Lab 6: Delta

Spring Semester 2020

Due: 6 April, at 8:00 a.m. Eastern Time

Corresponding Lecture: Lesson 10 (Delta Debugging)

Objective

In this lab, you will implement the Delta Debugging algorithm using Java to find 1-minimal failing test cases for buggy text encoder programs using both line and character granularity.

Resources

1. Delta Debugging webpage: http://www.st.cs.uni-sb.de/dd/

Caution

Dr. Zeller, the author of the Delta Debugging technique, and several other people have published implementations of the Delta Debugging algorithm. While we encourage you to research and read more about the algorithm, looking at someone else's implementation, will almost certainly cause your implementation to be similar to theirs. Therefore, make sure that you do not view or reference any implementations or code of the algorithm.

Setup

1. Download `delta.zip` from Canvas and decompress it. It will produce the directory `delta` where you should find:
    - `DeltaDebug.java` - skeleton file for you to implement your algorithm in
    - `MANIFEST.MF` - Manifest file for `DeltaDebug.java` program (do not modify)
    - `DeltaDebugTest` - Program to run and test your program
    - `deltadebugtest.properties` - Configuration file that `DeltaDebugTest` reads in
    - `SecretCoder` - first buggy encoder program
    - `MessageCoder` - second buggy encoder program
    - `long_failing_text.txt` - an input text file on which each buggy encoder program fails.
    - `[prg]_min_failing_text_line.txt`: the correct minimal test input that should be produced by your Delta Debugging program on `SecretCoder` ("sc") or `MessageCoder` ("mc") for part 1 (line granularity)
    - `[prg]_min_failing_text_char.txt`: the correct minimal test input that should be produced by your Delta Debugging program on `SecretCoder` ("sc") or `MessageCoder` ("mc") for part 2 (character granularity)

2. The `deltadebugtest.properties` file works with the `DeltaDebugTest` program to test `DeltaDebug.java`. The following fields are required:
    - `granularity`: `line` or `char`
    - `path.to.program`: path to program (e.g. `./SecretCoder`)
    - `error`: type of error (e.g. `java.lang.ArrayIndexOutOfBoundsException`, or other)
    - `path.to.input.file`: path to the input file
    - `output.file`: name of the output file
3. Each time you change the `DeltaDebug.java` file, you need to compile and compress the program with the following command lines:

    ```
    javac DeltaDebug.java
    jar cvmf MANIFEST.MF DeltaDebug.jar DeltaDebug.class
    ```
4. After compiling the `DeltaDebug.java` file, you may execute the `DeltaDebugTest` program as follows:

    ```
    ./DeltaDebugTest
    ```

Part 1: Line Granularity

Program Description

`SecretCoder` and `MessageCoder` implement a Caesar (or shift) cipher with a right shift of 98 places. Instead of the regular alphabet, these programs are intended to encode the 128 characters in the ASCII character set (value 0 to 127). When the programs run successfully, they create an output file (e.g. `mc_output.txt/sc_output.txt`) with the encoded contents of the input file.

The programs can be executed directly in the command line, followed by a single input that is the path to the input file, as shown below:

```
./SecretCoder <input file>
```

Note: In the VM, you may need to change the access mode of the executables in order to run them:

```
chmod +x <PROGRAM(s)>
```

Problem Summary

The bug in `SecretCoder` occurs when `long_failing_text.txt` contains a non-ASCII character. The program is using the UTF-8 character set to determine the number of characters in the file and creates a char array to store the encoded values for printing. It does not take into account that non-ASCII characters in the UTF-8 character set could be included into the file and these will take up more than one byte. When a non-ASCII character is read into the file as two or

more bytes, the char array runs out of room and throws a `java.lang.ArrayIndexOutOfBoundsException`.

`MessageCoder` is able to handle all characters, but the bug in the encoding program `MessageCoder` occurs when `long_failing_text.txt` contains a specific pattern: any four-letter pattern of alternating caps starting with a capital letter (e.g. “JaVa”). When this pattern is detected, the following exception occurs:
`java.lang.IllegalArgumentException`

Your task is to implement the delta debugging algorithm presented in the lecture to obtain the minimal test case on which `SecretCoder` fails for `java.lang.ArrayIndexOutOfBoundsException` and the minimal test case on which `MessageCoder` fails for `java.lang.IllegalArgumentException`.

To accomplish this task, we have provided a framework and you will need to implement the `deltaDebug` method in `DeltaDebug.java`. For part one of the lab, we will use line granularity. We have provided a `DeltaDebugTest` program that you can use to run `DeltaDebug` to test your changes. After adjusting the `deltadebugtest.properties` file to reflect the desired settings and compiling/compressing `DeltaDebug.java`, execute by running the following command:

```
./DeltaDebugTest
```

Note that `DeltaDebugTest` executable will call the `deltaDebug` method in `DeltaDebug.java`, which requires 5 parameters:

```
deltaDebug(Boolean char_granularity, String program,
      String failing_file, String error_msg,
      String final_minimized_file)
```

- char_granularity - If false, implement line granularity only
- program - this is the path to the program being tested
- failing_file - this is the path to the file with the large failing test case
- error_msg - this is the error message that will be used to determine if the program is failing for the same reason as the original file or not
- final_minimized_file - this is the final file that your program should print with the minimized input.

In this lab, the `deltadebugtest.properties` file and the `DeltaDebugTest` program handle this information and call on that method. Similarly, the grading script will call the `deltaDebug` method directly in `DeltaDebug.java`. **Therefore, it is very important that**

you do not make any changes to the type or method signature of `deltaDebug` in `DeltaDebug.java`. If your program does not run with `DeltaDebugTest`, chances are that it will not run with the autograder.

Ensure you are using the *failing_file* and *final_minimized_file* parameters for your file names in `DeltaDebug.java` instead of hardcoding files because the grader will be using additional files besides the ones provided in this lab.

We have provided an outline for some helper methods in `DeltaDebug.java` that you may find useful (there are comments in the code to explain what these do). You can modify these methods or add new helper methods in `DeltaDebug.java` in any way you see fit as long as your algorithm is invoked by calling `deltaDebug` with the original parameters.

If `deltaDebug` is written correctly, executing the test program will create a 1-minimal test case named `my_min_failing_text_line.txt` (or whatever you've named your output file) that is identical to the provided `[prg]_min_failing_text_line.txt.`

Verify they are identical (*including whitespace & new lines*) using the diff command (replace 'prg' with 'mc' or 'sc' depending on which files you are comparing):
`diff my_min_failing_text_line.txt [prg]_min_failing_text_line.txt`

For grading, we will also be testing your algorithm on hidden input files. We encourage you to come up with different inputs to test how your algorithm handles different cases. Feel free to edit `deltadebugtest.properties` to pass in different file names for testing.

The hidden test cases will be similar in format to the provided one, but will have a different file name, different number of lines, different failing character(s), different position for failing character(s), or other changes appropriate for a text file. You can assume that there will be a single failing character or pattern in the file.

Your algorithm should run in less than 5 minutes on the course VM.

If you add print statements for debugging, make sure to remove these before submission.

Part 2: Character Granularity

Now you will extend your algorithm to perform minimization at the character granularity level when the flag is set. Make sure to minimize the file by lines as much as possible, and then switch to character granularity when line granularity has reached its minimum.

After adjusting the granularity in `deltadebugtest.properties` files to reflect the desired settings and compiling/compressing your `DeltaDebug.java` code, you can run `DeltaDebugTest`:

```
./DeltaDebugTest
```

If `deltaDebug` method is written correctly, executing the test program will create a 1-minimal test case named `my_min_failing_text_char.txt` (or whatever you've named your output file) that is identical to the provided `[prg]_min_failing_text_char.txt.`

Verify they are identical (*including whitespace*) using the diff command (replace 'prg' with 'mc' or 'sc' depending on which files you are comparing):

```
diff my_min_failing_text_char.txt [prg]_min_failing_text_char.txt
```

The same grading notes and 5 minute timeout limit apply as in part 1.

Items to Submit

We expect your submission to conform to the standards specified below. To ensure that your submission is correct, you should run the provided file validator. You should not expect submissions that have an improper folder structure, incorrect file names, or in other ways do not conform to the specification to score full credit. The file validator will check folder structure and names match what is expected for this lab, but won't (and isn't intended to) catch everything.

The command to run the tool is: `python3 zip_validator.py lab6 lab6.zip`

Submit the following files in a single compressed file (`.zip` format) named `lab6.zip`. For full credit, there must not be any subfolders or extra files contained within your zip file.

- `DeltaDebug.java`

Upload your zip file to Canvas. Make sure the spelling of the filenames and the extensions are exactly as indicated above. Misspellings in filenames may result in a deduction to your grade. Also double-check that you are not accidentally submitting `DeltaDebugTest` or `deltadebugtest.properties`.

Grading

Part 1:

- 8 points: For `SecretCoder`, `DeltaDebug.java` correctly minimizes `long_failing_text.txt` to `sc_min_failing_test_case_line.txt`
- 12 points: For `MessageCoder`, `DeltaDebug.java` correctly minimizes `long_failing_text.txt` to `mc_min_failing_test_case_line.txt`
- 30 points: Other test cases

Part 2:

- 8 points: For `SecretCoder`, `DeltaDebug.java` correctly minimizes `long_failing_text.txt` to `sc_min_failing_test_case_char.txt`
- 12 points: For `MessageCoder`, `DeltaDebug.java` correctly minimizes `long_failing_text.txt` to `mc_min_failing_test_case_char.txt`
- 30 points: Other test cases