SSE2030: Introduction to Computer Systems (Fall 2010)

Programming Assignment #1:

Implementation of the X-match compression algorithm

Due: October 3, 11:59PM.

1. Introduction

이 과제는 파일 압축 함수를 C 언어를 이용하여 구현하는 것이다. 이 과제의 목적은 무손실 압축기법의 일종인 X-Match 알고리즘 구현을 통하여 파일 압축방법에 대해 이해하고, C 언어를 이용한 bit-level manipulation 방법을 학습하는데 있다.

2. Problem Specification

2.1. Overview

다음과 같은 원형을 갖는 `xmatch()` 함수를 구현한다. `xmatch()` 함수는 2개의 `char` 포인터(`input`, `output`)와 첫 번째 `char` 포인터가 가리키는 바이트열의 크기(`len`)를 인자로 받고, `xmatch()` 함수 실행을 통해 얻은 압축된 파일의 크기를 반환한다.

```
int xmatch (char* input, int len, char* output);
```

첫 번째 인자인 `char* input`은 압축이 될 바이트 배열의 시작주소를 가리키며, 그 바이트열의 길이는 두 번째 인자인 `len`을 통해 입력된다. 이 때 입력되는 바이트의 크기는 4의 배수로 가정하고, 최대 크기는 65532를 넘지 않는 것으로 한다. (4의 배수가 아니거나, 0 <= `len` <= 65532 이 아닌 경우 음수를 리턴, `len` == 0 인 경우에는 0을 리턴)
첫 번째 인자로 입력된 `input`을 X-Match 알고리즘을 통하여 압축하고 난 뒤, 세 번째 인자인 `output`이 가리키는 배열에 기록하고, 최종 압축 후의 바이트 수를 리턴 값으로 반환한다. 세 번째 output 인자가 가리키는 곳에는 `xmatch()` 함수를 호출하기 전 최악의 경우 필요한 크기를 가정하여 메모리를 이미 할당해 놓았기 때문에 `xmatch()` 함수 내에서 메모리 공간을 새롭게 할당할 필요는 없다.
`xmatch()` 함수 내에서는 입력 받은 4바이트 * N (N은 0 이상의 정수) 크기의 데이터를 4바이트씩 잘라서 압축을 수행한다. (자세한 사항은 2.2절을 참조)
함수의 실행 결과, 오류 없이 압축이 성공했을 경우 최종 압축 후의 바이트 수를, 오류가 발생했을 경우 음수를 반환한다.

본 과제 수행에 있어서 별도의 함수를 생성할 필요가 있는 경우, `xmatch.c` 파일 내에 구현할 수 있다.

2.2. Background

2.2.1 전반적인 구조 설명

- 4바이트 * N (N은 0 이상의 정수) 크기의 입력이 주어지면, 4바이트씩 잘라 아래 알고리즘을 활용해 압축을 진행한다.
- 4바이트 크기의 tuple이 입력되면, dictionary를 검사하여 2바이트 이상 매치가 되는 entry가 있는지 찾는다.
- 매치되는 entry가 있을 경우 <0 (1 bit), dictionary의 Match Location (7 bits), Match Type (4 bits), match 되지 않은 바이트 (0 – 2 bytes)>를 순서대로 기록한다.
- 이 때, 매치되는 entry가 복수개 존재 할 경우, Match Location의 선택은 일치하는 바이트가 많은 순서로, 일치하는 바이트가 같을 경우 dictionary의 상위에 있는 tuple을 우선순위로 선택한다.
- 매치되는 entry가 없을 경우 1로 표시(1 bit)하고, 입력된 4바이트를 그대로 기록한다.
- 이 때, 첫 번째 tuple일 경우 1은 입력하지 않는다. 첫 번째 tuple의 경우 항상 dictionary에서 찾을 수 없는 miss 상태가 되기 때문에 1을 굳이 쓸 필요가 없기 때문이다.
- Dictionary와 비교한 tuple이 4바이트 모두 일치하는 full hit일 경우, 다음 dictionary 검색 시 속도를 빠르게 하기 위해 dictionary에서 해당 entry를 최상위로 이동시키고, 2바이트나 3바이트가 일치하는 partial hit일 경우 해당 entry를 최상위에 추가한다.
- X-Match 압축 알고리즘을 통해 압축한 데이터는 `xmatch()`에 세번째 인자로 주어진 `output`의 3번째 바이트(즉, `output[2]`번째 주소)부터 기록한다. 그리고 `output[0]`과 `output[1]`에는 `xmatch()` 함수 호출시 두번째 인자로 주어진 바이트열의 크기를 unsigned short 형으로 저장한다. (이에 관한 자세한 내용은 2.2.5 절을 참조)
- X-Match 알고리즘에 대한 보다 더 자세한 내용은 다음의 논문을 참고한다. (주: 본 과제에서 구현하는 X-Match 알고리즘은 논문에서 제시된 알고리즘의 간략화된 버전이다.)

  1. M. Kjelso, M. Gooch, and S. Jones, "Design and Performance of a Main Memory Hardware Data Compressor," *Proceedings of the 22nd EUROMICRO Conference*, PP.324-430, Sep. 1996

  2. S. Jones, "Partial-Matching Lossless Data Compression Hardware," *IEE Proceedings – Computers and Digital Techniques*, vol. 147, no.5, pp.329-334, Oct.2000.

2.2.2 Algorithm

- X-Match는 Dictionary 기반의 무손실 데이터 압축 기법의 일종으로 X-Match의 전체적인 알고리즘은 아래와 같다.

```
clear the dictionary;
set the next free location (NFL) to 0;
DO
{
    read in a tuple T from the data stream;
    search the dictionary for tuple T;
    IF (full or partial hit)
```

```
    {
        determine the best match location ML and match type MT;
        output `0';
        output binary code for ML;
        output binary code for MT;
        output any required literal characters of T;
    }
    ELSE
    {
        IF (T is not the first tuple)
            output `1';
        output tuple T;
    }
    IF (full hit)
        move dictionary entries 0 to ML - 1 down by one location;
    ELSE
    {
        move all dictionary entries down by one location;
        increment NFL (if dictionary is not full);
    }
    copy tuple T to dictionary location 0;
}
WHILE (more data is to be compressed);
```

2.2.3 Dictionary

- X-Match 에서 사용하는 dictionary는 4바이트 배열 N개로 이루어져있다. (N: 0~127)
- 매치는 입력된 tuple과 dictionary 데이터의 4바이트 중 2바이트 이상이 같은 경우 이루어진다. 그 이외의 경우는 miss가 발생한다.
- 복수개의 매치가 이루어진 경우, Match Location의 선택은 매치된 바이트 수가 많은 순서로, 매치된 바이트 수가 같을 경우 dictionary의 상위에 위치하는 tuple을 우선순위로 선택한다.
- 최초 dictionary의 크기는 0부터 시작하고, partial 매치가 이루어지거나 miss가 발생했을 때마다 1개씩 쌓아서 최대 128개까지 입력시키고, full match 가 발생했을 경우 해당 entry를 최상위로 이동시킨다. dictionary가 꽉 찼을 경우 최상위에서부터 한 칸씩 미루고 난 뒤에(최하위에 있는 entry를 버리고) 최상위에 entry를 등록한다.
- Dictionary의 Match Location 값은 최상위를 0, 최하위를 127로 Binary 7자리 숫자로 표현한다.

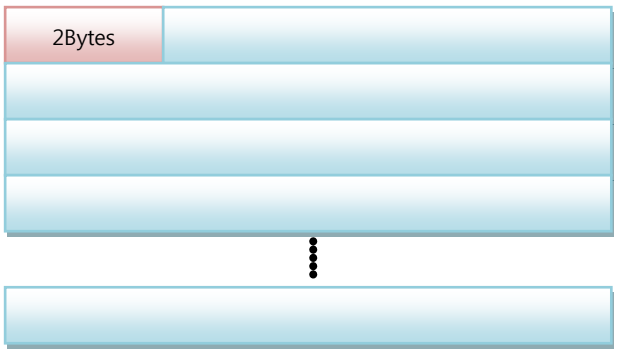
2.2.4 Match Type

- 앞서 설명한 것과 같이 match는 dictionary의 entry와 tuple이 2바이트 이상이 매치되었을 경우만 발생한다.
- 이 경우, 매치된 바이트들이 dictionary의 entry들 중 어느 부분에 매치되었는지 저장할 공간이 필요한데 이것이 Match Type이다.
- Match된 위치의 bit를 1로 mask하는 것을 기본으로 하여 Match Type은 아래와 같이 정의한다. (O은 해당 바이트 위치에 match가 일어난 경우, X는 그렇지 않은 경우를 의미함)

Byte Position				Match Type
1	2	3	4	
O	O	X	X	**1100**
O	X	O	X	**1010**
O	X	X	O	**1001**
X	O	O	X	**0110**
X	O	X	O	**0101**
X	X	O	O	**0011**
O	O	O	X	**1110**
O	O	X	O	**1101**
O	X	O	O	**1011**
X	O	O	O	**0111**
O	O	O	O	**1111**

2.2.5 최종 압축된 파일의 구조

- xmatch() 함수를 통해 압축된 데이터의 결과는 아래와 같은 형태가 된다.



: 압축 전 `input` 바이트열의 크기가 담겨있는 파일 헤더

: 압축된 파일 데이터

- 위 그림과 같이 최종 압축 결과의 상위 2 바이트 헤더에는 압축 전 바이트열의 길이(즉, `xmatch()` 함수의 두번째 인자 `len`)가 `unsigned short` 형으로 입력된다.
- 헤더의 바이트 순서에 주의한다. 원본 데이터의 길이를 나타내는 16 비트 중, 하위 8 비트는 `output[0]`의 위치에, 상위 8 비트는 `output[1]`의 위치에 저장되어야 한다.

- 결론적으로 본 압축 알고리즘은 최대 65532 바이트의 입력 바이트열을 한 번에 압축할 수 있다.

2.2.6. Example

- 입력으로 "abcdaecdabcdbdea"가 주어졌을 때 각 단계에서의 Output 은 아래와 같다.
- 첫 번째, abcd 가 입력으로 주어진 경우
  dictionary 에 데이터가 없고, 첫 번째 tuple 이기 때문에 abcd 가 그대로 출력된다.
  그리고, dictionary 에 입력된 tuple 이 추가된다.

- 두 번째, aecd 가 입력으로 주어진 경우
  dictionary 에 abcd 가 있고, 이 경우 첫 번째, 두 번째, 네 번째에서 partial matching 이 일어나므로 0, match location 은 0000000, match type 은 1011, 그리고 match 되지 않은 두 번째 문자가 출력된다.
  이 경우 마찬가지로 입력된 tuple 이 dictionary 에 추가된다.

- 세 번째, abcd 가 입력으로 주어진 경우
  dictionary 의 두번째 entry 에 abcd 가 있고, 이 경우 full matching 이 일어나므로 0, match location 은 0000001, match type 은 1111 이 된다.

  Full match 가 일어난 경우, dictionary 에서 해당 entry 를 최상위로 이동시킨다.

- 네 번째, bdea 가 입력으로 주어진 경우
  dictionary 에 match 가 일어나는 경우가 없으므로, 1 과 입력된 tuple bdea 가 바로 출력된다.

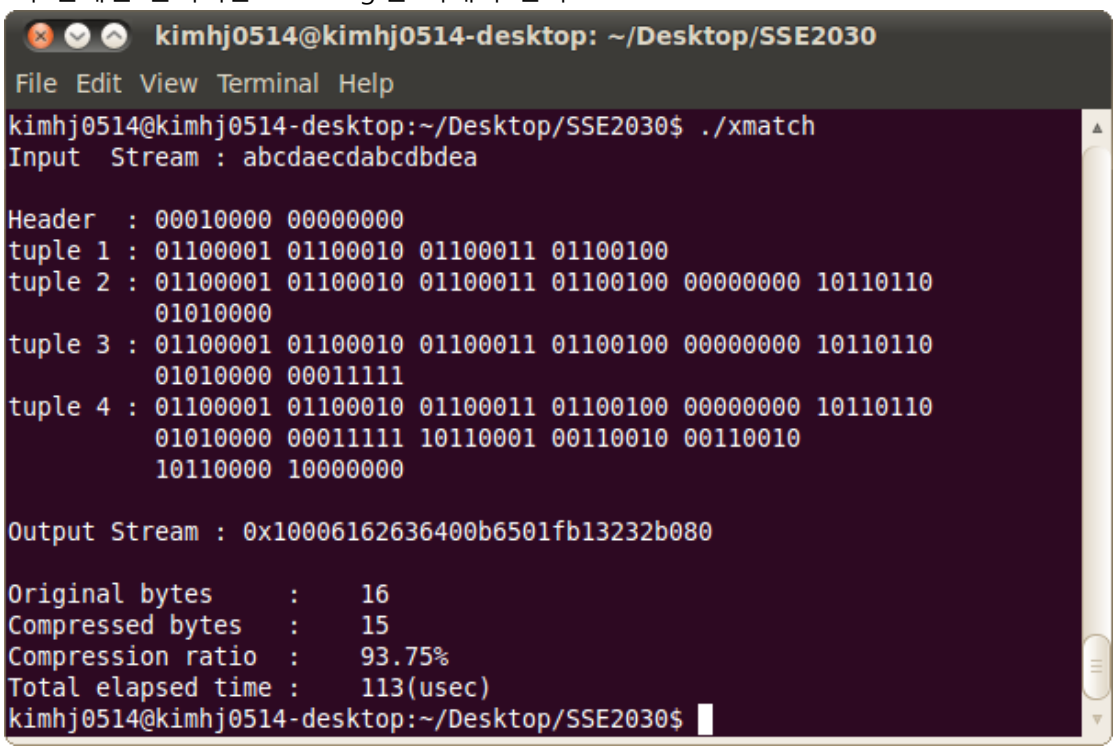
입력 전 Dictionary

abcd
aecd

입력 후 Dictionary

bdea
abcd
aecd

- 각 단계별 출력되는 bit string 은 아래와 같다.



2.3. Restrictions

- `xmatch()` 함수 내에서는 디버깅을 위한 `printf()` 함수 외에 다른 라이브러리 함수 사용이 불가능하다. 단, 필요에 따라 새로운 함수를 작성하여 사용하는 것은 무방하다.

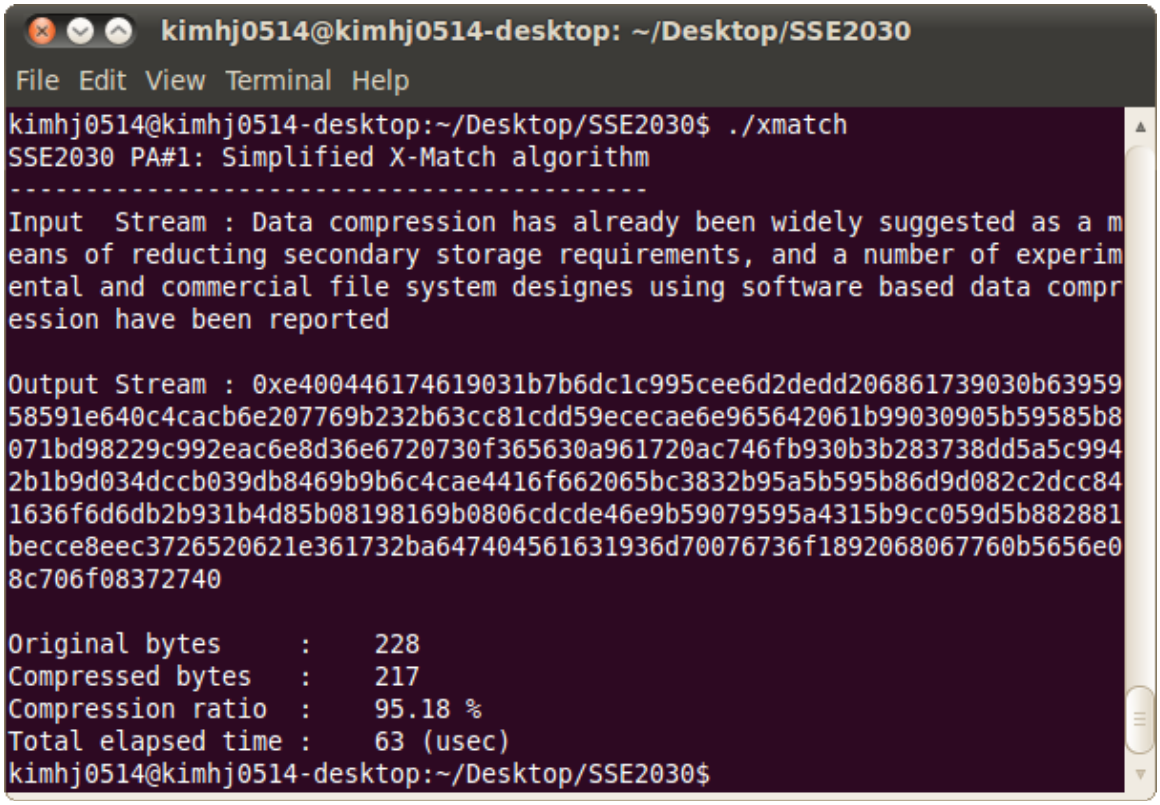
3. Skeleton Codes

본 과제 수행을 위하여 아래와 같이 3개의 파일이 주어진다.

`Makefile`:	GNU `make` utility를 위해 사용되는 파일.
`main.c`:	`xmatch()` 함수를 호출하는 `main()` 함수
`xmatch.c`:	`xmatch()` 알고리즘을 구현해 넣을 파일

과제를 위해서는 `xmatch.c` 파일만을 수정하면 된다. (`xmatch.c` 외에 다른 파일을 추가하지는 말 것)

4. Sample output



5. Hand in instructions

- 작성한 프로그램 코드 상단에 이름과 학번을 주석으로 넣는다.
- 과제 제출시 `xmatch.c` 파일을 "학번.c" 파일로 이름을 변경하여 제출한다 (예: `2009310123.c`)
- 프로그램의 설계, 구현에 관한 내용을 담은 보고서를 별도로 제출한다. 보고서는 워드, 한글 등의 형식도 상관 없지만 가능하면 PDF format으로 제출하고, 제출시 "학번.pdf" 형태로 파일을 저장하여 제출한다.
- 과제 제출은 프로그램 소스 코드인 "학번.c"와 보고서인 "학번.pdf"를 메일로 첨부하여 sse2030@csl.sku.edu 로 보낸다. 과제 제출시 메일의 제목은 아래와 같은 형식을 따른다.

  [SSE2030] PA#1, 학번, 이름

6. Logistics

- 과제 제출 결과는 본 수업의 홈페이지인 http://csl.skku.edu/SSE2030F10 에서 확인할 수 있다.
- 과제 제출 시간은 메일 도착시간을 기준으로 하며, 기한을 넘겨 제출할 경우 하루에 25% 씩 감점한다.
- 과제에 대한 의논은 함께 할 수 있으나, 프로그램 소스코드 작성은 스스로 해야 한다.
- 다른 사람의 과제를 copy한 경우, 두 사람 모두 0점 처리한다. 인터넷 등에서 찾은 소스 코드를 그대로 copy한 경우에도 0점 처리한다. 두 번 이상 이와 같은 이유로 0점 처리된 경우 F 학점을 받을 수 있다.

Good luck!

담당조교: 김형준, 031-299-4970, #400629 (hjkim at csl.skku.edu)