

Assignment 4

Context Adaptive

Huffman coding

|  |  |  |
| --- | --- | --- |
| 과목명 |  | 확률및통계 |
| 담당교수 |  | 심동규 교수님 |
| 학과 |  | 컴퓨터정보공학부 |
| 학년 |  | 3학년 |
| 학번 |  | 2019202009 |
| 이름 |  | 서여지 |
| 제출일 |  | 21.05.09(일) |



1. 과제설명

지난 프로젝트에서 작성한 허프만 인코더와 디코더를 확장하여 context adaptive huffman coding을 하는 인코더와 디코더를 작성한다. 인코더 프로그램은 training data를 읽어 일반 허프만 테이블과 context adaptive 허프만 테이블을 생성하고, 생성한 테이블의 내용을 이용하여 training data와 text input을 압축하여 .hbs로 출력한다. 디코더 프로그램은 인코더 프로그램에서 생성한 두 종류의 테이블과 총 4개의 압축파일을 읽어 원본과 동일한 txt파일로 재생성한다.

1. 접근방법

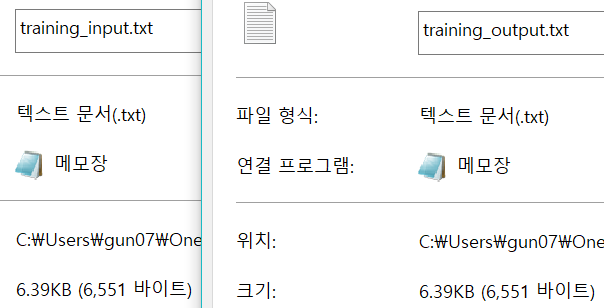
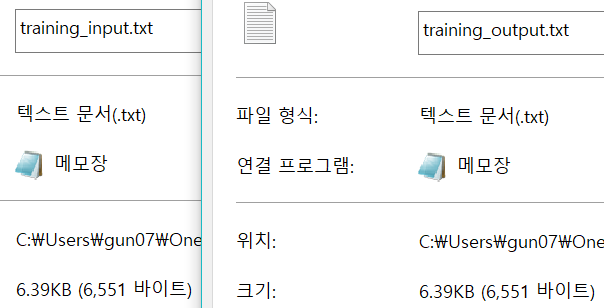
인코더는 지난 과제에서와 동일하게 training txt의 내용을 모두 읽어 각 문자의 등장 횟수를 센다. 그러나 이번 과제에서 context adaptive table을 작성하기 위해 직전의 문자와 이어지는 문자의 등장 횟수를 따로 저장한다. 이 정보를 바탕으로 normal table과 context adaptive table을 생성하기 위한 huffman tree를 구성하여 table을 작성한다. 이때 특정 조건에 의해 사용할 table만 생성하고, 조건에 부합하지 않는 table은 생성하지 않는다. 이것을 바탕으로 training input과 test input을 압축하여 .hbs파일을 작성한다.

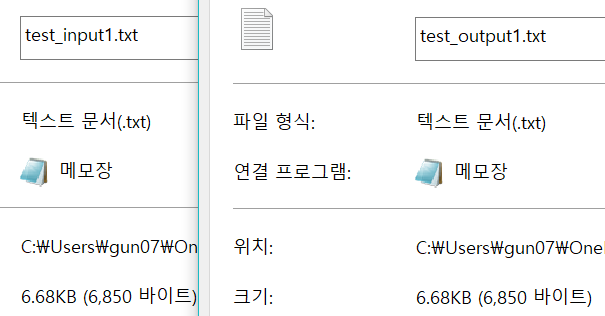
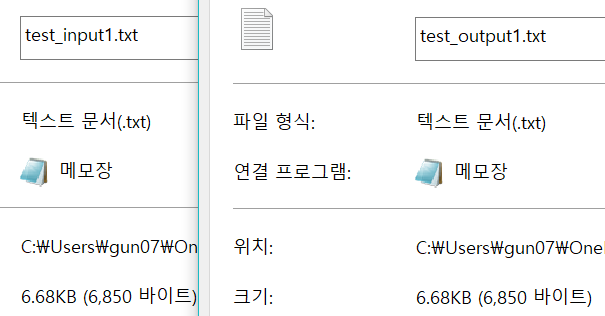
training input의 경우 모든 문자가 table에 존재하지만, test input의 경우 table에 존재하지 않는 문자가 등장할 수 있다. 이 경우의 예외처리를 위해 임의의 flag를 이용하였다. context adaptive table에서 이번 문자를 찾을 수 없는 경우 대신 normal huffman table에서 문자를 찾고, normal table에도 없는 경우 ASCII code를 삽입한다. adaptive table 대신 normal huffman table을 이용한 경우 모든 adaptive table에 각각 정의된 nTableFlag에 대한 codeword를 삽입하고 이어서 normal huffman table에서 찾은 문자의 codeword를 삽입한다. normal table에서 문자를 찾을 수 없는 경우 normal table에 정의된 ASCIIFlag에 대한 codeword를 삽입하고 이어서 해당 문자의 ASCII code를 삽입한다.

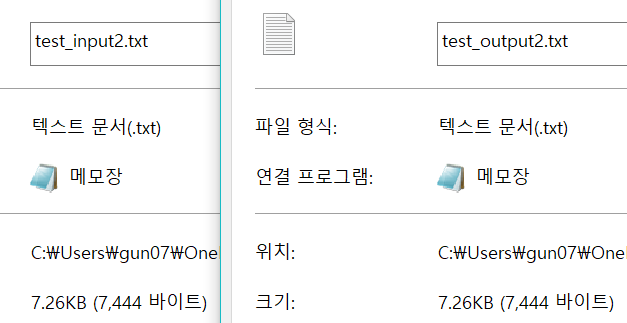
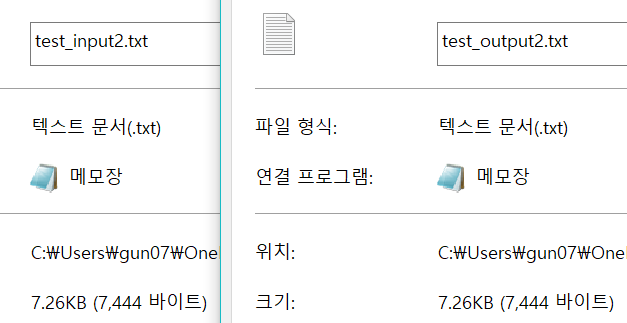
디코더 또한 지난 과제에서 작성한 것을 context adaptive table을 이용하도록 확장하였다. context adaptive table은 각각의 table의 끝에 EOD가 삽입되어있다. 따라서 adaptive table을 작성하는 과정은 normal table에서 얻은 EOD의 codeword에 대한 정보와 읽은 정보를 우선 비교한 후, EOD가 아닌 경우에 adaptive table을 작성하기 위해 이용하는 구조로 설계되었다. 이것은 서로 다른 테이블인 normal table과 adaptive table의 codeword가 중복될 수 있기 때문이다. EOD를 먼저 비교하여 해당 codeword가 adaptive table의 어떤 정보로 읽히는 것을 방지하였다.

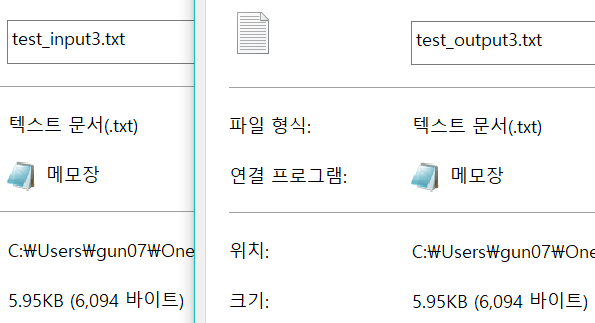
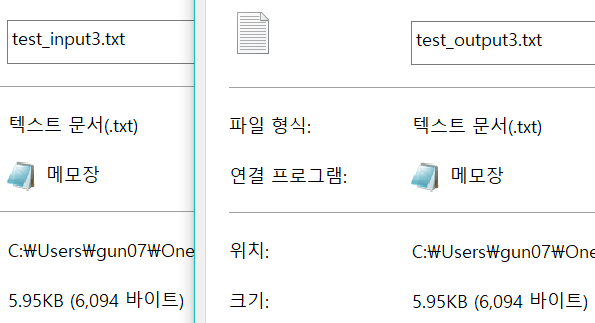
또한 디코더가 중간에 table을 교체해야 하는 경우 현재 테이블을 이용하여 읽을 수 있는 flag가 있기 때문에 해당 flag를 읽은 이후 적절하게 동작할 수 있도록 설계하였다.

1. 실험 및 검증 결과









압축 이전 파일과 재생성한 파일의 크기가 같은 것을 확인할 수 있다.

각 파일의 크기는 다음과 같이 나타났다.

|  |  |
| --- | --- |
| context\_adaptive\_huffman\_table.hbs | 1,863 byte |
| huffman\_table.hbs | 196 byte |
| training\_input\_code.hbs | 2,759 byte |
| test\_input1\_code.hbs | 3,159 byte |
| test\_input2\_code.hbs | 3,477 byte |
| test\_input3\_code.hbs | 2,978 byte |

cost가 최적이 되는 경우를 구하기 위해 몇 가지 조건으로 실험을 반복하였다.

adaptive table을 하나도 사용하지 않는 경우와 생성할 수 있는 모든 adaptive table을 사용한 경우에 대한 hbs 파일의 크기는 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| adaptive table | | training | test1 | test2 | test3 |
| 0 | 1 byte | 3631 | 3823 | 4164 | 3495 |
| 62 | 1863 | 2759 | 3159 | 3477 | 2978 |

이때 모든 adaptive table을 이용한 경우의 cost가 모든 실험에서 가장 작게 나타났다.

사용하는 table의 수를 줄이기 위해 두 가지 기준으로 실험하였다. 첫번째 기준은 adaptive table에서 정의하는 문자의 종류이다. 많은 종류의 문자에 대해 codeword를 지정하는 경우 각 문자에 대한 codeword의 길이가 늘어나기 때문에 adaptive table을 이용하여 얻는 이득이 줄어들 수 있다.

1. adaptive table에서 정의하는 문자의 종류가 k종류 이하인 table만 이용한 경우

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| k | adaptive table | | training | test1 | test2 | test3 |
| 1 | 5 개 | 36 byte | 3624 | 3815 | 4162 | 3491 |
| 2 | 15 | 128 | 3594 | 3792 | 4136 | 3471 |
| 3 | 22 | 210 | 3558 | 3769 | 4119 | 3458 |
| 4 | 28 | 296 | 3531 | 3760 | 4110 | 3443 |
| 8 | 35 | 452 | 3489 | 3748 | 4095 | 3442 |
| 16 | 51 | 1043 | 3177 | 3509 | 3818 | 3230 |
| 32 | 61 | 1729 | 2861 | 3255 | 3567 | 3024 |

실험 결과 사용하는 table의 수가 증가할수록 압축된 파일의 크기가 감소하는 것을 확인할 수 있었다. 이때 k가 증가하며 추가되는 adaptive table은 크기가 작은 순서이다. k의 값이 1에서 2로 변화하는 경우와 16에서 32로 변화하는 경우를 살펴보면 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 증가한 테이블 | adaptive table 크기 | output data 크기 |
| k = 1 -> k = 2 | +10 table | + 92 byte | - 20~30 byte |
| k = 16 -> k = 32 | +10 table | + 686 byte | -200~300 byte |
|  |  | 7.5배 | 10배 |

두 경우 모두 10개의 table이 증가하였지만 adaptive table file의 크기는 각각 92byte와 686byte로 약 7.5배의 차이가 발생한다. 이 경우에 압축된 파일의 크기는 k=1에서 k=2로 변화하는 경우에 20~30 byte 사이에서 감소하지만, k=16에서 k=32로 변화하는 경우에 200~300 byte 가 감소하여 약10배까지의 차이가 발생한다. 위 결과를 통해 adaptive table에서 정의하는 문자의 종류를 통해 adaptive table을 줄이는 것은 더 효율적인 table을 선택하는 것에 적절하지 않다.

두 번째 기준은 preceding symbol의 등장 빈도에 따라 상위 k개의 table을 남기는 것이다. preceding symbol이 자주 등장한다면 그 이후의 문자 또한 빈번하게 발생하게 될 것이다. 빈번하게 발생하는 문자의 codeword 길이가 줄어들면 적은 수의 table을 이용해서 더 큰 압축률을 얻을 수 있을 것이다.

1. preceding symbol의 등장 빈도가 높은 k개의 table을 이용한 경우.

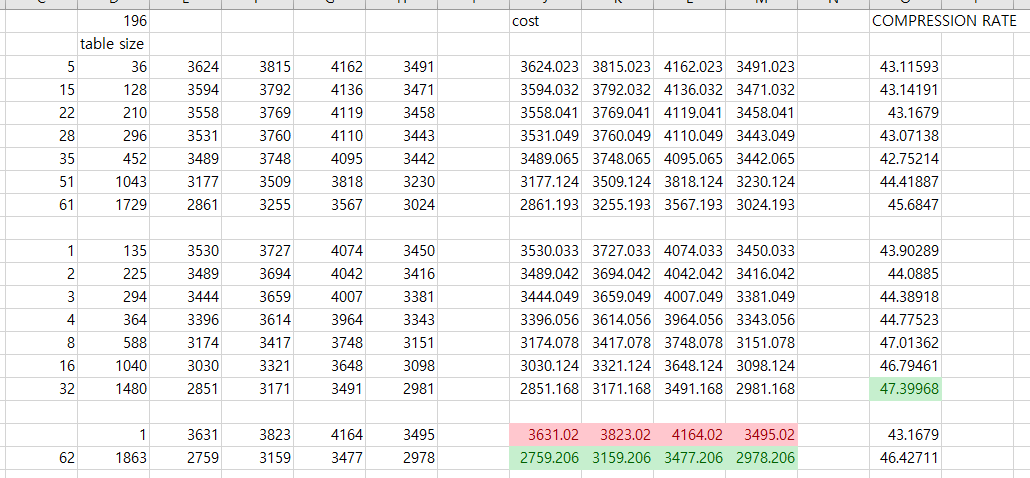
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| k | adaptive table | | training | test1 | test2 | test3 |
| 0 | 0 | | 3631 | 3823 | 4164 | 3495 |
| 1 | 1 | 135 | 3530 | 3727 | 4074 | 3450 |
| 2 | 2 | 225 | 3489 | 3694 | 4042 | 3416 |
| 3 | 3 | 294 | 3444 | 3659 | 4007 | 3381 |
| 4 | 4 | 364 | 3396 | 3614 | 3964 | 3343 |
| 8 | 8 | 588 | 3174 | 3417 | 3748 | 3151 |
| 16 | 16 | 1040 | 3030 | 3321 | 3648 | 3098 |
| 32 | 32 | 1480 | 2851 | 3171 | 3491 | 2981 |

실험 결과 첫 번째 실험에서와 동일하게 사용하는 table의 수가 증가할수록 압축된 파일의 크기가 감소하는 것을 확인할 수 있었다. k의 값이 0에서 8로 증가하는 경우와 8에서 16으로 증가하는 경우를 비교하면 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 증가한 테이블 | adaptive table 크기 | output data 크기 |
| k = 0 -> k = 8 | +8 table | + 587 byte | - 350 ~ 450 byte |
| k = 8 -> k = 16 | +8 table | + 452 byte | - 60 ~ 140 byte |
|  |  | 0.77배 | 0.25배 |

두 경우 모두 8개의 table을 추가로 사용하지만, output data의 크기는 k가 0에서 8로 증가한 경우가 8에서 16으로 증가하는 경우에 비해 약 4배 더 많은 크기를 줄일 수 있었다. 따라서 preceding symbol의 등장빈도에 따라 table을 선택하는 것은 더 효율적인 table을 선택하는 것에 적합하다.

실험 결과에 대해 cost와 압축률을 구하면 다음과 같다. adaptive table을 많이 사용할수록 cost를 낮출 수 있었다. 그러나 cost가 가장 작은 경우가 압축률이 가장 높은 경우와 일치하지 않았다. 압축률과 달리, cost를 구하는 과정에서 huffman table의 크기가 계수에 의해 결과에 주는 영향이 줄어들기 때문일 것이다. 이번 실험에서는 총 4개의 input을 처리했지만 처리하는 input의 수가 크게 증가한다면 압축률을 계산하는 경우에도 table의 크기가 결과에 주는 영향이 줄어들기 때문에 cost가 가장 작은 경우와 압축률이 가장 높은 경우가 일치할 것으로 예상된다.



1. 고찰

table을 전환하는 과정에 이용한 flag는 txt파일에서 사용된 적 없는 값으로 선택해야 한다는 주의점이 있고, training data에 나타나지 않았던 문자가 많이 포함된 input data를 encoding하는 경우 ASCII code를 빈번하게 사용하게 되어 압축률이 감소할 수 있다는 문제점이 있다. 과제를 진행하는 과정에서 table에 존재하지 않는 문자를 encoding하는 문제를 해결하기 위해 flag를 이용하였는데, 사용할 수 있는 다른 방법이 있을지 궁금하다.