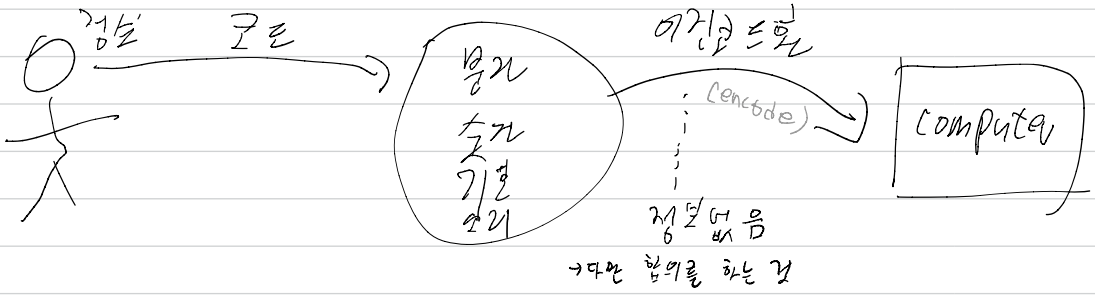
한글의 인코딩

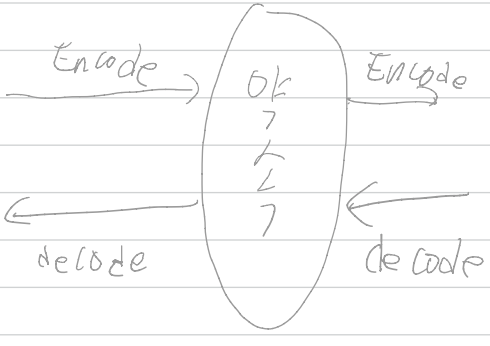
조합형 : 김 : ㄱ + ㅣ + ㅁ = 12bit

뷁: ㅂ+ㅜ+ㅖ+ㄹ+ㄱ = 48bit

완성형 : 16bit 무조건

유니코드가 나오면서 완성형과 조합형이 둘다 처리 가능해졌다.



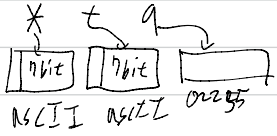


문자 데이터의 압축

코딩의 효율성은 얼마나 적은 bit를 가지고 얼마나 많은 정보를 표현할 수 있는가 라는 것이다
모스부호 ->정적으로 코드표가 정의
-> 동적이면 더 효율적?

ketword encoding : 매우 자주 사용되는 the, and, ad, who, Big, Party 등은
하나의 bit에 대응시키는 방법

run-left encoding은 여러 수가 반복되어서 나오는 경우,
kkkkkkkkk -> *k9
*c9*x5 -> ccccccccxxxxx
sss는 *s3로 압축하지 않는다.
어차피 3글자이기 때문이다.



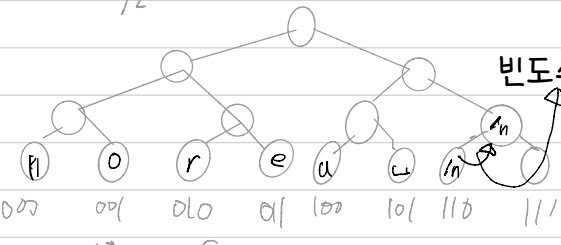
허프만 코딩 :
모스(정적), 허프만 코딩 (동적)

허프만 코딩 : 아스키 코드를 똑같이 쓴다고 했을 때,
그 길이가 똑같을때, 8bit씩 끊어 읽을 수 있는데,
허프만은 그것인 아니다.

모스부호는 시간으로 글자간 분리를 만들었다
허프만은 무엇으로 분리할 것인가?

어느 한 코드가, 다른 코드의 prefix가 되면 안된다.
모스부호는 이런 조건을 만족하지 않는다.

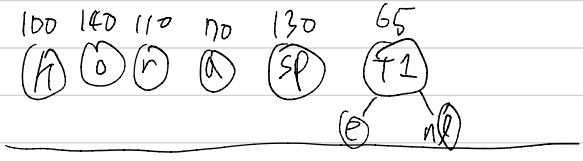
코드가 3bit를 사용한다고 가정하는 것이다.
7개의 문자라고 할때, $\lceil \log_2 7 \rceil = 3$

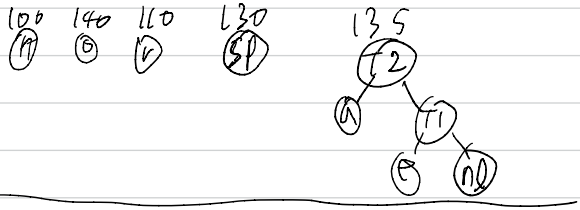


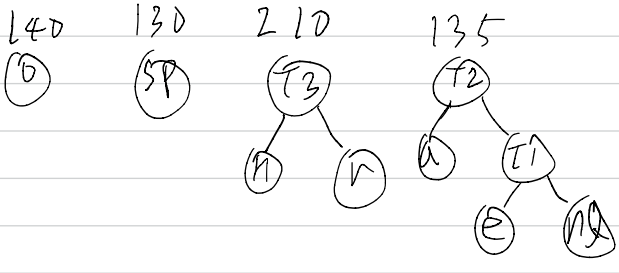
100 140 110 50 70 130 65

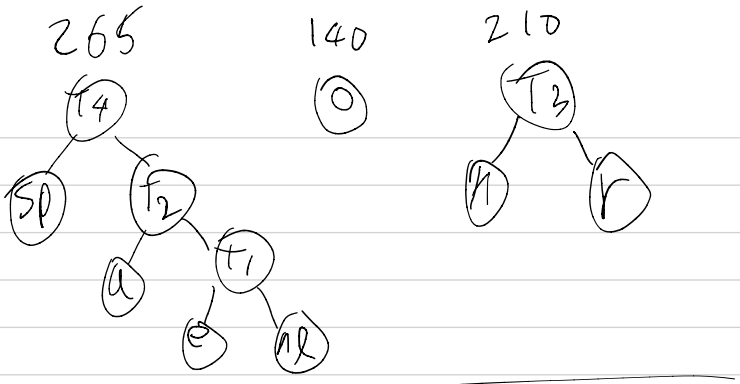
H o r e a sp nl

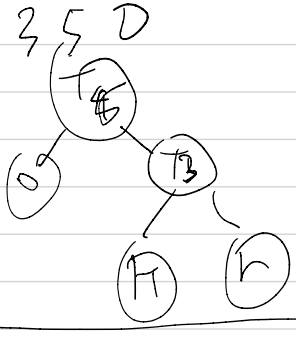
빈도수가 가장 적은 것 2개를 합침





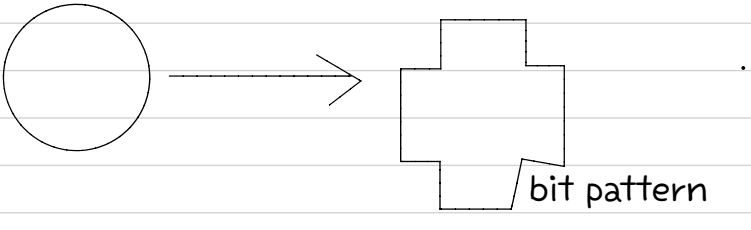






가장 기본은 utf-16인데, 영어권에서는 utf-8, 변형 영어권은 utf-16bit
한글, 한자는 utf-32bit이다.

이미지 encoding
이미지를 잘게 나눈다. 그 나눈 조각을 픽셀이라고 한다.
까만게 더 많으면 1, 하얀게 더 많으면 0이라고 하자



격자를 더 늘리면 정확해진다. 그러나 그만큼 픽셀의 수가 많아진다.
표현력과 효율성의 중간에서 합의를 봐야한다.

명암을 어떻게 배정?
8bit로 명암을 표현할 수 있기도 하다.

white and black : 6*6*1bit
8bit gray scale : 6*6*8bits

사진은 픽셀로
기기는 PPI(1인치 당 픽셀수)라고 표현한다.
기기가 구리면 좋은 사진 파일을 제대로 표현할 수 없다.