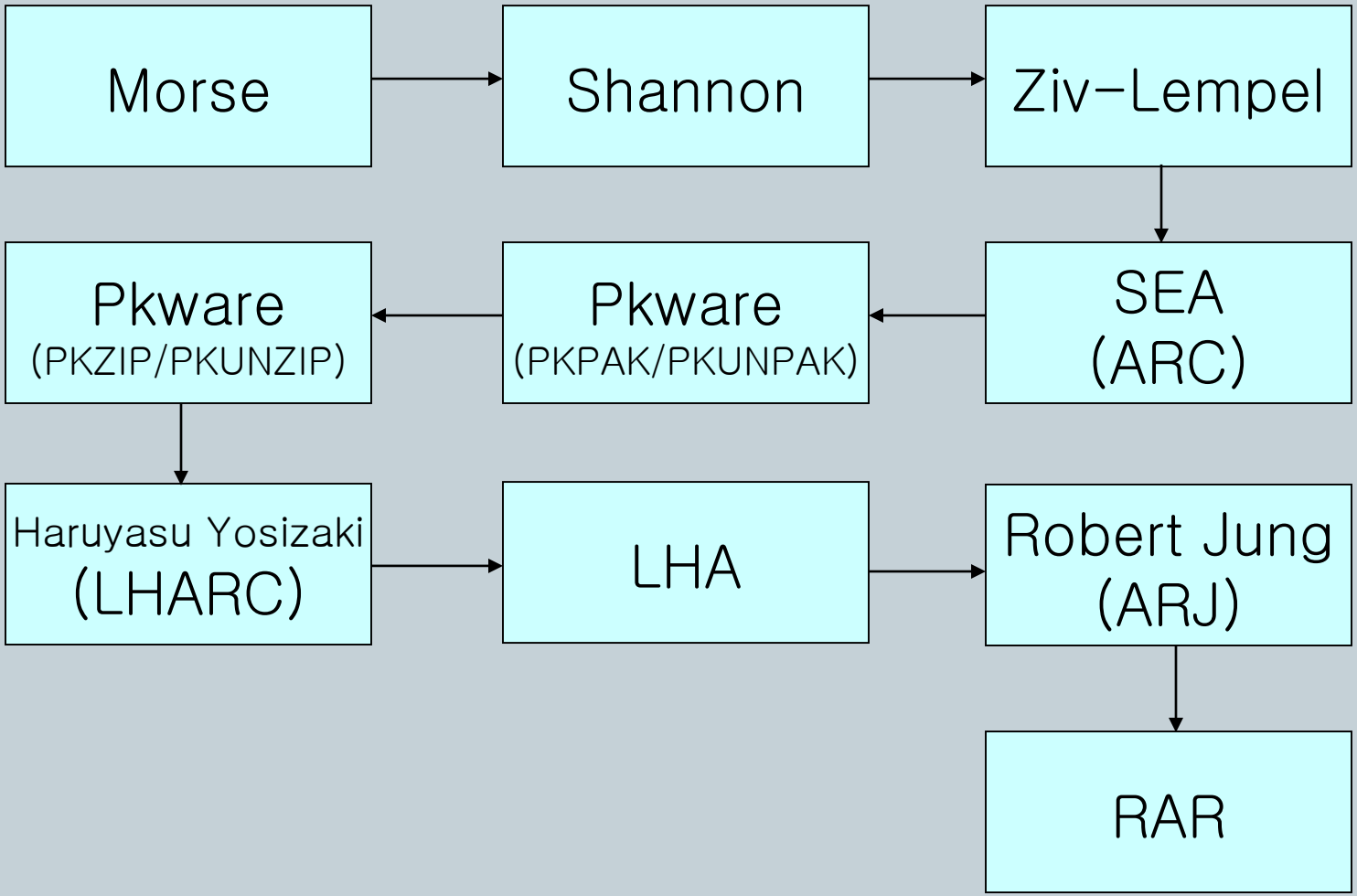
화일의 압축(file compression)

1. 화일 압축의 사용과 원리

- 화일 압축의 사용
  - 자주 사용하지 않는 화일들을 장기 보관할 목적으로 저장하는데 사용
  - 자료의 양을 줄임으로써 전송시간의 단축과 전송비용의 절감을 위해 사용
- 화일 압축의 원리
  - 반복문자를 문자와 반복횟수로
  - 자주 나타나는 문자는 짧은 부호어로, 드물게 나타나는 문자는 긴 부호어로 나타냄
  - 기타

2. 화일 압축의 변천


Morse
Shannon
Ziv-Lempel
SEA
(ARC)
Pkware
(PKPAK/PKUNPAK)
Pkware
(PKZIP/PKUNZIP)
Haruyasu Yosizaki
(LHARC)
LHA
Robert Jung
(ARJ)
RAR

모스(Morse)부호의 예

영문모스부호		한글모스부호		숫자모스부호		약부호
A	·－	ㄱ	·－··	1	·－－－－	·－
B	－···	ㄴ	··－·	2	··－－－	··－
C	－·－·	ㄷ	－···	3	···－－	···－
D	－··	ㄹ	···－	4	····－	····－
E	·	ㅁ	－－	5	·····	·····
F	··－·	ㅂ	·－－	6	－····	－····
G	－－·	ㅅ	－－·	7	－－···	－···
H	····	ㅇ	－·－	8	－－－··	－··
I	··	ㅈ	·－－·	9	－－－－·	－·
J	·－－－	ㅊ	－·－·	0	－－－－－	－
K	－·－	ㅋ	－··－			
L	·－··	ㅌ	－－··			
M	－－	ㅍ	－－－			
N	－·	ㅎ	·－－－			
O	－－－	ㅏ	·			
P	·－－·	ㅑ	··			
Q	－－·－	ㅓ	－			
R	·－·	ㅕ	···			
S	···	ㅗ	·－			
T	－	ㅛ	－·			
U	··－	ㅜ	····			
V	···－	ㅠ	·－·			
W	·－－	ㅡ	－··			
X	－··－	ㅣ	··－			
Y	－·－－	ㅐ	－－·－			
Z	－－··	ㅔ	－·－－			

3. 무손실압축과 손실압축

- 무손실압축(reversible compression: 가역압축)
  - 텍스트 문서나 실행형 화일처럼 완전히 원래의 같은 데이터로 복원할 필요가 있는 경우
- 손실압축(irreversible compression: 비가역압축)
  - 음성신호, 화상 및 그래픽 데이터처럼 대략 비슷한 데이터로 복원해도 좋을 경우
  - 예) JPEG, MPEG, MP3

4. PC용 압축 프로그램

- SQ : 1983, 허프만 코딩을 구현
  최초의 PC용 압축 프로그램
- ARC : 1985(SEA), LZW를 채택
- PKARC/PKXARC : 필 케츠,
  ARC보다 3~10배 빠름
- PKZIP : 1988 후반,
  ZIP이라는 표준 FORMAT 탄생
- LHARC : 1989, 하루야쯔 요시자끼,
  ZIP보다 10%정도 빠름, LHA의 전신
- ARJ : 1990, 로버트 정

5. 화일 압축기법

1. 숫자자료에 대한 압축
   1) 숫자 0의 압축
   2) 델타코딩기법
2. 문자자료에 대한 압축기법
   1) Run Length Encoding(RLE)
   2) Variable Length Encoding(Huffman Coding

5.1. 숫자 자료의 압축

1) 숫자 0의 압축 방법

[기존의 화일]

0;0;0;0;7;0;0;0;0;0;0;0;0;0;0;24;0;0;0;0;0;0;0;0;0;0;48;0;0;0;0;0;

[압축된 화일]

5,7: 16,24: 27,48:

의미있는 숫자의 내용

위치정보

5.1. 숫자 자료의 압축

2) 델타코딩의 방법

[기존의 화일]

125; 127; 132; 129; 131; 135; 140; 147; 145; 149

[압축된 화일]

125; ; 4비트; +2; +5; -3; +2; +4; +5; +2; -2; +4

두번째 이후의 델타 값

델타의 크기

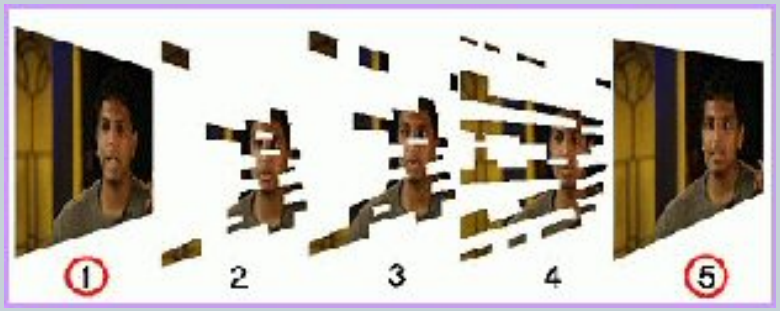
최초의 데이터 값

Delta(δ)는 차이(difference)를 의미함

적용사례: MPEG의 압축원리

- 압축 전

- 압축 후(Key frame 과 Delta frame)



- VKI 방식(Variable Key Frame Interval
  - 대개 일정한 간격으로 key frame을 삽입
  - 동작 변화가 적은 부분에서는 용량낭비
  - 동작 변화가 심한 곳에서는 화질 저하
  - Key frame의 삽입간격을 가변적으로 함
  - 최근의 Divx

5.2. 문자 자료의 압축(RLE)

1) RLE(Run Length Encoding)압축 방법

[기존의 화일 : 총 48 바이트]

FILE PROCESSING BOOK

10개의 공백 20개의 공백

[압축된 화일]

FILE<032><126><9>PROCESSING<032><126><19>BOOK

반복되는 횟수

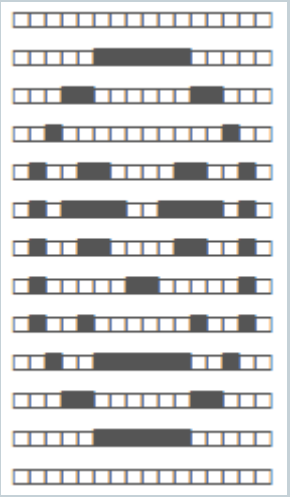
특수 코드

반복되는 문자의 코드(공백:ASCII코드)

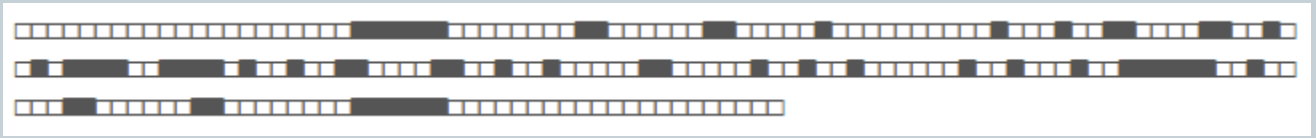
- 화일에서 가장 기본적인 중복은 같은 문자가 연속적으로 나타나는 것임
  - AAAABBCCC → 4A2B3C
  - 0000001111000 → 6 4 3 (반복횟수만 기록)
- 문자를 사용한 화일 압축 기법은 숫자와 문자가 혼합되어 있는 화일에는 적용할 수 없음
  - 텍스트에서 드물게 나타나는 문자를 탈출 문자(escape character) 로 사용
  - 탈출 순차(escape sequence) : 탈출 문자, 개수, 1개의 반복 문자
  - AAAABBCCCCC → QDABBQEC
  - Q : Q<space>
  - 50개의 B : QZBQXB

RLE의 예

- 그래픽 파일의 경우 대부분 무수히 많은 반복을 갖는다
- 다음과 같은 흑백비트맵을 보자



- 실제 저장된 모습은 다음과 같다(bmp화일형식)



- RLE로 압축하면 208비트가 124비트로 압축됨(pcx화일 형식)

21□6■8□2■6□2■5□1■10□1■3□1■2□2■4□2■2□1■2□1■1□4■2□4■1□1■2□1■2□2■4□2■2□1■2□1■5□2■5□
1■2□1■2□1■6□1■2□1■3□1■2□6■2□1■5□2■6□2■8□6■21□

5.2. 문자 자료의 압축(VLE)

- 가변-길이 인코딩(variable-length encoding)
  - 자주 나타나는 문자에는 적은 비트를 사용하고 드물게 나타나는 문자에는 많은 비트를 사용하여 인코드(encode)하는 압축 기법
- ABRACADABRA 에 다섯 비트를 사용해보자

문자	A	B	C	D	R
빈도수	5	2	1	1	2
코드	00001	00010	00011	00100	10010

  - 0000100010100100000100011000010010000001000101001000001(55비트)

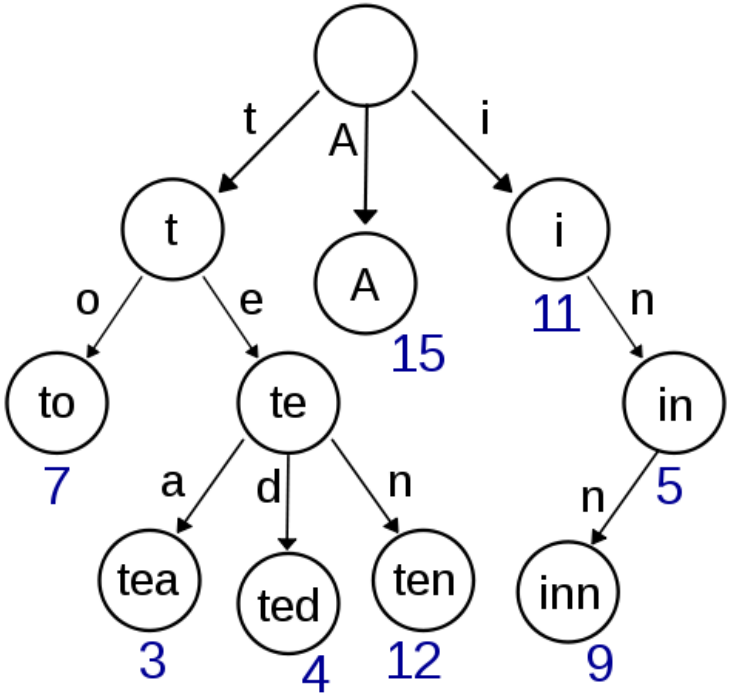
- ABRACADABRA에 대한 코드를 다음과 같이 할당해보면?

문자	A	B	C	D	R
빈도수	5	2	1	1	2
코드	0	1	10	11	01

  - 010101001101010 (25비트 delimiter를 사용하지 않으면 디코드가 불가능)
  - 0;1;01;0;10;0;11;0;1;01;0(delimiter를 사용하지 않을 수 있나?)

트라이(1)

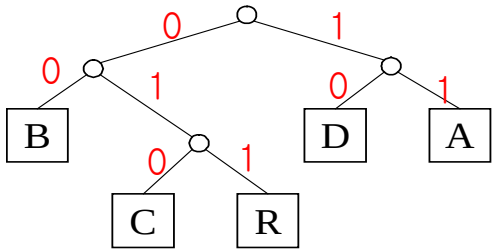
- 트라이(trie)
  - 스트링을 가지는 동적 셋 혹은 연관 배열을 저장하는데 사용되는 정렬된 트리 자료구조
  - 이진 검색 트리와는 다르게, 트리 상에서의 노드들은 그 노드와 연관된 키를 저장하지 않음
  - 트리상에서의 위치가 현재의 키를 나타냄
  - 특정 노드의 모든 후손들은 그 노드에서와 동일한 prefix를 가지며, 루트 노드는 빈 문자열을 가짐



트라이(2)

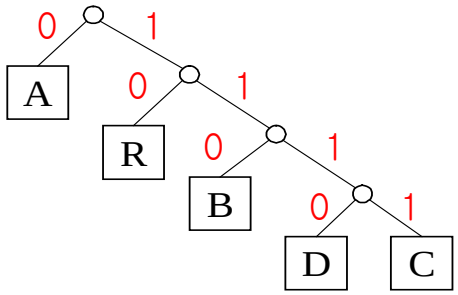
- 트라이(trie)
  - 하나의 문자 코드가 다른 문자 코드의 접두부가 되지 않는 것을 보장하는 자료구조
    - 즉, 구분문자(delimiter)를 사용하지 않아도 됨
  - 이진트라이로 비트 스트링을 유일하게 디코드 할 수 있음

예 : ABRACADABRA에 대한 두 가지 인코딩 트라이



A: 11, B: 00, C: 010 D: 10, R: 011

11000111101011101100011111(25비트)



A: 0, R: 10, B: 110 D: 1110, C: 1111

01101001111011100110100(23비트)

허프만 인코딩(1)

- 하나의 문자열로부터 여러 개의 트라이를 생성할 수 있음
  - 여러 개의 트라이 중에서 어떤 것이 가장 효율적인 것인지를 결정 방법이 있어야 함
- 허프만 인코딩(Huffman encoding)
  - 여러 트라이 중 가장 좋은 트라이를 결정하는 일반적인 기법
  - 우선순위 큐를 사용하여 빈도수가 가장 작은 문자부터 차례로 트라이를 만듦

허프만 인코딩(2)

허프만 압축법(Huffman Encoding)

① 압축할 화일을 읽어 각 문자들의 출현 빈도를 구한다.

② 빈도수가 가장 작은 문자끼리 연결하여 이진 트라이를 만든다.

③ 과정 ②를 계속 진행시켜 모든 문자에 대한 이진 트라이를 완성한다.

④ 완성된 이진 트라이에서 각 문자를 대표하는 값을 부여한다.

⑤ 파일 내의 문자들을 대표값으로 변환하여 압축화일로 만든다.

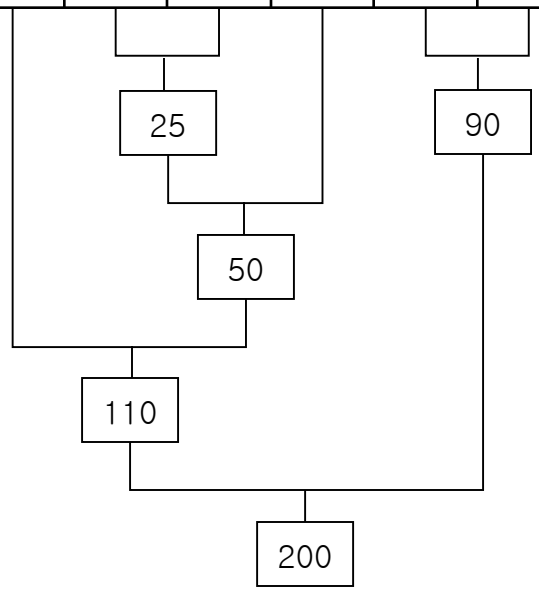
[화일의 문자 출현 빈도]

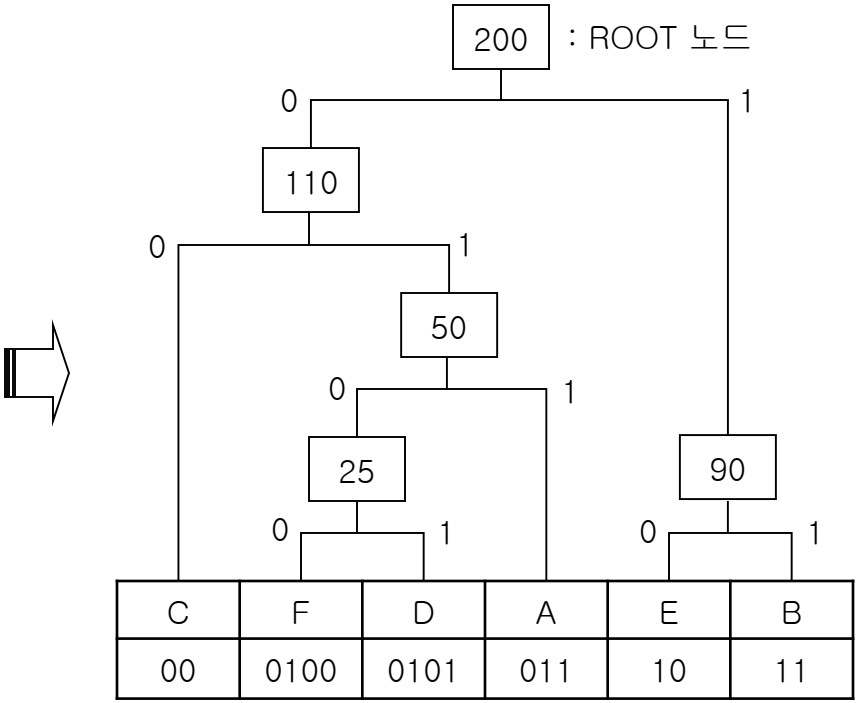
문자	A	B	C	D	E	F	전체
빈도수	25	40	60	10	50	15	200

문자의 임의 나열

빈도수	60	15	10	25	50	40
문자	C	F	D	A	E	B

이진 트리의
최종 생성





[원래 화일의 빈도수와 비트수]

문자	A	B	C	D	E	F	전체
빈도수	25	40	60	10	50	15	200
비트수	200	320	480	80	400	120	1600

[압축 화일의 비트수]

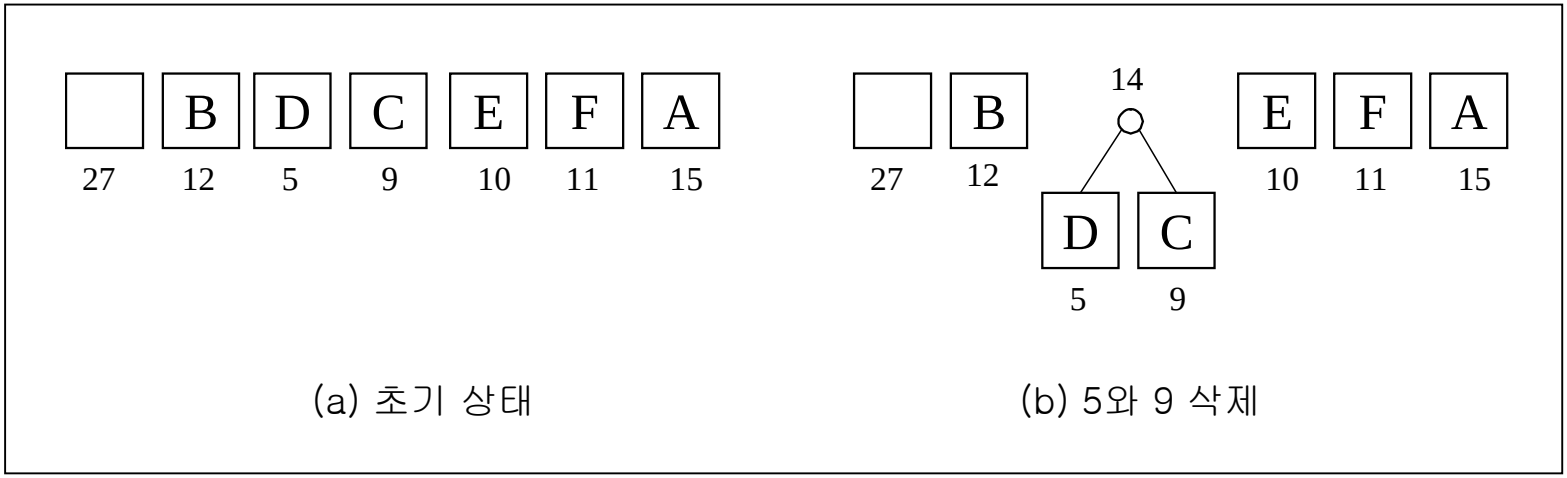
문자	A	B	C	D	E	F	전체
빈도수	25	40	60	10	50	15	200
대표값	011	11	00	0101	10	0100	
비트수	75	80	120	40	100	60	475

허프만 트리 구축 과정(1)

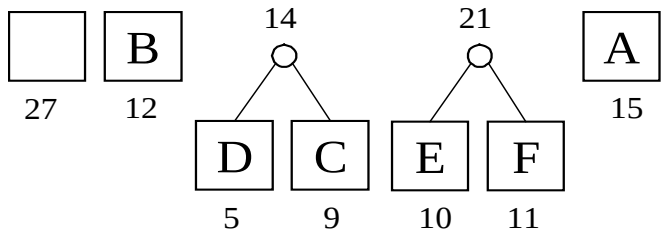
- 주어진 텍스트의 빈도수를 계산

		A	B	C	D	E	F
k	0	1	2	3	4	5	6
count[k]	27	15	12	9	5	10	11

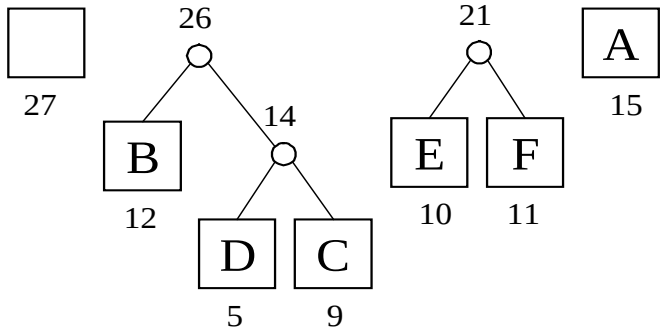
- 허프만 트리 구축



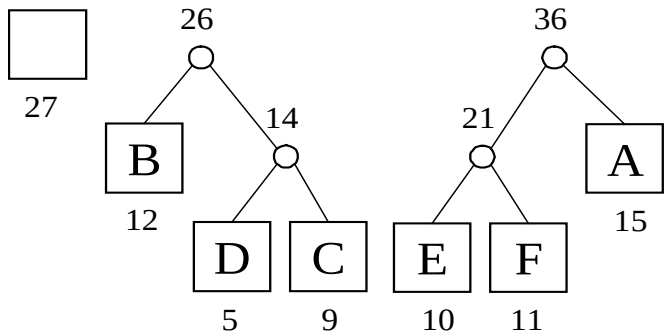
허프만 트리 구축 과정(2)



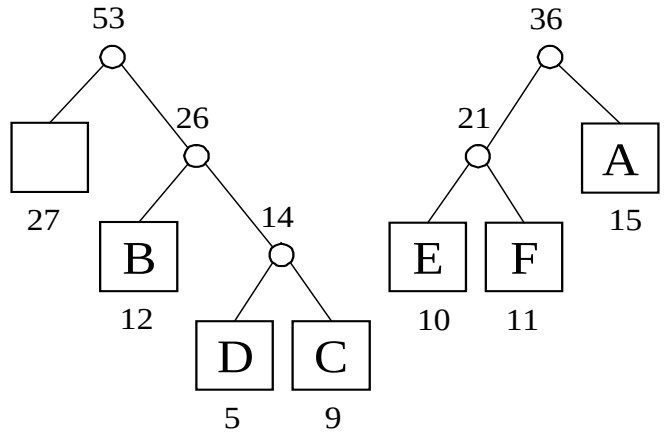
(c) 10과 11 삭제



(d) 14와 12 삭제

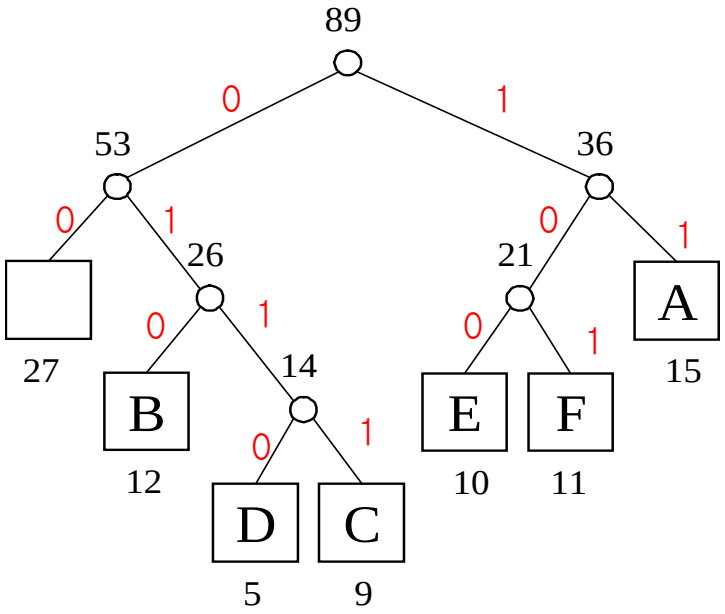


(e) 15와 21 삭제



(f) 26과 27 삭제

허프만 트리 구축 과정(3)



(g) 36과 53 삭제

허프만 트리 구축 과정(4)

- 허프만 인코딩에 사용될 코드를 할당

		A	B	C	D	E	F
k	0	1	2	3	4	5	6
len[k]	2	2	3	4	4	3	3
code[k]	0	3	2	6	7	4	5
	00	11	010	0110	0111	100	101

  - len[k] 허프만 코드의 길이
  - code[k] 십진수로 표현한 허프만 코드

허프만 인코딩 실행 예(1)

- VISION QUESTION ONION CAPTION GRADUATION EDUCATION
- 문자 빈도수

		A	C	D	E	G	I	N	O	P	Q	R	S	T	U	V
k	0	1	3	4	5	7	9	14	15	16	17	18	19	20	21	22
count[k]	5	4	2	2	2							1	2	4	3	1

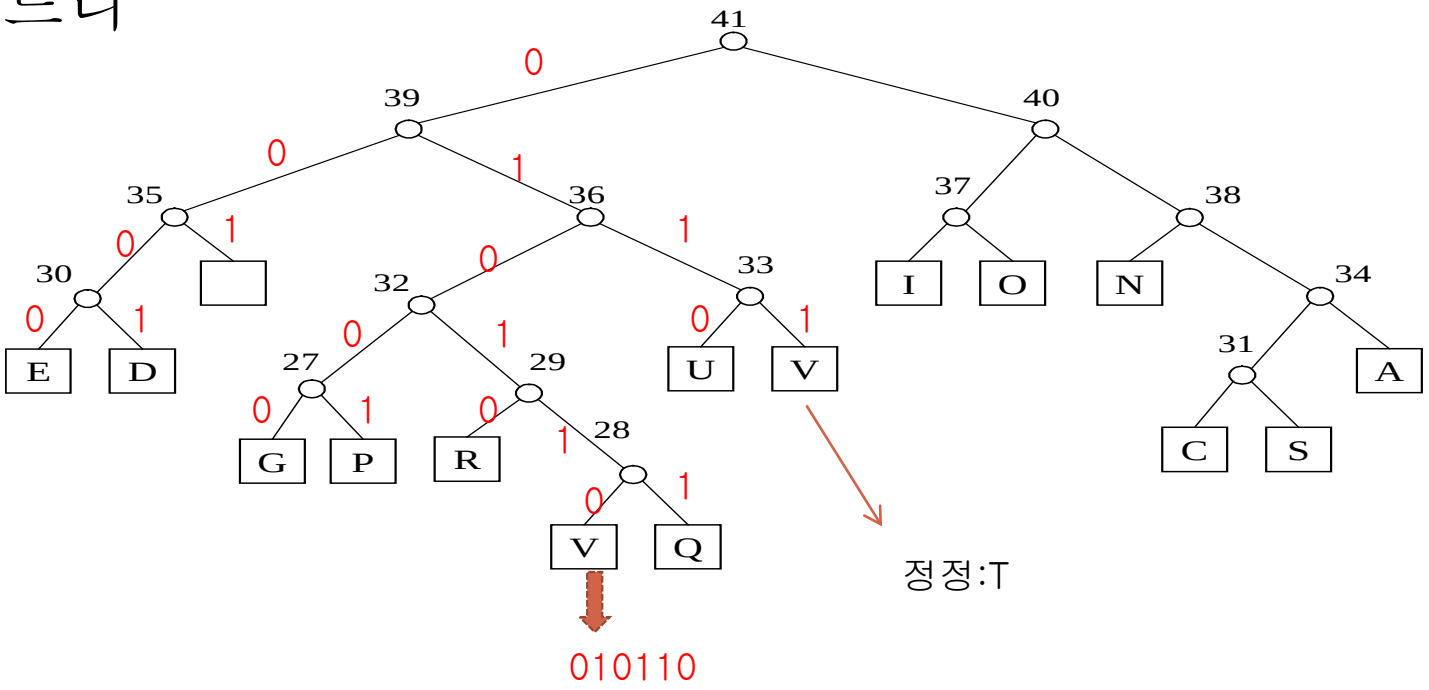
Count 중 최소값 두 개를 찾아 27부터 차례로 +, – 부호를 붙여 할당함

- 부모 노드 표현

k	0	1	3	4	5	7	9	14	15	16	17	18	19	20	21	22
count[k]	5	4	2	2	2	1	7	7	7	1	1	1	2	4	3	1
dad[k]	-35	-34	31	-30	30	27	37	38	-37	-27	-28	29	-31	-33	33	28
k	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	
count[k]	2	2	3	4	4	5	7	8	9	12	14	15	21	29	50	
dad[k]	32	-29	-32	35	34	36	-36	-38	39	-39	40	-40	41	-41	0	

허프만 인코딩 실행 예(2)

- 허프만 트리



허프만 인코딩 실행 예(3)

- “V(k=22)”에 대한 Encoding 사례
  - +28 : 0
  - -29 : 1
  - -32 : 1
  - +36 : 0
  - -39 : 1
  - +41 : 0
  - 따라서 010110

k	0	1	3	4	5	7	9	14	15	16	17	18	19	20	21	22
count[k]	5	4	2	2	2	1	7	7	7	1	1	1	2	4	3	1
dad[k]	-35	-34	31	-30	30	27	37	38	-37	-27	-28	29	-31	-33	33	28
k	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	
count[k]	2	2	3	4	4	5	7	8	9	12	14	15	21	29	50	
dad[k]	32	-29	-32	35	34	36	-36	-38	39	-39	40	-40	41	-41	0	

- 010110에 대한 Decoding
- Root(k=41부터 역순 탐색) : 22(k=V)에 도달함
  - 0 : +41 인덱스 탐색 : 39
  - 1 : -39 인덱스 탐색 : 36
  - 0 : +36 인덱스 탐색 : 32
  - 1 : -32 인덱스 탐색 : 29
  - 1 : -29 인덱스 탐색 : 28
  - 0 : +28 인덱스 탐색 : 22(“V”)

허프만 인코딩 실행 예(4)

- 허프만 코드

	k	code[k]	len[k]	
	0	1	3	001
A	1	15	4	1111
C	3	28	5	11100
D	4	1	4	0001
E	5	0	4	0000
G	7	8	5	01000
I	9	4	3	100
N	14	6	3	110
O	15	5	3	101
P	16	9	5	01001
Q	17	23	6	010111
R	18	10	5	01010
S	19	29	5	11101
T	20	7	4	0111
U	21	6	4	0110
V	22	22	6	010110