목차

1. Huffman 압축알고리즘 개요

허프만 코딩 알고리즘(Huffman coding algorithm)은 무손실 압축에 쓰이는 엔트로피 부호화의 일종으로, 데이터 문자의 등장 빈도에 따라서 다른 길이의 부호를 사용하는 알고리즘이다. 1952년 당시 박사과정 학생이던 데이비드 허프만이 《A Method for the Construction of Minimum Redundancy Codes란 제목의 논문으로 처음 발표했다.

이는 자료 압축의 가장 오래되고 기초적인 방법 중의 하나이며 최소 중복 코딩(minimum redundancy coding)에 기반한 알고리즘을 사용한다. 최소 중복 코딩이란 문자들이 자료 집합에서 얼마나 자주 발생하는지를 안다면 발생되는 문자들의 비율에 따라 자주 반복되는 문자들을 더 적은 비트로 부호화 하는 방법이다. 일반적인 경우 한 문자를 표현하기 위해 한 개의 바이트(ASCII코드)를 사용하나 허프만 코드는 문자 발생 비율에 따라 다른 크기의 비트로 표현한다.

2. Huffman 알고리즘 원리 및 설계

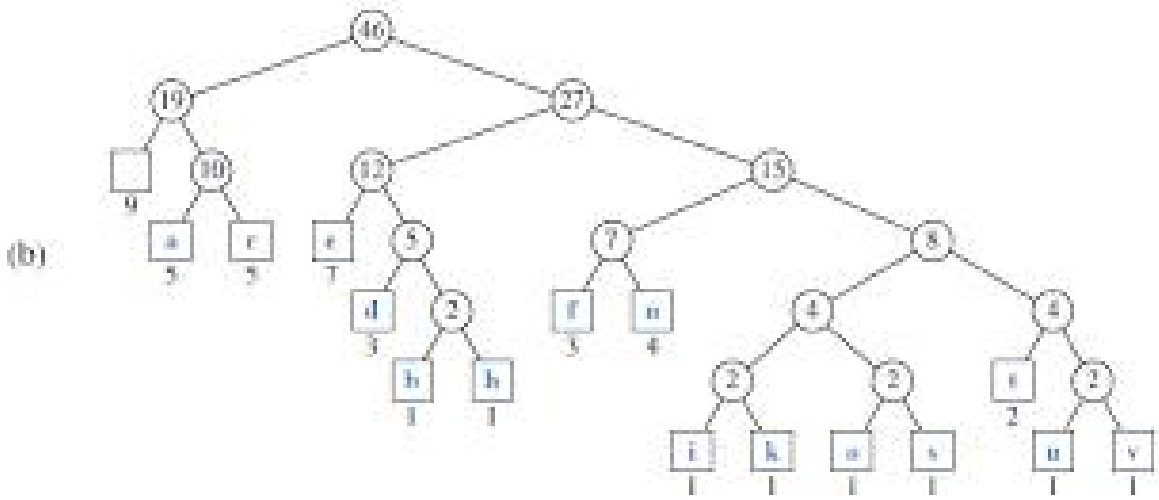
우선 엔트로피와 최소 중복의 원리를 이용해야한다. 이를 통한 엔트로피 부호화의 방법을 사용하면 문자의 빈도순으로 많으면 많을수록 짧은 코드(비트)를 부여하고, 적으면 적을수록 적은 수의 코드(비트)를 부여하는 방법을 사용한다.

이러한 원리를 적용하기 위한 허프만 코드는 모든 문자를 탐색을 해야 한다. 모든 문자를 탐색 시, 각 문자의 개수를 세어놓는다. 그 후에 문자의 개수대로 정렬을 시킨 후, 문자의 개수가 작은 수부터 많은 수로 트리를 생성해 나가는 방식을 취한다.

(a)

Character		a	b	d	e	f	h	i	k	n	o	r	s	t	u	v
Frequency	9	5	1	3	7	3	1	1	1	4	1	5	1	2	1	1

(b)



▲ 그림1. 허프만 알고리즘 중, 문자를 이진 트리에 넣어서 정렬하는 모습

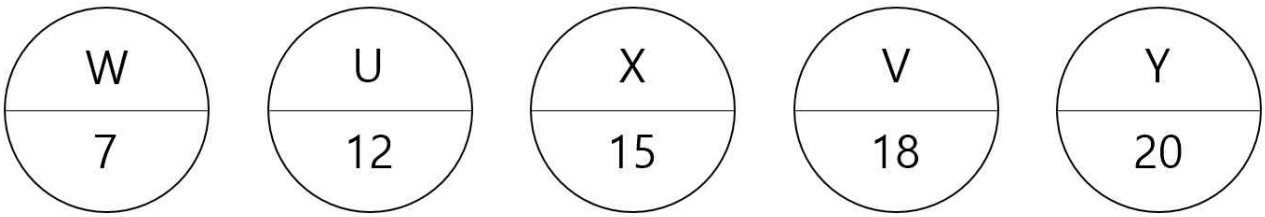
허프만 알고리즘의 예시를 들어보도록 하자. 아래와 같이 문자들이 주어졌다고 가정하자.

기호	확률
U	12/72
V	18/72
W	7/72
X	15/72
Y	20/72

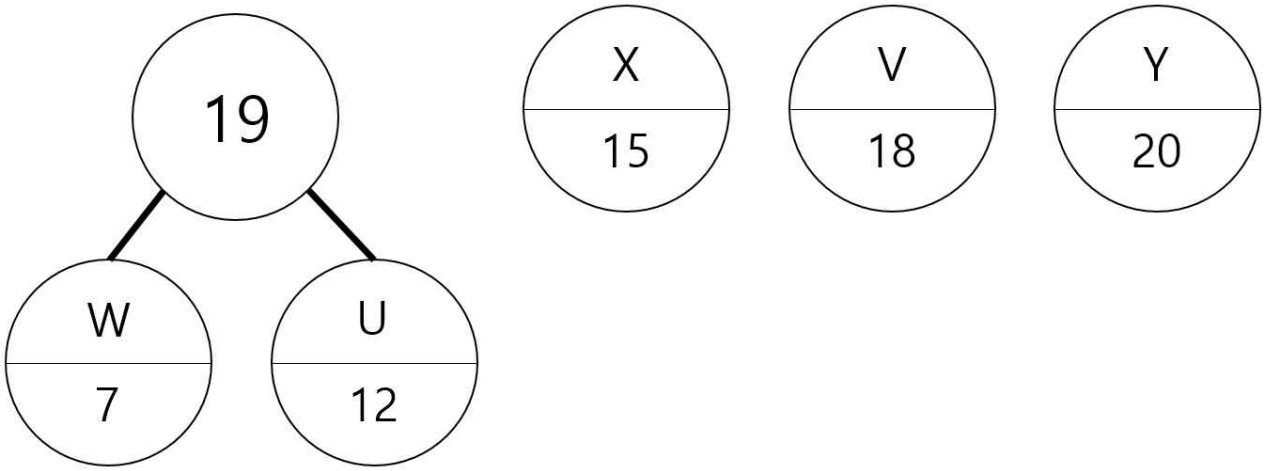
▲표1. 문자들을 탐색한 후, 모아놓은 표

이 표를 기준으로 허프만 트리를 만들자면, 우선 문자가 적은 것부터 많은 순서대로 정렬을 해야 한다. 즉, “W, Y, X, V, Y" 순서로 문자를 정렬해야한다. 그 후에는 문자가 작은 순서부터 허프만 트리를 구성해나가면 된다.

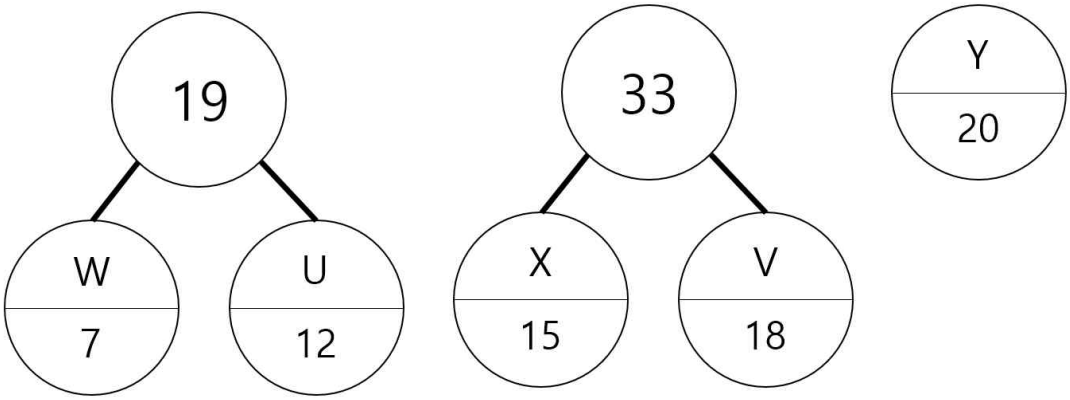
1. 문자가 적은 순서부터 정렬을 한다.



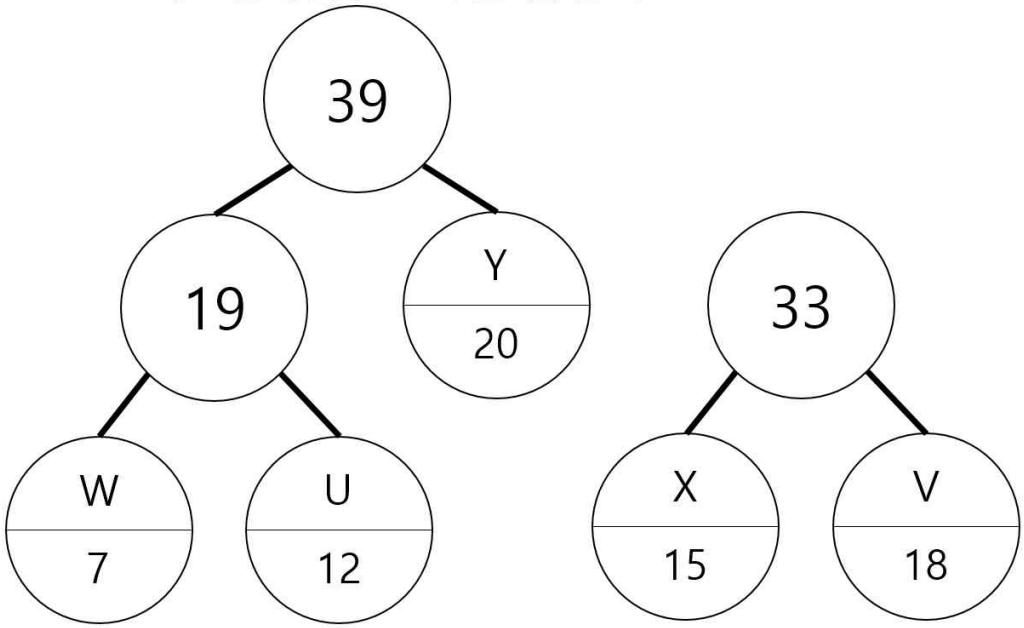
2. W와 U를 갖는 트리를 합친다.



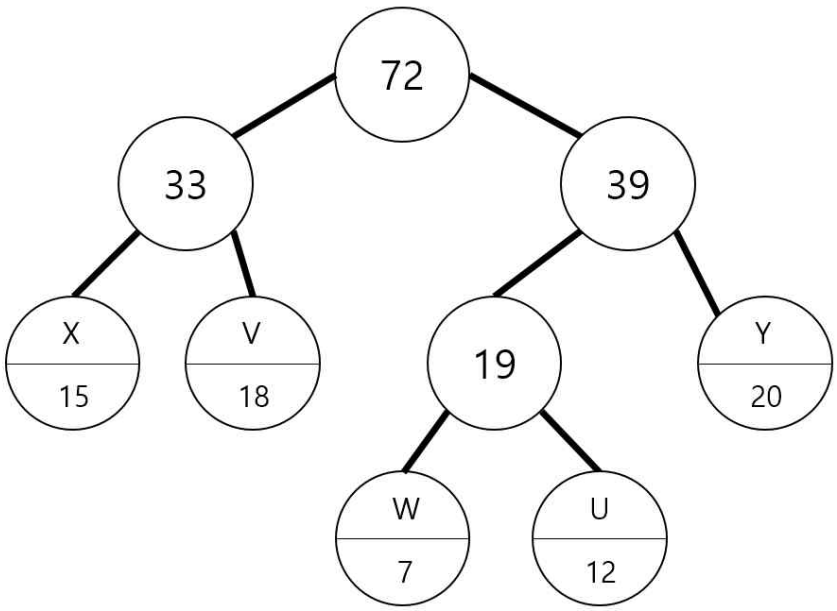
3. X와 V를 갖는 트리를 합친다.



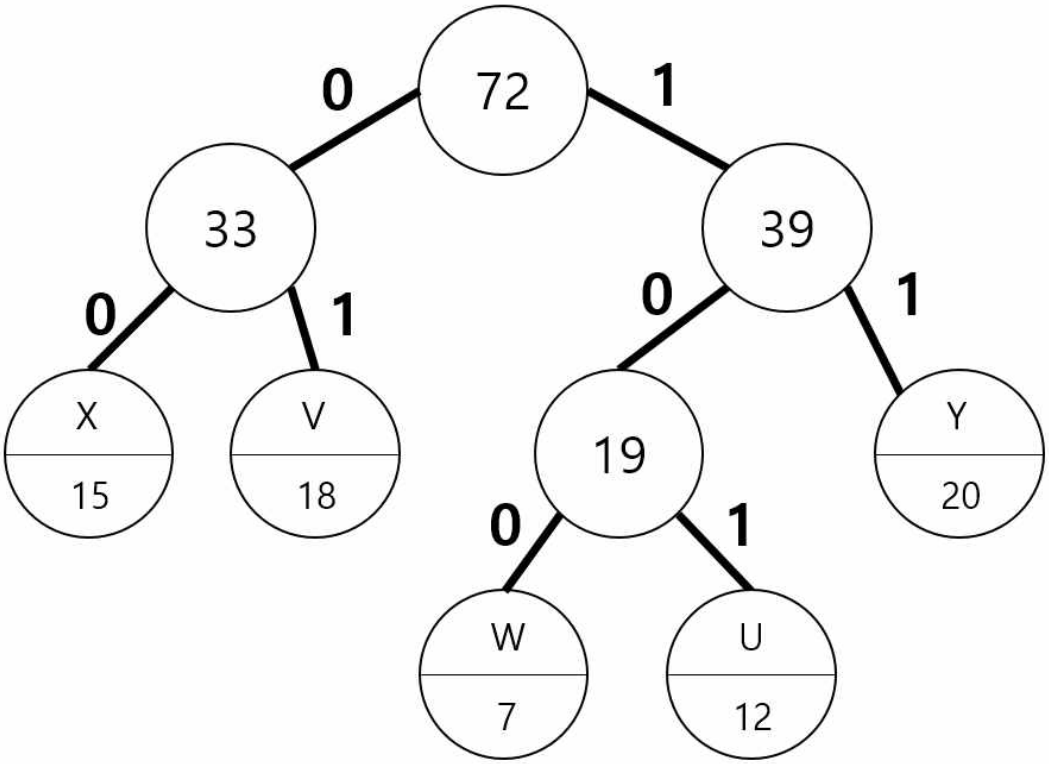
4. WU와 Y를 갖는 트리를 합친다.



5. WUY와 XV를 갖는 트리를 합친다.



6. 허프만 코드 생성



위의 6 과정을 거쳐서 허프만 트리로부터 허프만 코드를 생성할 수 있다. 허프만 코드의 해석방법은 Root트리(데이터가 72인곳부터)부터 트리를 내려가면서 이진수를 만들어가면 된다. 즉, X는 00, V는 01, W=100, U= 101, Y = 11이 된다. 기존의 문자 크기의 데이터가 8-bits라고 가정하면 압축률은 아래와 같이 계산할 수 있다.

- 원래의 데이터 크기 : 72*8 = 576-bits
- 압축후의 데이터 크기 : 12*3 + 18*2 + 7*3 + 15*2 + 20*2 = 163-bits
- 압축률(%) : (100 - (163/576))*100 = 71.7 %

즉, 압축률은 약 71.7%이 된다. 이렇게 비트를 기준으로 압축을 할 수 있고, 반대로 복호화는 위의 mapping 트리와 원래 문자의 비트들을 가지고 원래의 문자로 되돌릴 수도 있다. 이러한 원리를 가지는 것이 Huffman encoding algorithm이다.

설계는 위의 원래대로 먼저 “우선순위 큐(힙)”를 이용해서 데이터를 넣어준다. 물론 데이터 넣는 기준은 문자가 나온 개수를 기준으로 왼쪽으로 리프로 갈지, 오른쪽 리프로 갈지 선택하면 된다. 이렇게 힙을 완성했으면, 기존 테이블(설계를 위해서는 리스트를 사용하면 된다.)에 저장했던 데이터들을 모두 탐색하면서 그에 맞는 힙을 내려가본다. 그에 따라 비트가 정해지게 되고, 비트 수에 따라서 압축률이 달라지게 된다. 이를 기준으로 작성한 소스코드는 부록의 첨부파일로 올려두겠다.

3. Huffman coding algorithm 테스트 결과

C:₩Python35₩python.exe C:/Python/Huffman_encoding/main.py Input filename >> hello.txt ---------- Compress Result---------- Compressed file path: hello_compress.bin Compressed file size: 46822 bytes Compressed ratio: 65.110 % ---------- Decompress Result---------- Decompressed file path: hello_compress_restore.txt Decompressed file size: 134198 bytes Restoration ratio : 99.999 %	- test case 1 : hello.txt 파일 “hello ”만 연속으로 넣어놓은 데이터다. 이 경우 데이터의 압축률은 65.110%가 되었다.
C:₩Python35₩python.exe C:/Python/Huffman_encoding/main.py Input filename >> news.txt ---------- Compress Result---------- Compressed file path: news_compress.bin Compressed file size: 1012 bytes Compressed ratio: 46.027 % ---------- Decompress Result---------- Decompressed file path: news_compress_restore.txt Decompressed file size: 1875 bytes Restoration ratio : 100.000 %	- test case 2 : news.txt 파일 실제 이슈가 되는 뉴스의 일부 내용을 넣은 데이터다. 즉, 문자가 비교적 균등하게 섞인 경우다. 이 경우는 46.027%가 되었다.
C:₩Python35₩python.exe C:/Python/Huffman_encoding/main.py Input filename >> sample.txt ---------- Compress Result---------- Compressed file path: sample_compress.bin Compressed file size: 180782 bytes Compressed ratio: 45.019 % ---------- Decompress Result---------- Decompressed file path: sample_compress_restore.txt Decompressed file size: 328809 bytes Restoration ratio : 100.000 %	- test case 3 : sample.txt 파일 긴 문장이 반복적으로 나오는 케이스다. 이 역시 우와 근소한 차이를 보이는데, 데이터의 크기보단 문자의 다양성에 영향이 있다고 생각할 수 있다.
C:₩Python35₩python.exe C:/Python/Huffman_encoding/main.py Input filename >> test.txt ---------- Compress Result---------- Compressed file path: test_compress.bin Compressed file size: 15534 bytes Compressed ratio: 40.293 % ---------- Decompress Result---------- Decompressed file path: test_compress_restore.txt Decompressed file size: 26017 bytes Restoration ratio : 100.000 %	- test case 4 : test.txt 파일 C 소스의 일부를 길게 늘여쓴 것이다. 비교적 문자가 다양했기 때문에 압축률이 40% 극초반대까지 떨어진 경향을 보인다.
C:₩Python35₩python.exe C:/Python/Huffman_encoding/main.py Input filename >> ABC.txt ---------- Compress Result---------- Compressed file path: ABC_compress.bin Compressed file size: 240623 bytes Compressed ratio: 75.040 % ---------- Decompress Result---------- Decompressed file path: ABC_compress_restore.txt Decompressed file size: 964014 bytes Restoration ratio : 100.000 %	- test case 5 : ABC.txt 파일 "abc"문자만 반복적으로 나오는 데이터다. 이 경우가 바로 다양하지 않은 문자로 테스트 했다고 볼 수 있다.

위의 테스트 케이스들을 미뤄 보았을 때, 일반적으로 허프만 코딩 알고리즘의 압축률은 45%쯤 나왔다. 위의 테스트 케이스 이외에 다양한 테스트 케이스들로도 테스트 해보았으나, 거의 45%정도의 압축률이 나왔다. 하지만 위의 테스트 케이스들을 적용하였을 때, 문자의 수보단 문자가 얼마나 다양하지 않게 나오냐에 따라서 압축률이 높아지는 모습을 알 수 있었다. 즉, 위의 경우들을 따져보았을 때, 실제 허프만 코딩 알고리즘의 **압축률은 나오는 문자가 단순하지 않을수록 높은 압축률을 보인다**고 결론 내릴 수 있었다.

4. 개선된 알고리즘들 소개

허프만 알고리즘은 빠른 압축과 복원 시간을 갖는다는게 장점이기도 하다. 오래 걸린다면 빈도를 측정하기 위해 모든 데이터를 탐색한다는 점에서 데이터가 크면 클수록 당연히 실행시간이 늘어나게 된다. 하지만 빠른 압축과 복원 시간을 갖는 허프만 알고리즘이 좋다고 할 수 없으며, 이런 허프만 알고리즘을 개선하고자 나온 알고리즘들도 있다. 이러한 알고리즘들을 소개하고자 한다.

① LZH

LHA는 프리웨어 압축 프로그램 및 파일 형식의 하나이다. 요시자키 하루야스(일본어: 吉崎栄泰)가 1988년에 최초로 개발하여 LHarc라는 이름을 붙였다. 세계적으로 많이 사용되고 있지는 않지만, 일본에서는 아직도 널리 사용되고 있다. 이드 소프트웨어사는 둠 등의 초창기 게임 배포를 위해 LHA 형식을 사용한 바 있다. 또한 LHA는 아미가 컴퓨터용으로 포팅이 되기도 했다. 마이크로소프트에서는 윈도 XP의 압축 폴더 기능에서 LHA 형식의 압축 파일을 지원하는 애드온을 일본에서만 공개했다

② 7z

7z은 각기 다른 데이터 압축, 암호화, 전처리 알고리즘을 지원하는 압축 파일 포맷이다. 7z 포맷은 본래 7-zip 압축 프로그램이 제공한 데에서 비롯하였다. 7-zip 프로그램은 GNU 약소 일반 공중 라이선스 조항에 의거하여 사용할 수 있다. 이러한 7z 알고리즘은 다수의 폴더 및 파일을 담을 수 있다. 그리고 256비트 AES 계산을 따라 압축함으로 압축 파일 안의 파일들의 이름까지 암호화할 수 있다. 하지만 AES 특징상 1비트만 손실을 입어도 Avalanche Effect(쇄도효과)가 발생되어 쓰지 못하는 데이터가 될 수 있다는 단점을 안고 있다.

③ xz

xz 는 무손실 데이터 압축 프로그램 및 LZMA2 압축 알고리즘 파일 형식이다. XZ는 7-Zip 프로그램의 축소된 버전으로 간주 할 수 있다. XZ는 GNU coreutils 프로젝트, Debian, openSUSE, ,Fedora, Arch Linux, Slackware, FreeBSD, Gentoo, GNOME, 과 TeX Live 에서 패키지 압축으로 유명하다. 이 뿐만 아니라 리눅스 커널로 컴파일 된 파일을 압축하는 기능도 있다. 2013년 2월, 리눅스 커널 메인테이너는 그들의 2014년도부터 bzip2 대신 XZ를 압축 도구로 발표했다

④ tar

압축은 전혀 하지 않으며, 단순히 다수의 파일을 하나로 묶을 때 사용한다. 과거 테이프를 기억매체로 사용하던 시절 데이터 관리를 보다 효율적으로 하기 위해 만들어졌다. 파일을 하나로 묶는 역할만 하는지라 처리 속도가 빠르다. 유닉스 계열 운영체제(특히 리눅스)에서 주로 쓰인다.

⑤ gz

gzip은 파일 압축에 쓰이는 응용 소프트웨어이다. gzip은 GNU zip의 준말이며, 초기 유닉스 시스템에 쓰이던 압축 프로그램을 대체하기 위한 자유 소프트웨어이다. gzip은 Jean-loup Gailly와 마크 애들러가 만들었다. gzip은 ZIP (파일 포맷)과 같이 DEFLATE 알고리즘을 따르지만, 여러 파일을 하나의 파일로 압축하는 옵션이 없다는 점에서 차이가 난다. 여러 파일 또는 디렉터리를 하나의 파일로 압축하기 위해서 gzip은 보통 Tar (파일 포맷)와 같이 사용되는 것이 일반적이다. .tar.gz 로 압축된 파일의 경우 zip과 압축 알고리즘은 같지만 더 용량이 작다.

⑥ zip

ZIP 파일은 하나 혹은 여러 개의 파일들을 그 크기를 줄여 압축하고 하나로 묶어 저장한다. ZIP 파일 형식에서는 다양한 종류의 압축 알고리즘의 사용이 가능하나, 2009년 현재 Deflate 알고리즘만이 가장 많이 사용되고 지원되는 압축 알고리즘이다.

	Huffman	LZH	7z	xz	tar	gz	zip
hello.txt	65.110%	99.995%	99.997%	99.998%	-1.485%	99.995%	99.994%
news.txt	46.027%	49.333%	39.989%	44.533%	-91.2%	50.133%	43.093%
sample.txt	45.019%	99.000%	99.594%	99.622%	-0.590%	99.002%	98.961%
test.txt	40.293%	91.682%	91.805%	92.143%	-6.268%	91.632%	91.125%
ABC.txt	75.040%	99.666%	99.957%	99.967%	-1.678%	99.578%	99.705%

위에서 6가지의 알고리즘에 대해서 설명을 하였고, 위의 알고리즘들의 성능을 직접 테스트를 진행하였고, 위와 같은 결과가 나왔다. 여기서 Huffman 압축 알고리즘을 사용하는 것보단 LZH, 7z, xz, gz, zip 같은 압축 알고리즘을 사용하는 것이 훨씬 효율적이라는 것을 확인 할 수 있었다. (tar는 파일을 묶는 것이므로, 용량이 늘어난다.)

부록1. 참고파일 - Huffman 압축알고리즘 구현 코드

1) Python 소스코드 부분

- util/HeapNode.py - heap에 들어갈 노드를 정의한 클래스
- compress.py - Huffman 압축
- decompress.py - Huffman 복호화
- main.py - 위의 압축/복호화 알고리즘을 실행할 모듈

2) Test data

- 각 폴더 내에 실제 압축파일과 같이 들어있음.
- hello - hello.txt 같은 방식으로 묶여있음.
- “-compress.bin"같은 방식의 파일은 실제 허프만 압축 알고리즘을 사용한 파일

부록2. 참고자료

- 7z 알고리즘 : https://ko.wikipedia.org/wiki/7z
- LZH 알고리즘 : https://ko.wikipedia.org/wiki/LHA (LZH에서 이름이 바뀐 것)
- xz 알고리즘 : https://ko.wikipedia.org/wiki/Xz
- tar 압축 : https://www.gnu.org/software/tar/
- gz 알고리즘 : https://ko.wikipedia.org/wiki/Gzip
- zip 압축 : https://ko.wikipedia.org/wiki/ZIP_(%ED%8C%8C%EC%9D%BC_%ED%8F%AC%EB%A7%B7)

- Huffman 압축 알고리즘 : http://artlib.tistory.com/1
- Huffman encoding algorithm : https://en.wikipedia.org/wiki/Huffman_coding