숙제 4 : 압축 알고리즘 X

<Due : 2020-06-20 23:59>

압축 알고리즘은 어떤 input string 이 있을 때, input string을 더 작은 크기의 string 으로 표현하는 알고리즘을 뜻하며, 우리에게 익숙한 zip 파일에서 쓰이는 bzip 을 포함하여 Huffman, LZW 등등 여러가지 압축 알고리즘들이 있다. 마지막 숙제에서는 지금까지 배운 자료구조의 지식을 총동원 하여 무손실 압축 (lossless compression) 알고리즘 중 하나인 **알고리즘 X** 를 구현해 보는 것을 목표로 한다. (lossless compression 이란, **압축한 string 으로부터 정확하게 input string을 복구 (decompression) 가능**한 알고리즘을 뜻한다).

주의 : Input string 이 무엇이냐에 따라 같은 압축 알고리즘이라도 성능은 천차만별이며, 심지어 input string 보다 크기가 늘어날 수도 있다. 테스트 시 원본 string 보다 사이즈가 늘어났을 경우에도 당황하지 말자.

알고리즘 X 는 bit string (0과 1만으로 이루어진 string) 을 압축하는 알고리즘으로서 i) bit string을 여러개의 substring 의 조각들로 (예 : 0001110011 → 0001 + 110 + 011) 분리 (partitioning) 한 다음, ii) **각각의 substring 조각들을 code (코드) 라 하는 또 다른 bit string 으로 나타내는 과정**을 거친다. 알고리즘 X 를 이용한 압축 과정은 다음과 같다.

과정 1. input string 전체를 scan 하여 d0 및 d1 (input string에서 0 과 1 의 빈도) 을 계산한다. 여기서

d0 = (input string 에서 0의 개수) / (input string 크기)

d1 = (input string 에서 1의 개수) / (input string 크기)

로 정의된다. 예를 들어 input string 이

0000011000100 인 경우 d0 = 10/13 ~ 0.769 , d1 = 3/13 ~ 0.231 이 된다 (편의상 **본 과제 문서에서 나오는 double 형 값들은 모두 소수 넷째자리에서 반올림한 것으로 표기 (실제 프로그램 작성시에는 따로 변경하지 말 것)**).

앞으로 계속해서 이 input string 을 예제로 활용할 것이다.

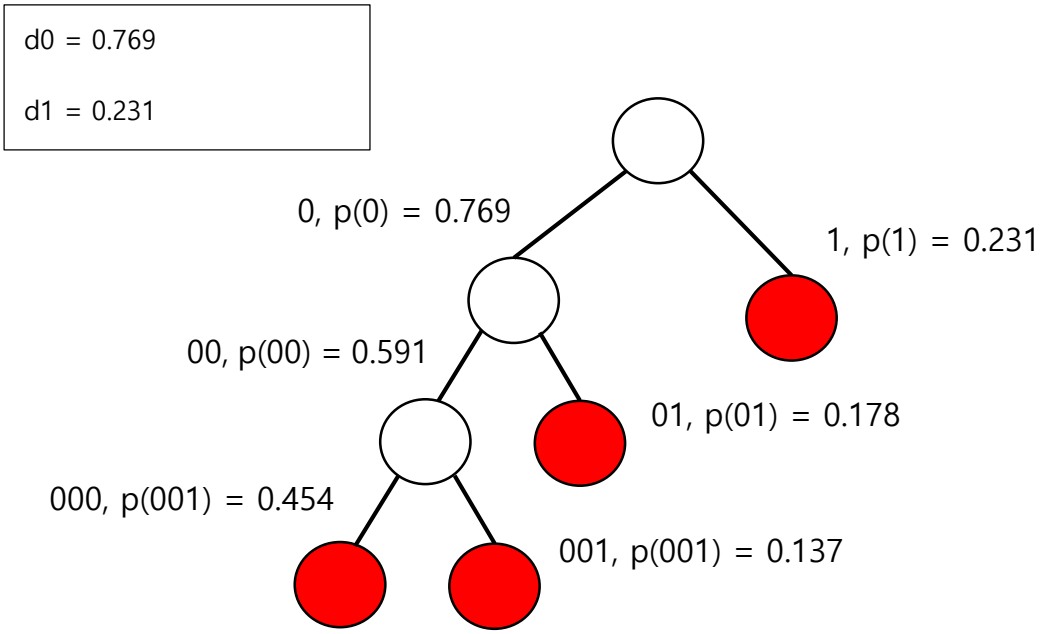
과정 2. (핵심 과정) 1에서 얻은 d0, d1 을 바탕으로 input string을 압축하기 위한 코드북 (code book) 을 생성한다.

Code book 이란 각각의 code 와 이에 대응하는 string 들을 정리한 table 을 뜻한다 (이 때 각 code 와 string 은 1 대 1 대응관계이다). 알고리즘 X 에서는 **각각의 code 의 길이가 모두 똑같으며** (이 때 각 code에 대응하는 substring의 길이는 다르다는 점에 주의하자), code 의 길이는 값 b로 미리 주어진다. Code 는 bit string 이므로 눈치가 빠르다면 code book 이 가질 수 있는 최대 code 수는 2^b 개가 된다는 것을 바로 알 수 있다 (예를 들어 b=2 이면 총 00, 01, 11, 10 4개의 code 가 가능). 이제 2^b 개의 code 를 어떻게 생성하는지 알아보자.

우선 input string 에서 구한 d0, d1 이 이미 주어졌을 때, string s 에 대한 **$p(s) = (d0)^{(s \text{ 에서 } 0 \text{의 개수})} \times (d1)^{(s \text{ 에서 } 1 \text{의개수})}$** 로 정의하자. 예를 들어 앞에서 구한 d0, d1 을 이용한 $p(110) = (0.769) \times (0.231)^2 \sim 0.137$ 이 된다. 이제 **binary tree Xtree** 를 생성한다. Xtree 는 총 leaf 의 수가 2^b 개인 binary tree 로서 **각각의 leaf node 가 하나의 code 에 대응되는 string 을 저장**한다. Xtree 는 다음과 같이 생성한다.

1. Root node 하나에 왼쪽 leaf 에는 string 0을 저장, 오른쪽 leaf 에는 string 1을 저장.
2. 이제 tree 의 leaf node 가 n개 될때까지 A, B 과정을 반복한다.
   A. 현재 leaf node 들 중 leaf node 에 대응되는 string s 에 대한 p(s) 의 **값이 제일 큰 node를 선택**
   B. 해당 node 의 left child, right child 를 생성한다. 이 때 left child 에 대응하는 string 은 s0 (s 뒤에 0 추가), right child 에 대응하는 string 은 s1(s 뒤에 1 추가) 가 된다. 이 과정을 통해 **leaf node 의 수가 정확히 1개 증가한다**.

예를 들어 b=2 일 때 예제 string 에 대한 Xtree 는 다음과 같다.



생성한 Xtree 를 바탕으로 이제 code book 을 생성한다. 앞서 말한대로 총 2^b개의 code 가 있으며 각 code 들은 0^b (0이 b개) ~ 1^b (1이 b개) 로 구성된, size 가 b인 모든 bit string 들이다. **이 code 들 중 lexicographical order (사전적 순서) 가 i 번째인 code 는 Xtree 의 leaf node 에 대응하는 string 들 중 i 번째 lexicographical order 에 해당하는 string 에 대응한다**. 예를 들어 위 예제에서 code 00 은 000 에, 01은 001 에, 10 은 01 에, 11 은 1 에 대응한다. 이 code book 은 xcode.h 에 전역 변수로 정의된 array **char * code_book[2^b]** 에 다음과 같이 저장 된다.

code_book[0] = 000, code_book[1] = 001, code_book[2] = 01, code_book[3] = 1.

code_book[i] 에는 모든 code들 중 lexicographical order 가 i 번째인 code 에 해당하는 string 을 저장한다는 것을 알 수 있다.

과정 3. 과정 2에서 생성한 code book 을 이용하여 input string 을 압축

압축하는 과정은 상대적으로 간단하다. Input string 을 맨 왼쪽부터 linear scan 하면서 code book 에 있는 어떤 code c 에 대응되는 string 을 발견할 때마다 해당 string 을 code c 로 변경해 준다 (Xtree를 이용하면 비교적 쉽게 구현할 수 있다). 예를 들면 위 예제인 0000011000100 의 경우 000 | 001 | 1 | 000 | 1 | 00 으로 나눠질 수 있고, 맨 뒤 00을 제외한 나머지 부분들은 code book 상에 이에 대응하는 code 들을 붙인 0001110011 로 나타낼 수 있다 (01303 이 아닌 것에 주의

하자!). 해당하는 code 가 없는 맨 뒤의 0 같은 경우 **'uncompressed string' (압축 안된 string) 이라 하여 따로 관리**해 준다. 압축된 결과물은 다시 file 로 출력하며, 이 때 file 형식은 다음과 같다.

```
압축된 string
Uncompressed string
```

압축된 string 과 uncompressed string 는 하나의 new line 문자로 구분한다. 따라서 위 예제의 최종 결과물을 저장하는 파일은 다음과 같다.

```
0001110011
00
```

<압축 해제 (decompression) 하기>

각 code 의 크기 b가 주어진 상태에서 압축한 file 과 codebook 이 주어지면 압축 해제는 매우 쉽다. 단순히 압축된 string 을 왼쪽부터 b 개씩 읽으면서 해당되는 code 에 대응하는 code book 상의 string 으로 변환해 주면 끝난다. 단 맨 마지막에 uncompressed string 을 붙이는 것을 잊지 말자!.

<주어지는 file 에 대한 간략한 설명>

xcode.h 에서는 다음과 같은 구조체가 정의되어 있다.

```
typedef struct xnode {
    char* string;
    double p;
   struct xnode * leftchild;
   struct xnode *rightchild;
}xnode;
```

→ Xtree 의 각 node 를 표현하는 구조체 (여기서 p = p(string) 를 뜻함).

각 code 의 길이를 나타내는 b 또한 동일 파일에 정의되어 있으며, 테스트 시 b=8 및 16 의 경우에 대해 test 할 예정.

마지막으로 에러 메세지를 출력하는 print_error() 함수가 정의되어 있다.

- 보조 자료 구조로 max heap-based priority queue 와 (heap.h에 정의) array-based stack (stack.h 에 정의) 이 주어지며, priority queue 및 stack item 으로는 xnode 가 (priority queue 에서 priority 는 각 node 의 p 값이 되며 **p값이 클수록 priority 가 높다**) 저장된다.

<xcode.c 에서 구현해야 하는 함수들>

1. xnode* construct_xtree (char * file) : input file "file" (따옴표 제외) 을 읽어서 이에 해당하는 Xtree 를 return 하는 함수. 앞서 설명한대로 file 은 bit string 만을 허용하며, new line character 는 존재하지 않는다 (즉 한 줄로 이루어진 파일이다). 0, 1 외에 다른 문자가 존재 하면 print_error() 를 호출하고 NULL 을 return 한다.

2. void codebook (xnode *xtree) : 주어진 Xtree 에 알맞게 전역 변수로 선언된 code_book array 의 값을 저장해 준다. 이 때 **생성된 array를 바탕으로 새로운 code book file codebook.txt 를 생성 및 저장 한다**. Code book file 은 code book array 을 저장한 file 로서 정확한 구성은 다음과 같다.

1. Code book file 은 2^b 줄로 이루어진 파일이다.
2. Code book file 의 i-번째줄은 **lexicographical order (사전적 순서) 가 i 번째인 code 와 그에 해당하는 string 을 표시한다**
3. **code 와 대응하는 string 사이는 단일 space 로 구분한다.**

앞선 input string 에 대한 codebook file 의 예시는 다음과 같다.

00 000 01 001 10 01 11 1

3. void compress_x (xnode *xtree, char *file) : 인자로 주어진 Xtree 와 전역 변수로 주어진 array code_book, 그리고 input file "file" (따옴표 제외) 을 이용하여 X 알고리즘의 결과물 output.txt 를 생성 및 저장하는 함수 (파일 양식은 위에 설명을 참고할 것). 파일을 저장한 뒤 compression ratio (압축률) 을 출력하는데 여기서 compression ratio 는 **(output 의 첫째줄 길이 + 둘째줄 길이) / input 파일 길이** 로 정의한다. 예를 들어 위 예제의 경우 compression ratio 는 (10+2)/13 ~ 0.923 이 된다. 이 경우 output.txt 저장한 뒤 다음과 같이 출력한다 (출력 시 **소수 넷째자리에서 반올림**).

Compression ratio : 0.923

4. void decompress_x (char *output, char *codebook) : 알고리즘 X 로 압축된 결과 파일 output 과 codebook file codebook 을 읽은 다음 이들을 이용하여 원본 input file 을 생성 및 저장하는 함수. 이 때 파일 이름은 input_original.txt 로 한다. Codebook file 에 없는 code가 output 에 있는 경우 print_error() 를 호출하고 그대로 종료한다.

3. 주의 **(이 중 하나라도 어기면 0점)**

- 주어진 파일은 다음과 같은 형식으로 되어 있음

1. xcode.h : xcode.c 에서 정의한 함수들을 모아둔 헤더파일.

2.**: xcode.c : xocde.h 에서 정의한 함수들의 세부 구현 (숙제에서 제출해야 할 파일).**

3. stack.h stack.c : Array-based Stack 구현 파일

4. heap.h, heap.c : Max-heap based priority queue 구현 파일

5. main.c : 테스트용 main file (결과값이 주석에서 언급한 답과 일치하는지 확인할 것.

6. 리눅스, 맥 환경에서 작업하는 학생들을 위한 Makefile.

7. input.txt : 본 문서 예제로 사용된 input 파일

8. Z-Pumsb2.txt.bz2 : 테스트용 대용량 input 파일 (압축 해제시 1.6GB 정도 차지)

- 숙제 제출은 due 전까지 e-campus의 과제탭의 **과제물 제출 – 파일 첨부를 이용**하여, 오직 **xcode.c 파일만을 학번_xcode.c 파일로 업로드** 할 것. (예 : 2019000000_xcode.c). **그 외 어떠한 파일도 받지 않음**.

- 숙제의 delay는 받지 않음.

- 문제의 skeleton code 에서 주어진 헤더 파일 외의 **어떠한 헤더 파일도 include 하지 말 것**.

- 주어진 코드에 정의된 함수 및 전역번수만을 구현 및 사용 하고, **자기가 따로 함수나 전역변수를 정의하여 xcode.c 에 추가하지 말 것 (**채점 시 xcode.c 제외 원래 주어진 파일들로 채점한다는 것을 잊지 말 것). **또한 모든 C 함수는 반드시 표준 C함수 (standard C) 함수 및 constant 들만 사용할 것** (Visual studio 에서만 정의된 C 함수를 사용시 0점).

Standard C 헤더 파일 및 함수, constant 목록은 (https://en.wikipedia.org/wiki/C_standard_library) 에서 확인 가능

- 채점 시 GCC 7.3.0 (컴파일 시 과제에서 주어진 Makefile 이용), 및 Visual studio 2019 에서 채점할 예정이며, 둘 중 한곳에서 컴파일이 안될 시 무조건 0점.

- 테스트 시 Z-Pumsb2.txt와 같이 매우 큰 size (10억 이상) 를 가진 input file 로 테스트할 예정이므로 구현 시 해당 부분을 고려할 것 (hint : input file 을 메모리상에 통째로 저장할 시 문제가 발생할 수 있음). 또한 일반적인 PC 환경에서 **Z-Pumsb2.txt 파일 기준으로 main 프로그램을 실행 후 완전히 종료할 때까지 (압축 및 압축해제) 5분이상 걸리지 않도록 구현할 것**.

- 제출하기 전에 예제로 주어진 main.c 파일을 컴파일 및 실행해서 실제로 올바르게 출력되는지 확인할 것. 해당 파일을 컴파일 후 실행 시 codebook.txt, output.txt, input_original.txt 세 가지 파일이 생성되며, input_original.txt 와 원본 input 파일의 내용은 정확히 일치하여야 함.