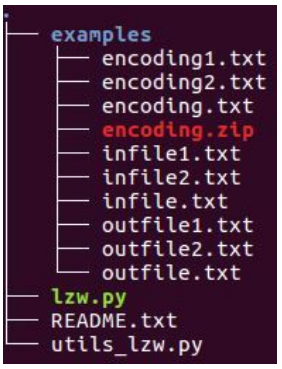
고급계산이론 HW2 보고서

2017-34165 김성국

개요

1) 프로젝트 구성

본 프로젝트의 구성은 다음과 같다.

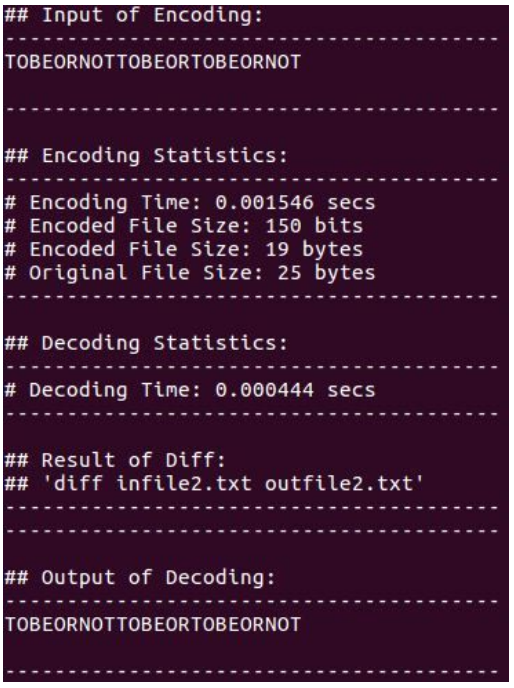


소스파일 lzw.py(제출 파일에서는 hw2.py)에는 문자열 데이터를 압축하고 압축해제하는 LZW 알고리즘을 구현하였다. 여기에 dictionary 자료구조가 사용되는데, 이를 trie로 구현하는 utils_lzw.py 파일을 포함한다. 모두 python3으로 구현하였다.

2) 출력 정보

옵션 -e를 주면 다음과 같은 출력이 나온다. 다음의 구성으로 맞추었다.

- Input of Encoding : 입력 데이터
- Encoding Statistics : 인코딩 시간, 압축 파일 사이즈, 원래 파일 사이즈
- Decoding Statistics : 디코딩 시간
- Result of Diff : infile과 outfile간의 diff 실행결과
- Output of Decoding : 디코딩 결과



LZW

1) LZW 구현

알려진 LZW 알고리즘을 그대로 구현하였다. 알고리즘 자체는 단순하여 구현이 크게 어렵지 않았다. 주어진대로 Trie 자료구조를 사용하여 dictionary를 구현하였다.
여기서 LZW 알고리즘은 phrase index 리스트를 대상으로 압축, 해제를 하도록 하였다. 즉, 이들을 비트화 시키는 연산은 bitfy(), unbitfy()로 따로 분리하여 구현하였다. 이 때 bitstring 모듈이 매우 유용하였다. 이는 별도로 설치가 필요하다.

2) 좋은 압축률을 위한 방법

압축률을 높이기 위하여, phase index를 비트화 하는데 dictionary 사이즈에 맞는 숫자만큼의 비트만을 사용하였다. 즉, math.ceil(math.log(dicSize, 2)) 만큼. 이렇게하여 최대한 비트 숫자를 줄이도록 하였다.

3) LZW 실행 결과

주어진 infile.txt에 대해서 결과는 다음과 같다.

- 1,521,996 바이트 → 713,644 바이트
- 48% 로 축소
- 인코딩 시간 : 15~20초, 디코딩 시간 : 3~5초

과제의 목표인 950kB 이하(714kB)를 달성하였다.

4) linux zip과의 비교

상용 프로그램과의 비교를 위하여 linux에서 기본으로 제공하는 zip 프로그램과 비교하였다.

- 원래 파일 : 1,521,996 바이트
- 나의 구현 : 713,644 바이트 (원래의 48%)
- linux zip : 644,480 바이트 (원래의 42%)
- linux zip 인코딩 시간 : 0.11초, 디코딩 시간 : 0.02초

위에서 보듯 linux zip에서 6%의 추가 이득이 있었다. 하지만 시간의 경우 많은 차이가 난다.

결론

주어진 프로그램의 크기를 반 이상 줄이는 압축 프로그램을 만들었다.
압축률은 6%로 정도 차이가 있었는데, 본 알고리즘은 알파벳에 제한했기 때문에 실제로는 더 차이가 있지 않을까 생각한다. 또한, LZW에서 variable-width encoding 등의 추가적인 최적화가 없었기에 격차가 나는 것이 아닌가 하는 생각이 들었다.
압축에 대한 benchmark들도 존재하는 등, 이 분야도 한가지 전지전능한 알고리즘이 존재하는 것이 아니라, 타겟하는 파일에 따라서 선호하는 알고리즘이 다르겠구나 하는 생각이 들었다.
실행 시간의 경우에도 구현을 python으로 하였고, 자료구조도 구현에 편의를 맞췄기 때문에 저 정도의 차이가 나는 것 같다.
전체적으로 많은 engineering이 필요한 분야라는 것을 느꼈다.