Steven Comer
CSC 705
Algorithm 2
25 Jul 2016

Class of Algorithm: General Purpose Lossless Compression

Overview: Compression aims to provide more efficient storage of a given set of information. This efficient storage need not be readable at rest, as long as it can be properly restored to its original form without loss of the original data, in the case of lossless compression. The driving force behind this need is the fact that we have few size restrictions for data at rest and many size restrictions for data in transit.

Compression is extremely important in a time where we constantly want to transmit and receive large quantities of information. Doing so more efficiently is a valuable contribution of compression.

Representative Algorithms in this Class:
Lempel-Ziv (LZ78): references a dictionary for repeated data occurrences, used by GIF images and many other applications
DEFLATE: combines LZ77 and Huffman encoding, RFC 1951, used by gzip, PNG images, zip
bzip2: uses Burrows-Wheeler algorithm, compresses single files, more efficient compression than DEFLATE, but slower
LZ-Markov chain algorithm (LZMA): uses dictionary scheme similar to LZ, better compression than bzip2, fast decompression, used by 7zip and xz
Run Length Encoding (RLE): straightforward scheme, inefficient for all but repetitive data sets, used in fax transmission

Algorithm: Run Length Encoding
aka: RLE

Techniques: Encoding scheme

Categories: Compression, lossless compression

Problem: Provide efficient storage of information that contains repeated values.

Applications: This algorithm was designed with a specific use case in mind. It does not work well outside of that specific use case, meaning data sets with frequent repetition and long sequences of those repeated bytes.

References:
1. https://en.wikipedia.org/wiki/Run-length_encoding

2. http://www.fileformat.info/mirror/egff/ch09_03.htm
3. https://www.rosettacode.org/wiki/Run-length_encoding
4. http://www.prepressure.com/library/compression-algorithm/rle

Implementation details:

- **Big Idea**: Represent repeated data efficiently.
- **Description**: Iterate through input. For compression, record sequentially the number of repetitions of each character, followed by that character.
- **Pseudo-code**: http://www.mat.univie.ac.at/~kriegl/Skripten/CG/node44.html

```
Loop: count = 0
        REPEAT
                get next symbol
                count = count + 1
        UNTIL (symbol unequal to next one)
        output symbol
        IF count > 1
        output count
        GOTO Loop
```

Figure 1: Run Length Encoding Pseudocode

- **Specific implementation**: See included file rle.py

Correctness:

Theoretical: The argument for the correctness of this program is a proof of the encoding scheme. The compression is counting the number of repetitions of a given byte and storing it with the count. The decompression is expansion of that compression by repeating each byte for its aforementioned count number.

Uncompressed = “aaabbbccc”
Compressed = “3a3b3c”
Decompressed = “aaabbbccc”

Empirical: We checked the program for correctness by running “diff” on the uncompressed and decompressed files to ensure that the program did not alter the data in the course of compression and decompression.

Performance:

Theoretical: The theoretical time complexity of this algorithm is O(n), as it iterates over the input data once. Again, this means that one iteration is performed per byte of input. The worst case space complexity is 2*n, if no bytes are repeated. The best case theoretical space complexity is a constant if all of the input bytes are identical.

Empirical: We wrote a Python script to test the performance factors of the Run Length Encoding algorithm with 99 data points of sizes (100, 200, …, 10000). It is important to note that this data is pseudorandom (from /dev/urandom). For this reason, we also tested the space complexity on an advantageous type of input data and confirmed that it has constant space complexity for that case. As expected, the empirical testing confirmed that the time complexity of RLE is O(n). The space complexity performed as expected on random data, that is, poorly. See charts below for more detail on the empirical testing.

```
#!/usr/bin/python

from rle import encode,decode
from os import system,stat

"""
    File: test.py
    Author: Steven Comer
    Description: Testing suite for rle.py
    Updated: 25 Jul 2016
    Data of interest:
        runtime_enc.csv -- file size vs encode runtime
        runtime_dec.csv -- file size vs decode runtime
        compression.csv -- file size vs compression ratio
"""

def main():
    d_run_enc = {}
    d_run_dec = {}
    d_comp = {}
    for i in xrange(100,10000,100):
        system("head -c " + str(i) + " < /dev/urandom > test.data")
        enc_time = encode("test.data","test.data.rle")
        dec_time = decode("test.data.rle","test.data.dec")
        file_size = stat("test.data").st_size
        enc_file_size = stat("test.data.rle").st_size
        d_run_enc[i] = round(enc_time,5)
        d_run_dec[i] = round(dec_time,5)
        d_comp[i] = round(float(file_size)/float(enc_file_size),5)
    f_run_enc = open("runtime_enc.csv","w")
    f_run_dec = open("runtime_dec.csv","w")
    f_comp = open("compression.csv","w")
    for item in d_run_enc:
        f_run_enc.write(str(item) + "," + str(d_run_enc[item]) + "\n")
        f_run_dec.write(str(item) + "," + str(d_run_dec[item]) + "\n")
        f_comp.write(str(item) + "," + str(d_comp[item]) + "\n")
    f_run_enc.close()
    f_run_dec.close()
    f_comp.close()

if __name__ == "__main__":
    main()
```

Figure 2: test.py

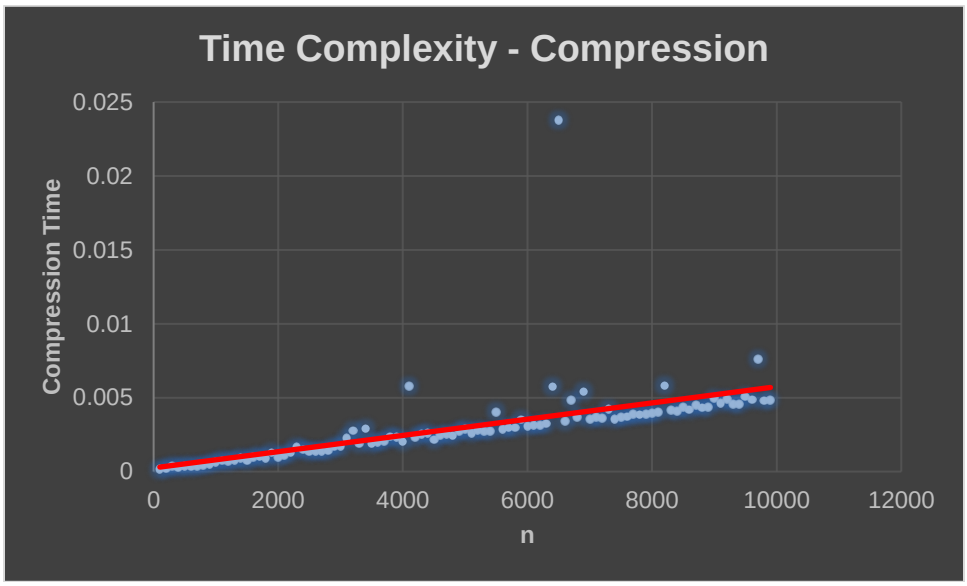

Figure 3: Time Complexity, sample size 99

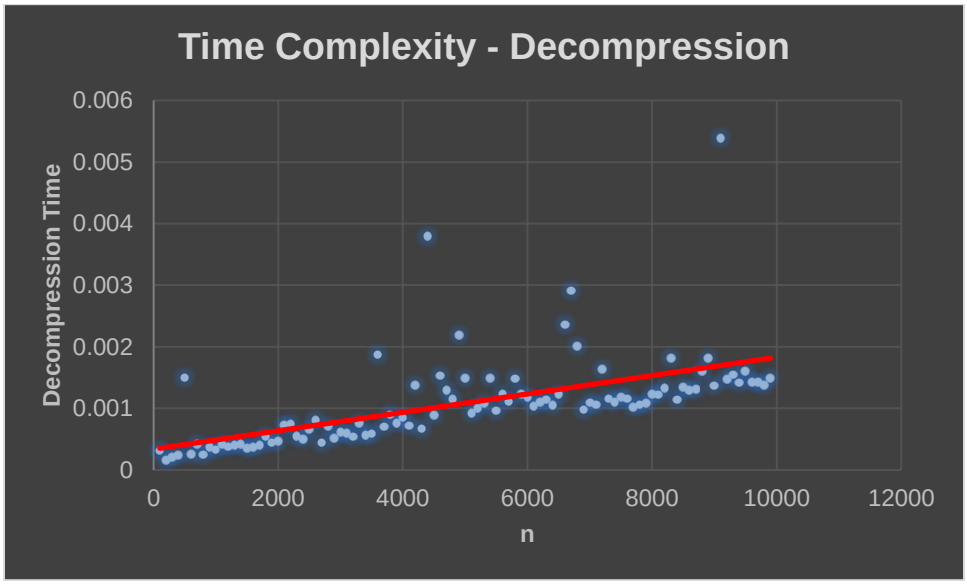

Figure 4: Time Complexity, sample size 99

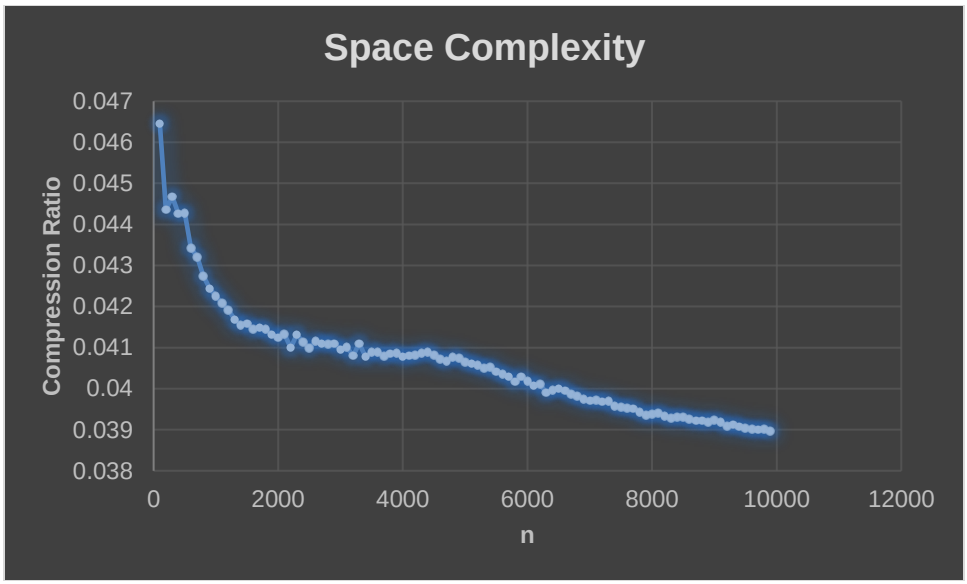

Figure 5: Space Complexity, sample size 99

Anecdotes: This algorithm will double the size of the original file if there are no repetitions in the data.

History: Run length encoding was first used in the transmission of TV signals in the late 1960's. It works well with low-quality images, particularly those which are black and white. It was used for image compression before more complex and efficient formats were available. It is still used in JPEG, but not directly on the pixels involved; it is used at a higher layer of abstraction where it can still be effectively applied. It is often efficient to use for fax data, because that is most often dominated by whitespace.

Variations: Modifications exist to prevent inefficient representation of non-repeated characters. These modifications make it slightly more applicable to normal use-cases for compression.

Alternatives: Many alternative compression algorithms exist. This algorithm is very straightforward and is good for data sets with a lot of repetition and very bad for those without large amounts of repetition. Alternatives include Lempel-Ziv (LZ78), used by GIF and others, and bzip2, which uses the Burrows-Wheeler transform.

Conclusion: The use case for Run Length Encoding is not terribly common at this time, but it is important to aid the student in understanding how far compression has come since the 1960's. Compression itself is and will continue to be very important as data transmission demands will only continue to increase.