TÍNH TOÁN ĐA PHƯƠNG TIỆN

XÂY DỰNG THUẬT TOÁN NÉN LOSSLESS FILE TEXT TRÊN NỀN WEB

14520529 – NGUYỄN CAO MINH, 14520178 – ĐOÀN TRÍ ĐỨC

Lớp Khoa học Tài năng 2014, Khoa Khoa học máy tính, Trường Đại học Công nghệ Thông tin;
14520529@gm.uit.edu.vn, 14520178@gm.uit.edu.vn

CS232.I11.KHTN

NGÔ ĐỨC THÀNH

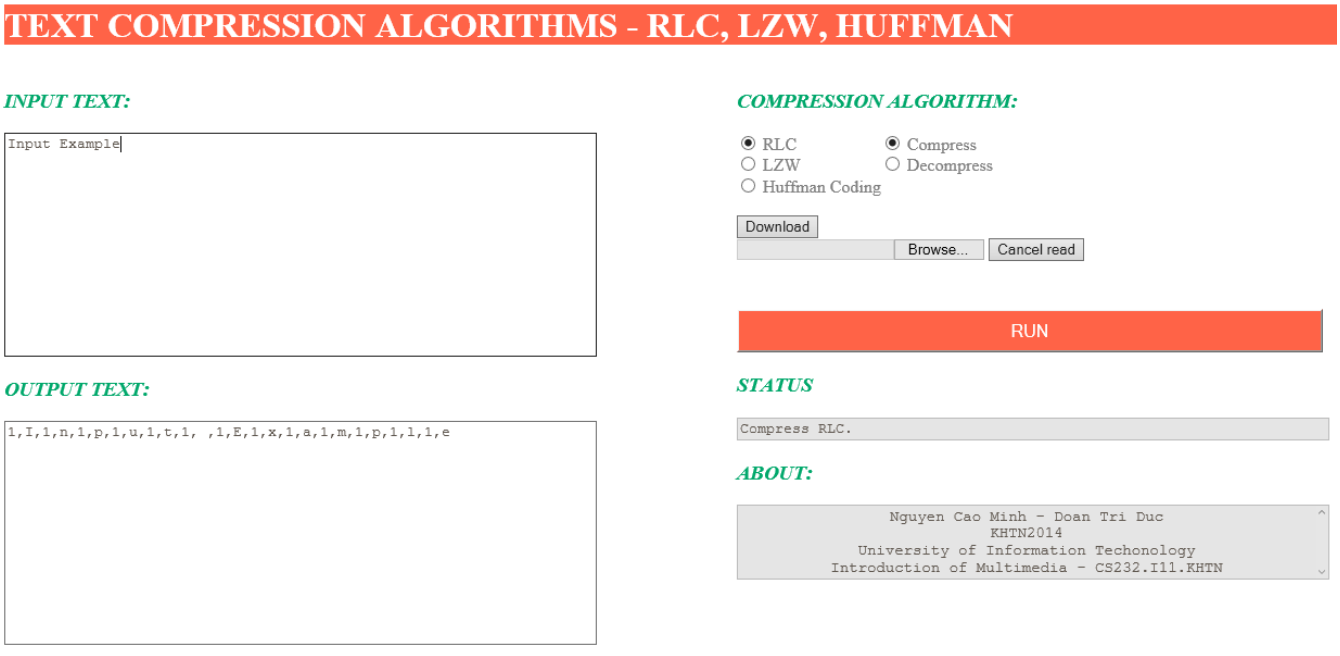
NGUYỄN VINH TIỆP

TP. HỒ CHÍ MINH, 18/1/2017

MỤC LỤC

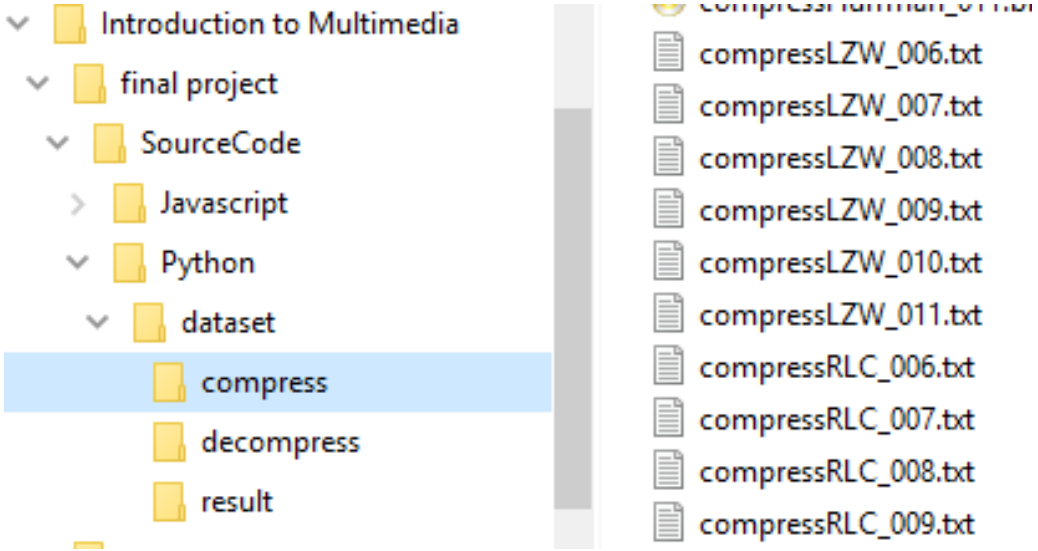
I. MỤC TIÊU:

- Tìm hiểu và cài đặt, xây dựng 1 ứng dụng áp dụng các thuật toán nén trên 1 loại dữ liệu đặc thù.
- **X**ây dựng 1 ứng dụng áp dụng các thuật toán nén lossless cơ bản trên dữ liệu text, với 2 nền tảng riêng biệt phục vụ cho 2 mục đích khác nhau:
    - Các thuật toán nén được sử dụng:
        - RLC – Run Length Coding.
        - LZW.
        - Huffman Coding.
    - **ỨNG DỤNG TRÊN NỀN PYTHON:** Với mục đích sử dụng cho việc nghiên cứu so sánh độ tối ưu giữa các thuật toán, đầu vào của ứng dụng là 1 tập dataset bao gồm các file text bất kỳ, kết quả trả về sẽ bao gồm: File compress, decompress, và file result.
    - **ỨNG DỤNG TRÊN NỀN WEB:** Với mục đích cho người sử dụng có cái nhìn trực quan hơn và hỗ trợ thao tác nhanh gọn cho các nhu cầu thực hiện với các file đơn lẻ.



II. PHƯƠNG PHÁP THỰC HIỆN:

1. Bộ dữ liệu (Đối với phiên bản trên Python):
   - Bộ dataset bao gồm 40 tập tin (tổng dung lượng 60MB) với các kích thước khác nhau. Nội dung và cấu trúc trong các tập tin rất đa dạng, bao gồm:
     - Bảng xếp hạng các trường đại học trên thế giới (chuyển từ .csv qua .txt)
     - Các bài báo khoa học, tin tức trên các trang web điện tử.
     - Các chương từ các tiểu thuyết, tác phẩm văn học.
     - Các thống kê của cảnh sát, bác sĩ về tai nạn và tội phạm.
     - Kết quả tỉ số của các trận bóng rổ, bóng đá trong tháng 1 – tháng 9 năm 2017.
   - *Do bộ dataset quá lớn (500MB) nên nhóm đã up lên github (link ở cuối báo cáo), chỉ đính kèm vài file mẫu trong file nộp.*
2. Xây dựng cấu trúc thư mục lưu trữ:
   - Thư mục lưu trữ có dạng như sau:



   - dataset: Thư mục chứa tập dataset từ người dùng
     - compress: Thư mục lưu trữ các tập tin đã được nén dưới dạng file .txt hoặc .bin
     - decompress: Thư mục chứa các tập tin đã được giải nén.
     - Result: Thư mục chứa các tập tin lưu trữ kết quả dung lượng của file gốc và file nén (đơn vị Byte) và compression ratio của file.

III. CÁC THUẬT TOÁN NÉN:

1. RLC – RUN LENGTH CODING[1]:
   - Là dạng nén ít mất mát dữ liệu rất đơn giản, ý tưởng của thuật toán đó chính là nén các dãy dữ liệu liên tục tương tự nhau (runs) thành 1 dữ liệu đơn lẻ kèm theo số lần lặp của dữ liệu đó.
   - Thuật toán RLC thể hiện rất tốt đối với các dạng dữ liệu chứa nhiều những chuỗi dữ liệu liên tục tương tự nhau. Sẽ tiết kiệm được rất nhiều dung lượng so với nguyên gốc.Ví dụ như các ảnh icons, các đường vẽ, các animations.
   - Đối với những dạng dữ liệu không có nhiều runs, thuật toán có thể làm tăng lượng lớn kích thước dữ liệu so với nguyên gốc.
   - Ưu điểm:
     - Đơn giản, dễ cài đặt, tốc độ xử lý nhanh.
     - Thích hợp với các dữ liệu có quy luật hoặc các dãy liên tục tương tự nhau.
   - Nhược điểm:
     - Phụ thuộc nhiều vào cấu trúc lưu trữ cho việc nén và giải nén.
     - Có thể gây ra tác dụng ngược, sản sinh ra thêm dung lượng với nhiều bộ dữ liệu đặc biệt.
   - Thực hiện cài đặt thuật toán:

```python
#Run Length Coding
import os
def compress(input_string):
    count = 1
    prev = ''
    lst = []
    for character in input_string:
        if character != prev:
            if prev:
                entry = (prev,count)
                lst.append(entry)
                count = 1
            prev = character
        else:
            count += 1
    else:
        entry = (character,count)
        lst.append(entry)
    return lst
```

```python
def decompress(lst):
    q = ""
    for character, count in lst:
        q += character * count
    return q
```

2. LZW – Dictionary-based coding[2]:
   - Thay vì khởi tạo từ điển từ các từ cho trước hoặc định trước, ta sử dụng bộ từ điển mặc định 256 của ASCII.
   - Tìm chuỗi W dài nhất có thể trong từ điển sao cho khớp với chuỗi đang xét hiện tại.
   - Đưa chỉ số index của W vào output và loại W ra khỏi chuỗi đang xét.
   - Đưa W và ký tự tiếp theo của chuỗi đang xét vào từ điển.
   - Quay trở về bước 2.
   - Ưu điểm:
     - Khai thác được tính chất lặp lại của từ và cụm từ. Có tính kế thừa cao.
     - Là một thuật toán 1 chiều (chỉ duyệt 1 lần), và không cần quan tâm đến xác suất và thống kê của các từ đằng trước.
   - Khuyết điểm:
     - Thông điệp cần phải dài và sự lặp lại giữa các cụm từ phải nhiều.
   - Thực hiện cài đặt thuật toán:

```python
#LZW Compression
import os

def compress(uncompressed):
    """Compress a string to a list of output symbols."""

    # Build the dictionary.
    dict_size = 256
    dictionary = dict((chr(i), i) for i in xrange(dict_size))

    w = ""
    result = []
    for c in uncompressed:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            # Add wc to the dictionary.
            dictionary[wc] = dict_size
            dict_size += 1
            w = c

    # Output the code for w.
    if w:
        result.append(dictionary[w])
    return result
```

```python
def decompress(compressed):
    """Decompress a list of output ks to a string."""
    from cStringIO import StringIO

    # Build the dictionary.
    dict_size = 256
    dictionary = dict((i, chr(i)) for i in xrange(dict_size))

    result = StringIO()
    w = chr(compressed.pop(0))
    result.write(w)
    for k in compressed:
        if k in dictionary:
            entry = dictionary[k]
        elif k == dict_size:
            entry = w + w[0]
        else:
            raise ValueError('Bad compressed k: %s' % k)
        result.write(entry)

        # Add w+entry[0] to the dictionary.
        dictionary[dict_size] = w + entry[0]
        dict_size += 1

        w = entry
    return result.getvalue()
```

3. Huffman Coding– Variable-Length Coding:[3]
    - Mỗi ký tự được xem như là một node, và giá trị của node chính là tần số xuất hiện của ký tự đó. Ta thêm tất cả các node theo thứ tự vào một hàng đợi.
    - Nếu hàng đợi vẫn còn chứa nhiều hơn 1 node:
        - Loại bỏ 2 node có tần số nhỏ nhất ra khỏi hàng đợi.
        - Gán 1 node bằng 0 và node còn lại là 1.
        - Tạo 1 node cha từ 2 node vừa mới lấy ra, với giá trị node cha bằng tổng tần số của 2 node con.
        - Đưa node cha mới tạo vào trong hàng đợi.
    - Node cuối cùng còn lại sau khi thực hiện quá trình trên sẽ là node gốc, và cây Huffman được hoàn thành.
    - Ưu:
        - Khắc phục được vấn đề của thuật toán Shannon.
        - Giảm thiểu được lượng lớn dữ liệu khi nén.
        - Vấn đề giải nén cực kỳ đơn giản nếu như vẫn tồn tại một bảng mã nén, cho dù dữ liệu có phát triển lớn đến mức nào.
        - Prefix code – không xuất hiện sự nhập nhằng.
        - Giảm thiểu được số bit cần cho 1 ký tự.
    - Nhược:
        - Yêu cầu chung vẫn giống như Shannon, tỷ lệ xác suất nên được phân bố rời rạc, không nên trải đều.
        - Yêu cầu phải thống kê được xác suất.
        - Xác suất thì có thể thay đổi theo thời gian do đó phải cập nhật.

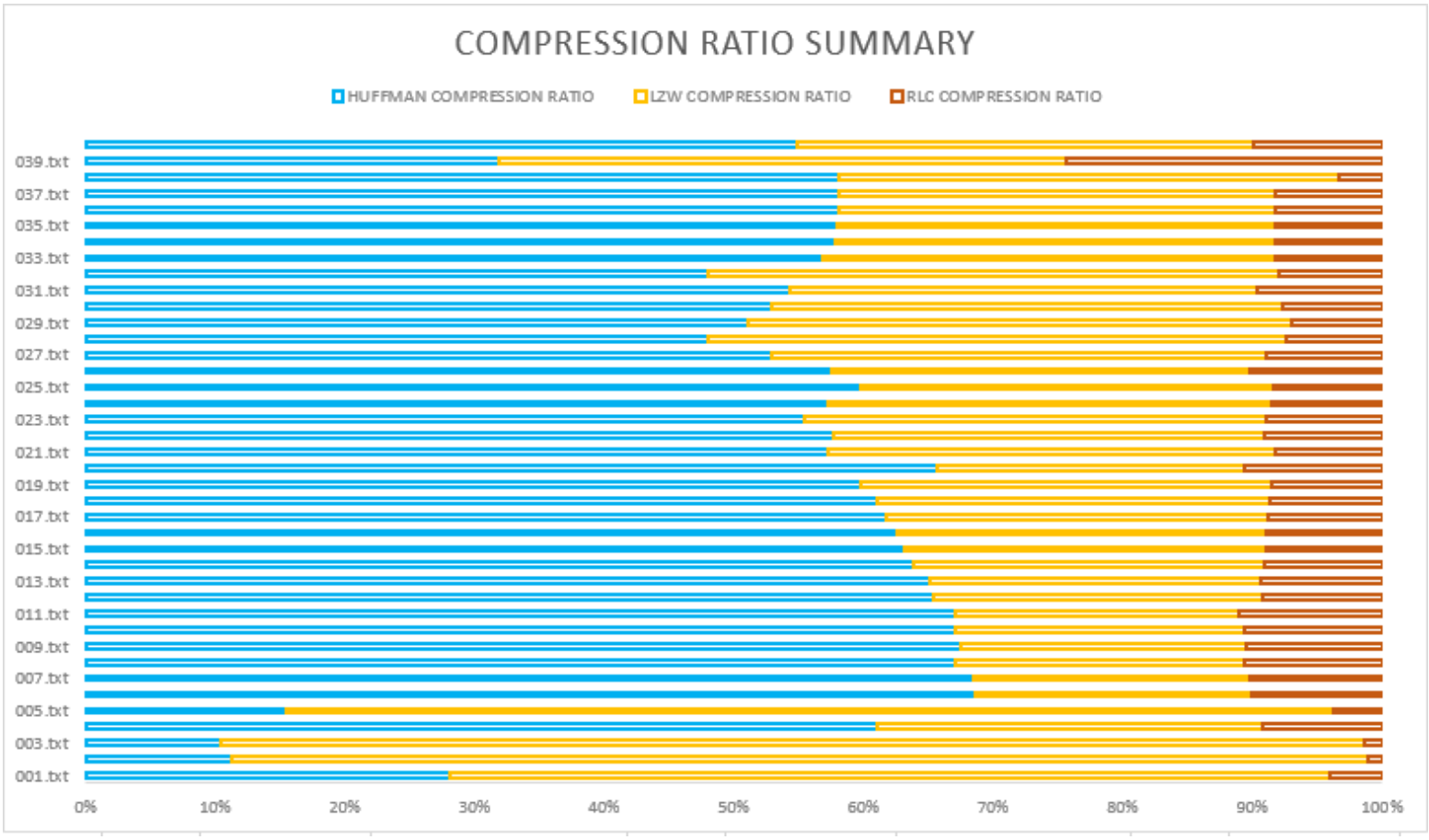
IV. ĐÁNH GIÁ KẾT QUẢ:

- Kết quả đạt được khi áp dụng các bộ phân loại trên 40 tập tin được thống kê trong file summary như sau:

FILE NAME	HUFFMAN			LZW			RLC		
	FILE SIZE (BYTE)	FILE COMPRESS (BYTE)	COMPRESSION RATIO	FILE SIZE (BYTE)	FILE COMPRESS (BYTE)	COMPRESSION RATIO	FILE SIZE (BYTE)	FILE COMPRESS (BYTE)	COMPRESSION RATIO
001.txt	117273	41024	2.858643721	117275	16945	6.920920626	117275	284840	0.41172237
002.txt	6835199	1507201	4.535028175	6835199	193429	35.33699187	6835199	15820795	0.432038908
003.txt	1367999	720001	1.899995972	1367999	84435	16.2018002	1367999	5255995	0.26027403
004.txt	30980	17817	1.738788797	30988	36565	0.847477096	30988	118039	0.262523403
005.txt	2041506	593256	3.441188964	2041506	113970	17.91266123	2041506	2430294	0.840024293
006.txt	4079	2284	1.785901926	4079	7305	0.558384668	4079	15659	0.260489176
007.txt	3410	1924	1.772349272	3412	6179	0.552192911	3412	12990	0.262663587
008.txt	7480	4519	1.655233459	7485	13604	0.550205822	7485	28576	0.261933091

	TOTAL FILE SIZE (MB)	TOTAL COMPRESS SIZE (MB)	AVERAGE COMPRESSION RATIO	NUMBERS OF HIGHEST COMPRESSION RATIO
HUFFMAN	61.05883217	32.26936436	2.187523593	35
LZW		45.93716908	3.104443496	5
RLC		220.0446148	0.434671994	0

- Ta có biểu đồ thống kê compression ratio giữa các phương pháp:



V. KẾT LUẬN:

- Từ các số liệu và biểu đồ thống kê khi thực hiện trên các bộ dữ liệu khác nhau, ta thấy rằng thuật toán của Huffman cho ra kết quả tốt nhất (35/40 kết quả compression tốt nhất), nguyên do vì Huffman coding dựa trên tần suất xuất hiện của ký tự trong văn bản, do đó tỉ lệ nén trả về tốt hơn hẳn so với LZW và RLC.
- Dù thuật toán LZW vẫn trả về vài kết quả compression ratio tốt nhất (5/40 tập tin), tuy nhiên các tập tin này đa phần là các tập tin tín hiệu (chỉ có một vài số và bit đặc trưng) do đó LZW thể hiện đặc tính vượt trội của mình (như đã nêu ở trên, ưu điểm khi văn bản có các cụm từ được lặp lại nhiều lần)
- Tuy nhiên, khi xét về thời gian thực thi các thuật toán thì thuật toán Huffman chiếm phần lớn thời gian hơn so với RLC và LZW – lí do là vì cách thức hoạt động của các thuật toán, cụ thể thời gian để compress 40 tập tin của cả 3 thuật toán là:
    - LZW: 3 phút.
    - RLC: 45 phút.
    - Huffman: 108 phút.

VI. TÀI LIỆU THAM KHẢO:

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

[2] https://en.wikipedia.org/wiki/Lempel–Ziv–Welch

[3] https://en.wikipedia.org/wiki/Huffman_coding

VII. THÀNH VIÊN NHÓM:

- Họ tên: Nguyễn Cao Minh – MSSV: 14520529
- Họ tên: Đoàn Trí Đức – MSSV: 14520178
- Material và Sourecode:

  https://github.com/bigredbug47/text_compression