Journal of KIIT. Vol. 17, No. 2, pp. 63-70, Feb. 28, 2019. pISSN 1598-8619, eISSN 2093-7571 63
http://dx.doi.org/10.14801/jkiit.2019.17.2.63

PreHuffman 알고리즘을 이용한 유사 패턴 데이터 압축에 관한 연구

맹일희*, 박지수**, 손진곤***

A Study on Similar Pattern Data Compression Using PreHuffman Algorithm

Il Hee Maeng*, JiSu Park**, and Jin Gon Shon***

이 논문은 2018년도 학술연구비 지원을 받아 작성된 것임.

요 약

허프만 코딩은 무손실 데이터 압축 알고리즘으로 출현 문자의 빈도수를 이용하여 사전을 만든 후 부호화한다. 그러나 데이터 압축할 때 연산은 단순하지만 데이터를 항상 두 번 읽어 부호화하기 때문에 데이터를 한 번 읽는 것으로 압축할 수 없는 단점이 있다. 본 논문에서는 패턴이 유사한 데이터를 압축할 경우 데이터를 한 번만 읽고 부호화하는 알고리즘을 연구하고, 압축 시간을 단축하는 PreHuffman 알고리즘을 제안한다. 제안 알고리즘의 성능을 검증하기 위해 PreHuffman과 허프만 그리고 동적 허프만의 성능을 비교 분석하였다. 실험에서 PreHuffman의 압축률은 허프만과 동적 허프만에 비해 1~3% 낮았으나, PreHuffman의 압축 시간은 허프만과 동적 허프만 보다 40~50% 시간을 단축하였다.

Abstract

Huffman coding creates and encodes the frequency of characters appearing due to lossless compression. Operations are simple, but because they always read and encode data twice when compressed, there is a disadvantage that data can not be compressed before reading the data once and completing the dictionary. This paper suggests a PreHuffman algorithm that studies algorithms that read and encode data once when the pattern compresses data similar to each other and shortens compression time. To test the performance of the proposed algorithm, we compared the performance of PreHuffman, Huffman and Dynamic Huffman. In the experiment, the compression rate of PreHuffman was 1 ~ 3% lower than that of Huffman and Dynamic Huffman, but the compression time of PreHuffman was 40 ~ 50% shorter than that of Huffman and Dynamic Huffman.

Keywords
prehuffman, huffman, dynamic huffman, lossless compression, real-time data

* 한국방송통신대학교 정보과학과
- ORCID: https://orcid.org/0000-0001-9802-9108
** 경기대학교 융합교양대학 교양학부 조교수
- ORCID: https://orcid.org/0000-0001-9003-1131
*** 한국방송통신대학교 컴퓨터과학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0002-0540-4640

· Received: Dec. 01, 2018, Revised: Feb. 19, 2019, Accepted: Feb. 22, 2019
· Corresponding Author: Jin Gon Shon
Dept. of Computer Science, Korea National Open University,
86 Daehak-ro, Jongro-gu Seoul, Korea (03087)
Tel.: +82-2-3668-4656, Email: jgshon@knou.ac.kr

Ⅰ. 서 론

최근 사물 인터넷과 정보 통신 기술의 발전은 다양한 종류의 많은 정보를 생산하고, 네트워크를 통해 송·수신하거나 저장매체에 저장한다. 그러나 정보 통신에서 전송 속도와 대역폭은 제한적이며 전송 속도와 전송량에 따라 높은 비용이 요구된다. 그러므로 제한된 전송 속도와 대역폭 그리고 기억용량 안에서 공간을 절약하고 처리 속도를 높이기 위한 데이터 압축 기술 연구가 필요하다[1]-[3].

대표적인 무손실 데이터 압축 알고리즘인 허프만 코딩은 문자의 출현 빈도수에 따라 이진트리를 만들고, 이진트리에 가중치를 부여하여 사전을 생성한다. 이렇게 생성된 사전을 참조하여 원본 문자의 부호를 교환함으로써 데이터를 압축한다. 허프만 코딩의 장점으로는 알고리즘이 간결하고 데이터 압축 처리 시간이 빠르다. 그리고 텍스트 문자열 압축에 높은 압축률을 보인다. 그러나 압축을 위해 원본 데이터를 전체적으로 두 번 읽는다. 첫 번째는 허프만 트리를 생성하기 위해 읽고, 두 번째는 문자의 부호를 교환하기 위해 읽는다. 따라서 호프만 코딩은 사전이 만들어지기 이전에는 데이터를 부호화할 수 없다[4]-[7].

동적 허프만 코딩은 허프만 코딩의 단점을 원-패스(One-pass) 방식으로 개선한 알고리즘이다. 실시간으로 데이터를 읽으면서 출현 문자의 빈도수를 검사하고, 이진트리 갱신, 가중치 부여, 사전 갱신, 데이터를 부호화하기 때문에 실시간으로 데이터를 압축하고 전송할 수 있다. 그러나 데이터를 읽으면서 빈도수 검사와 이진트리 갱신을 동시에 반복 수행하기 때문에 연산 처리가 많고 처리 속도가 느리다.

이에 허프만 코딩의 장점인 알고리즘의 간결함은 유지하면서 데이터를 두 번 읽어 처리하는 단점을 보완이 필요하다. 특히 무선 센서 네트워크에서 데이터 수집시 패턴이 유사한 데이터를 압축하는 환경과 같은 곳에서 더욱 효과적으로 적용할 수 있는 압축 알고리즘의 연구가 필요하다[8][9].

본 논문에서는 패턴이 유사한 데이터를 압축할 경우 데이터를 한 번만 읽고 효과적으로 압축할 수 있는 PreHuffman 알고리즘을 제안한다. PreHuffman 알고리즘은 허프만 코딩의 전처리 알고리즘(Preprocessing Algorithm of Huffman Coding)에 해당된다. 이는 유사한 패턴 데이터를 이용하여 공용 사전(이하. PreHuffman 사전)을 미리 만들고, 반복해서 사용함으로써 데이터의 압축 처리 시간을 단축한다.

Ⅱ. 데이터 압축 알고리즘

사전 기반 무손실 데이터 압축 알고리즘은 코드 부여 방식에 따라 가변 길이 코드와 고정 길이 코드가 있다. 가변 길이 코드 방식을 사용하는 알고리즘에는 허프만, 동적 허프만, LZW 등이 있다. 본 논문에서는 허프만, 동적 허프만 코딩 방법과 비교 평가를 한다.

허프만 코딩은 문자의 출현 빈도수를 검사하여 출현 빈도수가 많은 문자에 적은 수의 비트를 부여한다. 출현 빈도수가 적은 문자에는 많은 수의 비트를 할당하고, 가변 길이 문자 코드 사전을 생성한 후 사전을 참조하여 데이터를 부호화한다.

허프만 코딩의 부호화는 5단계를 거쳐 진행된다. 첫째, 문자의 출현 빈도수를 검사하고 우선순위 큐에 검사 결과를 넣는다. 둘째, 우선순위 큐에서 두 개의 노드를 추출하여 이진트리를 만들고, 두 노드의 빈도수 합으로 루트 노드를 정한다. 생성된 이진트리의 루트 노드를 우선순위 큐에 삽입한다. 셋째, 모든 원소가 이진트리의 원소가 될 때까지 둘째 과정을 반복한다. 넷째, 이진트리가 완성되면 루트를 기준으로 왼쪽은 0, 오른쪽은 1의 가중치를 부여하고, 루트 노드부터 문자 노드까지 경로에 부여된 가중치로 가변 길이 문자 코드표인 허프만 사전을 생성한다. 다섯째, 사전을 이용하여 문자를 부호화함으로써 데이터를 압축한다.

동적 허프만 코딩은 원-패스 방식으로 문자열로부터 문자의 빈도를 검사하고, 허프만 트리를 갱신하면서 동시에 부호화를 한다.

동적 허프만 코딩은 4단계를 거쳐 진행된다. 첫째, 이진트리에서 노드들은 왼쪽에서 오른쪽으로, 아래에서 위로 가중치를 부여한다. 둘째, 동적 허프만 트리는 양단 성질을 항상 유지하고, 모든 노드들을 빈도수 내림차순으로 정렬한다. 셋째, 양단 성질

에 위반되면 이진트리를 갱신하기 위해 노드들을 재 정렬하고 교체 과정을 수행한다. 넷째, 빈도수가 N인 노드와 빈도수가 N+1인 노드를 교체한다.

III. PreHuffman 알고리즘

3.1 PreHuffman 알고리즘의 구성

PreHuffman 알고리즘은 기존 허프만 코딩을 두 개의 알고리즘으로 나누어 구성한다. 첫 번째(①)는 패턴 데이터들을 이용하여 사전을 생성하는 PreHuffman 사전 생성 알고리즘이다. 두 번째(②)는 실시간으로 입력되는 데이터를 PreHuffman 사전을 참조하여 즉시 부호화하는 PreHuffman 데이터 부호화 알고리즘이다. 이는 PreHuffman의 사전 생성 알고리즘과 데이터 부호화 알고리즘을 분리함으로써 사전 생성 기능을 보완한다. PreHuffman의 사전 생성 알고리즘은 데이터 집합이 입력될 때 마다 사전을 생성하지 않고, 사용자가 옵션으로 사전 생성여부를 결정과 공용 사전 재사용을 여부를 선택할 수 있도록 한다. 다음 그림1은 PreHuffman 알고리즘의 구성이다.

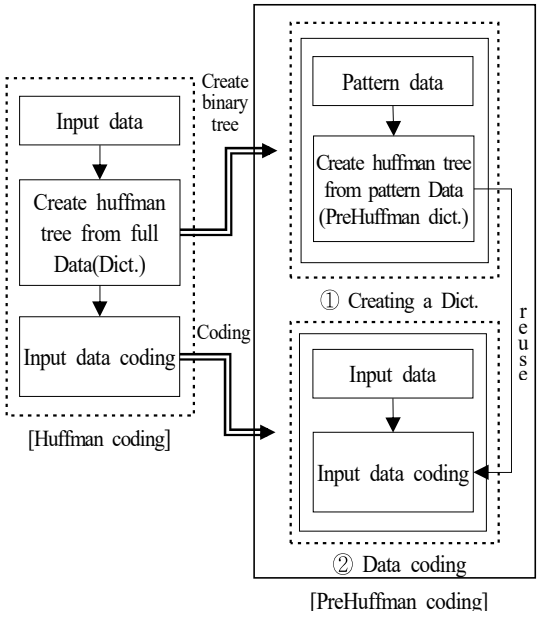

그림 1. PreHuffman 알고리즘의 구성
Fig. 1. Configure of PreHuffman algorithm

3.2 PreHuffman 사전 생성 알고리즘

허프만 사전은 빈도수에 따라 자주 발생하는 문자에는 적은 수의 비트를 부여하고, 드물게 발생하는 문자에는 많은 수의 비트를 부여하는 가변 길이 부호를 가지고 있는 부호의 집합이다. 허프만 사전 생성은 문자의 출현 빈도수에 따라 부호의 비트수를 결정한다. 비트수는 빈도수가 높은 문자는 적은 수의 비트를 부여하고, 빈도수가 낮을수록 많은 수의 비트를 부여한다. 빈도수는 입력 데이터 전체를 읽어 출현 문자수를 검사하여 오름차순으로 정렬하고, 정렬된 빈도수는 우선순위 큐에 저장하며, 빈도수가 높은 문자 두 개씩 인출하여 이진트리를 생성한다[4][5].

PreHuffman 사전 생성(이하 사전)은 생성된 이진트리의 전체 노드에 왼쪽은 0, 오른쪽은 1의 가중치를 부여한다. 본 논문에서는 출현 문자의 빈도수 분포가 유사한 데이터 집합을 유사 패턴 데이터라고 한다. 예로, 첫 번째 문자열(“abbcccddddeeeeeffffff”)에서 a는 1/21, b는 2/21, c는 3/21, d는 4/21, e는 5/21, f는 6/21이다. 두 번째 문자열(“fffffabbbeeeeecccddddd”)은 a는 1/21, b는 3/21, c는 3/21, d는 4/21, e는 5/21, f는 5/21로 문자가 분포되어 있다. 두 문자열의 차이는 b와 f에서 1/21 만큼 분포 차이가 있는 유사 패턴 데이터이다.

유사한 패턴을 보이는 두 종류의 입력 문자열을 이용하여 생성된 사전을 다음과 같다. 첫 번째 문자열로 만든 사전은 {'a', '1000', 'b', '1001', 'c', '101', 'd', '00', 'e', '01', 'f', '11'}이다. 두 번째 문자열로 만든 사전은 {'a', '1000', 'b', '1111', 'c', '100', 'd', '00', 'e', '01', 'f', '10'}이 된다.

또한 PreHuffman 사전 생성 알고리즘에서는 사전 재사용 기능을 추가함으로써 유사 데이터를 압축할 때마다 매번 사전을 생성할 필요 없이 그대로 사용한다. 이로 인해 사전 생성 시간을 단축한다.

PreHuffman 사전 생성 알고리즘은 그림 2를 참조한다.

① 사전과 이진트리 변수를 선언한다.
② 사전을 만들기 위한 패턴 데이터를 입력받아 변수에 저장한다.

③ 입력된 패턴 데이터의 길이만큼 순환하면서 ④ 작업을 수행한다.
④ 문자의 출현 빈도를 검사하고, 문자의 빈도수를 빈도 변수에 저장한다.
⑤ 빈도수를 기준으로 오름차순 정렬하고, 우선순위 큐에 삽입한다.
⑥ 우선순위 큐에 데이터가 없을 때까지 순환하면서 ⑦과 ⑧ 작업을 반복한다. 반복 작업이 완료되면 이진트리가 만들어지고, 만들어진 이진트리에서 ⑨를 실행한다.
⑦ 우선순위 큐로부터 우선순위가 높은 노드 2개를 인출하여 이진트리를 생성한다.
⑧ 우선순위 큐에 생성된 부모 노드를 삽입한다.

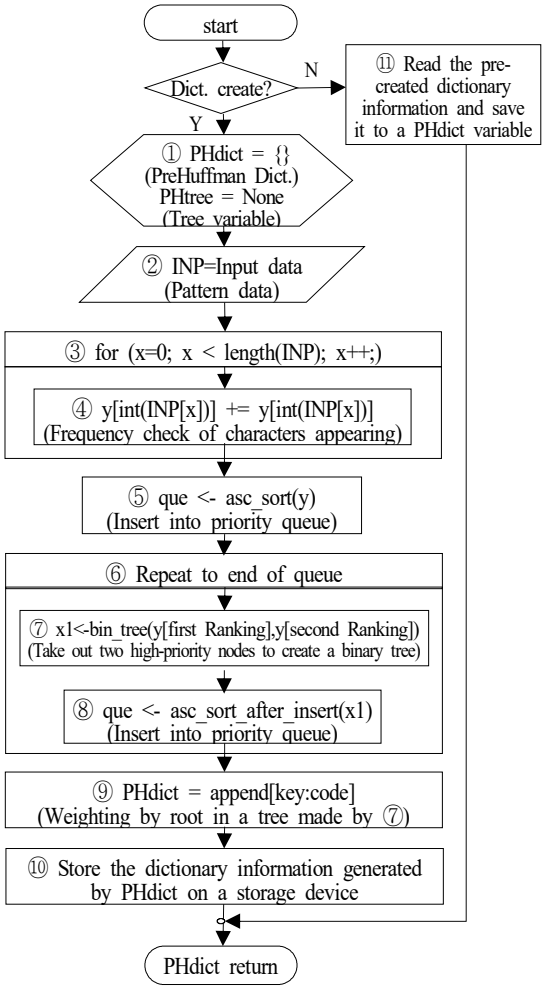

그림 2. PreHuffman 사전 생성 알고리즘
Fig. 2. PreHuffman dictionary create algorithm

⑨ 이진트리 전체 노드에 가중치를 부여하여 사전을 만든다.
⑩ 사전은 dictionary.dat 이름으로 저장장치에 저장한 후 사전을 반환한다.

3.3 PreHuffman 데이터 부호화 알고리즘

PreHuffman 데이터 부호화 알고리즘은 PreHuffman 사전을 사용하여 부호화를 한다. 다음 그림 3은 PreHuffman 데이터 부호화 알고리즘의 흐름도이다.

① PreHuffman 사전 생성 알고리즘을 호출한다. 호출된 알고리즘은 사전 생성 여부를 선택하는 옵션이 있다. 사용자 선택이 '예'이면 새로운 사전을 생성하여 반환하고 '아니요'이면 이전에 생성해 놓은 사전을 반환한다. 반환받은 사전은 변수에 저장하고 부호화 작업에 사용된다.
② 압축을 할 입력 데이터를 입력받는다.
③ 입력 데이터 길이만큼 반복하며, ④ 작업을 수행한다.
④ 한 문자씩 사전에서 찾아 부호로 교환하고, 교환이 전부 완료되면 현재 사전을 사전 영역에 추가할 것인지 결정한다. 사전 추가 여부는 알고리즘이 실행되기 전에 사용자에 의해 결정된다.

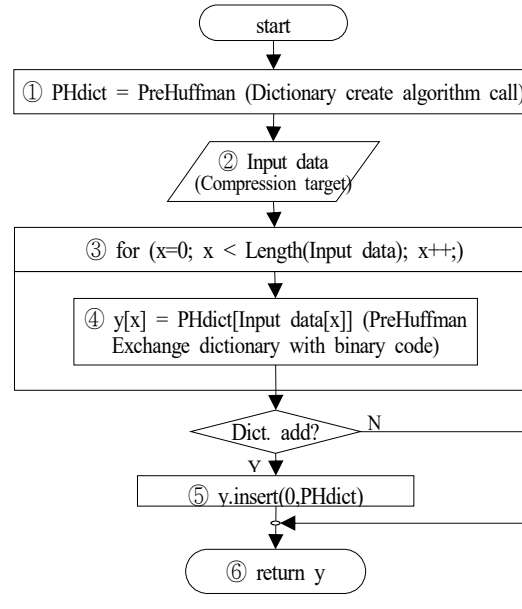

그림 3. PreHuffman 데이터 부호화 알고리즘의 흐름도
Fig. 3. Flowchart of PreHuffman data coding algorithm

⑤ ‘예’이면 부호화된 데이터 영역 앞에 사전 정보를 삽입하고, ‘아니요’이면 작업을 통과한다.
⑥ 부호화된 데이터를 반환 후 끝낸다.

압축 대상 데이터가 실시간으로 입력되면 제안하는 PreHuffman 데이터 부호화 알고리즘을 통해 PreHuffman 사전을 참조하여 문자가 즉시 부호화되고, 비트수가 감소한 데이터를 얻을 수 있다.

3.2의 예를 이용하면, 첫 번째 문자열을 부호로 교환하면 이진수 $1000_{(2)}$ $1001_{(2)}$ $1001_{(2)}$ $101_{(2)}$ $101_{(2)}$ $101_{(2)}$ $00_{(2)}$ $00_{(2)}$ $00_{(2)}$ $00_{(2)}$ $01_{(2)}$ $01_{(2)}$ $01_{(2)}$ $01_{(2)}$ $01_{(2)}$ $11_{(2)}$ $11_{(2)}$ $11_{(2)}$ $11_{(2)}$ $11_{(2)}$ $11_{(2)}$이 된다. 두 번째 문자열은 이진수 $10_{(2)}$ $10_{(2)}$ $10_{(2)}$ $10_{(2)}$ $10_{(2)}$ $1000_{(2)}$ $1111_{(2)}$ $1111_{(2)}$ $1111_{(2)}$ $01_{(2)}$ $01_{(2)}$ $01_{(2)}$ $01_{(2)}$ $01_{(2)}$ $100_{(2)}$ $100_{(2)}$ $100_{(2)}$ $00_{(2)}$ $00_{(2)}$ $00_{(2)}$ $00_{(2)}$이 된다. 부호화된 두 문자열을 압축하면 첫 번째 문자열은 168비트에서 51비트로 압축되고, 두 번째 문자열은 168비트에서 49비트로 압축된다.

Ⅳ. 실험 및 결과

4.1 실험 환경

본 논문의 실험 환경은 Intel(R) Core(TM) i3-6100U CPU @ 2.3GHz 2.3GHz, 램은 4GB, 운영체제는 VirtualBox 5.1.26으로 Ubuntu 16.04를 설치하였다. 개발 도구는 python 2.7.1을 이용하여 구현하였다.

실험 데이터는 다음과 같이 일곱 종류를 사용하였다. 첫째, Dictionary는 옥스퍼드 워드 3000[10]에서 3,839 단어와 SCOWL(Spell Checker Oriented Word Lists)[11]에서 세개 파일(2+2+3 cmn.txt, 2+2+3 frq.txt, 2+2+3 lem.txt)을 통합하여 157,507 단어로 구성하였다. 둘째, bible은 4,451,368 bytes 영문으로 작성된 성경[12]이다. 셋째, world192는 2,408,281 bytes 영문으로 작성된 세계 나라 정보[13]이다. 넷째, AHFp1은 562,591 bytes 영문도서로 Mark Twain 저서 The Adventures of Huckleberry Finn[14]이다. 다섯째, IFC는 651,145 bytes 영문도서로 G. A. Henty 저서 In Freedom's Cause[15]이다. 여섯째, TFR1은 933,703 bytes의 영문도서로 Hippolyte A. Taine 저서 The French Revolution Volume 1[16]이다. 일곱째, ToTE1은 697,523 bytes의 영문도서로 JAMES HUTTON 저서 Theory of the Earth[17]이다. 또한 실험 알고리즘들의 파일 크기에 따른 성능 변화를 측정하기 위해 샘플 파일 두 개를 추가하였다. 하나는 bible, world192, AHFp1, IFC, TFR1 파일을 병합하여 allTxt1를 만들고, 다른 하나는 allTxt1 파일에 지구의 이론(ToTE1) 파일을 병합하여 allTxt2를 만들었다.

4.2 실험 결과

실험 샘플 데이터 AHFp1, bible, world192, Dictionary에서 문자의 출현 빈도를 검사하고, 검사 결과 일부를 표 1과 표 2에 나타냈다. 빈도수가 없는 셀은 해당 문자가 사용되지 않았다는 것을 의미한다. PreHuffman은 사전을 만들 때 문자를 추가하고 빈도수를 1로 넣는다. 본 실험에서 PreHuffman 사전 생성 알고리즘은 실험 데이터인 AHFp1, bible, world192을 이용하여 공용 사전을 만들어 사용했다.

표 1. 문자 출현 비율 현황 - 대문자
Table 1. Character appearance ratio table - upper
(unit : %)

upper	AHFp1	bible	world192	Dictionary
A	5.40	15.28	9.72	10.26
B	3.78	4.01	3.06	4.25
C	1.35	1.44	8.39	7.60
...	...	...	...	...
Y	2.50	0.48	0.71	1.13
Z	0.04	0.77	0.53	0.57
total	100	100	100	100

표 2. 문자 출현 비율 현황 - 소문자
Table 2. Character appearance ratio table - lower
(unit : %)

lower	AHFp1	bible	world192	Dictionary
a	8.54	8.26	10.04	7.79
b	1.62	1.41	1.52	1.99
c	1.84	1.71	3.71	4.21
...	...	...	...	...
y	2.36	1.86	1.57	1.81
z	0.04	0.07	0.21	0.63
total	100	100	100	100

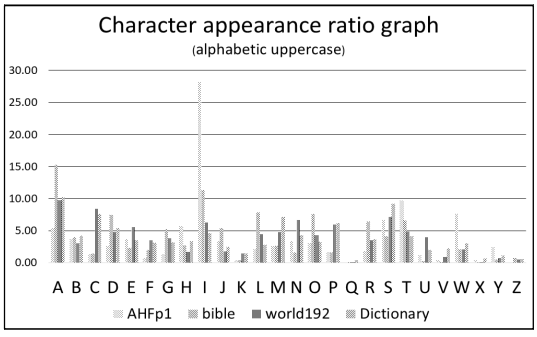

그림 4. 문자 출현 비율 그래프(알파벳 대문자)
Fig. 4. Character appearance ratio graph (alphabetic uppercase)

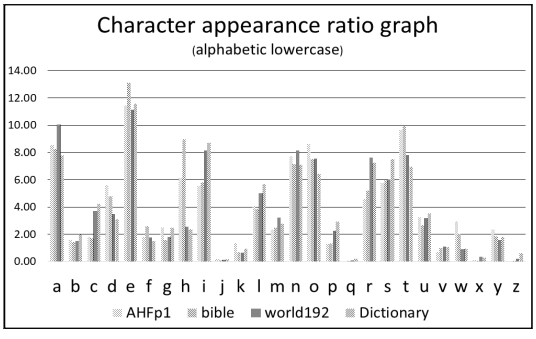

그림 5. 문자 출현 비율 그래프(알파벳 소문자)
Fig. 5. Character appearance ratio graph (alphabetic lowercase)

그림 4와 그림 5는 각 샘플 데이터에서 문자의 분포 현황과 데이터의 패턴을 나타낸다. 그래프에서 AHFp1은 알파벳 개수가 적기 때문에 그래프 상으로 구분되기 어려우나, bible, world192, Dictionary에서 문자의 출현 분포율이 유사한 패턴을 보인다.

본 논문에서는 유사한 패턴 데이터를 대상으로 허프만 코딩, 동적 허프만 코딩, 그리고 제안하는 PreHuffman 알고리즘을 이용한 압축을 실험했다.

허프만 코딩은 전체 처리시간의 단축보다는 압축률을 최대한 높이기 위해 최적화된 사전을 매번 생성하여 부호화했다. 그러나 PreHuffman은 공용 사전을 만들어 이용함으로써 다른 방법과 비슷하게 압축률을 최대한 유지하면서, 실시간으로 입력 데이터를 압축하고, 처리 시간을 단축하였다. 이를 증명하기 위해 샘플 데이터를 이용하여 원본 파일과 압축 파일의 크기 변화, 압축률, 수행 시간, 시간 복잡도를 분석하였다. 결과에서는 실험 데이터의 파일 크기가 출현 문자 수와 비례하고, 압축률은 출현 문자의 빈도수와 공용 사전의 부호 길이에 비례했다.

허프만 코딩과 동적 허프만 코딩이 PreHuffman 알고리즘보다 1~2% 만큼 압축률이 좋았다. 그러나 알고리즘의 수행 시간은 PreHuffman 알고리즘이 허프만 코딩보다 1.6~2.9배 빠르고, 동적 허프만 코딩보다 38.7~82.6배 빨랐다. 표 3은 데이터 압축을 비교 실험한 결과이다.

V. 결 론

본 논문에서는 문자의 출현 빈도가 유사한 데이터를 압축했을 때 다른 압축 방법과 비슷하게 압축률을 최대한 유지하면서, 압축 처리 시간을 단축하는 방법인 PreHuffman 알고리즘을 연구하였다.

표 3. 데이터 압축 실험 비교표 - 파일 크기, 압축률, 처리시간
Table 3. Data compression experimental comparison table - file size, compression rate, processing time

File name	Source file size	Compression file size (bytes)			Compression ratio (%)			Processing time (sec)		
		Huffman	Dynamic Huffman	Pre Huffman	Huffman	Dynamic Huffman	Pre Huffman	Huffman	Dynamic Huffman	Pre Huffman
① bible	4,451,368	2,580,095	2,580,003	2,580,727	42.04	42.04	42.02	0.7547	27.2532	0.3826
② world192	2,408,281	1,506,727	1,505,733	1,625,132	37.44	37.48	32.52	0.4191	15.3506	0.2537
③ AHFp1	562,591	321,702	320,279	339,302	42.82	43.07	39.69	0.1240	3.3893	0.0535
④ IFC	651,145	363,761	362,274	376,750	44.14	44.36	42.14	0.1332	3.8381	0.0811
⑤ TFR1	933,703	535,876	534,072	535,876	42.61	42.8	42.61	0.2815	5.9975	0.1546
⑥ ToTE1	697,523	383,126	381,587	400,095	45.07	45.29	42.64	0.1867	4.1475	0.0629
⑦ allTxt1	9,007,069	5,384,410	5,323,599	5,468,061	40.22	40.9	39.29	1.4891	56.2927	0.6961
⑧ allTxt2	9,704,592	5,780,061	5,706,099	5,865,218	40.44	41.2	39.56	1.5718	67.3750	0.8156

실험 결과 PreHuffman 알고리즘이 다른 압축 방법에 비해 빠르게 처리를 할 수 있었다. 특히 PreHuffman 알고리즘과 동적 허프만 코딩은 시간 복잡도가 O(n log n)로 같으나, 수행 시간에 많은 차이를 보인다. 동적 허프만 알고리즘에서는 데이터의 길이만큼 한 바이트씩 읽어 가면서 빈도수를 계산하고, 허프만 트리의 갱신 여부 검사, 허프만 트리 갱신, 가중치 부여, 문자 부호화 등 연속적인 작업이 반복 수행되기 때문에 많은 시간이 소요된다.

따라서 본 논문에서 연구한 PreHuffman 알고리즘은 유사 패턴을 가지고 있는 데이터 압축에서 효과적으로 사용된다.

References

[1] Young-Hye Min and Jai-Hoon Kim, "Optimal Data Compression Algorithm Selection for Efficient Data Store and Transfer", Journal of KIISE : Computer Systems and Theory, Vol. 38, No 2, pp. 107-115, Apr. 2011.

[2] Woosik Lee, Daeun Yu, and Namgi Kim, "Design and Implementation of an In-Memory File System Cache with Selective Compression", The Journal of Korean Institute of Information Technology, Vol. 13, No. 7, pp. 67-74, Jul. 2015.

[3] Jung-Hoon Kim, "Manchester coding of compressed binary clusters for reducing IoT healthcare device's digital data transfer time", Journal of KIIECT, Vol. 8, No. 6, pp. 460-469, Dec. 2015.

[4] Huffman and David, "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the IRE, Vol. 40, No. 9, pp. 1098-1101, Sep. 1952.

[5] Mashhur Sattorov, Heau-Jo Kang, "A Study on Huffman Coding to Improve Performance of OFDM", Journal of Korean Institute of Information Technology, Vol. 7, No. 6, pp. 115-119, Dec. 2009.

[6] Hyeran Kim, Yeojin Jung, Changhun Yim, Hyesook Lim, "A Balanced Binary Search Tree for Huffman Decoding", Journal of KICS, Vol. 30, No 5C, pp. 382-390, May 2005.

[7] Jae-Jeong Hwang, Kyong-Sig Jin, Hak-Su Han, Joon-Young Choi, Jin-Whan Lee, "Huffman Code Design and PSIP Structure of Hangul Data for Digital Broadcasting", Journal of Broadcast Engineering, Vol. 6, No. 1, pp. 98-107, Jun. 2001.

[8] Donald E. Knuth, "Dynamic Huffman Coding", Journal of Algorithms Vol. 6, pp. 163-180, Jun. 1985.

[9] JiHwan Park, "Multimedia Compression Technology", Korea Multimedia Society, Vol. 1, No. 1, pp. 41-51, Dec. 1998.

[10] Oxford University Press, (2017, Dec, 11) Oxford3000 Word List [Online], Available : https://github.com/OliverCollins/Oxford-3000-Word-List/blob/master/ Oxford 3000 Word List.txt. [accessed May. 08, 2018]

[11] SCOWL, (2018, Apr, 16) 12Dicts Package [Online], Available : http://wordlist. aspell.net. [accessed May. 06, 2018]

[12] Edward Gan, (2012, Dec, 10) Bible [Online], Available : https://raw.githubusercontent.com/mxw/grmr/master/src/finaltests/bible.txt. [accessed May. 08, 2018]

[13] David Turner, (2015, Jul, 18) The Project Gutenberg Edition of THE WORLD FACTBOOK 1992 [Online], Available : https://raw.githubusercontent.com/airlift/aircompressor/master/testdata/large/w orld192.txt. [accessed May. 08, 2018]

[14] Mark Twain, (1884, Dec, 4) The Adventures of Huckleberry Finn, Complete [Online], Available : http://www.fullbooks.com/The-Adventures-of-Huckleberry- Finn-Complete.html. [accessed May. 10, 2018]

[15] G. A. Henty, (1884) In Freedom's Cause [Online], Available : http://www.fullbooks.com/In-Freedom-s-Cause.html. [Accessed May. 10, 2018]

[16] Hippolyte A. Taine, (1878) The French

Revolution Volume 1 [Online], Available : http://www.fullbooks.com/The-French-Revolution-Volume-1-The-Origin s.html. [accessed May. 12, 2018]

[17] James Hutton, M.D. & F.R.S.E., (1795) Theory of the Earth Volume 1 [On line], Available : http://www.fullbooks.com/Theory-of-the-Earth-Volume-1-of-4-. html. [acessed May 12, 2018]

저자소개

맹 일 희 (Il Hee Maeng)

2018년 8월 : 한국방송통신대학교 정보과학과(이학석사)
2009년 9월 : 건영기술 솔루션 사업부 부장
관심분야 : 분산 시스템, 클라우드 컴퓨팅, 인공지능, e-Learning, 실감 미디어

박 지 수 (JiSu Park)

2013년 8월 : 고려대학교 컴퓨터교육과(이학박사)
2015년 12월 ~ 2018년 2월 : 충남대학교 초빙교수
2018년 3월 ~ 현재 : 경기대학교 융합교양대학 교양학부 조교수
관심분야 : 분산 시스템, 모바일 클라우드 컴퓨팅, e-Learning

손 진 곤 (Jin Gon Shon)

1991년 월 : 고려대학교 전산학전공(이학박사)
1991년 ~현재 : 한국방송통신대학교 컴퓨터과학과 교수
1997년~1998년 : State University of New York (Stony Brook) Visiting Professor
2000년 ~ 현재 : ISO/IEC JTC1/SC36 Korea Delegate
2010년 1월 ~ 2010년 12월 : 한국정보처리학회 부회장
2009년 ~ 현재 : 이러닝학회 부회장
관심분야 : 컴퓨터통신망, 분산시스템, 그리드 컴퓨팅, e-Learning, 정보기술 표준화