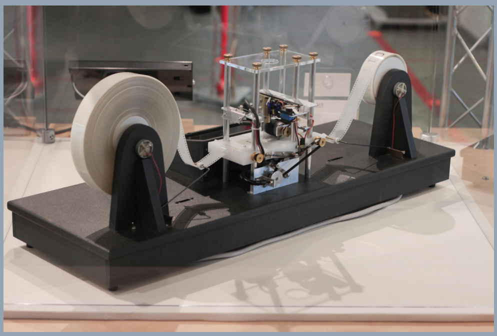
CoE 164

Computing Platforms

Midterm Problem

Academic Period: 2nd Semester AY 2022-2023
Workload: 24 hours
Synopsis: Obfuscated, compressed, and executable text

Description



Communication privacy is in the forefront of most people's priorities in the current era of very fast information exchange via the internet. Many unwanted people find a way to seek access into the internet usage of random and unsuspecting people, such as email inboxes, website accounts, and chat conversations, for various illicit purposes. This may include stealing money and property, impersonation, and blackmailing. A way to be able to deter such acts is through *obfuscation* of messages. Obfuscation is "scrambling" a message before sending it to the intended recipient, who will then "unscramble" it to get a hold of the true message. Since this method still enables third-party spies to get hold of this scrambled message, the unscrambling algorithm for the message should be known only to the intended recipient. A more secure version of this is called *encryption*, wherein the message is encoded with a *key*, which both only the sender and receiver should have. Without the key, the encoded message cannot be decoded by anyone. The encryption and decryption algorithms should ideally be open to everyone.

Before delving into encryption, you have decided to write some sort of obfuscation algorithm for the messages that you would like to send over some channel. A *channel* is some sort of medium where two or more parties exchange information. It can be through telephone, wireless, or the internet. Even barcodes can be treated as some channel! Having mulled over the possible workflows, you have decided to stick to encode a message as a computer program, which the intended recipient will run in their own machine. The program should have a very simple syntax and very few commands so that technically *any computer* can run it. As such, you have formulated diropql.

diropql Language

diropql (diro portable query language) is a simple language consisting only of seven commands. Each command corresponds to a letter in ASCII. diropql manipulates a virtual contiguous row of memory cells, each holding one byte, whose value can be changed. The first task is to create the *interpreter*, which reads a diropql program from a file. Then, the next task is creating the *encoder*, which accepts a message written using only ASCII characters, and creates a `.dpql` program that will output the message when run.

With a very limited command set, a diropql program will contain a very large amount of the same seven characters especially if the messages get very long. Having realized that this will not do especially if you wanted the program to be encoded as a barcode or sent to your recipients, you have suddenly thought that you can take advantage of this same fact and additionally *compress* the output program!

Program Compression

You have decided that before compression, any unrecognized characters in a diropql program should be removed. Then, the now-cleaned program will be treated as a single string and compressed by processing it through the following algorithms in order:

- Burrows-Wheeler transform
- Move-to-front transform
- Run-length encoding
- Huffman encoding

The algorithms above were selected to take advantage of the fact that a diropql program will have runs of repeated commands somewhere, and these runs can be encoded as succinctly as possible.

Combining Everything

The compression algorithm you have formulated consists of four different algorithms that exploit redundancies in a diropql program. The final result of the algorithm will be a binary string whose length may not be divisible by eight. Therefore, some arbitrary number of zeros may have to be appended to it such that the final length can be evenly split into groups of eight bits. In addition, the compressed program cannot be properly sent through a text message since some values in ASCII that are mapped into the numbers are unprintable. Hence, a way to make them "readable" should be formulated.

You have decided to encode the compressed program as a stream of printable ASCII characters. To make sure that you are not undoing the compression procedure substantially, you have decided to use Base85 encoding. This encoding will automatically append the appropriate amount of zeros needed to make sure that the length of the binary string to be encoded is divisible by eight. But, before encoding the compressed program, you have realized that the compression algorithms alone need some initial values for the recipient to be able to undo the obfuscation. Therefore, we need to append these values as *metadata before* encoding.

After the program is converted to Base85, a *magic string* (DIROPQLZ) is then appended before it to denote that this chunk of text is a compressed diropql program. This final obfuscated string can now be sent to recipients that have knowledge of the compression scheme and the program commands. If this string were to be sent as a file, it should have the `.dpqlz` extension to differentiate it from the uncompressed version of the program.

Test Suite

Testing a program is one of the most important aspects of computer programming. Hence, you have formulated various function signatures pertaining to the different parts of the obfuscator, which will be filled up as you progress with coding the obfuscator. The list of required function signatures, data structures, and modules to test and implement are as follows:

- Module `dpql`
    - `fn write(text: &String) -> String`
        - Convert `text` into a diropql program that outputs `text` and returns it.
    - `fn read(prog: &String) -> String`
        - Read `prog` as a diropql program, interpret it, and return the contents of the output queue.
    - Submodule `zip`
        - `fn write(text: &String) -> String`
            - Convert `text` into a diropqlz program that outputs `text` and returns it.
        - `fn read(prog: &String) -> String`
            - Read `prog` as a diropqlz program, decompress it, interpret it, and return the contents of the output queue.
        - `fn write_with_meta(meta: &DpqlzMeta, prog: &Vec <u8>) -> String`
            - Create and return a diropqlz program using `meta` as the metadata and `prog` as a compressed diropql program.
        - `fn read_with_meta(prog: &String) -> (DpqlzMeta, Vec <u8>)`
            - Read `prog` as a diropqlz program and output its metadata and resulting compressed program as a vector of bits, respectively.
        - `struct DpqlzMeta` - struct containing the following fields:
            - `mlen: u64` - length of the obfuscated message `msg` in bytes
            - `moffset: u8` - number of bits/elements to exclude or ignore from the end of `msg`

- `bwt_idx: u64` - suffix index associated with the Burrows-Wheeler transform
- `huf_bitlens: Vec <u8>` - array of 1-byte numbers corresponding to the bit lengths of the derived canonical Huffman codebook

- Module `compressor`
    - Submodule `bwt`
        - `fn encode(text: &String) -> (String, usize)`
            - Transform `text` using Burrows-Wheeler transform and return the result and index, respectively.
        - `fn decode(text: &String, index: &u64) -> String`
            - Return the inverse Burrows-Wheeler transform of `text` using the suffix index `index`.
    - Submodule `mtf`
        - `fn encode(text: &String, alphabet: &String) -> Vec <u8>`
            - Transform `text` consisting of the characters from `alphabet` using move-to-front and return the result as a vector of indices.
        - `fn decode(data: &Vec <u8>, alphabet: &String) -> String`
            - Decode `text` consisting of the characters from `alphabet` that was transformed using move-to-front and return the result as a string.
    - Submodule `rle`
        - `fn encode(text: &Vec <u8>) -> Vec <u8>`
            - Compress the list of numbers `text` using run-length encoding and return the result as a vector of numbers.
        - `fn decode(data: &Vec <u8>) -> Vec <u8>`
            - Decode the list of numbers `data` encoded using run-length encoding and return the result as a vector of numbers.
    - Submodule `huffman`
        - `fn encode(text: &Vec <u8>) -> (Vec <u8>, Vec <u8>)`
            - Encode the list of numbers `text` to its Huffman encoding and return the result as a vector of bits (each bit is an element) and the generated list of bit lengths of codewords of the corresponding canonical Huffman codebook, respectively.
        - `fn decode(data: &Vec <u8>, canon_freqs: &Vec <u8>) -> Vec <u8>`
            - Decode the list of bits `data` encoded using Huffman encoding and the corresponding list of bit lengths `canon_freqs`, and return the result as a list of numbers.

Note that the functions above may not have the complete signatures, and you may have to make the modules and functions public for them to be used in other functions, or annotate their lifetimes. Also, since this is your program, you have the liberty to add more functions and modules as needed.

Each function should have a `tests` module somewhere in it, which contains a collection of test cases. A sample snippet for testing the functions inside the `compressor::rle` module is as follows:

```
// src/compressor/rle.rs

pub fn encode(text: &Vec <u64>) -> Vec <u64> {
    // rle program here
}

pub fn decode(data: &Vec <u64>) -> Vec <u64> {
    // rle program here
}

#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn encode_pt1() {
        [...]
    }
}
```

Making the Program

You have expected that the final program will be a library that you can incorporate in future projects that will use this scheme, such as a custom instant messenger. The library consists of two major modules - the obfuscator, which encodes a message into a diropqlz string, and deobfuscator, which does the reverse of the obfuscator.

While outlining the obfuscation scheme, you have mulled over its complexity and the fact that you cannot realistically do it alone in a short amount of time. Hence, you have enlisted the help of two of your friends. You have decided to split the responsibility of creating the diropql interpreter, compression algorithm, and test suite among the three of you. Rust has been decided as the language of choice for the program for the opinion that this obfuscator should be fast but secure enough that it will not improperly access the memory of any computer that will run it. In addition, you have compiled a document outlining the diropql language, the compression algorithms, and the diropqlz file format so that the three of you can work efficiently together.

Additional Description/Requirements

For the purposes of easy set-up on any computer, you have decided that the program should not contain any imports to libraries that need to be downloaded through the `cargo` package manager (i.e. the internet) *except* for the following:

- base85

You have decided that the obfuscator will only accept messages that use only the printable characters in ASCII and whitespaces (i.e. the newline `\n`, carriage return `\r`, and space). In addition, since you have not incorporated any error correction in the already-complex obfuscation algorithm, you assume that all obfuscated messages entered through the program can always be deobfuscated in a valid and runnable diropql program.

After developing the (de)obfuscator, you three have decided to assess yourselves and your group's performance in creating the software. Hence, you have decided to enlist the help of another friend to make a form where you can send feedback anonymously, read the feedback in it, and inform all of you individually of the result. The form will be released after the deadline when the program should have been finished.

Upload the (de)obfuscator as a `cargo` package to your remote repository. The repository seems to hate compressed files, so make sure to not encase your codes in one. However, you can upload or make folders within the repository. Write on the submission notes, source code, and feedback form the following information:

- Module that you are mainly responsible for (choose between the following: diropql, compressor, test). Note that what you will mention here will be the component that will be individually graded to you
- Which module(s) did you substantially contribute to
- Acknowledgement of their group members with their respective names and student numbers

Grading Rubric

diropql Interpreter and (De)Obfuscator Pre-processing (A, 65%)
20% Message encoder to diropql
15% diropql interpreter (`dirol` commands)
20% diropql interpreter (`pq` commands)
5% Obfuscator output after compression
5% Deobfuscator splitter before decompression

Program (De)Compressor (B, 65%)
20% Burrows-Wheeler transform (encode)
15% Burrows-Wheeler transform (decode)
2% Move-to-front transform (encode)

2% Move-to-front transform (decode)
3% Run-length encoding (encode)
3% Run-length encoding (decode)
10% Variable-length encoding (encode)
10% Variable-length encoding (decode)

Test Suite (C, 65%)
10% Message encoder to diropql
10% diropql interpreter
10% Burrows-Wheeler transform
10% Move-to-front transform
10% Run-length encoding
10% Variable-length encoding
5% Obfuscator output and deobfuscator splitter

Combined Program (All Members, 35%)
10% Obfuscator program working
10% Deobfuscator program working
15% Self-assessment