論文
2-6-8

이진트리구조를 이용한 동적 재배치 알고리즘 설계 및 구현 (A Design and Implementation for Dynamic Relocate Algorithm Using the Binary Tree Structure)

최 강 희*
(Kang-Hee Choi)

요 약

데이터는 컴퓨터 시스템에서 파일구조로 나타난다. 그러나 파일의 크기는 매우 커지고, 그것을 제어하고 전송하기에 어려운 점이 있다. 그래서 최근에 데이터 압축에 대한 새로운 알고리즘이 개발되고 있다. 그래서 본 논문에서는 허프만 압축기법의 단점을 보완하여 새로운 동적 압축 기법을 제안하고자 한다.

허프만 압축 기법에는 두 가지 단점이 있다. 첫 번째로 처음 파일내의 문자의 빈도수를 구할 때와 실제로 압축하기 위해서 동작할 때, 파일을 두 번 읽어들인다는 것과, 두 번째로, 트리에 대한 정보를 같이 저장해야 되기 때문에, 그 만큼 압축 효율이 떨어진다는 것이다.

이러한 단점은 본 논문에서 제시한 방법은, 동적인 형태로 재배치된 데이터를 한번에 읽어들일 수 있고, 파이프라인 구조로 트리의 정보를 저장할 수 있기 때문에, 새로운 동적 재배치 방법으로 해결할 수 있다.

ABSTRACT

Data is represented by file structure in Computer System. But the file size is to be larger, it is hard to control and transmit. Therefore, in recent years, many researchers have developed new algorithms for the data compression. And now, we introduce a new Dynamic Compression Technique, making up for the weaknesses of huffman's.

The huffman compression technique has two weaknesses. The first, it needs two steps of reading, one for acquiring character frequency and the other for real compression. The second, low compression rate caused by storing tree information.

. These weaknesses can be solved by our new Dynamic Relocatable Method, reducing the reading pass by relocating data file to dynamic form, and then storing tree information from pipeline structure. The first, it needs two steps of reading, one for acquiring character frequency and the other for real compression. The second, low compression rate caused by storing tree information. These weaknesses can be solved by our new Dynamic Relocatable Method, reducing the reading pass by relocating data file to dynamic form, and then storing tree information from pipeline structure.

* 정회원 : 대원과학대학 인터넷 전자상거래과 전임강사

논문접수 : 2001. 6. 1.
심사완료 : 2001. 6. 13.

1. 서론

8비트 컴퓨터를 사용하던 시절에는 파일의 관리가 CP/M 환경 하에서 이루어 졌다. 당시에 파일들은 크기가 32K이하로 매우 작았기 때문에, 파일 관리는 크게 문제될 것이 아니었다. 그러나 하드웨어가 발달하고, 메모리와 보조기억 장치의 용량이 성장함에도 불구하고, 파일의 크기가 큰 프로그램이 개발되어 지면서, 보조기억 장치의 용량은 더 커지기 시작하였다.

그러나, 파일이 많이 들어가면 사용할 공간은 점점 줄어들게 되고, 컴퓨터 통신에서도 많은 파일을 전송하거나 용량이 큰 파일을 전송하고자 할 때 시간이 많이 걸린다. 이런 경우 파일의 크기를 줄여서 시간을 절약하는 방법을 생각하게 한다. 이런 생각으로 만들어진 것이 바로 자료의 압축이다[12].

압축하는 방법은 크게 두 가지로 나누어진다. 첫 번째로, 압축된 것을 원래 상태로 다시 만들 때 실제의 자료와 차이가 생기는 유손(猶孫)압축이 있다. 이것은 주로 화상 데이터나 음성 데이터를 다루는 경우 많이 사용된다. 두 번째로, 압축된 내용을 다시 원상태로 만들 때 실제 자료와 1비트의 오차도 없이 재생되는 압축을 유지(維持) 압축이라 한다. 유지 압축은 현재 통신상에서나 파일 관리에서 사용되고 있다.

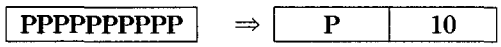
본 논문에서 제시한 압축 기법은 허프만 압축기법의 단점을 보안하여 구현하였다. 허프만 기법의 단점은 처음 파일 내의 문자의 빈도수를 구할 때와 실제로 압축하기 위해서 동작할 때, 파일을 두번 읽어 들인다는 것과, 트리에 대한 정보를 같이 저장해야 되기 때문에, 그 만큼 압축 효율이 떨어진다는 것이다. 4장에서는 허프만 기법을 보안한 새로운 알고리즘을 제시하여 단점을 해결했으며, 5장에서는 새로운 알고리즘의 성능을 평가해 보았다.

2. 자료 압축 방법

기본적으로 파일이 압축되어 크기가 줄어들어도 원래의 상태로 복원이 되어야 한다. 파일은 내부적으로 0과 1로된 디짓(digit)의 조합 형태이다. 그리고 파일을 구성하는 단위는 바이트이다.

또한 한 바이트가 표현할 수 있는 수는 28이므로 0부터 255까지를 기억시킬 수있다. 일반적으로 파일은 한 바이트 안에 0부터 255사이의 어떠한 값이 들어가 있다. 쉽게 말해서 비는 곳이 없다는 것이다. 만약 0부터 127까지만 값이 들어간다면 27=127이므로 파일을 이루는 각각의 바이트에서 1개의 비트는 남아 있는 것이다. 따라서 이 한 개의 비트를 줄이면 압축이 될 수 있다. 현재 사용되는 압축 기법은 첫 번째로, 관계 데이터베이스에서의 한 투플(tuple)에 같은 의미를 갖는 속성이 둘 이상 있을 때 자료가 중복이 되어 있는 경우 대표가 되는 속성을 하나만 남기고 나머지 속성을 삭제함으로써 중복된 자료를 감소시킬 수 있다. 이와 같은 방법을 중복성 감소 방법(Redundancy Reduction)이라 한다[11]. 두 번째로, 어떤 단어가 자주 반복되는 경우, 이 반복되는 단어들을 사전과 같이 짧은 코드(code)를 할당해서 참조하는 방법을 사전식 참조방법(Dictionary Encoding)이라 한다[10].

이 방법은 같은 단어가 자주 반복될 때 높은 압축률(compression rate)을 이룰 수 있다. 세 번째로, 같은 문자가 여러 번 반복되는 경우, 문자의 반복 수를 계산하여 문자부에는 문자를, 반복 수부에는 반복 수를 기록하는 것을 문자 반복 억제법(Character Repeat Suppression)이라 한다 [5]. 예를 들어서, 다음 [그림 1](a)는 P가 10번 반복되는 경우이다. 이것을 문자부와 반복 수부로 나누어 [그림 1](b)와 같이 각각 P는 문자부에 기록하고 10은 반복 수부에 기록을 한다.



(a) 압축되기 전 (b) 압축된 후

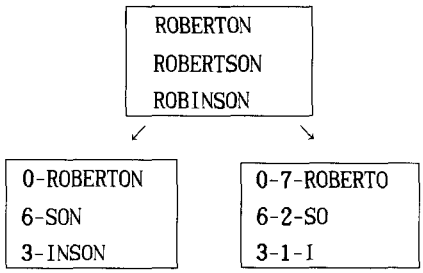
[그림 1] 문자 반복 억제법의 예

[Fig. 1] The example Character Repeat Suppression

그리고 네 번째로, 문자 자료에서는 같은 형태(pattern)의 문자들은 단어, 약어, 코드 등이 나타난다. 이들 중 하나를 특수한 문자로 대치시키는 방법이다 [6].

공통된 표현 억제법(Common Phrase Suppression)은 전위압축(front compression)과 후위압축(rear compression)으로 나눌 수 있다.

전위압축은 [그림 2](a)에서 보듯이 각 항목의 앞부분의 문자를 바로 전의 항목과 같은 문자의 수로 대체하는 방법이다. 그리고 후위압축은 [그림 2](b)에서 두개의 바로 인접한 항목을 구별하는데 필요한 것의 오른쪽에 있는 모든 문자를 제거해서 만들 수 있다. [그림 2]는 3개의 문자 데이터를 전위압축과 후위압축으로 한 예이다 [13].



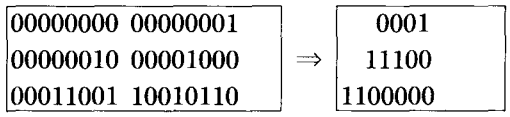
(a) 전위압축 (b) 후위압축

[그림 2] 전위압축과 후위압축의 예

[Fig. 2] The example front compression & rear compression

다섯 번째로는, 자료를 최소의 비트로 표현하는 방법을 간결한 표시방법(Compact Notation)이라 한다. 이 방법은 최대한 작은 단어로 주어진 단어의 의미를 잃지 않는 범위에서 압축시키는 방법을 말한다 [4].

예를 들어서, 다음 [그림 3](a)와 같이 年, 月, 日을 MM, DD, YY로 표시한다고 하면 총 48비트가 필요하게 된다. 그러나 [그림 3](b)와 같이, 月은 12달이므로 4비트로 표시할 수 있고, 달은 31日을 5비트만으로도 표시할 수 있다. 그리고 年은 100年을 7비트만으로 표시할 수 있으므로 총 16비트이면 年, 月, 日을 나타낼 수 있다. 그러므로 총 48비트에서 32비트를 절약할 수 있다.



(a)1996年 1月28日 (b)1996年 1月28日

[그림 3] 간결한 표시의 예

[Fig. 3] The example Compact Notation

여섯 번째로는, 저장장치를 정리하는 방법(Storage Compaction)이다. 이것은 데이터베이스 안의 모든 블럭(block)을 정리함으로서 평균 검색 시간을 감소시키고, 쓰레기 수집(garbage collection)하여, 저장장치(storage)의 활용도를 높인다. 즉, 데이터베이스 안에 있는 모든 블록들을 사용하는 것과 사용되지 않는 것으로 구분하여 저장 활용도를 높이는 기술이다 [7].

그리고 마지막으로, 허프만 방식(Huffman Coding)이 있다. 허프만 방식은 통계적으로 독립적인 원시자료를 최소 평균 단어 길이로 기록하기 위한 절차이며 트리(tree) 개념을 이용하여 처리한다. 허프만 압축법의 장점은 기존의 아스키코드의 값보다 더 작은 크기의 대표값을 새로 만들어 사용한다. 그리고 단점으로는 첫째, 처음 파일 내의 문자가 나오는 빈도를 계산할 때와 실제로 압축하기 위해 동작할 때, 파일을 두 번 읽어야 하기 때문에 처리 속도가 늦다. 둘째, 트리에 대한 정보도 같이 저장해야 하므로 그만큼 압축 효율이 떨어지게 된다. 이 허프만 방식은 다음 장에 자세하게 다루어져 있다 [12].

3. 허프만 압축(Huffman Compression) 방법

3.1 허프만 압축 과정

허프만(Huffman) 압축 기법은 네 가지 과정을 거쳐서 이루어진다. 첫 번째로 압축할 파일을 읽어 각 문자들의 출현 빈도수를 구하고, 두 번째로 이들 가운데에서 가장 빈도수가 적은 문자들끼리 연결시켜 이진 트리(binary tree)를 만든 다음, 세 번째는 이진 트리로부터 각 문자들을 대표하는 코드값을 얻고, 마

지막 네 번째는 파일의 문자들을 대표값으로 변환시켜 압축된 파일을 만든다[8].

이 허프만 압축기법의 장점은 기존의 아스키코드의 값보다 더 작은 크기의 대표값을 새로 만들어 사용한다는 것이다. 허프만 압축기법의 단점으로는 두 가지가 있다. 첫째로 처음 파일 내의 문자가 나오는 빈도를 계산할 때와 실제로 압축하기 위해 동작할 때, 파일을 두 번 읽어야 하므로 처리 속도가 늦다. 둘째로 트리에 대한 정보도 같이 저장해야 되기 때문에 그 만큼 압축 효율 떨어지게 된다 [12].

3.2 허프만 압축 방법을 이용한 예

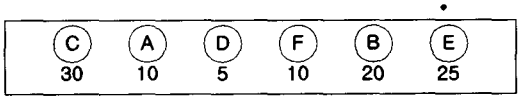
허프만 압축 방법을 설명하기 위하여, 각 문자들의 출현 빈도가 <표 1>과 같이 100바이트의 크기를 갖고 있고 6개의 문자만으로 이루어진 파일을 예를 들어서 설명하겠다.

<표 1> 출현 빈도수

<Table 1> Frequency number

문 자	빈 도 수	문 자	빈 도 수
D	5	B	20
A	10	E	25
F	10	C	30

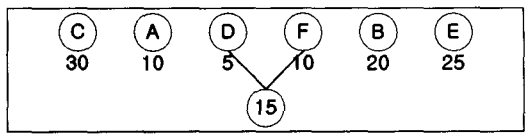
<표 1>과 같이 파일을 첫 번째로 읽어서 그 파일에 있는 문자의 빈도수를 계산하는 단계이다. <표 1>에 있는 문자와 빈도수를 [그림 4-1]와 같이 적당히 나열한다.



[그림 4-1] 노드 초기화

[Fig. 4-1] node initialize

우선 나열한 문자들 중에서 가장 빈도가 낮은 것을 두개 고른다. 여기서는 D가 가장 빈도가 낮고, A와 F가 그 다음으로 빈도가 낮다. 이때 D와 A 혹은 D와 F 가운데 어떤 조합을 선택해도 상관없지만 여기서는 D와 F를 사용하도록 하겠다. 이 둘을 묶어서 하나의 노드(node)를 만들고 두 문자들의 빈도의 합을 구한다 [그림 4-2].

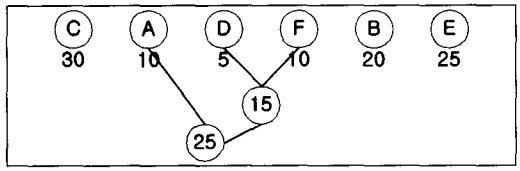


[그림 4-2] 노드 D와 F의 결합

[Fig.4-2] assembly for node D&F

하나의 노드가 완성된 다음에는 다시 가장 빈도가 낮은 두 문자를 고른다. 이때, 이미 노드로 묶여 있는 문자들은 한 개의 문자로 생각한다.

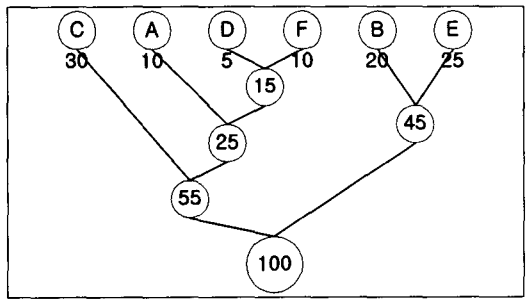
따라서 가장 빈도가 낮은 두 문자는(D+F) 노드와 A로, 이 둘을 묶어 또 하나의 노드를 만든다 [그림 4-3].



[그림 4-3] 노드 DF와 A의 결합

[Fig. 4-3] assembly for node DF&A

이러한 방법으로 모든 문자가 하나의 노드에 연결되도록 계속 노드를 만들어 나간다. 즉, 모든 문자들에 대한 트리(tree) 구조를 만드는 것이다 [그림 4-4].



[그림 4-4] 완성된 이진 트리

[Fig. 4-4] Binary Tree

트리 구조를 완성한 후에는 각 문자에 고유의 대표값을 부여한다. 즉, 가장 아래의 노드(root)에서 시작하여 왼쪽으로 가면 0, 오른쪽으로 가면 1의 값을 부여하는 것이다. 이 경우에 각 문자에 부여되는 대표값은 <표 2>와 같다.

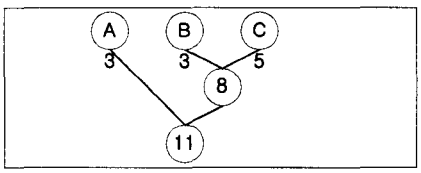
<표 2> 대표값

<Table 2> New code

문 자	대 표 값	비 트 수
C	00	2
A	010	3
D	0110	4
F	0111	4
B	10	2
E	11	2

따라서 8비트를 사용하여 표현되던 문자들은 2~4 비트만으로 표시할 수 있게 되며, 그 만큼 압축 파일의 길이가 줄어들게 된다. 이로써 빈도수가 많은 문자들일수록 작은 비트로 표현할 수 있고, 빈도수가 적은 문자들일수록 표현하는 비트수가 많아짐을 알 수 있다.

다음은 실행 결과 값을 표시한 예이다. 예를 들어 'ABCABCABCCC'라는 11바이트의 데이터가 있다고 보면 빈도수는 A는 3개 B는 3개 C가 5개임을 알 수 있다. 이것을 위의 네 가지 방법으로 트리 구조를 만들고 대표값을 나타내어 보면 다음 [그림 5]와 같다.



[그림 5] 'ABCABCABCCC' 때트리구조의 예

[fig. 5] Tree structure in 'ABCABCABCCC'

<표 3> [그림 5]에서 대표값

<Table 3> New Code for [Fig. 5]

문 자	빈 도 수	대 표 값
A	3	0
B	3	10
C	5	11

데이터 'ABCABCABCCC'의 비트 상태를 표시하여 보면 '0101101011010111111'이다. 이것을 8비트로 나누어 보면 '01011010 11010111 111'이다. 이와 같이 11바이트가 2바이트와 3비트의 출력 값을 얻을 수 있다. 물론, 구조에 대한 정보는 따로 필요하다. 허프만 방식을 이용하여 위와 같이 압축된 파일을 만들어 보았다. 허프만 방식은 빈도수를 구할 때 파일을 한번 읽고 실제로 압축할 때 파일을 다시 한번 읽기 때문에 처리 속도가 늦다. 그리고 트리에 대한 정보도 저장해야 되기 때문에 그 만큼에 압축 효율이 떨어진다.

위와 같은, 허프만 압축 방식의 단점인 처리 속도와 압축 효율의 감소를 보안하고자, 본 논문에서는 파일을 한번 읽을 때 트리를 동적으로 재배치하여 압축 파일을 만드는 방법을 제안한 것이다. 본 논문에서 제안한 압축 방법을 동적 재배치 방법에 의한 압축 기법(Compression Techniques by Dynamic Relocatable Method)이라 하겠다.

4. 동적 압축기법의 구현

4.1 구현 환경

본 장에서는 허프만 압축(Compression) 방식의 단점을 보안하여 동적 압축 기법을 구현하였다. 이 동적 압축 기법은 IBM 펜티엄(pentium processor)에서 C-언어를 사용하여 구현하였다. 그리고 시간은 C-언어의 기본 함수인 <time.h>을 사용하여 측정하였다. 허프만 기법에서는 파일을 한번에 읽어서 빈도수를 구하고 트리를 구성하는 반면, 동적 압축 기법은 문자 단위로 읽어서 트리를 구성하여 압축 단계까지 한번에 할 수 있게 구현하였다. 그리고 시간은 압축 단계나 복원 단계가 끝난 후에 초단위로 측정할 수 있도록 설계하였다.

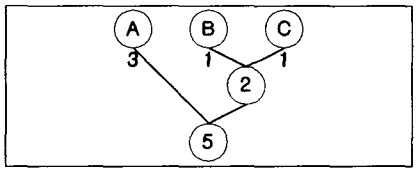
동적 압축기법의 단계는 파일을 읽은 후에 그 파일이 압축된 파일인가를 판단한다. 그런 다음 압축되기 전의 파일이면, 암호화(Encode) 단계와 동적재배치(Update) 단계를 거치면 압축된 파일을 출력할 수 있다. 만약, 압축된 파일이면 해독화(Decode) 단계와 동적재배치 단계를 거쳐서 압축되기 전의 파일로 복원된다. 4.3절의 알고리즘에 자세하게 나타나 있다.

4.2 동적 압축 기법

동적 압축 기법은 허프만 압축(Compression) 방식의 처리 속도와 압축 효율의 감소라는 단점을 보안하여 구현하였다. 첫 번째로 파일을 두 번 읽어야 하는 것을 제시한 압축 기법에서는 데이터를 한 문자(character) 단위로 읽을 때마다 트리를 재배치 해주고 압축을 하도록 설계하였다.

이렇게 함으로써 허프만 압축기법의 단점인 빈도수를 구할 때와 실제로 압축할 때 파일을 두 번 읽는 것을 한번만 읽으면 파일을 압축시킬 수가 있다. 두 번째로는 트리에 대한 정보를 스스로 트리에 저장하고 전송하는 방법, 즉 파이프라인(pipeline) 구조를 이용해서 단점들을 해결하였다.

다음에 [그림 10]에 있는 동적재배치 알고리즘을 이용하여 [그림 6]과 같이, 트리를 동적으로 재배치 해주는 예를 들어보았다. 입력 스트링은 'ABCAA BBB'이다. 우선 처음으로 'ABCAA'까지 들어왔을 때의 트리 구조가 똑같음을 알 수 있다. 여기서의 결과 값은 '0101100' 이다.

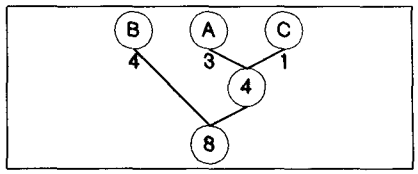


[그림 6] 'ABCAA'까지 일 때의 트리 구조와 대표값
[Fig. 6] Tree structure & New code until 'ABCAA'

<표 4> [그림 6]에서 대표값
<Table 4> New Code for [Fig. 6]

문 자	빈 도 수	대 표 값
A	3	0
B	1	10
C	1	11

다음은 'ABCAABB'까지 들어오는 트리 구조도 [그림 6]과 똑같다. 다음 [그림 7]은 다음에 B가 들어왔을 때의 트리가 재배치되므로 입력 스트링 'ABCAABBB'가 들어왔을 때의 트리가 변형된 모습을 보여준다.



[그림 7] 트리의 동적재배치 과정
[Fig. 7] Tree Dynamic Relocatable

<표 5> [그림 7]에서 대표값
<Table 5> New Code for [Fig. 7]

문 자	빈 도 수	대 표 값
A	3	10
B	4	0
C	1	11

다음과 같이 들어오는 문자별로 즉, 입력 숫자가 많아질 때 트리는 재배치하게 된다. 여기서의 최종 결과 값은 '100111010000'이 되고, 8바이트가 1바이트와 4비트로 축소가 되었다.

4.3 동적 압축 기법 알고리즘

자료(data)를 압축(Compression)하는 알고리즘은 다음 [그림 8]과 같다. 그리고, 압축된 자료를 다시 복원(Decompression) 시키는 방법은 압축 방법의 역순으로 트리의 루트로서 구성된 제일 큰 노드의 정보를 보고 트리로 구성한다. 그리고 압축되기 전의 데이터를 문자 단위로 정보를 추출해 냄으로서 트리를 압축과 마찬가지로 자료를 복원시킬 수가 있다. 다음에 압축파일이 복원되는 알고리즘은 다음 [그림 9]와 같다.

```
ENCODE
 IF(inf.get(c) == FALSE)  byte_left = FALSE;
 WHILE(byte_left)
[모든 문자가 encode될때까지 실행한다]
 CALL encodeandtransmit
 CALL update
        byte_left = inf.get(c)
 IF (shift !=0)
   FOR (x=shift; x<16; x++)
 CALL   transmit
ENCODE AND TRANSMIT
 IF (q<=m)
   IF (--q<2*r) t=e+1;
    ELSE  q-=r; t=e;
    FOR (ii=1; ii<t; ii++)
          stack[++i]=q%2; q=q/2; q=m;
    IF (m==N) root=N;
     ELSE   root=z;
 WHILE (q!=root)
   [트리를 아래에서 위로 검색한다]
 FOR (ii=i; ii>=1; ii--)
 CALL   TRANSMIT (stack[ii]);
 TRANSMIT [문자 단위로 스택에 삽입한다]
 CALL   chkshiftwriteword
         bitset (writeword, shift, i);
         shift++;
```

[그림 8] 압축 알고리즘

[Fig. 8] Compression Algorithm

```
 DECODE
 FOR (x=OLU; x<fsize; x++) outf.put
     (c = byte
 CALL reciveanddecode
 CALL update

 RECEIVE AND DECODE
 IF (m==N) q=N;
   ELSE q=z; [Set q to the root node]
 WHILE (q>N) [트리를 위에서 아래로 검색한다]
   q = findchild (q, receive( ))
 IF (q==m) [Decode 0-node]  q=0;
 FOR (i=0; i<e; i++)  q=q*2+receive( );
 IF (q<r)  q=q*2+receive( );
  ELSE  q+=r; q++;
 RETURN  alpha[q];
 RECEIVE [문자 단위로 값을 리턴한다]
 CALL   chkshiftreadword
         bitget (writeword, shift, value);
         shift ++;
 RETURN  value;
```

[그림 9] 복원 알고리즘

[Fig. 9] Decompression Algorithm

다음은 트리에서 노드를 찾고 그 노드가 자식이 있다면 그 노드를 분리시킨다. 트리를 동적으로 재배치 하는 알고리즘이다. 이 알고리즘은 [그림 10]에 제시되어 있다.

```
UPDATE
 CALL   findnode
 WHILE  (q>0)
 CALL   slideandincrement
 IF (leaftoincrement != 0)
   q=leaftoincrement;
 CALL   slideandincrement
FINDNODE
 IF (q<=m)
 CALL   interchangeleves (q,m);
  IF (r==0) r=m/2;
  IF (r>0)  e=e-1;
      m=m-1; r=r-1; q=m+1; bq=block[q];
  IF (m>0)
    [q의 블럭에서 첫번째 노드로 q를 바꾼다.]
 CALL interchangeleves (q, first[block[q]]);
  q = first[block[q]];
 IF  ((q==m+1) && (m>0))
   leaftoincrement = q;
   q=parent [block[q]];
 SLIDE AND INCREMENT
 [블럭에서 q는 첫번째 노드이다]
 IF(q<=N)&&(first[nbq]>N)
   &&(weight[nbq]==weight[bq]
   (q>N)&&(first[nbq]<=N)
   &&(weight[nbq]==weight[bq]+1)
 [다음 블럭에서 q를 쪼갠다]
[트리에서 다음 레벨로 자식 포인터를 연결]
 [블럭 nbq로 부모 포인터를 연결한다]
 IF [블럭에서 q를 병합한다]
   IF [오래된 q는 없앤다]
   ELSE  IF(last[bq]==q)
 ELSE  IF(parity[bq]==0)
[연결된 블럭에서 적당한 크기로 q의 블록에
 삽입한다.]
[트리에서 q를 한단계 높인다]
 IF (q<=N)  q=oldparent;
 ELSE  q=par;
```

[그림 10] 동적 재배치 알고리즘

[Fig 10] Dynamic relocatable algorithm

5. 성능 평가 및 결론

5.1 성능 평가

허프만 압축법을 사용한 결과에서는 800비트(100바이트)길이의 파일이 240비트(30바이트)로 줄었으므로, 파일의 길이는 240/800=30%로 단축되었다. 그러나 압축된 파일을 풀 때는 대표값을 생성한 트리에 대한 정보가 없으면 불가능하므로, 압축 파일과 트리에 대한 정보를 같이 저장해야 한다. [그림 4-1]에서, 트리는 5개의 노드와 6개의 문자로 구성되며 각 노드 및 문자를 표시하기 위해서는 4바이트 크기의 포인터가 필요하므로 (5+6)×4=44바이트가 요구된다.

또한 노드 숫자에 대한 정보와 그 외에 필요한 정보를 담기 위해 몇 바이트가 더 필요하다. 따라서 트리에 대한 정보가 50바이트가 필요하다고 가정하면, 압축 파일의 길이는 30+50=80바이트가 되므로 원래 파일의 길이와 비교해 볼 때 (30+50)/100=80%로 줄어들게 된다.

본 논문에서 제시한 동적 압축 알고리즘을 적용해서 다음 <표 6>과 같이, 압축률을 비교하여 보았다. 본 논문에서 제시한 압축률이 평균 65% 압축된 것을 보여주고 있다.

<표 6> 압축 비교표

<Table 6> Compression Compare

	허프만 방식		제안한방식	
파일 크기	압축된 후	압축률	압축된 후	압축률
1.2	0.8	66%	0.8	66%
5.7	3.8	66%	3.6	63%
9.6	7.0	73%	6.2	64%
50.9	39.1	77%	32.2	63%
99.6	78.6	79%	69.0	69%

<파일크기:10,000 byte>

그리고, 다음 <표 7>와 같이, 제안한 기법에서 압축되는 시간과 복귀되는 시간과의 차이점은 거의 없다고 볼 수 있다. 시간은 C-언어의 기본 함수 <time.h>를 사용하여, 압축되는 시간과 복귀되는 시간을 측정하였고 시스템은 펜티엄(pentium processor)을 사용하였다.

<표 7> 시간 비교표

<Table 7> Time Compare

시간 \ 바이트 수	5	10	50	100
압축 시간	1	2	10	22
복귀 시간	1	2	11	23

<파일크기 : 10,000byte> <단위 : sec>

이 동적 압축 알고리즘의 장점은 단일 패스(one-pass)이고, 트리를 동적으로 재배치 할 수 있다는 것이다. 암호화(encoder)단계에서는 단일 패스(one-pass)로 자료를 최적화 한다. 해독화(decoder)단계에서는 아무런 과부하 없이, 스스로 트리에 저장하고 보낸다. 그리고 암호화(encoder)단계와 마찬가지로 바로 트리를 만들 수 있다.

그리고 이 동적 압축 알고리즘의 단점으로는 첫 번째, 허프만 형식 압축 루틴과 비교해 보면 매우 큰 메모리를 요구하고, 두 번째로는 정형화된 허프만 트리 보다는 좋지 않다. 허프만 방식을 비공식적으로 테스트 해본 결과 약 30%~80%의 압축률을 보인 반면 본 논문에서 제시한 기법은 63%~69%의 압축률을 보여주고 있다.

본 논문에서 제시한 기법은 압축률이 매우 고르게 나타났으며, 정규화된 허프만 방식보다는 압축률이 매우 낮지만 비정규화된 것보다는 압축률이 높게 나타났다.

5.2 결론

본 논문에서는 트리를 동적으로 재배치해 주는 방법으로 동적 재배치 방법에 의한 압축 기법(Compression Techniques by Dynamic Relocatable Method)을 제시하였다. 본 논문의 동적 압축기법의 장점은 첫째, 파일을 한번만 읽어서 압축하는 단일 패스이고, 둘째, 트리를 동적으로 재배치하여 새로운 코드값을 부여하는 것이다. 그리고, 단점으로는 첫 번째로 허프만 기법보다는 매우 큰 메모리를 요구한다는 것과, 두 번째로는 자료가 정형화된 허프만 트리보다는 좋지 않다 라는 것이다.

허프만 기법의 압축률은 30%~80%이고, 그에 반해 동적 압축 기법은 63%~69%의 압축률이 나타났다. 압축률은 평균적으로 동적 압축 기법은 65%이고, 허프만 압축 기법은 55%이다. 동적 압축 기법이 10%의 성능 저하가 나타났다. 그러나 파일의 형태가 비정형 일 때는, 허프만 압축 기법은 80%이고, 본 논문에서 제시한 동적 압축 기법은 69%의 압축률이므로, 동적 압축기법의 성능이 11% 향상되었다.

자료들은 형태가 여러 가지이고 매우 복잡하다. 그러므로, 자료의 형태가 정형 일 때 보다는 비정형 일 때가 더 많다. 그러므로, 본 논문에서 제시한 동적 압축 기법을 적용하는 것이 좋다. 앞으로 연구해야 할 방향은, 이 기법의 단점인 메모리를 많이 요구하는 것을 해결하는 것과 여러 기종에도 호환성 있게 구현하는 것이 향후 연구 방향이다.

※ 참고문헌

[1] D.Comer., "The Ubiquitous B-tree", ACM Computing Surveys V.1.11, No.2 1979. pp.121-137.

[2] R.A.Baeza-Yates, P.A.Larson,. "Performance of B+-tree with Partial Expansions",. IEEE Transaction on Knowlege and Data Engineering. Vol. 1, No.2,June 1989, pp.248-257.

[3] K.L.Kelby and M.Rustnktewtcz., "Multikey Extensible Hasing for Relational Database",. IEEE software, July 1988, pp.77-85.

[4] Alsberg, P., "Space and Time Savings Through Large Database Compressionand Dynamic Restructuring", Proc. of The IEEE, Vol. 63, No.8, Aug.1975, pp.1114-1122.

[5] Ruth, S. and Kreutzer, P., "Data Compression for Large Business Files", Datamation, Vol. 18, No.9, Sept.1972, pp.62-66.

[6] Lea, R.M., "Text Compression with Associative Parallel Processor". IEEE Brutush Computer Society., The Computer, Vol.21, No.1 Feb.1978, pp.45-56.

[7] Pike, J., "Test Compression Using a 4-bit Coding Schema". The Computer Joural, Vol.24, No.4, Nov. 1981, pp.324-330.

[8] Jeffery Scott Vitter "Implementation of the dynamic Huffman Coding algorithm" Journal of ACM, Vol.34, October 1987, pp.825.

[9] Debre, A., Lelewer and Daniel, S., "Hirschberg. Data Compression", Departement of Information and Computer Science. Univ of California, Irvine, California 92717.

[10] Korth, H. and Silberschatz, A. "Database System Concepts", McGraw-Hill., 1986.

[11] 조태남, "데이타베이스 시스템의 이론과 설계", 1994년 9월

[12] 이해원, "자료압축의 요구와 배경", MYCOM, 1991,7,pp.94-111

[13] 박석, "데이타베이스 시스템", 1991년

최 강 희

1988.3-1993.2 관동대학교
정보처리과 이학사
1993.3-1995.2 관동대학교
전자계산공학과 공학석사
1996.6-2000.2 관동대학교
전자계산공학과 공학박사
1996.3-1999.8 (주)토탈정보통신
부장
1999.9-2001.2 (주)컴투어 이사
1998.3-1999.2 동우대학
사무자동화과 겸임교수
2000.9-2001.2 안동과학대학
정보처리학과 겸임교수
2001.3-2001.7 현재
대원과학대학
인터넷 전자상거래과 전임강사