CMSC5714 Multimedia Technology

Programming Assignment

LZW Compression Program

Deadline: 18th March, 2015 23:59

I. Introduction

File compression/decompression is a common task used by people in computer. In 1984, Terry Welch proposed a compression algorithm, *the LZW algorithm*. This algorithm is based on the LZ77 algorithm, which is proposed by A. Lempel and J. Ziv in 1977. Similar to LZ77 and LZ78, LZW is a dictionary based compression algorithm and does not perform any analysis on the input text. LZW is widely known for its application in GIF and in the V.42 communication standard. The idea behind the LZW algorithm is simple but there are implementation details that one needs to take care of. In this assignment, you are required to write a variant of the LZW algorithm. The following sections explain the details.

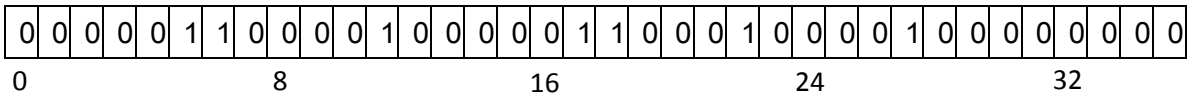
II. Requirements

In this programming assignment, you should implement a LZW compression/decompression program in either *C/C++* or *JAVA* programming language. Please refer to the lecture notes for LZW technical details. The program should support the followings:

1. **Compress file** – The compressor should be able to compress a file of any length. We required using *12-bit codeword dictionary* in this assignment. The initial dictionary

contains 256 entries (ASCII characters) with index from 0 to 255. During compression, when the word dictionary is full (i.e., after adding a new dictionary with index 4095), you should clear up the current dictionary and create a new dictionary, starting with new 256 entries (without adding new dictionary) for the next input data.

2. **Binary output** – The output of compressed data should be stored in a new file using binary form. As our program use 12-bit codeword, the output should be ***unsigned 12-bit word*** as well. For instance, if the sequence of index output is { 97, 98, 256 }, your unsigned 12-bit word data should be { 0000 0110 0001, 0000 0110 0010, 0001 0000 0000 } and stored as follow:



Note that you should pad the final code with zero if total number of bit is not multiple of 8.

3. **Decompress file** – Your program should be able to decompress the compressed file to the original file without changing the content.

4. **Archives (Multiple files)** – Your program should be able to support compression of multiple files into one compressed archive. Therefore, you have to save a list of file in the beginning of the compressed file with following format:

   The header is stored in plain text, as it queries what files are included. Following the header is the compressed data. When one single file is completely compressed, you have to insert a ***special code 4095*** (2^N-1) in the compressed data immediately to indicate the End-Of-File (EOF). Then you can start to compress the next file using the current word dictionary. The first character in the next file will be the new prefix.

<filename1>\n
<filename2>\n
<filename3>\n
....
\n
<Compressed File 1>4095<Compressed File 2>4095<Compressed File 3>4095
<...>4095

Compressed File

5. **Command Line Control** – To simplify your work, graphical user interface is NOT required. Your program should support the following format for execution.

   Compression

   - *lzw –c <output file name> <list of files name>*

   Decompression

   - *lzw –d <lzw filename>*

6. **Performance** – You can use any data structures to implement the dictionary so as to speed up the dictionary lookup process. There will be bonus for the students who have done any optimization.

III. LZW Algorithm

Here is the detail of LZW algorithm. You can refer to lecture note for more detail of dictionary-based compression/decompression algorithm.

Compression

Here is the pseudo code of LZW compression algorithm.

```
1. Initialize the dictionary, and set P to an empty string
2. While there are still characters to read,
    a. Get the next character, C, from the input text
    b. Search for the string <P, C> in the dictionary
    c. If found,
        i. Remember the code
        ii. Append C to the end of the string P
    d. Else if not found,
        i. Output the code that corresponds to P
        ii. Add to the dictionary the new entry <P, C>
        iii. P := C
3. Output the code that corresponds to P
```

The string P denotes a prefix. During compression, we create a dictionary of strings on the fly. A code corresponds to the location of a string in the dictionary. Assume we are at a particular instance of compression (just after 2.d.iii) and a new character is read from the input text. Then, we search for the presence of the new string that is created by appending the character to the prefix P (2.b) in the dictionary. If this string could be found, we will use the new string as the prefix (2.c.i) and repeat the search by appending another new character. Meanwhile, the codeword is memorized (2.c.i). Conceptually, the iteration is equivalent to reading the longest string that is already contained in the dictionary. When the string could not be found, we will output the codeword for the prefix (2.d.1) and add a new entry to the dictionary (2.d.ii). A practical issue about implementation is how to store the dictionary. Obviously, a linear array suffices, but the performance (i.e. the execution speed) will be unacceptable since the dictionary lookup step (2.b) will take too much time. Alternatives will be to use a tree structure or a hashing function.

Decompression

Here is the pseudo-code for the decompression procedure:

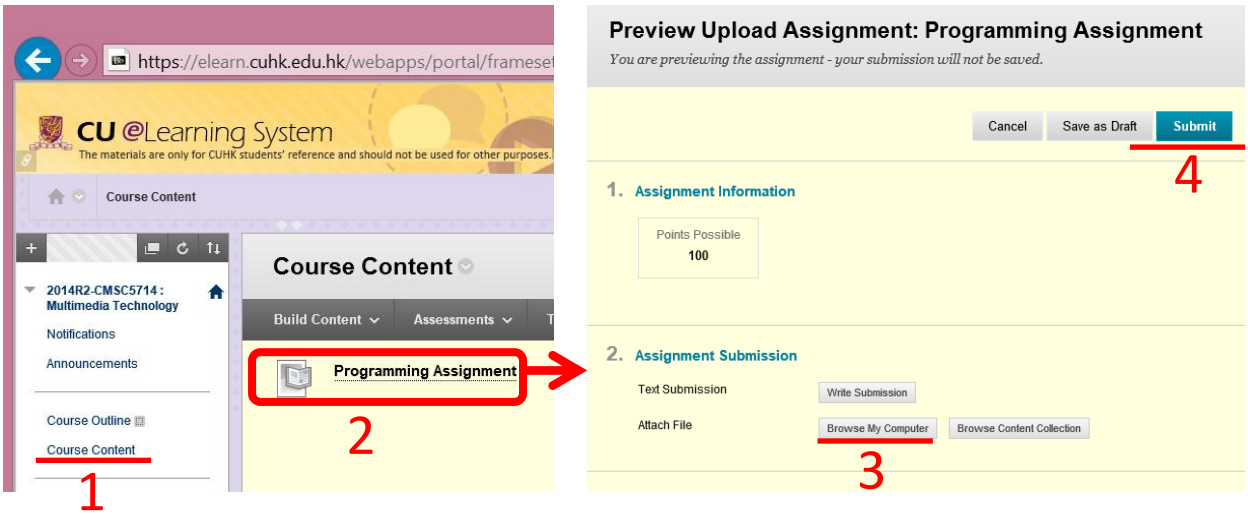
```
1.  Initialize the dictionary,
2.  Get the first code, cW
3.  Find cW in dictionary and output the string dictionary(cW)
4.  While there are still codes to read,
    a.  pW := cW
    b.  Get the next code, cW, and find it in the dictionary
    c.  if found,
        i.   Output the string dictionary(cW)
        ii.  Let C be the first character of dictionary(cW)
        iii. P := dictionary(pW)
        iv.  Add the string <P, C> to the dictionary
    d.  if not found, (exception)
        i.   P := dictionary(pW)
        ii.  Let C be the first character of P
        iii. Output the string <P, C>
        iv.  Add the string <P, C> to the dictionary
```

The decompression process is even easier to understand. It is basically the reverse of the compression process. For a code read from the input stream, the decompression routine will

output the corresponding entry in the dictionary (4.c.i) and update the dictionary (4.c.ii, iii, iv). In closer examination, one would find that the decompression process actually *lags* behind the compression process. Hence, there will be the case when a code refers to an entry that has not yet been created. To handle this, we just need to follow the routines (4.d). Besides, cautions must be taken when a new dictionary is created in the middle of the decompression process. Once a file is decompressed, you should go back to step 2 for the next file with current dictionary..

IV. Submission Guidelines

You should submit the source code of your programming assignment before the deadline via our CUHK e-learning system (Link: https://elearn.cuhk.edu.hk/). After login the e-learning system using your student ID and CWEM password. You first need to open course content page (Step 1). And then you can enter the submission page by clicking “Programming Assignment” (Step 2). In the submission page, attach your zipped source code from computer (Step 3). Finally, click the “Submit” to submit your assignment (Step 4). Multiple submissions are allowed but only the last submission will be graded. Submission using another method is not allowed unless there are strong reasons.



V. Remarks

1. You will fail the course if your work is plagiarism.
2. The programing will be tested in PC with windows using Visual Studio (C/C++) or latest Java runtime (JAVA). Please make sure your program can be compiled and executed in this environment.
3. A win32 executable sample program will be provided.
4. To simplify your work, you can assume that the files always exist, and the filename will never exceed 1024 character. Hence, you do not need to do error checking for this part.
5. If you do not follow the required format in this assignment, mark will be deducted.
6. 20% marks will be deduced per day for late submission, including holidays and Sundays.
7. You can send email to kckwan@cse.cuhk.edu.hk if you have any question.
8. You will fail the course if your work is plagiarism. (Yes, it is very important)