Data Compression 과제 보고서

2018-28420 이혜수

구현언어: python 3.6.4
운영체제: OSX

- encoding run: python3 encoding.py infile.txt _com_infile

```
(venv) graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$ python3 encoding.py infile.txt _com_infile
(venv) graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$ ls -l
total 12904
-rw-r--r--  1 grace  staff      303 Nov 19 17:09 README.md
-rw-r--r--  1 grace  staff   951156 Nov 19 17:23 _com_infile
```

- decoding run: python3 decoding.py _com_infile _dec_infile

```
(venv) graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$ python3 decoding.py _com_infile _dec_infile
(venv) graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$ ls -l
total 15880
-rw-r--r--  1 grace  staff      303 Nov 19 17:09 README.md
-rw-r--r--  1 grace  staff   951156 Nov 19 17:23 _com_infile
-rw-r--r--  1 grace  staff  1521996 Nov 19 17:24 _dec_infile
```

- 원본 파일과 동일한지 확인

```
(venv) graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$ diff -w infile.txt _dec_infile
(venv) graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$
```

1. LZ78 구현에 관한 간단한 설명

Lossless 압축 기술은 크게 static, dynamic, hybrid로 나뉘게 되는데 LZ78의 경우, 한번의 스캔으로 압축을 진행하는 dynamic 압축 방식 중의 한가지이다.[1] LZ78은 overlapping되지 않는 하나 혹은 여러 캐릭터를 식별가능한 패턴화해서 사전에 추가하는 형식이다. 패턴의 prefix는 마지막 캐릭터를 제외한 전체 캐릭터가 된다.
따라서 LZ78의 output은 다음과 같이 출력되게 된다.[1]
만약 사전에 해당 1캐릭터 패턴이 존재하지 않는다면 -> (0, char)
만약 사전에 해당 multi-캐릭터 패턴이 존재하지 않는다면 -> (prefix_index_in_dictionary, last_char)
만약 사전에 존재한다면 -> (prefix_index_in_dictionary,)

사전 구성을 위한 기본 Trie 생성하기.

```
# aho-corasick때 구현한 trie와 동일한 상태로 배열을 활용하여 trie를 구성
# 총 알파벳의 갯수는 70이기 때문에 70열로 구성된 trie를 생성
# 총 가용가능한 알파벳을 index로 trie의 컬럼 index를 찾음

def get_default_trie(data):
    # 총 활용 가능한 알파벳은 = [a-zA-Z0-9 !?,.;:\n]
    pos_chars = [i for i in range(ord('a'), ord('z')+1)] + \
                [i for i in range(ord('A'), ord('Z')+1)] + \
                [i for i in range(ord('0'), ord('9')+1)] + \
                [ord('!'), ord('?'), ord(' '), ord(','), ord('.'), ord(':'), ord(';'), ord('\n')]
    available_char_dict = {}
    for i, c in enumerate(pos_chars):
        available_char_dict[chr(c)] = i

    # 초기에 사전은 -1값으로 전체 설정하여, 사전에 존재하지 않는 여부를 -1로 비교할 수
```

[1] http://faculty.kfupm.edu.sa/ICS/jauhar/ics202/Unit31_LZ78.ppt

```
있도록 설정
    default_trie = [[-1]*70 for _ in range(len(data))]
    return default_trie, available_char_dict
```

encoding을 진행하여, 사전을 구성하고 결과 값을 출력하기.

```
def encoding(data):
    # 기본 Trie구성 및 char 에 따른 컬럼 index를 찾을 수 있는 캐릭터 사전 구성
    trie, char_dic = get_default_trie(data)
    out = []

    present_state = 0
    data_state = 1

    key = ''
    for c in data:
        char = char_dic[c]
        # trie내 매칭 char가 없다면
        if trie[present_state][char] == -1:
            trie[present_state][char] = data_state
            out.append((present_state, c))

            # trie내 재 검색을 위해 리셋
            present_state = 0
            # 다음 노드 값
            data_state += 1
        # trie내 매칭 char가 있다면
        else:
            present_state = trie[present_state][char]

    # 사전에 존재한다면
    if present_state >0:
        out.append((present_state, ''))

    return out
```

예시로 'ABBCBCABABCAABCAAB' 를 입력데이터로 추가하였다면

1	2	3	4	5	6	7
A	B	BC	BCA	BA	BCAA	BCAAB

사전 및 결과 구성

사전		결과
index		
1	A	(0, 'A')
2	B	(0, 'B')

3	BC	(2-prefix B의 index, ‘C’)
4	BCA	(3-prefix BC의 index, ‘A’)
5	BA	(2-prefix B의 index, ‘A’)
6	BCAA	(4-prefix BCA의 index, ‘A’)
7	BCAAB	(6-prefix BCAA의 index, ‘B’)

얻어진 결과값을 가지고 다시 decoding을 진행 할 때는, 입력값에서 사전을 재구성하면서 동시에 결과값을 생성할 수 있다. 재구성 할 경우에는 prefix_index와 캐릭터를 가지고 재구성.

decoding을 통해 원본 데이터 복구

```
def decoding(data):
    # 사전을 재구성함과 동시에 결과값을 생성
    d_out = []
    for (n, c) in data:
        d_out.append(c if n == 0 else d_out[n-1] + c)
    return ''.join(d_out)
```

위의 encoding값이 입력으로 들어온 경우에, (0, A)(0, B)(2, C)(3, A)(2, A)(4, A)(6, B) -> 다음과 같이 구성할 수 있다.

결과	사전	
	Index	
A	1	A
B	2	B
BC	3	BC
BCA	4	BCA
BA	5	BA
BCAA	6	BCAA
BCAAB	7	BCAAB

2. **좋은 압축율을 얻기 위해서 사용한 방법**

1) ‘ABBCBCABABCAABCAAB’를 압축예시로 하였을 때, 원본 입력값을 그대로 저장하게 된다면 1캐릭터는 8bit를 차지하기 때문에 18*8bit = 144bit = 18byte를 차지하게 된다.
2) encoding결과를 데이터타입 변환없이 그대로 저장하게 되면 (0, A)(0, B)(2, C)(3, A)(2, A)(4, A)(6, B). 각 정수 8bit(python에서의 정수 기본 크기), 캐릭터 8bit를 차지하게되어 7*(16bit) = 112bit = 14byte를 차지하게 된다.
3) 정수가 차지하는 사이즈를 줄이기 위해서 최대값을 기준으로 bit 형식으로 저장하는 것이 가능하다.
   위의 결과를 0A0B10C11A010A100A110B 형태로 저장하는 것이다.

4) 초기에 사전 내 index값을 기준으로 bit사이즈를 지정하려하였으나, decompress할 때 어떤 형태로 변환이 되었는지 확인할 방법을 찾지 못해서 전체 결과값의 sequence index값을 기준으로 bit사이즈를 지정하였다. 결국 사전 내 index가 될수 있는 최대값은 결과 sequence index값과 같거나 작을 수 밖에 없다고 가정하였다.
5) 바이너리 형태로 저장하기 위해 python의 struct 패키지를 활용하였다. **struct의 pack(byte object로 패킹), unpack(tuple로 변환) 함수들을 활용하여 바이너리형태로 변환하고 다시 튜플형태로 변환한다.** 위의 함수를 통해 변환할 때, format strings를 제공하여 형식을 제공하게 되는데, sequence index는 양수만을 고려하기 때문에(0포함) unsigned char형태로 B, H, I만을 고려한다.
6) 따라서 sequence index를 기준으로 정수의 크기를 예상하여 각 1byte, 2byte, 3byte, 4byte로 저장한다. 정수는 캐릭터의 ascii code값 (1byte)와 합쳐서 결국 각 2byte, 3byte, 4byte, 5byte 형태로 저장한다.

max(sequence index)	format strings	size
255	**Bc**	2byte
65535	**Hc**	3byte
16777215	**Ic - Ic에서 마지막 byte를 제거**	4byte
그외	**Ic**	5byte

```
def compress(origin_file, compressed_file):
    import struct

    data = read_file(origin_file)
    encoded_data = encoding(data)

    with open(compressed_file, 'wb') as compressed_file:
        for i, (idx, ch) in enumerate(encoded_data):
            # 압축 된 파일의 마지막이 공백인 경우를 표기하기 위해
            save_char = ch.encode() if len(ch) > 0 else b'\x00'

            # sequence index가 256 (2^8 = 1 byte) 보다 작다면
            # 2 byte = 1 byte of idx and 1 byte of ascii code for char
            if i <= 255:
                data = struct.pack('Bc', idx, save_char)
                compressed_file.write(data)
            # sequence index가 65536 ((2^8)^2 = 2 byte) 보다 작다면
            # 3 byte = 2 byte of idx and 1 byte of ascii code for char
            elif i <= 65535:
                data = struct.pack('Hc', idx, save_char)
                compressed_file.write(data)
            # sequence index가 16777216 (((2^8)^2)^2 = 2 byte) 보다 작다면
            # 4 byte = 3 byte of idx and 1 byte of ascii code for char
            elif i <= 16777215:
                data = struct.pack('Ic', idx, save_char)
                compressed_file.write(b''.join([data[0:3], data[4:]]))
            # sequence index가 16777216 (((2^8)^2)^2 = 2 byte) 보다 크다면
```

```
        # 5 byte = 4 byte of idx and 1 byte of ascii code for char
        else:
            data = struct.pack('Ic', idx, save_char)
            compressed_file.write(data)
```

예시, infile.txt를 압축하면 다음과 같이 61%정도 압축할 수 있다.

```
graces-MacBook-Pro:[HW2]2018-28420_이혜수  grace$ ls -l
total 9712
-rw-r--r--  1 grace  staff      297 Nov 19 12:48 README.md
-rw-r--r--  1 grace  staff   951156 Nov 19 12:46 compressed_test
-rw-r--r--  1 grace  staff     6903 Nov 19 12:48 encoding.ipynb
-rw-r--r--@ 1 grace  staff  1555051 Nov 15 21:34 infile.txt
-rw-r--r--  1 grace  staff  1521996 Nov 18 16:53 test_out.txt
```

```
def decompress(compressed_file, decompressed_file):
    import struct
    bin_file = open(compressed_file, 'rb')
    bin_type = {2: 'Bc', 3: 'Hc', 4: 'Ic', 5: 'Ic'}

    encoded_data = []
    seq = 0

    while True:
        # sequence index가 256 (2^8 = 1 byte) 보다 작다면
        # 2 byte = 1 byte of idx and 1 byte of ascii code for char
        if seq <= 255:
            bin_size = 2
        # sequence index가 65536 ((2^8)^2 = 2 byte) 보다 작다면
        # 3 byte = 2 byte of idx and 1 byte of ascii code for char
        elif seq <= 65535:
            bin_size = 3
        # sequence index가 16777216 (((2^8)^2)^2 = 2 byte) 보다 작다면
        # 4 byte = 3 byte of idx and 1 byte of ascii code for char
        elif seq <= 16777215:
            bin_size = 4
        # sequence index가 16777216 (((2^8)^2)^2 = 2 byte) 보다 크다면
        # 5 byte = 4 byte of idx and 1 byte of ascii code for char
        else:
            bin_size = 5

        # sequence index값으로 지정된 bin_size를 기준으로 입력값을 읽어온다
        binary = bin_file.read(bin_size)

        # 파일 마지막 확인
        if binary == b'':
            break

        # bin_size 4 인 경우, 압축 시에 마지막 4번째 byte를 지웠기 때문에, 마지막에 \x00를
추가한다
```

```
        if bin_size ==4:
            encoded_data.append(struct.unpack(bin_type[bin_size], b''.join([binary[0:3], b'\x00',
binary[3:]])))
        else:
            encoded_data.append(struct.unpack(bin_type[bin_size], binary))

        seq += 1

    #압축 된 파일의 마지막이 공백인 경우에 추가한 내역을 삭제
    if encoded_data[-1][1] == b'\x00':
        encoded_data[-1] = (encoded_data[-1][0], b'')

    encoded_data = [(d[0], d[1].decode()) for d in encoded_data]
    decoded_data = decoding(encoded_data)

    with open(decompressed_file, 'w') as decompressed_file:
        decompressed_file.write(decoded_data)
```

decompression된 파일과 원본 파일을 비교하면 같은 것을 확인할 수 있다.

```
graces-MacBook-Pro:[HW2]2018-28420_이혜수  grace$ diff -w infile.txt decompressed
graces-MacBook-Pro:[HW2]2018-28420_이혜수  grace$
```

3. 구현한 LZ78과 다른 압축 프로그램과의 비교(첨부한 입력 파일 및 자신의 예제)

구현한 압축 알고리즘을 비교하기 위해 MacOS의 기본 압축 방식(PKZIP 2.0 encryption)[2]와 linux의 기본 압축 방식(GZIP)을 활용하였다.

	size(bytes)	**구현 압축방식**	**MacOS**	**Linux**
주요 implementation		LZ78 + 정수 저장 byte manipulation	PKZIP 2.0	GZIP
infile.txt	1555051	951156 61%	651763 41%	625413 40%
titles_condition.csv (example1.txt)	2400075	1221296 51%	712525 30%	681174 28%
aladdin.txt (example2.txt)	9664	7820 80%	4906 50%	4355 45%
aesops_fables.txt (example3.txt)	1331814	730932 55%	518776 39%	498138 37%

3. 결과 분석

1) encoding결과

[2]https://security.stackexchange.com/questions/186128/what-encryption-method-is-used-by-the-zip-program-in-macos

```
%%timeit

encoding(read_file('./infile.txt'))
```

```
1.61 s ± 41.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
(0, 'T'), (0, 'h'), (0, 'e'), (0, ' '), (0, 'P'), (0, 'r'), (0, 'o'), (0, 'j'), (3, 'c'), (0, 't'), (4, 'G'), (0, 'u'), (10, 'e'), (0, 'n'), (0, 'b'), (3, 'r'), (0, 'g'), (4, 'E'), (0, 'B'), (7, 'o'), (0, 'k'), (4, 'o'), (0, 'f'), (4, 'U'), (0, 'l'), (0, 'y'), (0, 's'), (27, 'e'), (27, ','), (4, 'b'), (26, ' '), (0, 'J'), (0, 'a'), (0, 'm'), (3, 's'), (4, 'J'), (7, 'y'), (0, 'c'), (3, '\n'), (0, '\n'), (1, 'h'), (0, 'i'), (27, ' '), (3, 'B'), (20, 'k'), (4, 'i'), (43, 'f').....
```

예제 file	file size (bytes)	encoding time
infile.txt	1555051	1.61 s ± 41.9 ms
titles_condition.csv (example1.txt)	2400075	2.76 s ± 43.6 ms
aladdin.txt (example2.txt)	9664	8.02 ms ± 238 µs
aesops_fables.txt (example3.txt)	1331814	1.42 s ± 17.2 ms

2) encoded된 내용을 decoding하는 결과

```
%%timeit

decoding(encoded)
```

```
128 ms ± 4.5 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
print(decoding(encoded))
```

```
The Project Gutenberg EBook of Ulysses, by James Joyce

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
reuse it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: Ulysses

Author: James Joyce

Posting Date: August 1, 2008 EBook 4300
Release Date: July, 2003
Last updated: November 17, 2011

Language: English
```

예제 file	file size (bytes)	decoding time
infile.txt	1555051	128 ms ± 4.5 ms
titles_condition.csv (example1.txt)	2400075	165 ms ± 2.68 ms
aladdin.txt (example2.txt)	9664	682 µs ± 14.7 µs
aesops_fables.txt (example3.txt)	1331814	96.8 ms ± 1.74 ms

3) diff infile.txt outfile.txt: 예제로 decoding 결과를 저장한 test_out.txt과 infile.txt가 동일한 것을 확인 할 수 있다.

```
graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$ diff -w infile.txt test_out.txt
graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$
```

4) 압축 시간 결과

예제 file	file size (bytes)	compression + encoding time
infile.txt	1555051	2.03 s ± 57.2 ms
titles_condition.csv (example1.txt)	2400075	3.18 s ± 92.2 ms
aladdin.txt (example2.txt)	9664	9.74 ms ± 150 µs
aesops_fables.txt (example3.txt)	1331814	1.64 s ± 24.3 ms

5) 압축 푸는 시간 결과

예제 file	file size (bytes)	decompression + decoding time
infile.txt	1555051	418 ms ± 16.4 ms
titles_condition.csv (example1.txt)	2400075	545 ms ± 18.4 ms
aladdin.txt (example2.txt)	9664	3.19 ms ± 37.4 µs
aesops_fables.txt (example3.txt)	1331814	325 ms ± 10.3 ms

압축 해제하여 decoding한 결과가 원본 파일과 동일한지 확인


graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$ diff -w infile.txt decompressed
graces-MacBook-Pro:[HW2]2018-28420_이혜수 grace$
graces-MacBook-Pro:sample1 grace$ diff -w titles_condition.csv decompressed_titles
graces-MacBook-Pro:sample1 grace$
graces-MacBook-Pro:sample2 grace$ diff -w aladdin.txt decompressed_aladdin
graces-MacBook-Pro:sample2 grace$
graces-MacBook-Pro:sample3 grace$ diff -w aesops_fables.txt decompressed_aesopes
graces-MacBook-Pro:sample3 grace$