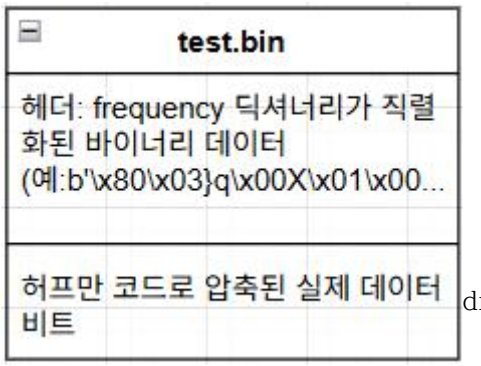
-파이썬 파일처리 기말과제

-풀이과정
저는 저에게 익숙한 인코딩 방식인 '허프만 인코딩'을 응용하여 데이터를 압축시키는 코드를 구현해 보았습니다. 먼저 기존의 허프만 인코딩은 문맥이나 순서를 고려하지 않고, 문자가 등장하는 빈도수만 파악하여 코드를 부여하기 때문에 반복 패턴에는 취약하다는 특징을 가지고 있습니다. 이러한 취약점을 해결하기 위해 반복 패턴을 뭉쳐서 처리할 수 있는 알고리즘을 공부하다 BWT(Burrows Wheeler Transform)알고리즘에 대해 알게 되었습니다.
이 알고리즘은 문자열을 재정렬하여 비슷한 문자끼리 모으는 변환 기법으로 세 번의(회전 행렬 생성 -> 사전순 정렬 -> 정렬된 행의 마지막 문자를 순서대로 추출)단계를 거쳐 데이터의 순서를 바꿔 비슷한 글자끼리 뭉치도록 해주는 알고리즘입니다.
이 알고리즘을 도입하여 구현해보니 또 이 알고리즘은 데이터를 뭉치게만 하고 압축에는 도움이 되지 않는다는 것을 알게 되었습니다. 그래서 관련 자료를 찾아보다 MTF(Move TO Front)라는 알고리즘을 찾게 되었습니다.
이 알고리즘은 압축 대상이 되는 데이터에 포함될 수 있는 초기 값들을 리스트에 저장 -> 압축 데이터를 하나씩 읽기 -> 그 값이 몇 번째 인덱스에 있는지 작성 -> 그 값을 리스트의 맨 앞으로 이동하는 단계를 반복하며 작은 숫자들로 바꿔주는 알고리즘입니다.
이 알고리즘을 BWT 알고리즘이 변환한(뭉쳐준) 데이터에 적용하면 0, 1 등 아주 작은 숫자의 빈도수가 매우 증가합니다.
이러한 전처리 과정을 거쳐 허프만 코드를 부여하면 데이터 압축의 효율을 극대화 할 수 있다는 것을 알게 되었고, 코드 구현에 적용했습니다. 또한 코드 구현 과정에서 파이썬의 file.write() 함수는 최소 단위가 1Byte인데 허프만 코드는 비트 단위로 코드를 부여하기 때문에 비트단위 처리 함수가 필요했습니다. 그래서 데이터 버퍼를 만들고 비트 단위의 shift, or 연산을 이용하여 8비트를 채우고 저장하는 방식을 사용했습니다.
이렇게 압축기를 구현하고 테스트하는 과정에서 데이터의 용량이 MB단위가 되면 BWT의 속도가 느려진다는 것을 알게 되어 관련 자료를 찾아보던 중 suffix array(접미사 배열)라는 알고리즘을 추가로 적용해야한다는 사실을 알게 되었습니다. 이 배열은 문자열의 모든 접미사를 사전 순으로 정렬하고, 그 시작 인덱스를 적어놓은 배열입니다. 이 배열은 주어진 문자열의 모든 접미사를 생성 -> 생성한 접미사들을 사전순 정렬 -> 정렬된 순서에 따라 원래 문자열에서 시작하는 인덱스 나열 단계로 만들어지는 배열입니다.
이 배열을 사용하면 문자열의 모든 회전 행렬을 생성하지 않고, suffix array에 저장된 위치(접미사의 시작인덱스)에서 -1을 한 위치(마지막 문자 인덱스)를 구하여 BWT의 값을 얻어낼 수 있습니다.
BWT 방식을 이용하면 1MB의 데이터를 처리하는데 1TB(1MB*1MB)의 메모리가 필요하지만 이 방식은 원본 문자열과 suffix array만 있으면 BWT 값을 구할 수 있어 메모리를 적게 사용하고, 속도 훨씬 빠른 방식입니다.

이제 해독기를 만들어 테스트를 하려고 하는데 해독을 하기 위해서는 허프만 트리가 있어야 하고, 이 허프만 트리를 만들기 위해서는 빈도수 딕셔너리가 있어야 한다는 것을 알게 되어 압축 파일에 이 딕셔너리를 저장하는 방법을 공부하다 파이썬 표준 라이브러리의 내장 모듈인 pickle 모듈을 알게 되었습니다.

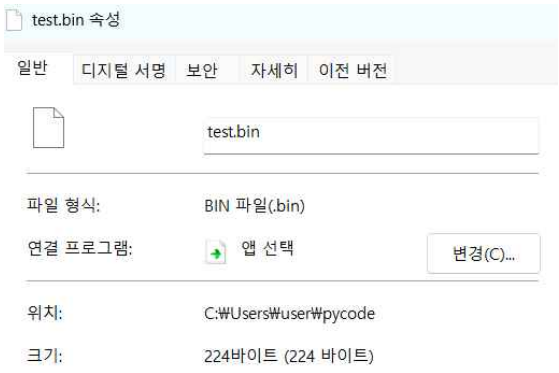
이 모듈은 파이썬 객체를 바이트 스트림으로 변환하여 파일에 저장할 수 있도록 해주는(직렬화) 모듈입니다. 이 모듈을 이용하면 빈도수 딕셔너리를 바이너리 덩어리로 만들어 저장하고, 해독기에서 읽을 때도 알아서 딕셔너리 끝나는 부분까지만 읽어 복구하도록 할 수 있습니다.



draw.io를 이용해 표현한 압축 파일 내부 구조

해독기는 먼저 압축한 바이너리 파일의 헤더 부분을 파싱하여 허프만 트리를 복구하고, 비트 스트림을 읽어가며 MTF 숫자 리스트를 복원합니다. 그 뒤에는 MTF 숫자 리스트를 다시 BWT 문자열로 변환하고, BWT 문자열을 다시 원본으로 복구해야합니다. 그런데 원본으로 복구하기 위해서는 또 새로운 방법이 필요하다는 것을 알게 되어 공부하던 중에 LF 맵핑이라는 방식을 이용하면 압축된 데이터 내부에서 효율적으로 패턴을 검색할 수 있다는 것을 알게 되었습니다.
이 방식은 압축 파일에서 꺼낸 마지막 열(L)만 가지고 있는 현재 상태에서 이 L을 알파벳 순으로 정렬하면 첫 번째 열(F)를 구할 수 있고, 절대 규칙인 “L열에 있는 k번째 문자 ‘a’는 F열에 있는 k번째 ‘a’와 동일한 문자이다.”를 이용합니다.
먼저 L열에서 문장의 끝이 ‘$’를 찾고 그 행의 F열의 문자는 ‘b’인 것을 보면 그 문장은 “...b$” 형식의 문장이라는 것을 알 수 있습니다. 그 다음 F열의 첫 번째 b는 L열의 첫 번째 b와 같은 문자입니다. 이 사실을 통해 문장은 “...ab$” 라는 것을 알 수 있습니다. 이 과정을 반복하면 BWT 문자열을 원본 문자열로 빠르게 복구할 수 있습니다.

-실행 결과




```
[test.txt] 압축 시작
1단계: BWT 변환
2단계: MTF 변환
3단계: 허프만 트리 생성
4단계: 파일 저장 중...
압축 완료! (715B -> 224B, 압축률 68.67%)
[test.bin] 해제 시작
1단계: 허프만 디코딩 완료
2단계: MTF 역변환
3단계: BWT 역변환
해제 완료! 원본 복구 성공.
```

테스트 결과 파일의 압축률은 68.67%였습니다.