Project 2: Code Compression for Embedded Systems

CIS 4930/CDA 5636: Embedded Systems

Posted: Oct 25, 2012 **Due:** Nov 14 (EDGE: Nov 17), 2012 11:55 PM

Project Description: Please start this project after code compression is covered in the lectures.

Assume that the dictionary can have four entries (index 2 bits) and the four entries are selected based on frequency (the most frequent instruction should have index 00). If two entries have same frequency, priority is given to the one that appears first in the original program order. The original code consists of 32-bit binaries. You are allowed to use only three possible formats for compression (as outlined below): i) *Original Binaries*, ii) *Direct Matching*, and iii) *Consecutive Mismatches* starting from fixed locations. Note that if one entry (32-bit binary) can be compressed in more than one way, choose the most beneficial one i.e., the one that provides the shortest compressed pattern.

Format of the *Original Binaries*

00	Original Binary (32 bits)

Format of the *Direct Matching*

01	Dictionary Index (2 bits)

Format of the *Consecutive Mismatches*

10	Mismatching Patterns (2 bits)	Starting Location (5 bits)	Dictionary Index (2 bits)

(Location is counted from left/MSB)

Mismatching Patterns:
00: 1 bit mismatch
01: 2 consecutive bits mismatches
10: 3 consecutive bits mismatches
11: 4 consecutive bits mismatches

You are expected to implement the compression and decompression functions using C, C++ or Java. You need to show a working prototype that will take any 32-bit binary (0/1 text) file and compress it to produce a output file that shows compressed patterns arranged in a sequential manner (32-bit in each line, last line padded with 0's, if needed), a separation marker "222", followed by four dictionary entries, another separation marker "222", and finally the compression ratio (in *dd.dd* format). Your program should also be able to accept a compressed file and decompress to generate the decompressed (original) patterns. Please see the sample files posted in the webpage.

Grading Policy: If your implementation works correctly with the given input file (i.e., produces both correct compressed output and correct decompressed output), you will receive 60% of the score (6 points). The remaining 40% (4 points) will be given by running another set of inputs (you will not have access to these inputs prior to submission). **There can be significant penalty (up to 100%) if you do not follow the submission policy (see next page).**

Submission Policy:

Please follow the submission policy outlined below. There can be significant **score penalty** (up to 100%) based on the nature of submission policy violations.

1. You are not allowed to take or give any help in completing this project.
2. Please submit only one source file (*see next comment if you have multiple source files*). Your file name must be SIM (e.g., SIM.c or SIM.cpp or SIM.java). Please do not submit any object files. On top of the source file, please include the sentence: "/* On my honor, I have neither given nor received unauthorized aid on this assignment */".
3. If you decide to submit more than one source files, Please use "**tar cvf project2.tar file1 file2 file3…"** to pack your source files. Please do not put your files in a separate folder and pack. Please provide a **Makefile** to compile your code.
4. Please test your submission:
    - Download your submission from eLearning (ensures your upload was successful).
    - Login to **thunder.cise.ufl.edu** (get a CISE login, if you do not have one)
    - If you provided a .tar file (if you have multiple source files), untar it first and then compile to produce an executable named **SIM**
        - tar xvf project2.tar
        - make
    - If you submitted only one source file, please compile to produce an executable named **SIM**.
        - gcc SIM.c –o SIM **or** javac SIM.java **or** g++ SIM.cpp –o SIM
    - Please do not print anything on screen.
    - Please do not hardcode input or output filenames, accept them as command lines options.
    - Compress the input file (original.txt) and check with the expected output (compressed.txt)
        - ./SIM –c original.txt cout.txt (or java SIM –c original.txt cout.txt)
        - diff –w –B cout.txt compressed.txt
    - Decompress the input file (compressed.txt) and check with the expected output (original.txt)
        - ./SIM –d compressed.txt dout.txt (or java SIM –d compressed.txt dout.txt)
        - diff –w –B dout.txt original.txt
5. *In the last project there were many cases where output format was different, filename was different, command line arguments were different, or e-Learning submission was missing, …. All of these led to un-necessary frustration and waste of time for TA, instructor and students.* **Please use the exactly same commands as outlined above to avoid any score penalty.**