Lab 1

CS 3793/5233 – Fall 2016 assigned August 23, 2016
Tom Bylander, Instructor due midnight, September 9, 2016

In Lab 1, you will complete a program for solving Cryptoquip puzzles. The initial program is `ailab1.zip`, which you can download from the course web site. This will be done by implementing an algorithm that searches for solutions to the puzzle. The initial program already has code to create the data structures that you need to perform a search.

Initial code is provided for you on the course web site. Your grade on the lab will depend on the performance of your program when I run it on test problems. Graduate students must also include a description of how your program performs and what improvements could be made if you had more time.

Cryptoquip

Cryptoquip is a puzzle that appears in the daily newspaper. It is a cryptogram (http://en.wikipedia.org/wiki/Cryptogram) with a pun. The puzzle is described by providing a *cipher text* such as:

```
CVAWDEV H WB ZVEAJHCHRT EVOVJWU ZHQQVJVRX EIDRZ EKEXVB OWJHVXHVE, H
EDFFIEV H'B EXVJVI XKFHRT.
```

and a *hint* such as:

```
B equals m
```

The problem is to find a substitution cipher so that the cipher text maps to a reasonable sentence. The *plain text* answer for this particular Cryptoquip is:

```
because i am describing several different sound system varieties, i
suppose i'm stereo typing.
```

The file `cryptoquips.txt` in the download has many examples of Cryptoquips.

Each alphabetical character in the cipher text will be called a *cipher char*. Similarly, each alphabetic character in the plain text will be called a *plain char*. For convenience, cipher chars will be upper case, and plain chars will be lower case.

A *cipher word* is a sequence of cipher chars plus possibly an apostrophe (e.g., `H'B` in the above cipher text). Similarly, a *plain word* is a sequence of plain chars plus possibly an apostrophe (e.g., `i'm` in the above plain text). Apostrophes are the only non-alphabetic character allowed in a cipher word or plain word.

Initial Program

The initial program is organized so that an "environment" thread interacts with an "agent" thread. In this lab, the environment sends a cipher text and a hint to the agent, the agent responds with a plain text answer, the environment scores the answer, and the sequence is repeated for all the Cryptoquips in `cryptoquips.txt`. The separate threads make it difficult for the agent to cheat. The Cryptoquip file will have a different name when I grade your labs.

- `Interact.java`. This class sets up the threads and supervises their interactions. You should not make any changes to this class.

  In more detail, `Interact.java` creates Java pipes so that the environment and agent have their own input and output streams. It monitors these streams so that any line output by one will be input to the other. `Interact.java` also prints to the console anything output by the environment and agent. The environment and agent can use `System.out.println` to print directly to the console.

- `CryptoquipEnvironment.java`. This class contains the code to initialize and run the environment thread. The `in` and `out` fields are how the environment reads from and prints to the agent. You should not make any changes to this class.

- `CryptoquipSolver.java`. This class contains the code to initialize and run the agent thread. The `in` and `out` fields are how the agent reads from and prints to the environment. The fields are described in more detail below. Your job is to code the `search` method.

There are two other classes to help you solve Cryptoquips.

- `Cryptoquip.java`. This class is used to store the information of a Cryptoquip: the `ciphertext`, `plaintext`, and `hint` fields. The `plaintext` field should be `null` if the plain text is unknown. It also has an `extractCipherwords` method.

- `Words.java`. This class is used to store the list of plain words in `lab1-words.txt`. It has a data structure and a `match` method (described in more detail below) to efficiently determine if there is any plain word that matches a cipher word given a (partial) cipher substitution.

Agent

Your task is to complete the agent program (`CryptoquipSolver.java`) to to solve Cryptoquip puzzles. First, the fields:

- `in`. Used to read information from the environment. Initialized by the constructor. The `run` method takes care of this field.

- `out`. Used to print information to the environment. Initialized by the constructor. The `run` method takes care of this field.
- `words`. Used to store the words in `lab1-words.txt`. Initialized in the constructor. You need to use the `match` method in your code.
- `quip`. Used to store the current cipher text and hint. Initialized by the `run` method. The `run` and `solve` methods take care of this field.
- `cipherwords`. Used to store the cipher words from the cipher text. Initialized by the `solve` method. You need to loop through the cipher words repeatedly to check substitution ciphers.
- `plainOrder`. This is an ordering of plain chars by frequency in my Cryptoquips. `e` is the most common, then `a`, and so on.
- `cipherOrder`. This is an ordering of cipher chars by frequency in the current Cryptoquip. This is assigned within the `solve` method. For example, `V` is the most common cipher char in the above example Cryptoquip.

  Generally, you want to test assigning more common cipher chars to more common plain chars. This will create a greater chance of a correct assignment as well as a greater chance of rejecting wrong assignment (and so avoid deep failing searches).
- `cipherToPlainMapping`. This represents the substitution cipher. For example, if the cipher char `B` is mapped to the plain char `m`, then 13 should be assigned to `cipherToPlainMapping[2]`. The 13 comes from `m` mod 32 because `m` is encoded by the number 109. The 2 comes from `B` mod 32 because `B` is encoded by the number 66.

  An assignment of 0 indicates no assignment yet.
- `plainToCipherMapping`. This also represents the substitution cipher, but in the other direction. For example, if the plain char `m` is mapped to the cipher char `B`, then 2 should be assigned to `plainToCipherMapping[13]`.

  An assignment of 0 indicates no assignment yet.

The *search* method should return `cipherToPlainMapping` if the substitution cipher results in mapping cipher words to plain words; otherwise, it should return `null`.

search should be implemented recursively as follows:

Let `cipherchoice` be a cipher char that has not been assigned.
If all cipher chars have been assigned, return `cipherToPlainMapping`.
Loop over possible `plainchoice`s (those not assigned)
  Assign the choices to each other:
  `cipherToPlainMapping[cipherchoice % 32] = plainchoice % 32;`
  `plainToCipherMapping[plainchoice % 32] = cipherchoice % 32;`
  Use `words.match` to check every `cipherword`. Incomplete code:

```
for (String cipherword :  cipherwords) {
   boolean matched = words.match(cipherword, cipherToPlainMapping);
}
```

If all words match,

`int[] answer = search();`

If answer is not null, return answer.

Undo the choice assignments:

```
cipherToPlainMapping[cipherchoice % 32] = 0;
plainToCipherMapping[plainchoice % 32] = 0;
```

At this point, return null because nothing was successful.

Note that much of the above is pseudocode, so several lines of code might be needed to implement one line of pseudocode.

Extra Credit

The above algorithm will solve (or nearly solve) the first 10 Cryptoquips in `cryptoquips.txt`. However, the last 10 Cryptoquips contain one plain word that is not contained in `lab1-words.txt`.

The extra credit is to solve (or nearly solve) these harder Cryptoquips without losing any performance on the easier Cryptoquips. One implementation is to first try the above search algorithm. If it fails, then call another search algorithm which will allow one cipher word to be unmatched.

Turning in Your Lab

Students taking this class for graduate credit must write 100-200 words (txt file please, not a pdf or doc or docx or whatever) on the performance of your program and how any part of the program might be improved if you have more time and motivation. If need be, you will get a second chance to provide a coherent text.

Submit the folder containing all your Java files (and all text files including `lab1-words.txt`, `cryptoquips.txt`, and, for graduate students, a short analysis) to Blackboard. Your program should run under the following conditions: after all the .class files are deleted, then something like:

```
javac Interact.java
java Interact
```

should at least solve (or nearly solve) the easier Cryptoquips. You may also submit your program as an Eclipse project.

The score of your lab will primarily depend on the performance of the program. Your program will be tested on an additional 20 puzzles (10 easy and 10 hard). Assuming your program finishes within one minute, iff x is the average score on the first 10 puzzles and y is

the average score on the last 10 puzzles, then your grade is the maximum of $2x - 100$ and $2x + y - 150$. To repeat, this scoring assumes that the program finishes within one minute or so. Your score will likely be based on the puzzles it finishes within one minute.

Note there is no credit if your program does not solve any puzzles.