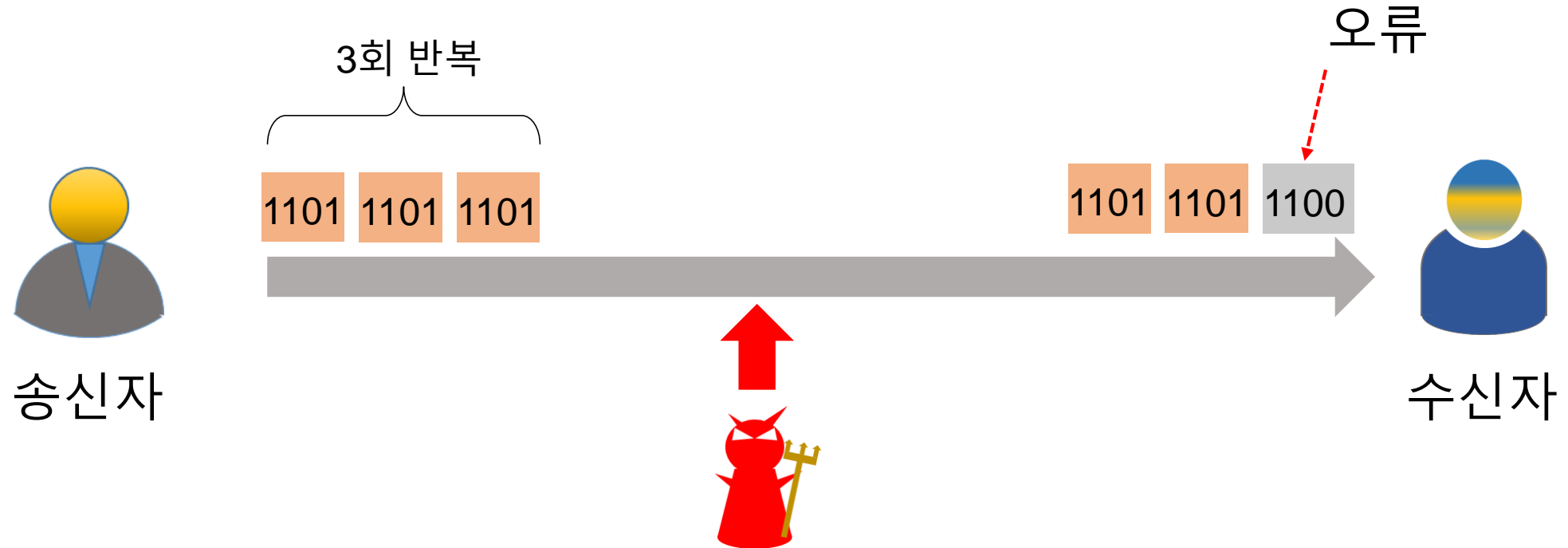
# 반복 코드(Repetition Code)



- 송신자는 정보를 정해진 횟수만큼 반복하여 수신자에게 전송
- 3개의 정보가 같으면 변조가 없음을 확인
- 3개의 정보가 모두 동일하게 달라질 확률은 없다고 가정



# 반복 코드



- 3개의 정보가 모두 같지 않으면 변조 발생 확인



## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

## ● 문제 해결 전략

- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용



## Computational thinking

- 일상 생활에서 컴퓨터의 파일을 별도의 공간에 backup 하는 것과 같은 개념
- 여러 개 복사하는 것(복잡도 증가)과 안정성과의 trade-off
- 단순한 방법

# 패리티 비트(Parity Bit)

데이터 A

1 0 0 0 1 1



송신자



데이터 B

1 0 0 0 1 0



수신자



변조 또는 오류



[problem]

- 수신자가 수신한 데이터 B에는 송신자가 보낸 원래의 데이터 A와 다른 정보가 존재하는지를 수신자가 알아낼 수 있는 방법을 설계하라.

## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

## ● 문제 해결 전략

- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용



[해결방안] 데이터 A에 패러티 비트 추가. 데이터 A' 생성

1	0	0	0	1	1	1
---	---	---	---	---	---	---

데이터 A'

패러티 비트 추가

- ✓ 이 경우는 짝수 패러티(even parity)이다.
- ✓ 1의 개수가 짝수





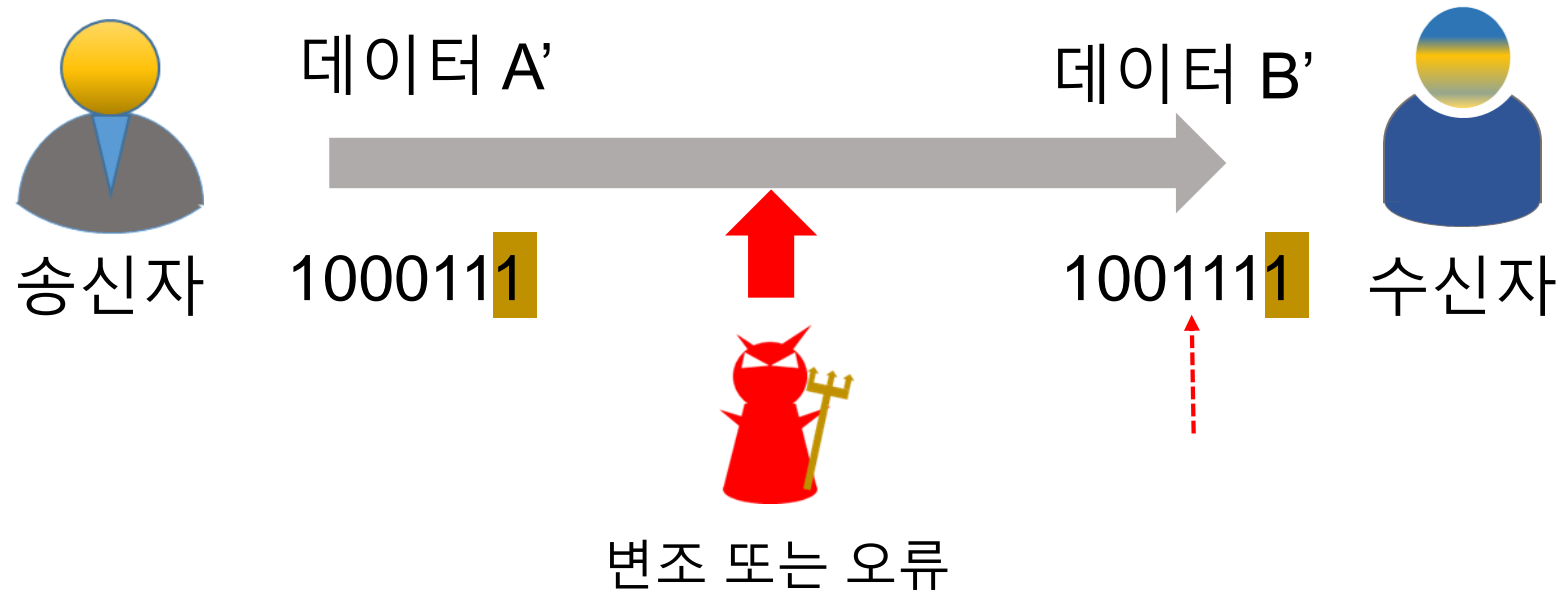
- 추가 된 패리티 비트에 의해 1의 개수가 짝수가 된다.
- 짝수(even) 패리티: 1의 개수가 짝수가 되도록 설정하는 방법
- 홀수(odd) 패리티: 1의 개수가 홀수가 되도록 설정하는 방법

## 정상 수신



- 송신자는 패리티 비트가 포함된 데이터 A'를 송신
- 수신자는 데이터 A'를 수신
- 패리티 비트의 정보가 데이터와 일치하므로 오류가 없다는 것을 알 수 있다.

# 비정상 수신



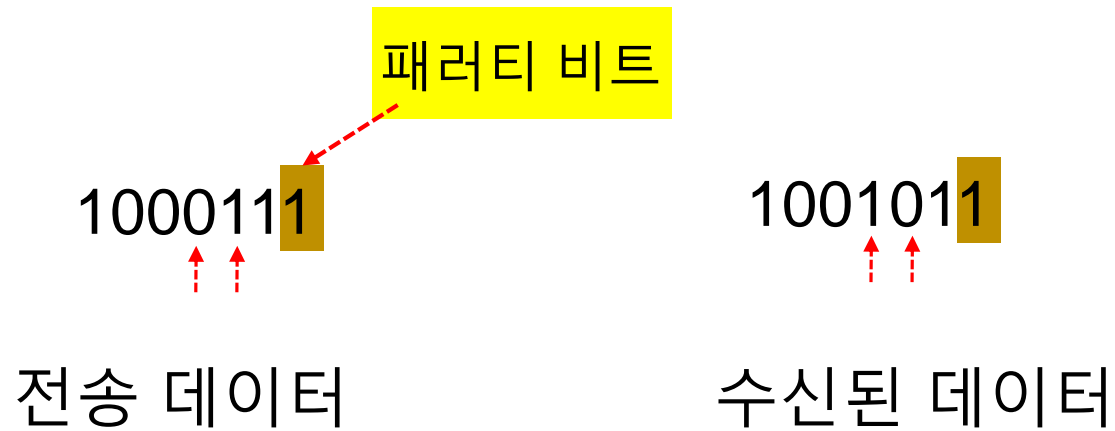
- 송신자는 패리티 비트가 포함된 데이터 A'를 송신
- 제3자가 A'를 변조
- 수신자는 변조된 데이터 B'를 수신
- 패리티 비트의 정보가 데이터와 일치하지 않는다는 것을 알 수 있다.
- 일치하지 않는 정보의 위치는 확인 불가



## Computational thinking

- 데이터를 어떻게 표현하느냐에 따라 문제의 해법이 존재
- be creative

만일 데이터에 2번의 오류가 발생하면,....



- ✓ 짝수개의 위치에서 변조 또는 오류가 발생하면, 패리티 비트로는 변조 또는 오류가 발생하였다는 것을 알 수 없다.
- ✓ 두 군데에서 동시에 오류가 발생할 확률은 작다고 가정



## 2차원 패리티 비트

- 송신자가 데이터를 수신자에게 전송
- 수신자가 데이터의 변조 또는 오류 확인 가능

데이터:

1 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 0



송신자



수신자



변조 또는 오류



데이터:

1 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 0

- 이 데이터를 5 비트 씩 끊어 2차원으로 배치하면,

1	0	0	0	1
0	1	0	0	1
1	0	1	1	0
1	1	0	0	1
1	0	0	1	0

- 짝수 패리티 가정
- 행 패리티와 열 패리티를 추가하면

1	0	0	0	1	0	행(row) 패리티
0	1	0	0	1	0	
1	0	1	1	0	1	
1	1	0	0	1	1	
1	0	0	1	0	0	
0 0 1 0 1						열(column) 패리티



원래데이터:

1 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 0

패러티가 추가된 데이터:

1 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 0

0 0 1 1 0

0 0 1 0 1

데이터

행 패러티

열 패러티





# 데이터의 변조 또는 오류가 발생한다면

송신자

1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	0	1
1	1	0	0	1	1
1	0	0	1	0	0
0	0	1	0	1	

수신자

1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	0	1
1	1	1	0	1	1
1	0	0	1	0	0
0	0	1	0	1	

오류

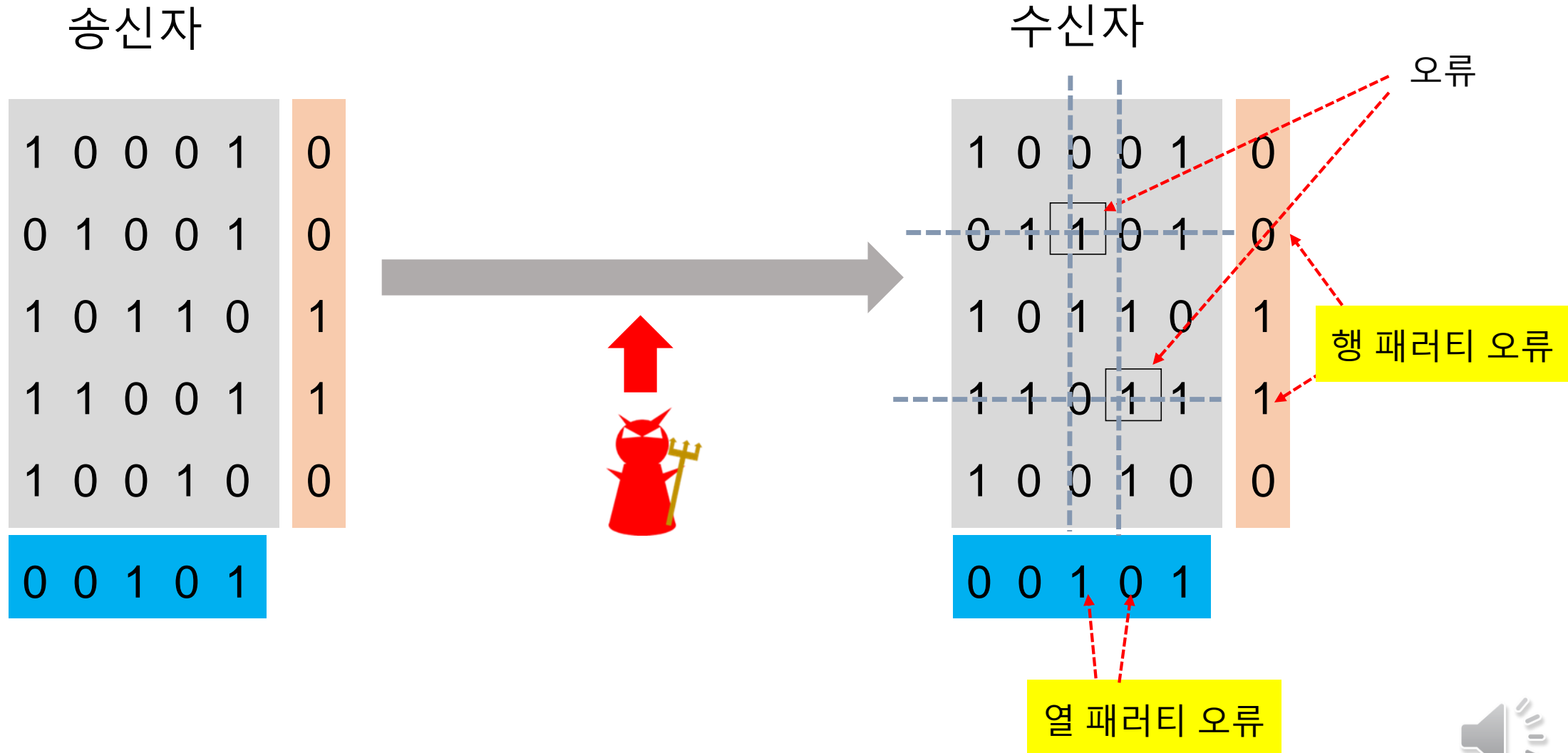
행 패러티 오류

열 패러티 오류

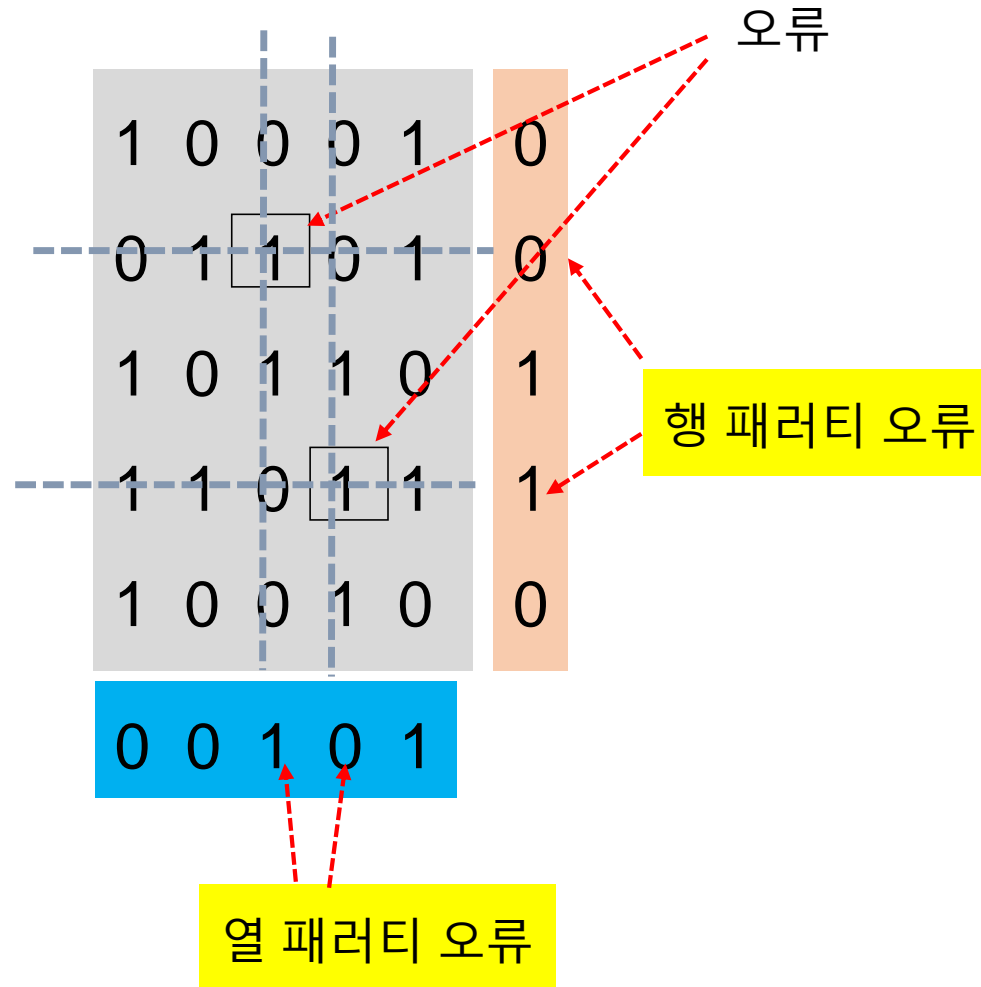
- ✓ 패러티 오류가 겹치는 행, 열의 위치에 오류 발생
- ✓ 오류 위치를 정확한 알아낼 수 있다.



[질문] 만일 서로 다른 행, 열에서 두 곳의 비트에 변조가 발생하면 어떻게 될까?



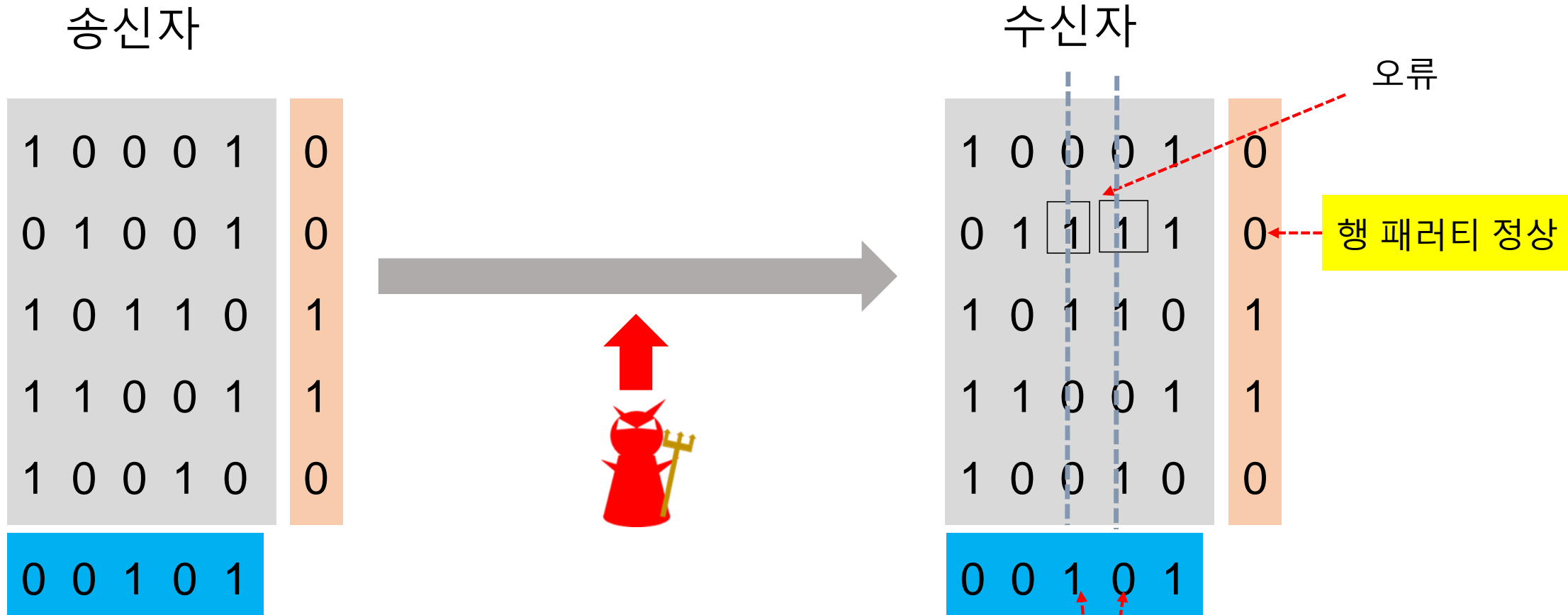
수신자



- 총 4비트의 변경 가능한 비트 존재
- 2행의 3열, 4열, 4행의 3열, 4열이 변경되었을 가능성만 확인 가능



[질문] 만일 하나의 행에서 두 곳의 비트에 변조가 발생하면 어떻게 될까?



- 행 패리티는 변동 없음. 열 패리티는 변동 발생
- 해당 열만 찾을 수 있다.
- 정확한 위치 확인 불가



## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

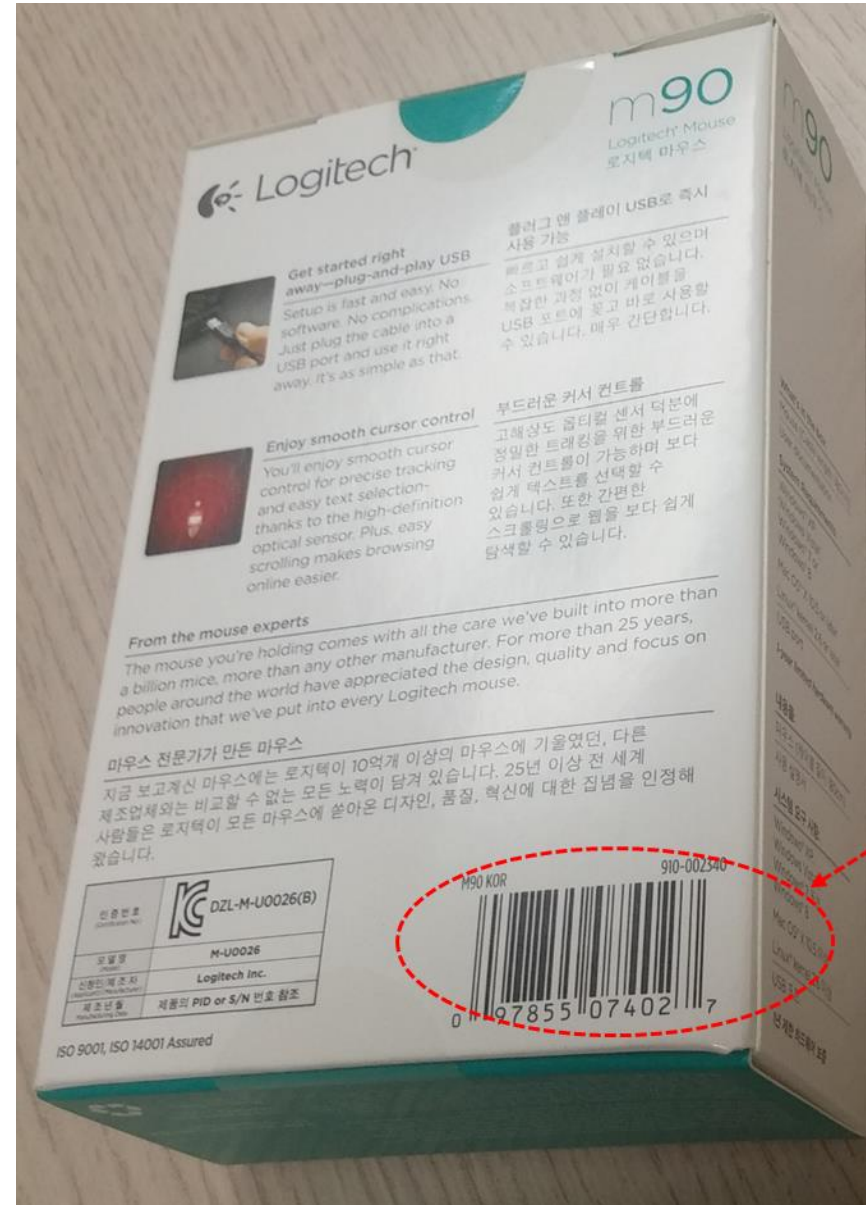
## ● 문제 해결 전략

- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용

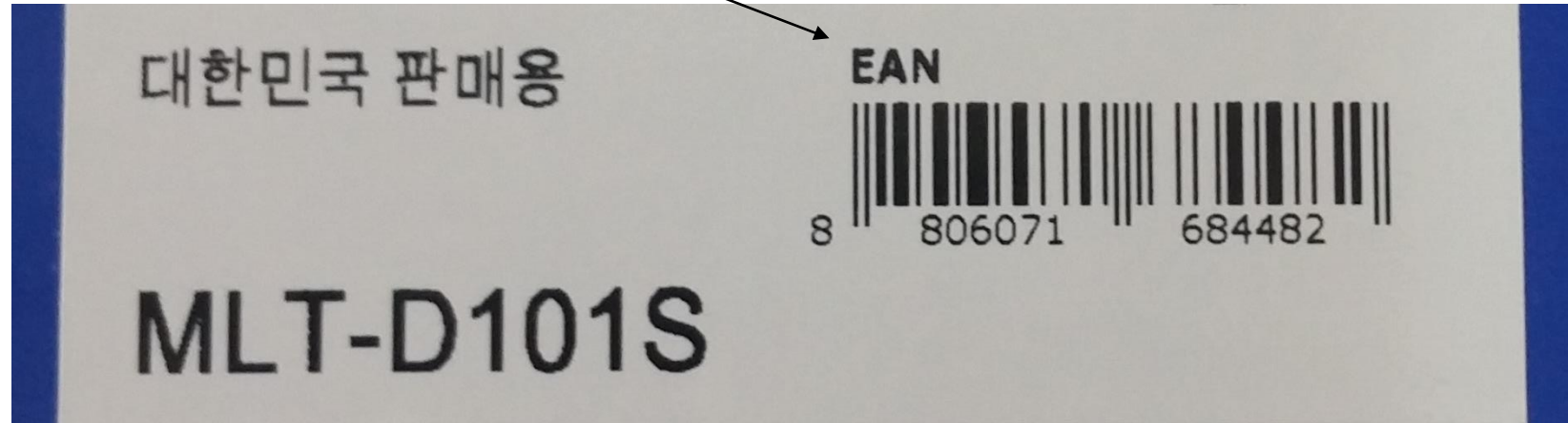


# 바코드(bar code)

- 제품의 고유 번호를 가는 막대형 표시 나타냄
- 마트에서 콜라를 의자로 계산 ?????
- 빠르고 정확한 상품가격 인식이 필요
- 소개하는 바코드는 EAN 코드의 바코드
- 다양한 코드가 존재



- European Article Number

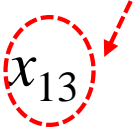


체크디지트(check digit)

880 607 168 448 2

## 체크디지트(check digit)을 계산하는 방법

체크디지트

$$\text{바코드} = x_1x_2x_3\dots x_{12}x_{13}$$


$$x_{13} = 10 - (x_1 + 3x_2 + x_3 + \dots + x_{11} + 3x_{12}) \bmod 10$$





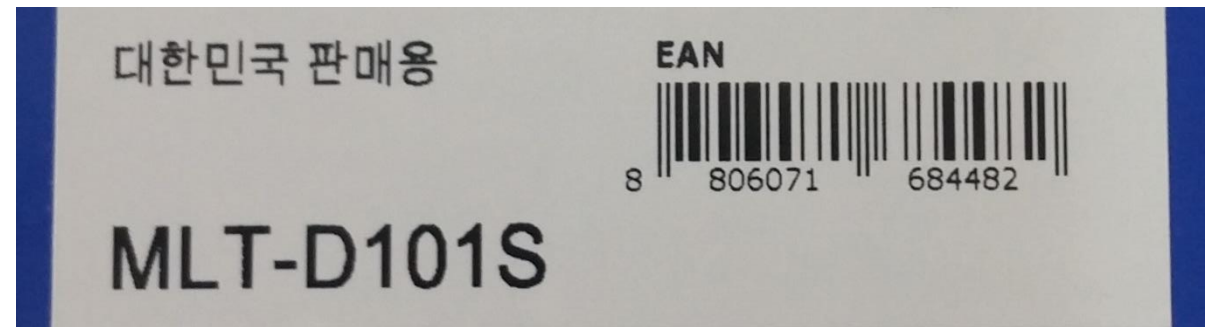
880 607 168 448 2

$$\begin{aligned}s &= 8 \times 1 + 8 \times 3 + 0 \times 1 + 6 \times 3 + 0 \times 1 + 7 \times 3 + 1 \times 1 + 6 \times 3 + 8 \times 1 + 4 \times 3 + 4 \times 1 + 8 \times 3 \\ &= 8 + 24 + 0 + 18 + 0 + 21 + 1 + 18 + 8 + 12 + 4 + 24 \\ &= 138\end{aligned}$$

$138/10=13$  나머지 8

$$10-8=2$$

체크디지트





바코드 리더



979 115 600 0129



979 115 605 0129

체크디지트를 이용해 오류 확인 가능

# ISBN

- International Standard Book Number
- 책의 고유 번호
- 바코드의 규정을 따른다.
- ISBN 데이터의 완전성을 확인하기 위한 방법으로 check digit 설정



ISBN-13: 십진수 13자리로 표시



979 115 600 012 9 체크디지트

$$\begin{aligned}s &= 9 \times 1 + 7 \times 3 + 9 \times 1 + 1 \times 3 + 1 \times 1 + 5 \times 3 + 6 \times 1 + 0 \times 3 + 0 \times 1 + 0 \times 3 + 1 \times 1 + 2 \times 3 \\&= 9 + 21 + 9 + 3 + 1 + 15 + 6 + 0 + 0 + 0 + 1 + 6 \\&= 71\end{aligned}$$

$$71/10=7 \text{ 나머지 } 1$$

$$10-1=9 \text{ check digit}$$



## Computational thinking

- 패러티 비트의 개념에서 출발하여 쉽게 데이터의 무결성을 확인할 수 있는 방법을 고안했다.
- 데이터의 변조 확인하는 방법을 컴퓨터에 접목하여 물건관리의 자동화를 할 수 있다. – 자동화(automation)
- be creative
- EAN bar code 무결성을 확인하는 알고리즘을 만들 수 있는가?
- 학번의 무결성을 확인하는 방법을 고안하라.

- Computational Thinking
  - 분해(decomposition)
  - 패턴 확인
  - 추상화(abstraction)
  - 알고리즘 설계
  - 데이터 모음
  - 데이터 표현
  - 데이터 분석
  - 모의실험
  - 자동화
  - 병렬화
  - 패턴 일반화



# 체크섬(Checksum)

	국어	영어	수학
김 ○ ○	100	90	85
이 ○ ○	90	90	90
...	...	...	...
한 ○ ○	80	90	90

T: 원래의 데이터

[problem]

- 전송 중 데이터의 변조가 발생하거나,
- 저장 중 데이터의 변조가 발생한 것을 어떻게 확인할 수 있는가?



	국어	영어	수학	체크섬
김 ○ ○	100	90	85	275
이 ○ ○	90	90	90	270
...	...	...	...	...
한 ○ ○	80	90	90	260

### T: 체크섬 열을 생성

- 체크섬 값 = 국어+영어+수학
  - ✓ 설명을 위한 단순한 방법. 다양한 다른 방법 가능
- 누군가가 과목의 성적을 바꾼다면 체크섬  $\neq$  국어+영어+수학

# 데이터 변조 시

	국어	영어	수학	체크섬
김 ○ ○	100	90	85	275
이 ○ ○	90	90	90	270
...	...	...	...	...
한 ○ ○	80	90	90	260

T: 원래의 데이터

	국어	영어	수학	체크섬
김 ○ ○	100	90	85	275
이 ○ ○	90	90	90	270
...	...	...	...	...
한 ○ ○	80	100	90	260

T': 수신자가 받은 데이터 또는 1년  
후의 데이터 상태

$$80+100+90 \neq 260$$

- 변조되었다는 사실만 파악.
- 이 예에서는 변경된 데이터의 위치 확인 불가





# 데이터 변조 시

	국어	영어	수학	체크섬
김 ○ ○	100	90	85	275
이 ○ ○	90	90	90	270
...	...	...	...	...
한 ○ ○	80	90	90	260

T: 원래의 데이터

변조

	국어	영어	수학	체크섬
김 ○ ○	100	90	85	275
이 ○ ○	90	90	90	270
...	...	...	...	...
한 ○ ○	80	100	90	270

T': 수신자가 받은 데이터 또는 1년  
후의 데이터 상태

$$80+100+90 = 270$$

- 과목의 성적과 체크섬을 동시에 변조하면, 변조 사실 확인 불가
- [해결방안] 체크섬을 계산하는 방식을 예상하기 어렵게 만든다. 단순 합이 아닌 방법으로 설정.
- (예) 체크섬 =  $(35*국어+56*영어+2*수학*수학)/6$



과목 체크섬 9+5=14

	국어		영어		수학		체크섬
김 ○ ○	100	1	91	10	85	13	276
이 ○ ○	94	13	93	12	92	11	279
...	...		...		...		...
한 ○ ○	82	10	95	14	91	10	268

T: 원래의 데이터

변조 불일치

	국어		영어		수학		체크섬
김 ○ ○	100	1	91	10	85	13	276
이 ○ ○	94	13	93	12	92	11	279
...	...		...		...		...
한 ○ ○	82	10	99	14	91	10	272

T': 수신자가 받은 데이터 또는 1년  
후의 데이터 상태

$$82+99+91=272$$

- 각 과목 성적의 체크섬을 추가한다.
- 95점인 경우 각 자리수의 합 9+5=14 추가
- 모든 과목 성적의 합 체크섬이 과목 성적과 일치하더라도, 각 과목의 체크섬 확인으로 과목 성적 변조 확인 가능



## Computational thinking

- 체크섬을 계산하는 공식을 복잡하게 하여 유추할 수 없도록 한다.
- 데이터의 주기적인 체크섬 확인으로 해킹에 의한 변조를 확인할 수 있다.
- 데이터 표현을 어떻게 하느냐에 따라 변조 사실을 파악할 수 있다.
- be creative



## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

## ● 문제 해결 전략

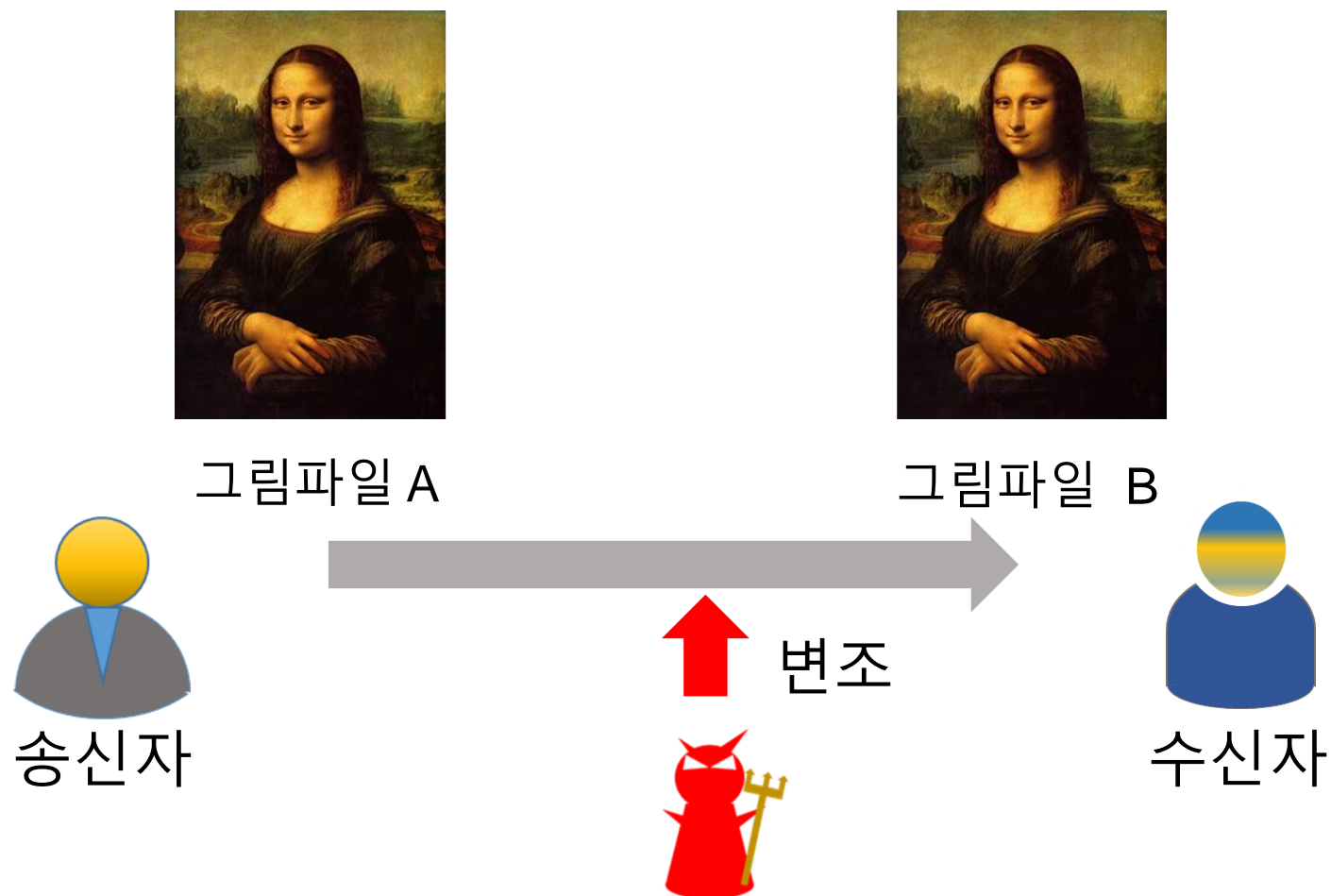
- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용



# 메시지 다이제스트(Message Digest)

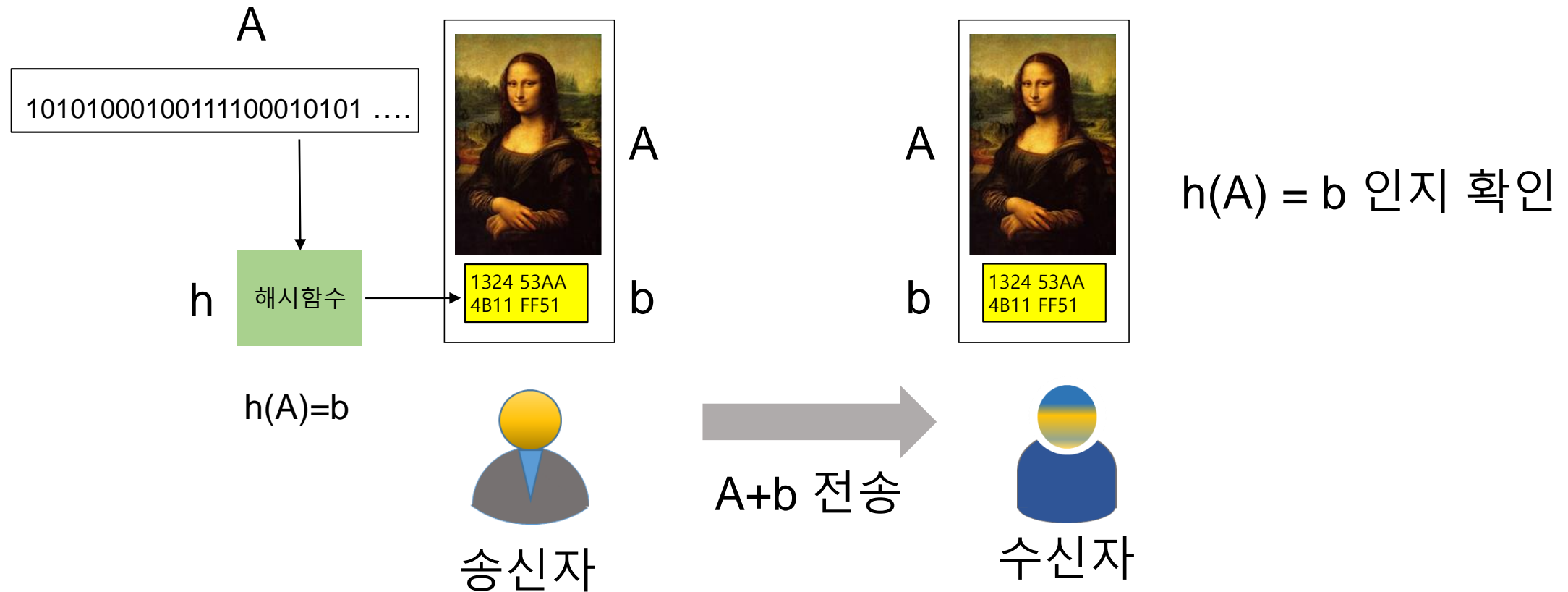
[problem]

그림 파일 B는 그림 파일 A와 동일한 것인지 확인할 방법이 있는가?

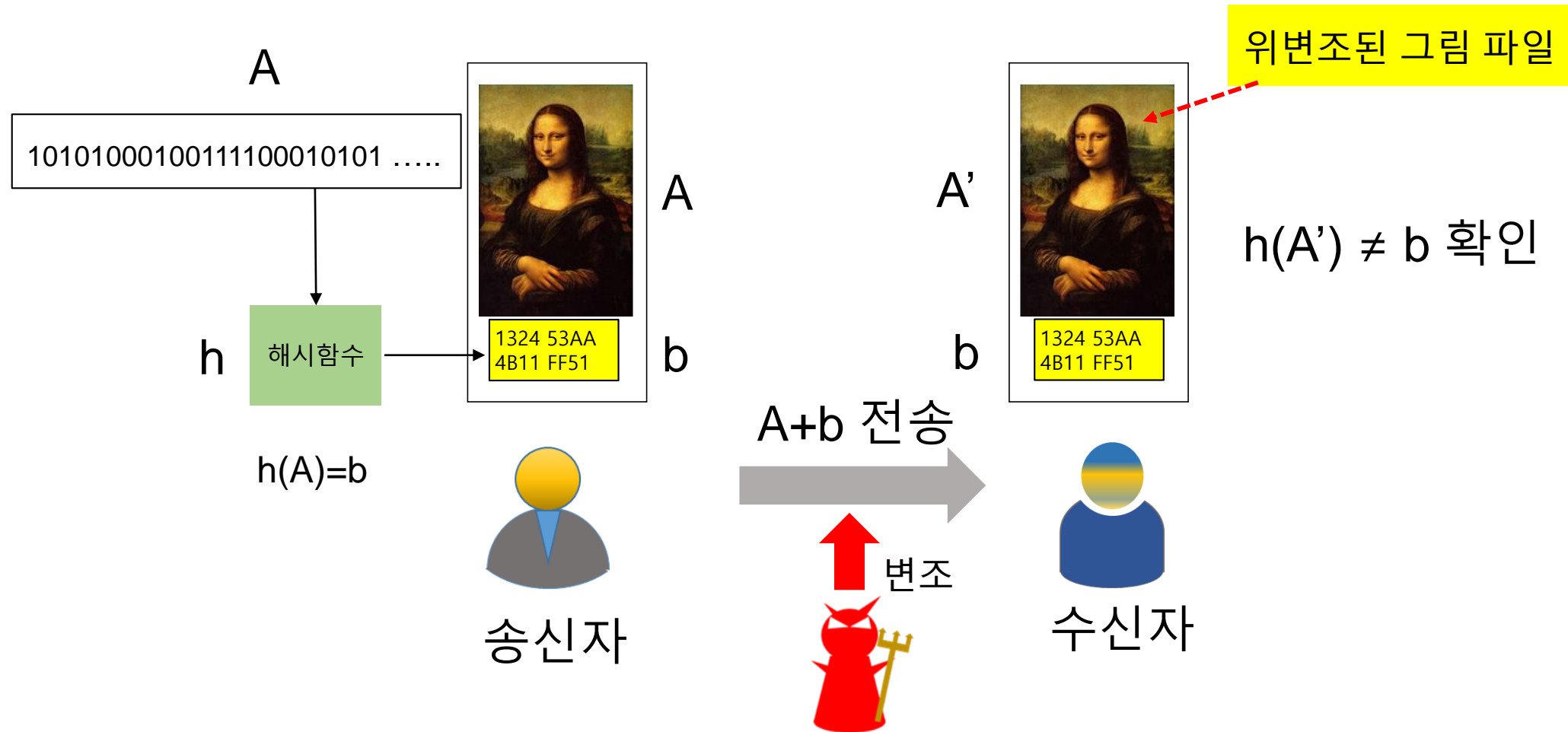


- 이미지 파일이든지 텍스트 파일이든지 모든 데이터는 0 또는 1로 구성되어 있다.
- 복잡한 해시함수( $h$ )에 이들 데이터( $A$ )를 입력으로 주면, 결과값(메시지 다이제스트,  $b$ )이 생성, 즉  $h(A)=b$
- 그림 파일( $A$ )과 메시지 다이제스트( $b$ )를 송신
- 수신자는 데이터  $A$ 를 미리 공유한 해시함수  $h$ 에 입력으로 주고, 그 결과값이  $b$ 가 되는지를 확인
- 수신자는 결과값의 동일 여부를 확인하여, 위변조 여부를 확인할 수 있다.

## [해결 방안] 정상 수신인 경우



변조에 의해 A'가 수신된다면,  $h(A') \neq b$ 이므로, 위변조 사실 확인 가능





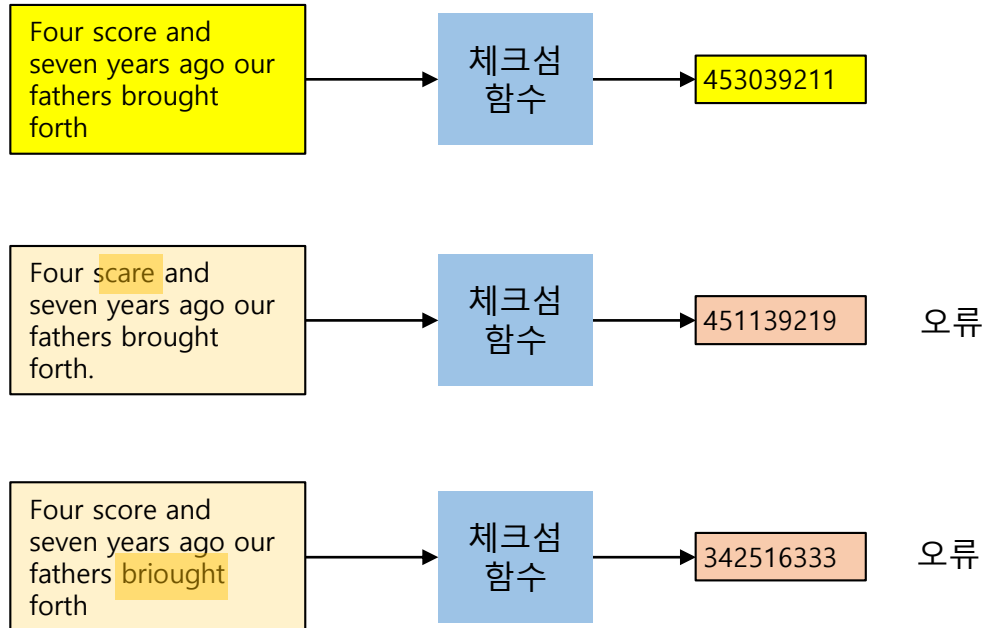
# 메시지 다이제스트(Message Digest)

- 암호화 해시함수(cryptographic hash function)
- 체크섬보다 더 긴 데이터 길이의 메시지 다이제스트를 생성한다.
- 생성된 메시지 다이제스트를 이용해서 원래의 데이터를 재현하는 것이 거의 불가능하다.
- 데이터가 바뀌면 메시지 다이제스트도 바뀐다
- 문서의 위변조 유무를 파악
- 이미지의 변조가 있는지 파악 가능

# 체크섬과 메시지 다이제스트

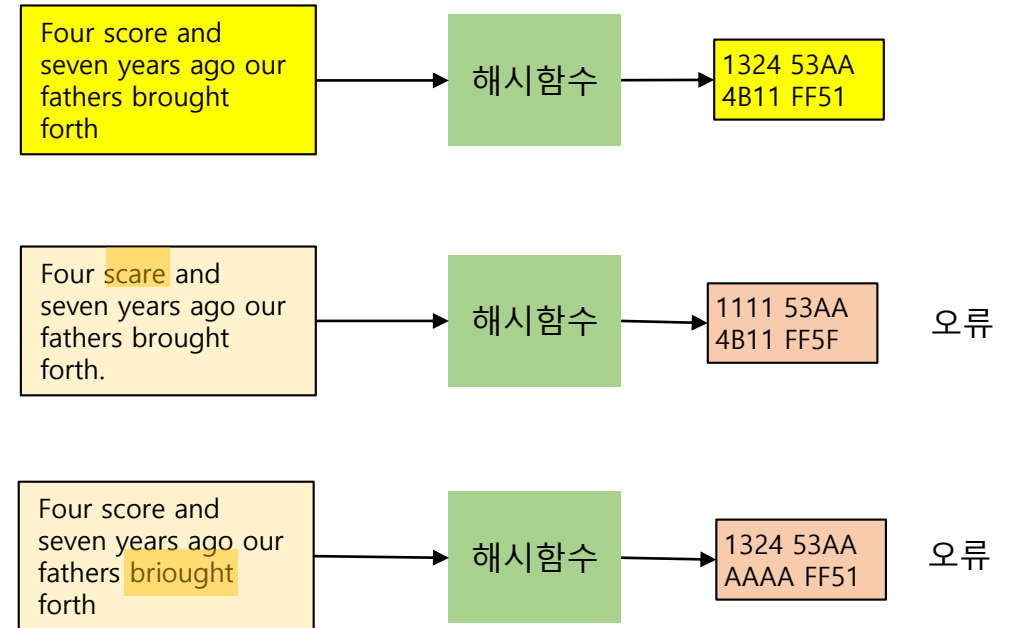
입력

체크섬



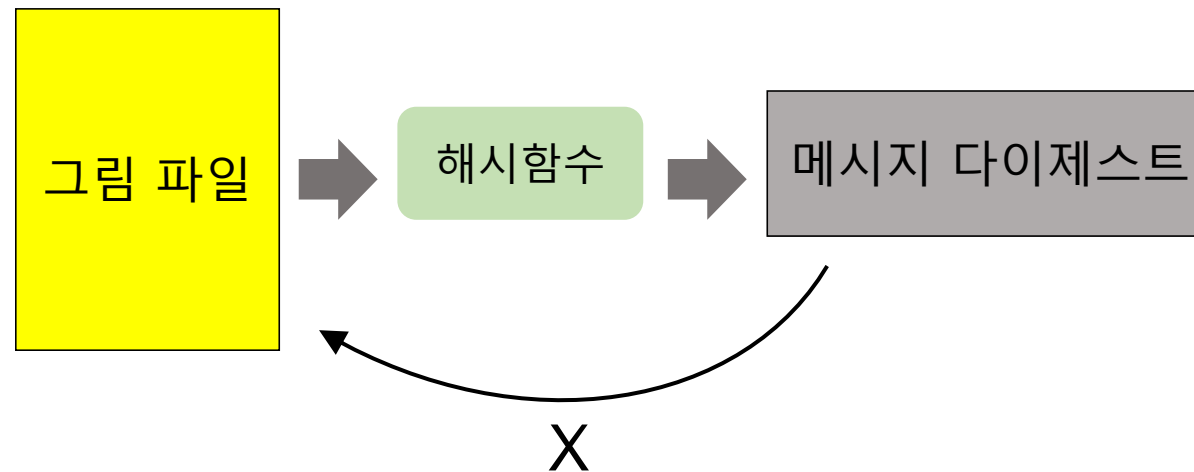
입력

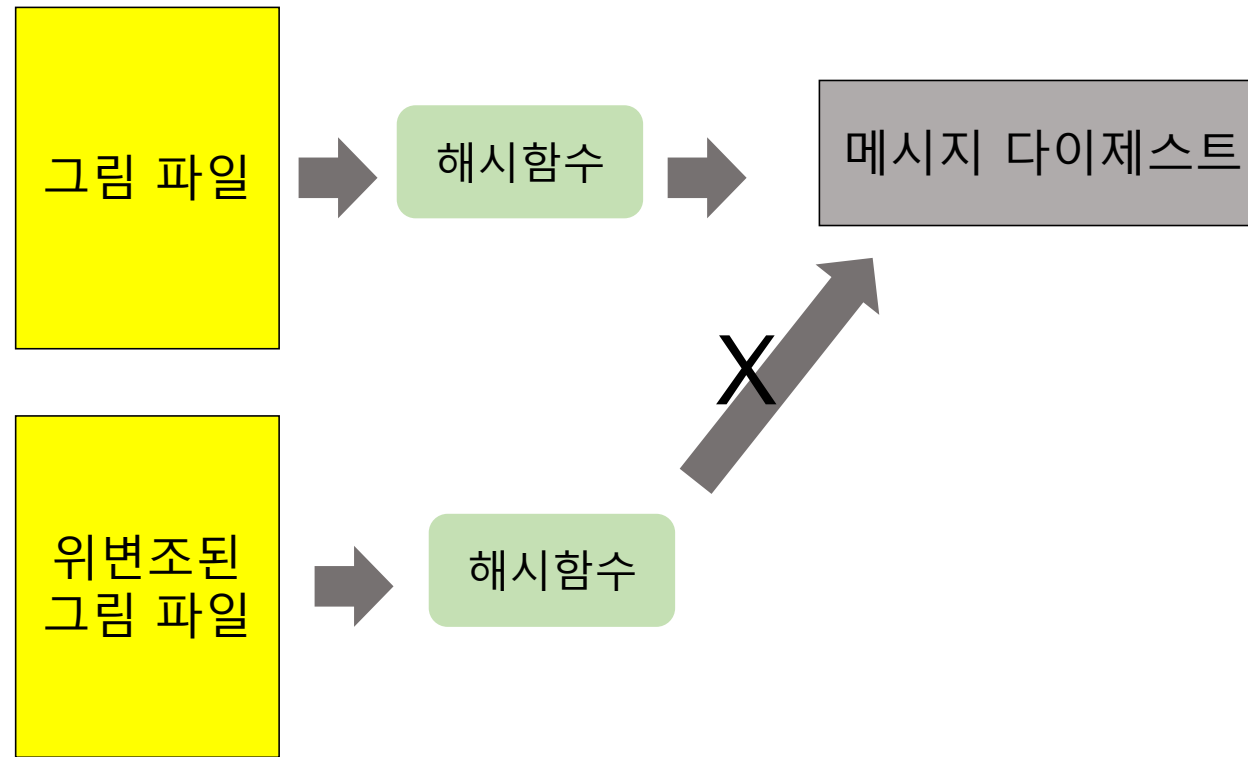
메시지 다이제스트



이상적인 메시지 다이제스트의 변환 함수(hash)가 가져야 할 성질

- 어떠한 데이터가 주어지더라도 해시값을 쉽게 계산할 수 있어야 한다.
- 해시값으로부터 원래의 메시지를 생성하는 것이 불가능해야 한다.





- 데이터가 일부 수정되면 원래 데이터의 메시지 다이제스트와 동일한 메시지 다이제스트를 생성할 수 없다.

## Computational thinking

- 패리티 비트의 개념에서 출발하여 서로 주고 받는 문서, 이미지 데이터의 무결성을 확인할 수 있는 방법을 고안했다.
- 데이터의 변조를 확인하여 전자상거래, 저작권 등에 활용한다.



## ● 문제 해결 과정

- 1.문제를 이해한다
- 2.계획을 세운다
- 3.계획을 실행한다
- 4.풀이과정을 재점검한다

## ● Computational Thinking

- 분해
- 패턴 확인
- 추상화
- 알고리즘 설계
- 데이터 모음
- 데이터 표현
- 데이터 분석
- 모의실험
- 자동화
- 병렬화
- 패턴 일반화

## ● 문제 해결 전략

- 추정과 확인
- 순서 리스트 작성
- 가능성들을 제거
- 대칭 성질을 이용
- 특별한 경우를 고려
- 직접 추론 사용
- 방정식을 푼다
- 패턴을 찾는다
- 그림을 그린다
- 단순한 문제 해결 시도
- 모델 사용
- 역방향으로 시도
- 공식 사용
- 창조적 생각
- 잔피를 사용



## 9주차 강의 요약

### 데이터의 표현

- 인코딩 및 압축
  - ✓ run-length 인코딩
  - ✓ LZSS 인코딩
  - ✓ 후프만코드
  - ✓ 반복 코드
  - ✓ 패러티 비트
- 오류확인
  - ✓ 바코드
  - ✓ 체크섬, 메시지 다이제스트





9주차 끝

수고하셨습니다.