알고리즘

Huffman Code 압축 실험

소프트웨어학부

2019203021

이승헌

1-1. 인코딩 체계의 표현 방법

필자가 헤더에 사용한 인코딩 체계는 이하의 예를 들어서 설명하겠습니다.

CABBAB라는 문자열에 있는 문자들을 Huffman Code알고리즘을 이용하면 B:1, A:01, C:00으로 압축할 수 있습니다.

CABBAB를 위 결과를 이용해 압축하면 000111011, 총 9비트로 압축할 수 있습니다.

그러나, 파일은 바이트 단위로 읽고 쓰기 때문에 위 과정에서 두 가지 문제점이 발생합니다.

첫 번째로는 A와 B의 Huffman code는 1바이트로 확장하고 본다면 같은 수라는 점입니다. 분명 표현하는 데 쓰이는 비트 수는 다름에도 불구하고 바이트로 확장되면 같은 수가 되어버려 구분하는 것이 불가능합니다. 헤더를 만들 때 한 문자를 구분하기 위해 문자와 그에 대응하는 Huffman code만을 저장한다면 결국 문자끼리 구분이 불가능하다는 의미입니다.

두 번째는 문자를 Huffman Code와 일대일 매칭시켜 저장시킨다고 생각했을 때 발생합니다.

즉, 인간이 하는 일을 컴퓨터에게 그대로 시킨다면 이하와 같이 저장됩니다.

00000000 00000001 00000001 00000001 00000001 00000001

결국 6바이트 그대로가 되며, 헤더 파일이 추가되어 용량은 원본보다 더 늘어난다는 문제점이 있었습니다. 비트 단위로 파일에 쓰는 방법을 생각해야 한다는 의미입니다.

따라서, 저는 두 가지 문제점을 이하와 같이 해결했습니다.

첫 번째 문제는 헤더를 만들 때 한 문자를 구분하기 위해 문자 1바이트와 Left Shift를 통해 1바이트로 확장한 그에 대응되는 Huffman Code, 그리고 유효 비트 수를 저장하는 방법을 사용하여 해결하였습니다.

CABBAB 예제로 돌아가서, 압축 파일에 이 문자열에 대한 헤더를 만든다면 다음과 같습니다.

A2(0100000)B1(10000000)C2(00000000)

이 방법을 사용하면 문자 하나당 3바이트를 쓰게 됩니다. 이 방법의 장점은 유효 비트 수를 저장함으로써 압축 해제 시 유효 비트 수 까지만 남기는 Right Shift를 통해 A와 B의 정확한 비트열을 알 수 있다는 점입니다.

1-2. 인코딩 체계의 표현 방법

두 번째 문제는 Shift와 논리 연산을 활용하여 1바이트에 8비트를 저장할 수 있게 하는 것입니다. 동작 방식을 CABBAB 예제를 통해 설명하겠습니다.

1. 1바이트 단위 저장을 위해 1바이트 크기의 Buffer를 하나 만들어 초기화합니다.

2. C의 Huffman Code 00에 대해 (00>>1) AND 1와 같이 연산하여 최상위 비트를 추출해 bit에 저장합니다.

3. Buffer에 대해 Right Shift를 1회 하고, 그 다음 Buffer OR bit 연산을 거쳐서 Buffer의 맨 마지막 비트에 추출한 bit를 저장합니다.

4. C의 Huffman Code 00에 대해 (00>>0) AND 1과 같이 연산하여 그 다음 비트를 추출해 bit에 저장합니다.

5. 3을 수행합니다. 이렇게 하여 현재 Buffer에는 최하위 2개 비트에 00이 저장 되어있습니다.

즉, 2부터 5까지의 과정은 Huffman Code를 한 비트씩 쪼개어 Buffer에 집어넣는 작업입니다.

6. 2와 3을 Buffer크기가 1바이트가 될 때까지 수행합니다.

7. Buffer가 1바이트가 되었다면 압축 파일에 Buffer를 저장하고 버퍼를 초기화합니다.

8. 문자열의 끝까지 2, 3, 6, 7을 반복합니다.

9. 8까지 수행했을 때 버퍼의 크기가 0이 아니라면 버퍼의 크기를 변수에 저장해둔 다음, 버퍼를1바이트 크기가 될 때까지 Left Shift하고 압축 파일에 저장하고 저장해둔 기존의 버퍼 크기를 압축 파일의 끝에 저장해둡니다. (실제로는 압축 파일의 시작점에 저장했습니다.)

위 과정을 거쳤을 때 압축 파일에 저장되는 것은 다음과 같습니다.

A2(0100000)B1(10000000)C2(00000000)

0001110110000000

2. 실험 결과

i) 헤더 파일이 없는 Huffman Code알고리즘의 압축률

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 원본 크기 | 1,019B | 10,131B | 101,425B | 1,014,623B | 2,029,079B |
| 압축 크기 | 762B | 7,676B | 76,994B | 770,544B | 1,541,198B |
| 압축률 | 25.22% | 24.23% | 24.08% | 24.05% | 24.04% |

ii) Huffman Code 알고리즘의 최종 압축률

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 원본 크기 | 1,019B | 10,131B | 101,425B | 1,014,623B | 2,029,079B |
| 압축 크기 | 973B | 7,887B | 77,205B | 770,755B | 1,541,409B |
| 압축률 | 4.51% | 22.14% | 23.87% | 24.03% | 24.03% |

iii) 분석

I. 필자가 구현한 Huffman Code는 압축률이 24%에 수렴하는 것으로 보입니다.

II. 과제에서 제시된 조건에서 헤더 파일의 최대 크기는 211바이트인 것으로 보입니다.

III. 위 표를 분석한 결과, 1만 글자가 넘는 파일을 압축할 때 유의미한 압축을 할 수 있는 것으로 보입니다.