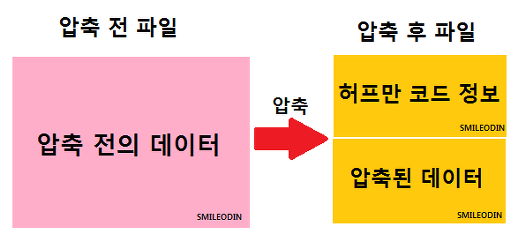
과제 9 - 허프만 코딩 조사

201411802 응용통계학과 전민규

-허프만 코딩에서는 압축 대상이 되는 파일마다 최적의 부호 체계를 구축하고, 이것을 기준으로 삼아 압축을

한다.

-파일마다 특정 부호(허프만 부호)에 다른 데이터를 할당함. 그 덕에 허프만 코딩으로 압축을 할 시, 압축파일에는 부호의 정보와 압축된 데이터가 함께 저장된다



**1. 허프만 트리 만들기**

-기호의 빈도는 길이가 짧은 접두어 코드를 빈도가 높은 기호에 부여하기 위해 사용

-빈도가 높은 기호에 작은 접두어 코드를 부여하면 그만큼 저장 공간을 아낄 수 있음

예 ) 어떤 문자열 중 ‘a’기호 20개와 ‘b’기호 5개로 구성된 것을 허프만 코딩을 해보면

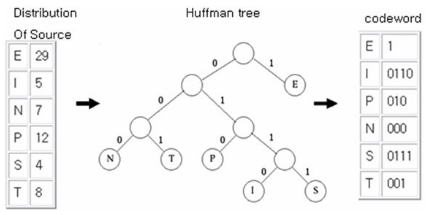
-‘a’코드에 100을, ‘b’에 11을 부여한다면, 변환된 데이터의 크기는 3(100의 비트 수)\*20 + 2(11의 비트 수)\*5=70비트

->원본 문자열이 200비트였으니 원본 크기 대비 35%압축

-반대로 ‘a’코드에 11을, ‘b’에 100을 부여한다면, 변환된 데이터의 크기는 2(11의 비트 수)\*20 + 3(100의 비트 수)\*5 = 55비트

-이진트리는 접두어 코드를 표현하는데 사용

-트리의 노드에서 왼쪽 자식 노드는 0, 오른쪽 자식 노드는 1을 나타냄



-위 트리에서 모든 기호는 leaf node에만 기록되어 있는데, 루트 노드에서부터 leaf node까지 이르는 경로가 기호의 접두어 코드

-루트 노드에서 leaf node에 이르는 경로가 길어질수록 접두어 코드는 길어짐

-빈도가 높은 기호일수록 경로를 짧게, 빈도가 낮은 기호일수록 경로를 길게 가져가야 압축률이 높아짐

-탐욕 알고리즘의 3단계를 적용해서 위 조건을 만족

1)해 선택

-우선 해를 선택하는데에 있어서, 기준은 현 시점에서 빈도가 가장 작은 노드 두개 I(5), S(4)

-이 두 노드 위에 부모 노드를 새로 만들어 이 두 노드를 왼쪽과 오른쪽에 연결

-이 때 부모노드의 빈도는 자식 노드들의 빈도의 합이 됨, 기호는 따로 갖지 않음

-따라서, I와 S의 부모 노드의 빈도 수는 9가 됨. 마지막으로 새로 만든 부모 노드를 노드 집합에 추가

2)실행 가능성 검사

-기호를 가진 노드는 leaf node여야 하는데, 아직까지는 이 규칙을 위반하지 않음. 따라서 실행 가능성 통과

3)해 검사

-허프만 트리를 완성하지 못했으므로 해 검사는 통과하지 못하고 다시 해 선택 단계로 돌아감

-나머지 해들 중 I와 S의 부모 노드 빈도 수와 같은 해가 없지만 빈도가 가장 작은 노드를 선택기준으로 진행하는 방식으로 트리를 구성하다보면 위의 트리 그림과 같게됨.(트리 형태는 유지)

Cf)참고

1. 고정 길이 코드

-모든 코드의 길이가 똑같은 값을 갖는 코드 체계 (예, ASCII)

-장점 : 다루기 쉽다 (ASCII로 문자열을 표현하고 싶으면 8bit 길이의 데이터를 연속으로 이어 붙이면 되고 반대로 문자열의 각 요소를 알고 싶으면 8bit단위로 끊어 읽으면 됨)

-인코더와 디코더 작업과정에서 일정한 기준을 적용하여 쉽게 코드 체계를 분석. 연산의 편의를 위함

-단점 : 저장 공간의 문제

2. 접두어 코드

-VLC(Variable Length Code, 가변길이 코드)의 한 종류.

-코드 집합의 어느 코드도 다른 코드의 접두어가 되지 않는 코드를 말함.(=접두어 속성)

예) {“0”,”1”,”01”,”010”}은 접두어 코드가 아님. “0”이 “01”과 “010”의 접두어가 될 수 있기 떄문

반면 {“00”,”010”,”100”,”101”}은 어느 코드도 다른 코드의 접두어가 되지 않기 때문에 접두어 코드라고 할 수 있음

a=00, b=010, c=100, d=101에서 ‘abcd’는 00010100101로 표현하며 ASCII로는 8bit가 4개 있으므로 32bit가 필요하지만 접두어 코드로 표현하면 11it만이 필요. 허프만 코딩은 이런 아이디어에 기반한 알고리즘

**2. 데이터 압축하기(Encode)**

-압축은 문자열의 각 요소를 차례대로 읽으면서 허프만 트리가 나타내는 해당 문자의 접두어 코드로 변환

-접두어 코드를 알아내기 위해 매번 허프만 트리를 순회할 수 밖에 없다

-이 문제에 대한 해결책으로 보통 접두어 코드 테이블을 만들어 놓고(배열로) 별도의 심볼을 이용하는 방식 검토

**3. 압축 해제하기(Decode)**

1) 우선 압축을 위해 만들었던 허프만 트리와 압축 해제된 데이터를 담을 버퍼를 준비. 여기서는 허프만 트리의 루트 노드로부터 시작해서 잎 노드까지 순회

2) 압축 데이터에 읽지 않은 부분이 남아 있으면 데이터를 한 비트 읽음

2) 읽어낸 비트가 0이면 현재 노드의 왼쪽 자식 노드, 1이면 오른쪽 자식 노드로 이동

-현재 노드가 잎 노드이면 저장되어 기호를 버퍼에 추가하고 다시 루트 노드로 이동

-위 3단계를 요약하면 비트를 읽을 때마다 잎 노드를 만나기 전까지 허프만 트리의 왼쪽/오른쪽으로 노드를 순회하게 됨

**\*개념**

-자주 쓰이는 문자에 작은 bit를 비교적 덜 쓰이는 문자에 큰 bit를 할당해서 문자열을 전체적으로 압축하는 개념

Ex) AABBAC에서 A : 3번, B : 2번, C : 1번 출현

-가장 많이 사용하는 A에는 0을, 그 다음 B에는 01, C에는 11을 부여하여 A(0)A(0)B(01)B(01)A(0)C(11)로 만듦

=>000101011

-영문자 하나는 1byte이기 떄문에 6byte가 사용됨(6\*8=48bit)

-그러나 압축된 허프만 코드는 9bit가 된다 => 9/48=0.18로 20%가량 압축되었다

**\*작동 방법**

CDDCACBCBCCCBBCDA라는 문자열(총 17byte)이 있다 하자.

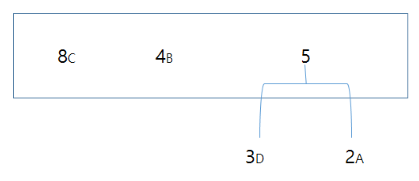
1) 빈도수 검사

- 2A, 4B, 8C, 3D

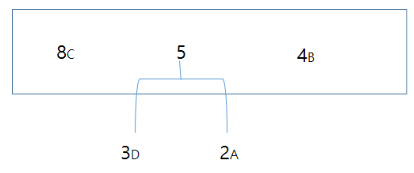
2) 빈도수대로 정렬

- 8C, 4B, 3D, 2A

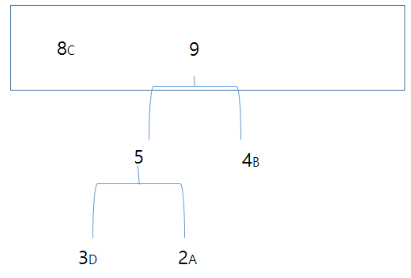
3) 가장 두 그룹을 묶어서 하나의 그룹으로 만듦



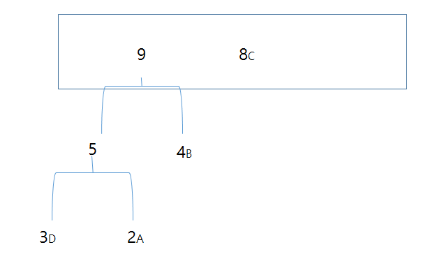
4) 빈도수 대로 정렬



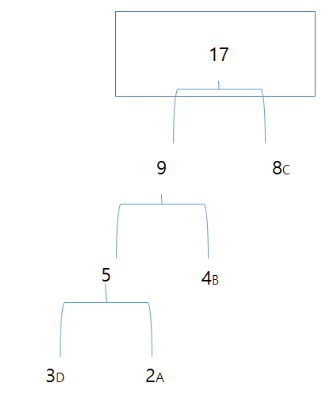
5) 가장 낮은 두 그룹을 묶어서 하나의 그룹으로 만듦



6) 빈도수 대로 정렬



7)가장 낮은 두 그룹을 묶어서 하나의 그룹으로 만듦

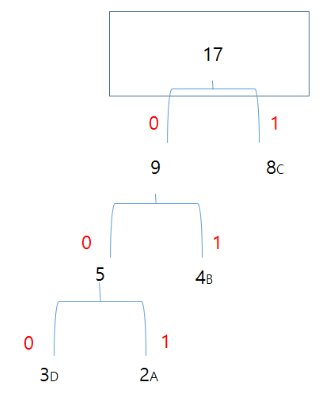


8) 트리 완성(위와 같이 하나의 루트노드가 남을때까지!)

**\*허프만 코드 작성**

- 가장 윗 트리에서 빈도수가 큰 트리에 0을, 작은 트리에 1을 부여하여 아래 그림처럼 만든다

- 최상위 트리 노드를 기준으로 비트를 참조한다



-부모노드에서 출발하여 문자열에 다다를 때 까지 비트를 더하여 참조한다

D=>000, A=>001, B=>01, C=>1

**\*디코드 하는 방법**

-호프만 트리를 타고 내려가면서 문자를 만나면 치환함(복호화 과정)

예) 1000000100110110111101011000001에서

-최상위 노드를 기준으로 비트의 수를 따라가며 알파벳을 만날 때 까지 따라간다

-1C/000(D)/000(D)/…. => CDDCACBCBCCCBBCDA가 됨