Theory Of Computation 2017 Fall (고급계산이론 2017 가을학기)

Prof. Kunsoo Park (지도교수. 박근수)

HW2. Lempel-Ziv-Welch Compression

2017-21586 Wonseok Lee (이원석)

**NOTE. 입력 파일에 대하여**

실험 환경 및 구현 환경이 Linux이므로 압축 대상이 되는 파일 역시 Unix 형식을 따른다고 가정합니다. 즉, 배표된 예시 입력 파일인 infile.txt의 경우 NOTEPAD++ 등을 사용하여 EOL (End-Of-Line)문자를 Unix 형식으로 변환한 후 사용하였습니다. 또한, 파일의 크기 역시 256MB 까지만을 허용합니다.

1. **Lemple-Ziv-Welch Compression and Decompression**

Lempel-Ziv-Welch (LZW) compression/decompression 알고리즘은 압축 과정에서 구성한 딕셔너리를 압축 해제 단계에 전달하지 않고도 손실 없는 압축 해제를 할 수 있는 엔코딩/디코딩 알고리즘으로 아래의 표1, 표2와 같이 나타낼 수 있다.

|  |
| --- |
| **표1.** Lempel-Ziv-Welch Compression 알고리즘[1] |
| w = ""  while( GetNextChar( K ) )  {  if( w + K is in the dictionary )  w = w + K  else  {  send CODE(w) to output  Add w + K to the dictionary  w = K  }  } |

|  |
| --- |
| **표2.** Lempel-Ziv-Welch Decompression 알고리즘[1] |
| GetNextCode( code )  w = D(code)  send w to output  while( GetNextCode(code) )  {  if( there's a string whose index is code in the dictionary )  {  tmp = w  w = D(code)  Add tmp + w[0] to the dictionary  }  else  {  Add w + w[0] to the dictionary  w = D(code)  }  send w to output  } |

1. **Optimization**

LZW 엔코딩/디코딩에서 압축률의 최적화를 위하여 조정해 볼 수 있는 Knob 중 하나로 딕셔너리 코드의 사이즈가 있습니다. 본 과제 구현 과정에서는 딕셔너리 코드의 사이즈를 16bit로 설정하였습니다. 즉, 65536개의 딕셔너리 엔트리만을 허용합니다. LZW의 구현에 사용된 딕셔너리의 매핑을 아래의 표3에서 보입니다. 0 ~ 127의 경우 ASCII code와 1:1 매핑하였으며, LZW가 생성하게되는 Longest Prefix에 대한 매핑은 128부터 사용되었습니다.

|  |  |
| --- | --- |
| **표3.** LZW 딕셔너리 매핑 | |
| 0 | NULL |
| 1 | SOH |
| ... | ... |
| 65 | A |
| 66 | B |
| ... | ... |
| 90 | Z |
| ... |  |
| 127 | DEL |
| 128 | LZW prefixes |
| 129 |
| ... |
| 65534 |
| 65535 | EOF |

1. **Environment**

OS : Ubuntu 14.04.5 LTS

Language : C++

Compiler : g++ 4.8.4

Compile Options: std=c++11

CPU : Intel® Core™ i7-3770 @ 3.40GHz

RAM : 32 GB

**4. Evaluation**

작성한 프로그램의 평가를 위하여 주어진 infile.txt, 시스템의 Kernel 레벨 로그 (dmseg.txt), 명령어 기록 (history.txt)에 대하여 LZW 알고리즘을 수행한 결과는 아래의 표4와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| **표4.** My LZW algorithm | | | |
| File name | Origin (byte) | Encoded (byte) | Compression ratio (%) |
| dmseg.txt | 84950 | 42138 | 49.60 |
| history.txt | 24239 | 10072 | 41.55 |
| infile.txt | 1521996 | 685390 | 45.03 |

리눅스의 “zip” 명령어를 통하여 동일한 파일을 압축하였을 때 얻어지는 결과는 아래의 표5와 같으며 그림1과 같이 직접 구현한 LZW와 비교할 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| **표5.** Linux "zip" command | | | |
| File name | Origin (byte) | Encoded (byte) | Compression ratio (%) |
| dmseg.txt | 84950 | 20456 | 24.08 |
| history.txt | 24239 | 4589 | 18.93 |
| infile.txt | 1521996 | 644480 | 42.34 |

**그림1.** LZW와 zip의 성능 비교

**5. Conclusion**

Zip 명령어와의 비교를 통한 평가에서 직접 구현한 LZW는 압축률 측면에서 zip에 비해 효율적이지 못한 것이 확인되었다. 이에 대한 원인으로는 직접 구현한 LZW 알고리즘이 구현의 편의를 위하여 항상 딕셔너리 코드로 16bit을 사용하는 것을 생각해볼 수 있다. 실제 LZW를 사용하는 상용 툴의 경우 12bit 크기의 딕셔너리 코드를 사용한다.

**6. Reference**

[1] http://www.cs.carleton.edu/faculty/jondich/courses/cs337\_w00/Assignments/lzw.html