Code and Algorithmics Documentation for byte compress/decompress.

Introduction

This manuscript presents the reasoning behind the algorithmics used on the C program compress-decompress.

The idea is to create an index table with all the different elements(max size 16).This way the algorithm will reduce at least form 1 byte (8 bit) to at most 4 bits for each of the elements of the input. and in the best case scenario it can be reduced to 1 bit per byte from the input file.

The structure of the intermediate/compressed file would be:

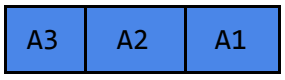
- 1 byte for table size
- [table size] bytes for the index table
- [total_number_of_elemtns]* size bits.(ranging from 1 to 4)

Main Issue

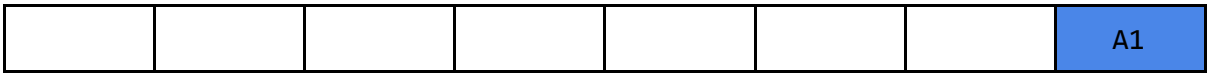
For small pieces of data [total_number_of_elemtns] < 16 with many different elements will cause the compressed file to be bigger than the input file due to the table being as long as the actual data.

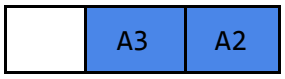
How the data is stored

EXAMPLE bit size 3
say there are 5 different 3 bit elements
A B C D E
first A is broken into

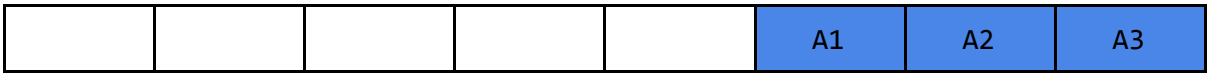


first element (unit position) of A is pushed into the byte

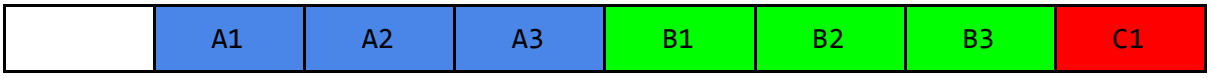




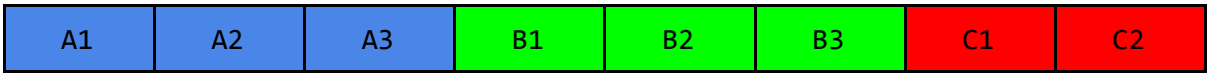
then the second and thrid



A is depleted now is time to perform the same process with B and C
...
...
...



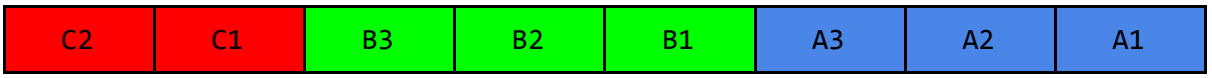
once the last fitting element is pushed into the byte the write_to_memory is triggered.



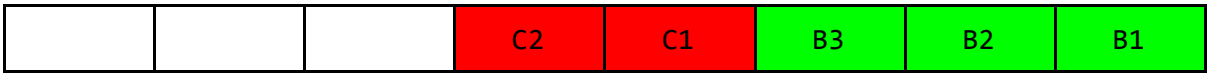
The leftover of C is pushed first into the next byte.

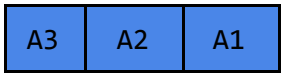
How the data is retrieved

The first step is to inver the byte. This way the data will pop ordered.

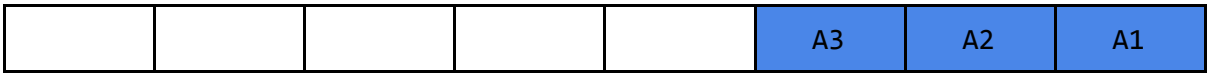


Then the data is stored on bit_size containers again.

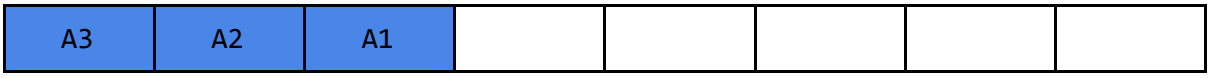




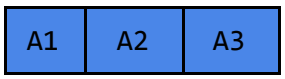
This containers need to be inverted to reproduce the initial state of the data. Since this inversions are performed byte wise it is needed to do it in two steps



- first move data to the other side of the byte
  (byte = byte <<(8-bit_size)



- then invert the bite and perform the search on the index table



Result A