20151547 박재상

1. 우선 대략적인 알고리즘을 소개하겠다.

압축이라는 것은, 반복되는 부분을 최소화하여 파일의 부피를 줄이는 것이 그 목적이다. 그래서 greedy algorithm의 일종으로 huffman coding을 응용하여 가장 많이 반복되는 문자에 가장 짧은 bit string을, 그리고 가장 적게 반복되는 문자에 가장 긴 bit string을 할당하여 전체 파일의 크기를 줄인다.

이를 위해서는 해당 파일이 어떤 분포를 이루고 있는지에 대한 정보가 압축파일에 담겨야한다. 그래야 압축을 해제할 때 어떤 식으로 bit string이 할당되었는지 알 수 있기 때문이다.

본인은 huffman code를 어떻게 구현해낼까 고민을 많이 해보았다. 구글이나 수업자료에는 min-heap의 구현을 해놓았다. 이런 자료구조에 너무 신경쓰고 싶지 않아서, huffman code를 가장 많이 반복되는 문자 0~26위까지에 대해서 미리 만들어놓고, 그 다음부터는 순서대로 bit string을 할당 받는다.

즉, 코드로 구현하면 아래와 같다.

char T[26][7]={"10","110","010","1110","0110","0010","11110","01110","00110","00010","111110","011110","001110","000110","000010","1111110","0111110","0111111","0011110","0011111","0001110","0001111","0000110","0000111","0000010","0000011"};

이렇게 미리 huffman code를 할당 한 뒤, 나머지 문자에 대해서는 순위별로

for(int k=0;k<j-18;k++){

fprintf(temp,"1");

}

fprintf(temp,"0");

이런식으로 11111111110으로 할당하여 파일을 압축했다.

Bit string으로 원본파일을 변환한 뒤, 이 bit string을 8개단위로 끊어서 unsigned char의 data와 1대1 대응시켜서 변환한 파일의 용량을 1/8로 줄인다.

변환된 최종파일의 가장 첫 번째 문자는 8개단위로 끊었을 때, 뒤에 남는 문자의 개수를 뜻하고, 두번째부터 256번째 까지의 문자는 원본파일의 분포를 분석하여 frequency로 정렬해놓은 문자들이다. 이 두 가지를 이용하여 decompression을 진행할 것이다.

이제 decompress를 할 때는, frequency순으로 정렬된 순서를 알아내고서, 각 unsigned char의 data값들을 한글자씩 읽어서 8글자의 bit string으로 변환하는 과정을 거친다. 이 과정은 ascii code값에 대응 시키는 것과 같다. 그리고 나서 bit string으로 바뀐 파일을 앞에서부터 차례로 읽어가면서 미리 선언한 huffman code를 참조하여 원래 파일로 복원한다.

1. Text file

|  |  |  |
| --- | --- | --- |
| Input file size | Output file size | Ratio |
| 4229 | 4144 | 1.02 |
| 435 | 256 | 1.69 |
| 1142 | 1794 | 0.63 |

1. JPG

|  |  |  |
| --- | --- | --- |
| Input file size | Output file size | Ratio |
| 24654 | 208457 | 0.11 |
| 33041 | 276179 | 0.11 |
| 45880 | 468082 | 0.17 |

Textfile은 반복도가 높은 문자열이 있을수록 압축률이 높았다. 그러나 JPG파일은 문자들이 정말 고르게 분포되어있는지 용량이 줄어들 생각을 않는다.