허프만 복호화를 위한 균형이진 검색 트리

준회원 김 혜 란*, 정 여 진*, 정회원 임 창 훈, 임 혜 숙*

A Balanced Binary Search Tree for Huffman Decoding

Hyeran Kim*, Yeojin Jung* *Associate Member*
Changhun Yim, Hyesook Lim* *Regular Members*

요 약

허프만 코드는 영상이나 비디오 전송뿐만 아니라 여러 분야에서 광범위하게 사용되고 있는 데이터 압축 알고리즘으로서, 실시간 데이터의 양이 증가함에 따라 효율적인 디코딩 알고리즘에 관한 많은 연구가 진행되고 있다. 본 논문에서는 호프만 디코딩을 위해 균형 트리를 형성하여 효율적인 이진 검색을 수행하는 구조를 제안하고 타 구조와의 성능을 비교하였다. 제안하는 구조는 길이가 다른 코드워드 간의 크기 비교를 가능하게 하는 정의를 사용하여 비어있는 내부 노드를 포함하지 않는 완전 균형 트리를 구성하므로, 디코딩 테이블을 위해 필요로 하는 메모리의 크기에 있어 매우 우수한 구조이다. 실제 영상 데이터를 사용하여 실험한 결과, 256개의 심볼 set에 대해 제안하는 구조는 매우 적은 수의 테이블 엔트리를 요구하며, 디코딩 성능은 최소 1번, 최대 5번, 평균 2.41번의 메모리 접근을 소요함을 보았다.

Key Words : Huffman decoding, binary search, balanced tree, variable length coding, entropy coding

ABSTRACT

Huffman codes are widely used for image and video data transmission. As the increase of real-time data, a lot of studies on effective decoding algorithms and architectures have been done. In this paper, we proposed a balanced binary search tree for Huffman decoding and compared the performance of the proposed architecture with that of previous works. Based on definitions of the comparison of codewords with different lengths, the proposed architecture constructs a balanced binary tree which does not include empty internal nodes, and hence it is very efficient in the memory requirement. Performance evaluation results using actual image data show that the proposed architecture requires small number of table entries, and the decoding time is 1, 5, and 2.41 memory accesses in minimum, maximum, and average, respectively.

Ⅰ. 서 론

허프만 코드[1]는 1952년에 발견된 이래로 널리 쓰이는 variable length code로서 구현이 간단하면서 압축효율이 높은 장점으로 인해 다양한 데이터의 압축에 이용되고 있으며 이미지나 비디오 압축 표준인 JPEG, MPEG-1, MPEG-2, MPEG-4에도 포함되어 있다.

어떤 심볼 set에 대하여 출현 빈도에 관한 확률 분포가 주어졌음을 가정할 때, 허프만 코드는 출현 빈도가 높은 심볼은 짧은 길이의 코드워드로 표현하고, 출현 빈도가 낮은 심볼은 긴 길이의 코드워드로 표현하여 평균 전송 데이터의 양을 줄이는 것을 목적으로 한다. 심볼을 같은 길이의 코드워드로 표

* 이화여자대학교정보통신학과 SoC Design 연구실(hlim@ewha.ac.kr)
논문번호 : KICS2005-02-058, 접수일자 : 2005년 2월 2일

현한 경우에는 연속적으로 들어오는 입력 비트스트림에 대하여, 일정한 길이로 잘라서 디코딩 하면 되지만 길이가 다른 경우에는 guard 비트가 없으므로 하나의 코드워드는 하나의 심볼에만 mapping 될 수 있도록 하여야 한다. 이는 임의의 코드워드가 다른 코드워드의 프리픽스가 되지 않도록 함으로서 가능한데, 허프만 코드는 이러한 원리를 사용하여 만들어진 코드이다.

이러한 코드워드의 특징 때문에 허프만 인코딩을 위해서 이진 트리 형태를 주로 이용한다. 이진 트리를 만드는 방법은 먼저 출현 빈도가 가장 낮은 두 개의 심볼을 선택하여 노드에 mapping 한후 각 노드에 0과 1을 할당하고, 이 둘을 묶어 parent 노드를 만든다. 두 노드의 출현빈도의 합이 parent 노드의 출현 빈도가 되는데, 이러한 과정을 반복적으로 진행하면, 그 결과 이진 트리가 형성된다. 각 심볼은 트리의 leaf 노드에만 mapping 되고, 트리의 루트로부터 leaf에 이르는 path에 따라 그 심볼의 코드워드가 정해진다.

이미지나 영상 데이터의 크기가 커지고 인터넷 어플리케이션들이 발달함에 따라 실시간 데이터의 양이 늘어나면서 메모리를 적게 사용하면서도 빠르게 디코딩을 수행할 수 있는 구조가 요구되고 있다. 이진 트리 구조는 최대 코드워드 길이가 늘어날수록 트리의 깊이가 깊어짐으로, 비어있는 내부 노드의 수가 증가하게 되는데, 비어 있는 내부 노드에도 하위 노드로 가는 포인터를 저장해야 하므로 메모리 사용에 있어 매우 비효율적이다. 또한 디코딩에 걸리는 시간은 메모리 접근 횟수와 매우 밀접한 관계를 갖는데, 이진 트리 구조는 코드워드의 길이에 따라 거쳐 가야 하는 노드의 수가 정해지기 때문에 최대 디코딩 시간이 가장 긴 코드워드의 길이에 비례하게 된다. 이에 따라 좀 더 효율적으로 메모리를 사용하고, 디코딩 시간을 줄이기 위한 많은 연구가 진행되고 있다.

본 논문에서는 허프만 디코딩을 위해 비어 있는 노드를 포함하지 않는 균형 트리 구조를 제안한다. 본 논문에서 제안하는 구조는 비어있는 내부 노드를 포함하지 않으므로, 메모리 사용량에 있어 매우 우수한 구조일 뿐만 아니라, 트리의 형태가 완전 균형구조를 갖기 때문에, 디코딩 시간에 있어서도 매우 효율적인 구조이다. 본 논문의 구성은 다음과 같다. II장에서 이전의 허프만 디코딩 알고리즘을 살펴보고 III장에서 제안하는 구조에 대해 설명한다. IV장에서는 제안하는 구조와 타 구조의 성능을 비교하고, V장에서 결론을 맺는다.

II. 기존의 구조

엔트로피 코딩은 확률 분포에 따라 코드 길이를 다르게 정하므로써 데이터를 압축하는 방법으로 어떤 심볼 집합 S에 대한 엔트로피는 다음과 같은 식으로 표현된다.

$$H(S) = \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i} \quad (1)$$

여기서 p_i는 집합 S에 속하는 심볼 S_i가 일어날 확률이 되고 N은 전체 심볼의 개수를 나타낸다. 식에서 $\log_2(1/p_i)$ 는 심볼 S_i 에 포함된 정보의 양으로 볼 수 있는데, 다시 말해 심볼 S_i 를 인코딩하는데 필요한 비트수라 할 수 있다. 엔트로피 H(S)는 가중치를 갖는 $\log_2(1/p_i)$ 의 합이므로, 심볼 집합 S에 포함되어 있는 각 심볼들을 표현하는데 필요한 평균 비트수의 최소값으로 볼 수 있다. 이 식을 통해서 보면 엔트로피 코딩은 확률 분포가 고르게 퍼져있을 때 보다 한쪽으로 몰려 있을 때 더 낮은 엔트로피를 가지게 되며 이 특성을 이용하여 압축 효율을높인 것이 바로 엔트로피 코딩이다. 허프만 코딩은 엔트로피 코딩 중 가장 널리 알려진 것으로서, 허프만 코드의 효율적인 디코딩을 위한 연구가 활발하게 진행되고 있다.

허프만 디코딩을 위한 가장 단순한 구조는 가장 긴 코드워드의 길이를 h라고 했을 때 2^h 크기의 lookup 테이블을 만들어 입력 값을 메모리 인덱스로 이용하여 검색을 수행하는 구조이다. 이 구조는 한번의 메모리 검색으로 디코딩이 가능하나, 심볼 수에 비해 너무 큰 메모리를 필요로 하기 때문에 실용적이지 않다.

앞서 간단히 언급된 바와 같이 허프만 디코딩을 위해 가장 널리 쓰이는 것으로 이진 트리를 이용한 구조를 들 수 있다. 이진 트리는 루트 노드에서 출발하여 입력으로 들어온 비트가 0이면 왼쪽, 1이면 오른쪽으로 한 비트, 한 비트씩 검색을 수행하는 방식으로, leaf 노드를 만날 때까지 진행하여 디코딩하는 구조이다. 그러나 이 구조는 트리를 구성함에 있어 내부에 빈 노드가 많게 되어 메모리 효율이 떨어질 뿐 아니라, 메모리 검색 횟수가 트리의 최대 높이인 가장 긴 코드워드의 길이가 되는 단점이 있다. [2]는 이진 트리의 이러한 단점을 보완하기 위해 multibit 트리를 이용한 알고리즘을 제안하여 검

색 횟수와 메모리 사용을 줄였다.

Level-compressed-tree (LC-tree) [3, 4]에서는 이진 트리를 breadth-first search (BFS)방식으로 저장하여 효율적으로 메모리를 사용하는 알고리즘을 제안하였다. 이진 트리의 루트에서 가장 짧은 코드의 길이를 d 라고 할 때, d-1의 높이만큼 트리의 윗부분을 자르고 남은 트리를 앞의 노드부터 순차적으로 저장하는 코드워드 어레이를 만든다. 레벨 d부터 심볼이 저장되어 있으므로 레벨 d부터 오른쪽으로 지나가면서 코드워드 어레이를 만든다. 중간에 비어있는 내부 노드에는 jump 값을 미리 계산하여 저장하는데, jump 값은 자신의 첫 번째 child로 가기 위한 포인터가 된다. 내부 노드는 valid 비트를 0을 갖고, 심볼이 저장된 leaf 노드는 valid 비트를 1을 갖는다. 코드워드 어레이를 따라가며 디코딩을 하게 되는데 먼저 코드워드의 가장 짧은 길이까지를 인덱스로 사용해 코드워드 어레이에 접근하고 저장된 값이 jump 값이라면 입력된 비트스트림의 다음 비트에 jump 값을 더한 값을 인덱스로 사용하여 다시 코드워드 어레이에 접근하고 심볼을 만날 때까지 이 과정을 반복한다. 이진 트리 구조와 비교할 때, 이 알고리즘은 길이 d 이전 트리의 빈 노드만을 제거한 구조로서, d 레벨 이하에 있는 빈 노드들은 jump 값을 갖는 노드로서 저장 되어야하는 단점을 지닌다.

[5]의 first-bit-change 구조에서는, 처음 비트가 바뀌는 위치에 따라 전체 코드워드 트리를 여러 개의 트리로 분류한 후, 분류된 각 트리에 속한 코드워드들을 같은 길이를 갖는 코드워드로 확장하여 디코딩 하는 알고리즘을 제안하였다. 이렇게 하면 확장된 트리에서의 코드워드 들의 numerical 값은 1씩 차이가 나므로 이 특성을 이용하여 빠르게 디코딩을 할 수 있다. 비트스트림이 들어오면 한 비트씩 보다가 0에서 1 또는 1에서 0으로 바뀌는 위치에 따라 그 비트스트림이 속한 트리가 정해진다. 해당된 트리에서 정한 길이까지의 입력을 저장된 코드워드와 비교하고 그 차이를 인덱스로 하여 심볼이 저장된 테이블에 접근하여 디코딩을 수행한다. 이 알고리즘은 항상 두 번의 메모리 접근으로 디코딩을 수행할 수 있는 특성을 갖지만, 4장의 실험 결과에서 보이려는 바와 같이, 코드워드들을 같은 길이로 확장해야 함에 따라 매우 큰 메모리를 필요로 한다.

Canonical 허프만 트리 혹은 single-side-growing-huffman (SGH) 트리는 이진 트리의 모양이 한 쪽으로 쏠리도록 코드워드를 할당한 트리이다. 즉 왼쪽으로 쏠리는 SGH 트리에서의 긴 길이의 코드워드는 항상 짧은 길이의 코드워드 보다 왼쪽 노드에 위치하게 된다. 이러한 형태로 인코딩을 할 경우 매우 효율적인 디코딩이 가능하다. [7]에서 제안한 방법은 canonical 허프만 트리에만 적용될 수 있는 방법으로, 두 단계로 이루어지는데, 입력된 비트스트림에 해당되는 코드워드의 길이를 먼저 찾아낸 후, 해당하는 코드워드 값을 찾아내는 구조이다. Condensed-huffman-table (CHT)에는 길이 정보와 각 길이에 해당하는 코드워드들 중 제일 작은 코드워드, 그리고 심볼이 저장된 또 다른 테이블의 인덱스가 저장되어 있다. 입력 비트 스트림에 대해 CHT 테이블을 순차적으로 검색하여, 값을 비교함으로써, 길이 정보를 찾고, 일치된 길이에 저장된 코드워드 값과 입력과의 차를 메모리 포인터로 사용하여, 디코딩을 수행한다. 매우 효율적인 구조이나, canonical 트리의 형태로 인코딩된 경우에만 적용할 수 있다는 제약점을 갖으며, 짧은 길이의 코드가 확률적으로 더 많이 나온 다는 것을 이용하여 코드 길이별로 순차적으로 검색하므로 최악의 경우 검색 횟수는 가장 긴 코드 길이인 h가 된다는 단점을 지닌다.

[8]은 first-bit-change 구조가 SGH 형태의 트리에서는 하나의 트리에서 확장되어야 하는 노드의 수가 급격히 감소하게 되는 원리를 적용한 알고리즘이다. 즉 일반적인 이진 트리를 노드스와핑을 수행하여 SGH 형태의 트리로 변환하고, first-bit-change 알고리즘을 적용한 구조이다. 그러나 노드스와핑을 위한 매우 복잡한 pre-processing이 추가로 요구되는 단점을 지닌다.

Ⅲ. 제안하는 구조

본 논문에서는 비어있는 노드를 포함하지 않는 새로운 방식의 균형이진 트리를 구성하는 알고리듬을 제안한다. 이에 더하여 발생 빈도가 높은 짧은 길이의 코드워드를 효율적으로 디코딩하고, 트리를 여러 개로 나누어 보다 짧은 트리들로 만들기 위하여 range 테이블을 이용한 새로운 디코딩 구조를 제안한다.

3.1 이진 균형 트리

그림 1은 표 1의 코드워드를 이진 트리로 구성한 것이다. 검정색 노드가 심볼이 저장된 노드이고 흰 노드는 비어있는 노드를 나타낸다. 그림 1에서 보듯

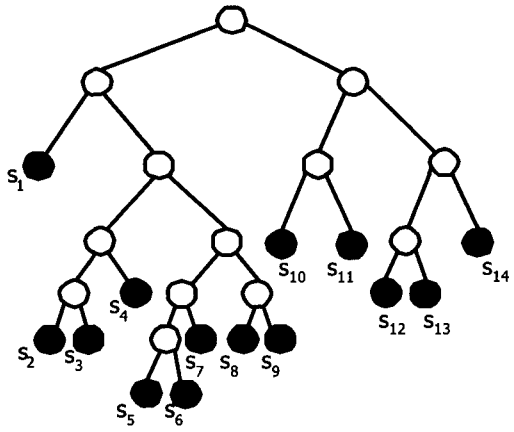


그림 1. 이진 트리

표 1. 허프만 코드워드의 예

심볼 S_i	코드워드	심볼 S_i	코드워드
S1	00	S8	01110
S2	01000	S9	01111
S3	01001	S10	100
S4	0101	S11	101
S5	011000	S12	1100
S6	011001	S13	1101
S7	01101	S14	111

이 이진 트리는 트리의 leaf에 있는 심볼까지 가기 위해 빈 노드를 저장해야 하며 디코딩 할 때도 빈 노드를 거쳐 가야하므로 효율성이 떨어진다. 본 논문에서 제안하는 이진 균형 트리는 길이가 다른 코드워드간의 크기 비교를 가능하게 하는 새로운 정의를 사용하여 가능하게 된다. 이진 균형 트리의 구성은 코드워드 들을 크기 별로 정렬하는 것으로부터 출발하는데, 코드워드들의 정렬을 위하여 그림 2의 정의를 사용하였다. 그림 2의 정의를 살펴보면, 길이가 같은 코드워드들의 크기 비교는 코드워드의 numerical 값이 비교되고, 길이가 다른 코드워드들은 짧은 길이까지만 비교하여, 짧은 길이까지의 numerical 값이 큰(작은) 쪽이 큰(작은) 코드워드가 된다.

균형 트리를 만드는 방법은 그림 3의 알고리즘과 같다. 그림 3의 알고리듬은 정렬과정 후에 수행되는데, 그림 4의 코드워드 리스트는 이진 균형 트리를 만들기 위해 표 1의 코드워드들을 크기 순으로 정렬한 것이다. 이렇게 정렬된 리스트의 가장 중간에 위치한 코드워드가 루트로 선택되고, 루트의 왼쪽

정의 Compare
a. 두 codeword의 길이가 같으면 numerical 값이 비교된다.
Ex) A=1001 B=1100인 경우 B>A
b. 두 codeword의 길이가 다른 경우 짧은 길이까지만 비교된다.
Ex) A=1001, B=110000인 경우 B>A

그림 2. 길이가 다른 코드워드들 간의 크기 비교에 관한 정의[9]

```
BuildTree(List)
        if List is empty, return.;
        let m be the median of List
        root<-m;
        let leftList and rightList contain all elements in the left and right of m
        leftChild(root)<-BuildTree(leftList);
        rightChild(root)<-BuildTree(RightList);
        return address of root
end BuildTree;
```

그림 3. 트리 구성 알고리즘

Codeword list
(00, 01000, 01001, 0101, 011000, 011001, 01101, 01110, 01111, 100, 101, 1100, 1101, 111)

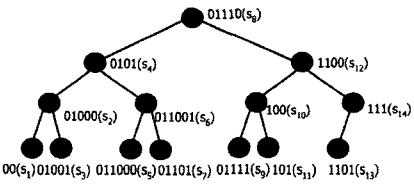


그림 4. 제안하는 이진 균형 트리

리스트의 중간 코드워드가 루트의 왼편 child 로, 루트의 오른쪽 리스트의 중간 코드워드가 루트의 오른편 child로 선택된다. 이들 children에 대해서도 같은 방식으로 이들의 children을 선택하여 리스트에 있는 모든 코드워드가 선택 되어질 때까지 반복하는 방식이다. 그림 4에 주어진 코드워드 리스트를 예로 설명하면, 코드워드 리스트 중에서 가운데 있는 01110이 루트 노드로 선택되고 루트의 왼쪽과 오른쪽 리스트의 가운데에 있는 0101과 1100을 다음 노드로 선택되는 것을 recursive 하게 반복하여 트리를 만들어 나간다.

그림 4의 트리는 표 1의 코드워드들을 가지고 이진균형 트리를 구성한 예이다. 그림 4와 그림 1을 비 교해 보면 그림 4에서는 흰색으로 표시되는 비어있는 내부 노드가 없고 트리의 코드워드가 균형있게 퍼져 있는 것을 볼 수 있다. 내부 노드가 없고 트리가 한 쪽으로 치우치지 않았기 때문에 전체 트리의 높이가 6에서 3으로 줄었는데, 이는 단순한 이진 트리 대신 이러한 균형 트리를 이용하면 검색 횟수가 크게 줄어듦을 보여주는 것이다. N개의 심볼을 갖는 set 에 대하여, 이진 트리를 구성하기 위해 저장되어야 하는 최대 노드 수가 $2N$-1인 반면에, 이진 균형 트리의 경우는 빈 노드를 저장하지 않으

므로 써 저장되어야 하는 노드의 수를 N으로 줄일 수 있게 된다. 또한 이러한 이진 균형 트리는 완전 균형 트리로서 메모리 인덱스를 사용하여 검색이 수행되므로, 다음 노드로 가는 포인터를 저장할 필요가 없다.

이진 균형 트리에서의 코드워드를 검색하는 과정은 그림 5의 알고리즘과 같다. 루트 노드로부터 출발하여 입력된 비트스트림을 최대 코드워드 길이로 잘라, 이 값과 현재의 노드에 저장된 코드워드 값을 비교한다. 입력된 값과 노드에 저장된 코드워드를 비교하는데 있어, 그림 2에 보여준 정의가 사용된다. 입력된 값이 현재의 노드에 저장된 값보다 작으면 현재의 노드의 왼쪽에 있는 child로, 크면 오른쪽에 있는 노드로 검색을 진행하여 비교 량을 지수적으로 줄이는 검색 방법이다. 입력된 값과 현재의 노드에 저장된 코드워드의 값이 일치한다면, 입력된 값에 해당하는 코드워드를 찾은 것으로, 입력 값 중 일치된 코드워드의 길이보다 긴 부분은 다시 입력 비트스트림으로 되돌린 후 검색을 종료한다.

```
Search(tree, extracted_bits)
        if tree=NIL, return NULL;
        if(extracted_bits<tree(root)),
                symbol<-Search(tree(leftChild), extracted_bits);
        else
                symbol<-Search(tree(rightChild), extracted_bits).
        if extracted_bits matches tree(root),
                symbol<-tree(root);
                return the remaining part which is not included in symbol
                to the bit stream.
        Return symbol;
end Search;
```

그림 5. 트리 디코딩 알고리즘

3.2 Range 테이블

이진 균형 트리를 이용하면 $\log_2 N$ 만큼의 최대 검색 횟수를 갖게 된다. 그러나 입력된 값의 검색 범위를 보다 빨리 좁힐 수 있다면, 검색 성능은 더욱 향상 된다. 이러한 목적을 위하여 range 테이블을 제안한다. Range 테이블은 코드워드 중 일부 길이를 먼저 검색 하여 다음 디코딩 테이블로 연결하는 구조로서 range 테이블에서 비교하는 코드 길이를 r이라고 했을 때 range 테이블은 2^r 의 크기를 갖는다. 허프만 디코딩은 짧은 길이의 코드워드가 빈도수가 더 높고 그것을 빨리 찾는 것이 더욱 중요하기 때문에 이러한 range 테이블을 이용한다면, 길이가 r보다 작거나 같은 입력이 왔을 때에는 한 번에 디코딩을 가능하게 하여, 전체 검색 성능을 더욱 향상시킨다. 길이가 r보다 긴 코드워드들은 길이가 r인 프리픽스에 따라 깊이가 작은 이진 균형 트리를 구성하게 된다.

3.3 테이블 구조

본 논문에서 제안하는 구조는 2개의 테이블을 가진다. 표 1의 예를 들어 테이블을 구성하면 먼저 range 테이블의 크기를 2^3이라고 했을 때 그림 6의 (a)와 같이 구성 된다. range 테이블의 길이 3 보다 긴 코드워드들을 가리키는 엔트리에는, 다음 테이블로 연결되는 포인터와 같은 범위의 프리픽스를 갖는 엔트리가 몇 개인지가 저장되고, 레인지 테이블의 길이가 3보다 작은 코드워드가 가리키는 엔트리에는 심볼 자체가 저장된다. 표 1의 예를 들면, range 테이블은 들어온 비트스트림의 처음 3 비트를 사용하여 그것을 인덱스로 하여 접근한다. 디코딩 테이블은 그림 6의 (b)와 같이 구성된다. 이 테이블에는 range 테이블에 저장된 코드워드를 제외한 코드워드와 그 길이, 그리고 심볼이 저장 된다. 그림에서 보면, 코드워드의 프리픽스가 010인 코드워드는 S_2부터 S_4까지 3개 이므로 range 테이블의 세 번째 엔트리에는 포인터 0과 엔트리 수 3이 저장되고 디코딩 테이블에는 S_2부터 S_4이 차례로 저장된다. 코드워드 길이가 r보다 작거나 같은 심볼은 range 테이블에만 저장되는데, 포인터를 저장하는 자리에 심볼을 저장하고 심볼인지 포인터 인지는 'Number of entries' 필드가 0인지 아닌지로 구분된다. 코드워드의 길이가 r보다 작은 S_1의 경우에는 길이를 r로 확장하여 두개의 엔트리에 저장되고 코드워드의 길이가 r보다 작기 때문에 일부의 길이는 다음 코드워드가 되므로 'Length' 필드에 코드워드의 길이를 저장하여 다시 입력으로 되돌려준다.

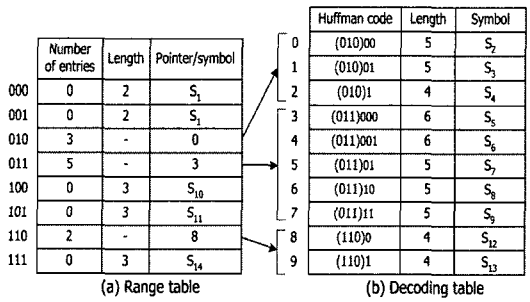


그림 6. 테이블 구조

3.4 디코딩 과정

디코딩을 위해서는 먼저 비트스트림 중 길이 r까지를 range 테이블의 주소로 이용해 range 테이블에 접근하게 된다. Range 테이블에 접근하여 'Number of entries' 필드가 0 이면 'Pointer' 필드에 저장된 것이 심볼이라는 의미이므로 한 번의 메모리 접근

으로 입력이 디코딩 된다. 그렇지 않고 엔트리 수가 하나 이상 저장 되어 있다면 다음 디코딩 테이블 검색이 필요하게 되는데, range 테이블에서 시작 인덱스와 엔트리 수를 주기 때문에 디코딩 테이블에서는 주어진 범위 내에서의 이진 검색이 수행되고, 일치하는 코드워드를 만나면 그 엔트리에 저장된 심볼로 디코딩 된다.

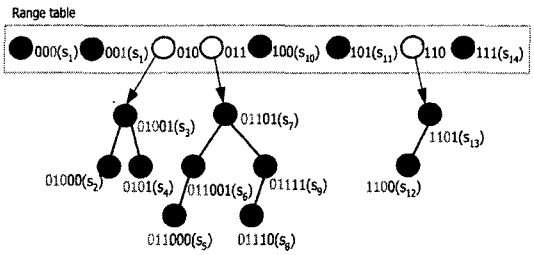


그림 7. Range 테이블을 사용한 제안하는 구조

그림 7은 range 테이블을 사용하였을 때 구성되는 여러 개의 균형이진 트리를 보여주고 있다. 예를 들어 011111000..의 비트스트림이 들어온다면 먼저 range 테이블에 접근하는 코드워드 길이인 011, 즉 십진수 3을 인덱스로 하여 range 테이블에 접근한다. 이 엔트리에는 포인터 3과 'Number of entries' 5가 저장되어 있으므로, 디코딩 테이블의 인덱스 3부터 7사이에서 이진 검색이 진행되어야 함을 알 수 있고, 입력 011111이 메모리 인덱스 5에 저장되어있는 01101과 비교된다. 입력 값이 크므로 다음은 메모리 인덱스 7에 저장되어 있는 01111과 비교되어 일치하므로 S_9가 출력되고 검색이 종료된다. 이미 디코딩 된 5비트를 제외한 다음 비트스트림을 디코딩 하기 위해 다시 3비트인 100, 즉 십진수 4를 인덱스로 하여 range 테이블에 접근하면, 이 경우 range 테이블의 'Number of entries' 필드가 0이므로, 포인터 필드에는 심볼 자체가 저장되어 있음을 알 수 있고, 심볼 S_{10}이 출력된다.

Ⅳ. 실험 결과 및 비교

본 논문에서는 Peppers, Lena, Barbara, Zelda 4개의 gray-scale 이미지의 차이영상을 이용하여 실험하였다. 차이영상은 $I(x, y)$를 원영상이라 했을 때, 다음과 같은 식으로 표현되는데, 원영상보다 gray scale이 특정 값에 집중되므로, 적은 엔트로피를 갖고 따라서 JPEG 등에서 이용하는 허프만 코드와 더 가까운 결과를 얻을 수 있다.

$$d(x, y) = I(x, y) - I(x-1, y) \quad (2)$$

각 차이영상의 크기는 512 x 512 이고 각 픽셀은256 level의 gray-scale로 표현되는데, gray-scale의 출현빈도를 가지고 허프만 인코딩을 하면 표 2와 같은 특성을 갖는다.

표 2. 실험 영상

	Peppers	Lena	Barbara	Zelda
심볼의 수	256	256	256	246
Tree depth	14	15	15	18
Min. code length	4	4	4	4

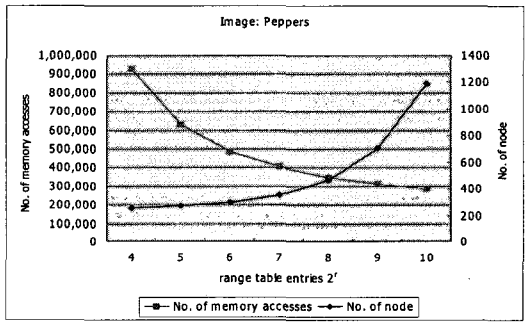


그림 8. range 테이블 크기에 따른 비교

그림 8은 표 3은 실험 영상 peppers에 대한 메모리 사용과 디코딩 시 메모리 접근 횟수를 비교한 그래프이다. 실험 영상의 코드워드는 가장 짧은 길이의 코드워드가 4이므로 range 테이블의 크기를 2^4부터 2^{10}까지 다르게 하여 실험하였다. 그래프를 보면 디코딩에 필요한 메모리 접근 횟수가 지수적으로 감소하며 메모리 사용량은 선형적으로 증가하는 것을 알 수 있다. Range table의 크기를 가장 짧은 코드 길이인 2^4에서 한 단계 증가시켜 2^5로 한다면 디코딩 시 필요한 메모리 접근 횟수는 크게 주는 것에 반해 필요한 메모리 크기는 약간 증가하게 된다. 이러한 실험 결과를 바탕으로 디코딩 시 필요한 메모리 횟수와 저장해야할 엔트리 수의 trade off 관계에서 적합한 지점을 택해 range 테이블의 크기를 정하면 된다. 본 논문에서는 r이 5인 경우의 실험 결과를 다른 알고리즘과 비교하였다.

표 4는 제안하는 구조와 앞서 설명한 5개의 다른 구조에 대하여 각 구조에서 필요로 하는 메모리 크기를 나타낸 것이다. 이진 트리는 내부 노드를 모두 저장해야 하므로 $2N$-1 만큼의 엔트리가 필요하다. LC-tree는 이진 트리에서 깊이가 d-1까지의 부분을

표 4. 메모리 크기 (2^5 range 테이블의 경우 전체 엔트리 수)

	Peppers	Lena	Barbara	Zelda
Proposed scheme	**274**	**275**	**276**	**363**
이진 트리	511	511	511	491
LC-tree	496	496	496	476
First-bit-change	145,145	354,096	428,667	5,139,333
*SGH-tree	6,319	5,744	4,474	3,658
*CHT	267	267	267	260

*canonical tree only

표 5. 전체 메모리 접근 횟수(2^5 range 테이블의 경우)

	Peppers	Lena	Barbara	Zelda
Proposed scheme	632,323	702,100	732,563	609,088
이진 트리	1,494,024	1,526,741	1,630,367	1,449,043
LC-tree	707,592	740,309	843,934	662,611
First-bit-change	524,288	524,288	524,288	524,288
*SGH-tree	524,288	524,288	524,288	524,288
*CHT	969,710	1,002,379	1,106,038	924,743

*canonical tree only

없애 $(2N\text{-}1)\text{-}(2d\text{-}1)$ 의 엔트리를 요구하므로, 표 4에서 보인 값과 같은 결과를 보인다. 앞서 설명한 바와 같이 first-bit-change 구조는 실험 결과 확장되는 엔트리의 수가 매우 커져 실용적이지 못한 것을 볼 수 있다. SGH-tree 구조와 CHT 구조를 위해서는 주어진 영상을 canonical 형태로 다시 인코딩한 후 실험한 것으로서, SGH-tree 구조의 경우 first-bit-change 구조보다는 메모리 사용량이 줄었으나 여전히 큰 메모리를 요구함을 알 수 있다. CHT 구조는 메모리 사용에 있어 매우 우수한 성질을 갖으나, canonical 형태로 인코딩되어 있는 허프만 코드에만 적용될 수 있다는 단점을 갖는다. 본 논문에서 제안한 구조는 빈 노드를 저장하지 않으므로 메모리 사용량에 있어 canonical 제약점을 갖지 않는 구조 중에 가장 우수함을 알 수 있다.

표 5는 디코딩 시에 필요한 메모리 접근 횟수를 비교한 것으로, 그림 전체를 디코딩 할 때 필요한 메모리 접근 횟수를 나타내고 있다. LC-tree는 앞서 설명된 바와 같이 빈 노드에 jump 값을 미리 계산하여 저장해 두어야 한다는 단점을 갖는다. 이진 트리는 트리의 깊이가 4이 되는 지점에서 처음으로 leaf 노드가 나타남에도 심볼이 없는 내부 노드를 다 따라가야 하므로 전체 접근 횟수가 매우 크게 나타난다. First-bit-change 구조와 SGH-tree 구조는 심볼 당 항상 두 번의 메모리 접근이 필요하므로, 이미지 크기가 정해지면 필요로 하는 메모리 접근 횟수가 정하여져서 가장 좋은 디코딩 성능을 보이나 앞서 설명했듯이 이 구조는 메모리 크기가 타 구조에 비해 월등히 많이 필요하다. 본 논문에서 제안하는 구조는 영상에서의 gray-scale 출현 빈도에 따르는 호프만 코드의 효율성을 매우 잘 반영하고 있으며 우수한 성능을 보임을 알 수 있다.

표 6는 픽셀 하나를 디코딩 하는데 필요한 최소, 최대, 그리고 평균 메모리 접근 횟수를 나타내고 있다. 본 논문에서 제안하는 구조는 코드워드 길이가 d인 심볼은 한 번의 메모리 접근으로 디코딩 할 수 있고 그 이상의 길이를 가지는 코드워드는 빈 노드 없는 이진 균형 트리를 구성하게 되므로 최대 5~6번의 메모리 접근으로 디코딩이 가능하다. 이진 트리는 루트부터 차례대로 트리를 따라가는 구조 이므로 최소, 최대 횟수가 트리의 깊이와 비례하여 나타나게 되고, LC-tree는 이진 트리의 leaf노드가 나

표 6. Number of Memory Accesses (Minimum, Maximum, Average)

	Peppers			Lena			Barbara			Zelda		
	Min	Max	Avr	Min	Max	Avr	Min	Max	Avr	Min	Max	Avr
Proposed scheme	**1**	**5**	**2.41**	**1**	**6**	**2.68**	**1**	**5**	**2.79**	**1**	**6**	**2.32**
이진트리	4	14	7.56	4	15	7.47	4	15	7.31	4	18	7.31
LC-tree	1	10	2.70	1	11	2.82	1	11	3.22	1	15	2.53
First-bit-change	2	2	2	2	2	2	2	2	2	2	2	2
*SGH-tree	2	2	2	2	2	2	2	2	2	2	2	2
*CHT	2	12	3.67	2	12	3.82	2	12	4.22	2	17	3.53

*canonical tree only

타나기 전의 비어있는 내부 노드들을 없앤 것으로 이진 트리에 비해 d 만큼의 접근 횟수가 줄게 되지만, 그 아래 부분에 있는 빈 노드들을 따라가야 하므로 본 논문에서 제안하는 구조에 비해 최대 메모리 접근 횟수가 크게 나타났다. 앞서 설명한 바와 같이 first-bit-change 구조와 SGH-tree 구조의 메모리 접근 횟수는 항상 2이다. CHT 구조는 코드워드가 가지는 길이의 종류 (중간에 없는 길이가 있을 수 있으므로 트리의 깊이가 아님)에 비례 하여 순차적 검색 횟수가 증가하고, 심볼이 있는 메모리에 다시 접근해야 하므로 한 번의 추가적인 메모리 접근이 더 필요하게 된다.

V. 결 론

본 논문에서는 이진 검색을 이용한 효율적인 허프만 디코딩 구조를 제안하고 기존에 나와 있는 타 구조와의 성능을 비교하였다. 제안하는 구조는 길이가 다른 코드워드 간의 크기 비교를 가능하게 하는 정의를 사용하여 비어있는 노드를 저장할 필요가 없는 이진 균형 트리를 구성하여, 이진 검색을 수행하는 구조이다. 실험 결과 디코딩을 위한 메모리 사용량에 있어 가장 우수한 성능을 보였으며 검색 횟수에 있어서도 매우 우수한 성능을 보였다. 또한 심볼의 확률 분포에 따르는 허프만 코드의 효율성을 매우 잘 반영하는 것으로 평가될 수 있었다.

참 고 문 헌

[1] Z. Li and M. S. Drew , Fundamentals of Multimedia, Pearson Education, 2004

[2] Y.S. Lee, B.J. Shieh, C.Y. Lee "A Generalized Prediction Method for Modified Memory-Based High Throughput VLC Decoder Design," IEEE Transactioins on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 46, No. 6, 1999, pp. 42-754.

[3] K.L. Chung, "Efficient Huffman Decoding," Inf. Process. Lett, vol. 61, 1997, pp. 97-99.

[4] K.L. Chung, "Level-Compressed Huffman Decoding," IEEE Trans. communications, vol. 47, Oct. 1999, pp. 1455-1457.

[5] M. Aggarwal and A. Narayan, "Efficient Huffman Decoding," Proc. Internation Conference on Image Processing, 2000, pp. 936-939.

[6] E.S. Schwartz and B. Kallick, "Generating a canonical prefix encoding," commum. ACM, vol.7, 1964, pp. 166-169.

[7] Reza. Hashemian, "Condensed Table of Huffman Coding, a New Approach to Efficient Decoding," IEEE Transactions on Communications, Vol. 52, No. 1, Jan. 2004, pp.6-8.

[8] Y.J. Chuang, "An SGH-Tree Based Efficient Huffman Decoding," Proc. ICICS, Dec. 2003, pp. 1483-1487.

[9] N.Yazdani and P.S.Min, "Fast and Scalable Schemes for the IP Address Lookup Problem," Proc. IEEE HPSR2000, 2000, pp 83-92

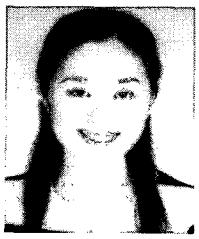
김 혜 란 (Hyeran Kim) 준회원

2004년 2월 이화여자대학교 정보통신학과 학사

2004년 3월~현재 이화여자대학교 정보통신학과 석사과정

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계, Huffman coding Architecture

정 여 진 (Yeojin Jung) 준회원



2003년 2월 이화여자대학교 정보통신학과 학사

2005년 2월 이화여자대학교 정보통신학과 석사

2005년 3월~현재 삼성전자 정보통신총괄

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

임 창 훈(Changhun Yim) 정회원

1986년 2월 서울대학교 제어계측공학과 학사
1988년 2월 한국과학기술원 전기 및 전자공학과 석사
1996년 12월 The University of Texas at Austin, Electrical and Computer Engineering 박사
1988년 3월~1991년 6월 한국방송공사 기술연구소 연구원
1996년 12월~1999년 3월 Sarnoff Corporation, 연구원
1999년 3월~2000년 7월 Lucent Technologies, Bell Labs, 연구원
2000년 8월~2002년 3월 KLA Tencor Corporation, Sr. Software Engineer
2002년 5월~2003년 8월 삼성전자 디지털미디어 연구소, 수석연구원
2003년 9월~현재 건국대학교 인터넷미디어공학부, 조교수
<관심분야> 멀티미디어 통신, 비디오 압축, 디지털 영상처리, 멀티미디어 네트워크

임 혜 숙(Hyesook Lim) 정회원

1986년 2월 서울대학교 제어계측공학과 학사
1986년 8월~1989년 2월 삼성 휴렛 팩커드 연구원
1991년 2월 서울대학교 제어계측공학과 석사
1996년 12월 The University of Texas at Austin, Electrical and Computer Engineering
1996년 11월~2000년 7월 Lucent Technologies, Bell Labs, Member of Technical Staff
2000년 7월~1002년 2월 Cisco Systems, Hardware Engineer
2002년 3월~현재 이화여자대학교 공과대학 정보통신학과 조교수
<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계, 이미지 코딩