# CSE 490 GZ
# Project 1

Due on Wednesday, February 20, 2002.

## 1 Introduction

The purpose of this project is for you to become familiar with some of the major lossless compression programs that are available in the unix/linux world. Many of these programs are also available in the PC world too, but it is easier to study the performance of the programs in the unix/linux setting. These programs are based on the compression techniques we are learning in class so that experimenting with these programs will help you understand the strengths and weaknesses of the methods. The programs we will study are: gzip (LZ77), compress (LZW), bzip2 (Burrows-Wheeler transform), compact (adaptive Huffman), arith (adaptive arithmetic), ppm, ppm* (adaptive arithmetic coding with context), and sequitur (context-free grammar based). The former three programs are supported by unix/linux while the latter four are unsupported.

There are a number of aspects of performance that you will investigate. The first and primary one is compression performance. That is, how well does the program compress files. Naturally, this depends on the file being compressed. Thus, you will measure the compression performance for a variety of files, some from the well known Calgary Corpus, and other more modern large files, like XML files, graphics model files, DNA files, and at least one of your own choosing. Also, the files in the Calgary Corpus are fairly small which may not be a good test for a program. For some of these programs there are parameter settings that could yield different compression performances. You will explore some setting to see how they affect performance.

Other aspects of performance are the times to compress and decompress. The supported programs we have given you are highly engineered to minimize compression and decompression times. The unsupported programs are less engineered and slower. We are also interested in whether the a program has asymmetic running times where the time to compress and time to decompress are very different.

## 2 Experimental Methodology

We will provide you with the executable programs within some Perl wrappers to help you design your experiments. In addition, we will provide you with a suite of files on which to run the programs. However, it is up to you produce meaningful data from the programs and files. For example, if you are doing a timing study for a compression program it may be the case that a file is so small that the timing measurement (which has a resolution of about .01 of a second) is too coarse to measure the time. To get an accurate measurement you may have to form a larger file from the original by concatenating it to itself many times. At the other extreme some of the programs are very slow (e.g. ppm) so that compressing a very long file would take hours to days. To gain insight about the program on the type of data the file represents it might be a good idea to truncate the file to a smaller size.

## 3 Reporting Your Results

Your project should include the compression performance and time performance (compression and decompression) for all the programs listed above on each of the files we provided. In addition, you should find an additional interesting class of files to include in your study.

Some of the programs have options for parameter settings. For at least one of the programs you should explore some parameter settings to see how they affect performance.

Your project report should be written to an audience of peers, other students who are taking CSE 490gz. You paper should be no more than 5 pages in length including the title page and should be type set. Your project report will consists of several components:

**Title page:** Title, author, date.

**Introduction:** A one or two paragaph introduction to your project. In your own words explain the goal of the project and its major findings.

**Methodology:** A one or two paragraph explanation of your experimental methodology. You will want to include information about programs and their settings, the files, the platform used and what data was gathered and how it was gathered. Explain how the data is normalized and reported.

**Results:** A description of the results of the project. I would suggest using at least two tables. The first has one row for each file and one column for each program reporting the compression performance of each program on each file. The compression performance should be normalized to compression ratio or some other metric such as bits per character. The second should be similarly organized for reporting compression and decompression times. These times should also be normalized. You may want separate tables for your experiments that vary the parameters of a program. In addition to the tables you should have several descriptive paragraphs explaining what to look for in the tables.

**Conclusion:** A description of your conclusions about the programs.

## 4 Details

The programs gzip, compress, and bzip2 are directly supported by Linux while compact, arith, ppm, ppm*, sequitur are not. The source code and binaries for the non-supported algorithms is in the project directory located at:
/cse/courses/cse490gz/02wi/project01/
on the CSE instructional machines. For information see
http://www.cs.washington.edu/lab/about/ugrad-computing.html.

There are 4 directories within the "project01" directory:

1. `bin` - contains executables for the unsupported algorithms
2. `src` - contains source code used to compile the executables
3. `data` - contains sample files to encode/decode
4. `scripts` - contains perl scripts used to run the experiments

### 4.1 Setup

Create a working directory. From your working directory do the following two steps:

1. Copy the perl scripts as follows:
   ```
   cp /cse/courses/cse490gz/02wi/project01/scripts/* .
   ```
2. Add a symbolic link to the "data" directory as follows:
   ```
   ln -s /cse/courses/cse490gz/02wi/project01/data/ data
   ```

This will create a directory called "data" in the directory that you executed the "ln" command from. This will allow you to easily access the files in the "data" directory that you need to encode/decode for the project.

### 4.2 Running Experiments

We will be using the perl scripts that you copied to your working directory to run the experiments. The scripts take as input the file that you want to compress/decompress. Each script uses a different compression/decompression algorithm pair, and outputs timing and compression ratio information. As an example of how to run an experiment, typing the following will run experiments by compressing/decompressing the file called "trans" with the "gzip" algorithm (note that "%" is the Linux prompt and is not something you type in).

```
% ./gzip-exp.pl data/calgarycorpus/trans
```

## 4.3 Command-Line Arguments

The scripts that we have provided for you use default parameters for each of the compression/decompression algorithms. If you are interested in changing some of these parameters, you need to directly modify the perl scripts. If you need help with this, please see me (Justin).

For the algorithms that are supported by Linux (gzip, compress, bzip2) you can get detailed information by reading the man pages. The algorithms that are not supported by Linux have the following command-line arguments:

1. compact (adaptive Huffman coder)

   ```
   usage: compact [-d] file
     -d decompress the file
   ```

2. arith (adaptive arithmetic coder)

   ```
   usage: arith [-e [-t s] [-f n] [-b n] [-m n] [-c n] | -d] [-v][file]
     -e: Encode
         -t s  Encoding method       (char, word, bits, default = char)
         -b n  Code bits to use          (27, fixed at compile time)
         -f n  Frequency bits to use     (32, fixed at compile time)
         -m n  Memory size to use (Mbytes)   (1..255, default = 1)
         -c n  Bits of context to use        (0..20, default = 16)
     -d: Decode
     -v: Verbose Give timing, compression, and memory usage
   ```

3. ppm, ppmstar (adaptive arithmetic coding with context)

   ```
   encoding usage: ppmenc training.txt < input > compressedfile
   decoding usage: ppmdec training.txt < compressedfile > original
   ```

4. sequitur (context-free grammar based)

   ```
   usage: sequitur [-cdpqrtTuz -k <K> -e <delimiter> -f <max symbols>]
     -p  print grammar at end
     -d  treat input as symbol numbers, one per line
   ```

```
-c  compress
-u  uncompress
-q  quiet: suppress progress numbers on stderr
-r  reproduce full expansion of rules after each rule
-t  print rule usage in the grammar after each rule
-T  print rule usage in the input after each rule
-z  put rule S in file called S, other rules on stdout as usual
-k  set K, the minmum number of times a digram must occur to form
    rule (default 2)
-e  set the delimiter symbol. Rules will not be formed across
    (i.e. including) delimiters. If with -d, 0-9 are treated as
    numbers
-f  set maximum symbols in grammar (memory limit).
    Grammar/compressed output will be generated once the grammar
    reaches this size
```