Dear students,

Below are the most frequently asked questions of this week's lab (week 2):

Question 1: We have implemented the branch distance computation, what is it used for?

Answer:

The goal of the assignment is to use the branch distances to guide your fuzzer/hill climber in fuzzing new inputs for the program to trigger more branches. The branch distance information is thus used to guide your hill climber. An example would be to keep track of the sum of distances for each trace run and use the trace with the smallest distance (smallest sum) to generate new inputs

Question 2: In the assignment, it says that a simple random fuzzer is implemented, but where is this fuzzer called?

Answer:

If you look at the "*fuzz*" method, you should see that it always returns a random trace ("*generateRandomTrace*"). The actual fuzzer itself is not being run. You can run it by going into the while-loop of the "*run*" function of "FuzzingLab.java" and adding the following line (within the try-catch clause):

```
DistanceTracker.runNextFuzzedSequence(fuzz(DistanceTracker.inputSymbols).toAr
ray(new String[0]));
```

This line generates a new input trace using the input symbols of the given problem and the trace is run on the problem.

Question 4: Are we allowed to use libraries for Edit Distance?

Answer:

Yes, you are allowed to use libraries for Edit Distance. We do not expect you to implement it yourself. You are free to implement it yourself if you prefer

Question 5: In the assignment, it says to compute a permutation of a trace. Is this correct?

Answer:

No, this is indeed a small mistake in the assignment, it should be mutation instead. We apologize for any confusion it might have caused. We will update the PDF of Lab 1, make sure to download the new version on BrightSpace or pull the new PDF from the repository

Question 6: Can we use distance or condition of the if-statement as ID for a branch?

Answer:

We do not recommend using this information as the ID of a branch, as a branch could be visited multiple times when a trace is run on the problem. This means that the condition and distance might be different every time that you visit this branch. Using the distance or the condition as ID might mean that you visited more unique branches but actually you visited the same branch multiple times.

Question 7: When do we normalise the branch distance?

Answer:

One way is to normalise as soon as the distance is computed (before it is returned by the method you're using for computing branch distance).

Question 8: I see the output 'Invalid Input: null' when running a problem, what is going on?

Answer:

The instrumentation automatically catches exceptions, so if you throw any exception, it will show up there. If you have not added messages to the exception: eg: `throw new IllegalArgumentException()` instead of `throw new IllegalArgumentException("Oopsie")` you will see null. If you add a message, you will see that message.

As an alternative to throwing exceptions, you can also enable assertions by giving java the flag `-ea` and in your code use something like `assert false`. You then get a stack trace, and you can even specify a message like so: `assert false: "Oopsie"`.