Assignment #3 Software Testing

Com S/SE 417 Spring 2023

Handed out March 21h, 2023

Due at 11:59 PM, Tuesday March 30th, as a pdf uploaded to Canvas

Homework Policy

Homework Policy: The homework assignment should be done individually. You may talk to classmates about the problems in general, and you can help each other with getting the tools running, but you must complete the homework on your own. You are not permitted to use published answers from websites, etc. Assistance by others must be specifically credited in the solution to the problem that is turned in, describing what the contribution was (e.g., "Thanks to [name] for explaining X in Problem Y", not "Thanks to [name] for help with HW1."). The Dean of Students Office offers several good resources to build understanding of how to avoid plagiarism, such as Purdue's "Safe Practices" site.

The goal is for each of you to learn the material sufficiently well to use it productively, to think innovatively, and to develop confidence in your problem-solving abilities. Feel free to talk to me individually about this if you have any questions.

Late policy: *10% penalty per day* (or part of day) for late homework. ***Assignments will not be accepted after April 1st***, unless otherwise arranged/discussed with me.

Late Policy One Slack Period for all Homework
During the semester you have one "slack period". This means that you can hand in one homework late – up until the late acceptance date –(e.g. Feb 14th for this assignment) without penalty. No questions asked. Once you use up your slack period late penalties will apply as per the late policy

Some homework problems are adapted from the course textbook, "Introduction to Software Testing", 2nd edition, Ammann & Offutt, 2017. There is a Student Solution Manuals available online with answers to other practice questions https://cs.gmu.edu/~offutt/softwaretest/exer-student.pdf.

Background:

In this exercise we are going to use two tools for input partitioning (Chapter 6). First we will use the TSL (Test Specification Language Tool) which is part of the Software Infrastructure Repository (SIR) (https://sir.csc.ncsu.edu/portal/index.php). If you ***re-use this tool***, you should give that repository credit.

The second tool is a research tool that generates pairwise samples (i.e. pairwise combination strategies) or t-wise samples. The reference for this tool is: https://ieeexplore.ieee.org/document/1201186

We are going to use the "sort" program as our example for this exercise.

You can download both tools as well as some input files for sort from Canvas. The TSL tool will run directly on pyrite (there is a compiled program called "tsl" in the main directory). However, you can run "make clean" followed by "make build" and recompile to run on any other Linux/Mac machine. For the other tool you only have an executable. I have included the pyrite one. I will also upload one that should run on the current version of Mac OSX.

Note – there are 4 questions below (in the gray boxes), each with multiple parts, for a total of 10 items to be handed in (via a single .pdf document)

Part I: Working with sort

First, we want to get familiar with the Unix "sort" utility. This program can be run on any unix-like system. There are 6 input files for sort. You can use these to learn how sort works. We are going to focus on a small set of parameters.

(a) Sorting order
   a. Default – this is alphanumeric
   b. -n – this is numeric
   c. -R – this produces a random order (except it will keep duplicates together)
   d. -r -reverse order
   e. -h -human numeric sort (consider units like K, M at end of number when sorting)

Read the man pages (> man sort) to read about this parameter. Then try running some of these using the input files (you may need to look at those files to determine what they are doing). For instance:

- **sort -r sorted-numbers-dup.txt**

should sort that file in reverse alpha order:

9
7
4
25
25
25
15
100

And

- **sort -n sorted-numbers-dup.txt**

should sort in ascending numerical order

4
7
9

15
25
25
25
100

- sort -h random.huan.txt

33
1000
2200
35k
10M
45M

1. Working with the TSL Tool

Now go to the tsl directory. There is a file called "sort.tsl". Open up this file and take a look at it. Spend a few minutes trying to understand what the different parameters and environments are. Note there are 6 files that are associated with the various environments.

Now run the tsl tool:

- ./tsl sort.tsl

This will create a test frame file called sort.tsl.tsl

You should have 480 test frames.

Question 1:

Hand in the first and last 10 test frames of this file and answer the following questions

(a) Pick **one test frame** and explain (on paper) what this frame means (i.e. what combinations of options you would test based on the frame – note you will want to choose one that does not have the help and/or the version turned on for this question). Use the existing input files to pick a file that would make sense to use for this test frame. Give this frame as a concrete test case and show a screen shot of the output (you will need to show the command line so we know which input you used).

(b) Now run the tsl on the file called "sort.refined.tsl". How does sort.refined.tsl modify the tests cases that used help and version. Explain the differences and list how many frames it has

2. Working with a pairwise testing tool

The second tool is a tool to generate pairwise (or t-wise) samples. This tool uses only numbers (no names like the TSL so you will need to do some mapping to use).

First go to the directory called CIT_Tool. To run the tool:

- ./cit_generate input.txt -F

The tool creates an output file called "cit_sample.out". Note this uses randomness so each run will be different. There is README that explains the formatting of the input file. I explain both files here. Below we see the input. The first line tells use what the "combination strength is – in this case – 2 -- or all pairs". This is the parameter t. The next line tells us how many parameters there are (this is both the parameters + environments in TSL). The next lines tell us for each parameter how many choices they have. The tool uses a shortcut so consecutive parameters with the same number of choices can be combined. Below we see 4 parameters. This is the parameter k. The first 3 have 3 choices each (e.g. high, medium, low) and the last one has only 2 choices (maybe on/off). The right size shows the output. The first 2 lines are just information about the model. The next line lists out each of the numbers of choices for each parameter (in this case: 3 3 3 2). The next line after that tells us the sample size (in this case 9). Then next k lines (one for each parameter) gives us a numerical mapping of unique integers for the choices from that parameter. For instance if, the first parameter had "high, medium, low" then 0 = high, 1 = medium and 2=low. In the next line 3 is the first parameter choice for second parameter. Assume we also have high, medium, low for that parameter, 3 is high, 4 is medium, and 5 is low. Last, we have a number (the number of tests in the sample) followed by the actual samples. In this case, suppose we use all high/medium/lows for the 3-choice parameters and on/off for the binary, our first test case is:

- low, medium, low, on

```
2
4
3  3
2  1
```

Input.txt

```
t   k   v  TCount
2 4 11 45

 3 3 3 2

  0  1  2
  3  4  5
  6  7  8
  9  10

9
 2 4 8 9
 1 5 8 10
 0 5 7 9
 2 3 7 9
 2 5 6 10
 1 4 7 10
 1 3 6 9
 0 4 6 10
 0 3 8 10
```

cit_sample.txt

There are two input files for sort:

cit-sort-input.txt

cit-sort-input-3way.txt

These models each have 5 parameters (they match the TSL but leave out the “version” and “help” parameters.

Try running these (note if you use the -F option you will overwrite the cit-sample.out file each time). If you leave that out you will get a cit-sample.#####.out file where ##### is a random number.

- ./cit_generate cit-sort-input.txt -F
- ./cit_generate cit-sort-input-3way.txt -F

Question 2:

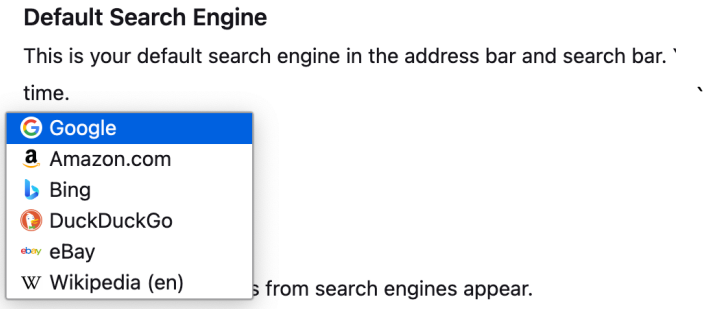
(a) How many combinations (test cases) are there for each of the above files?
(b) Provide the output from the first one (cit-sort-input.txt)

(c) own program) and **map the test cases so that they use the choices from the TSL**. For instance, you might have the header.

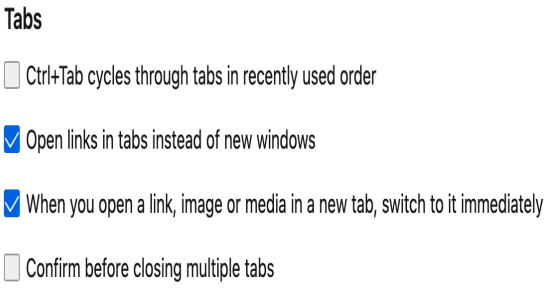
Part II: Modeling Input Configurations

The following three figures represent various configuration options for Firefox that can be set during testing. #1 defines the browser’s default search engine. #2 has three options for tabs that can be turned on or off independently (i.e. all three can be checked, all three can be unchecked and/or any combination of these can be checked). #3 has a radio button that can allow updates to either be installed automatically or to be installed manually. These cannot both be checked at the same time (they represent a choice of two options). The last option for this screen is a binary choice for allowing updates of searches.

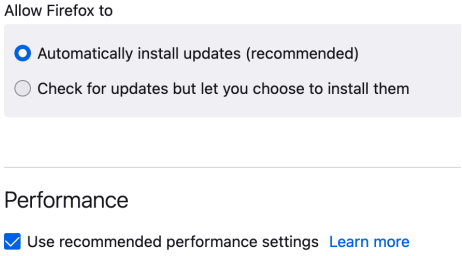
1



2



3



1. TSL for Firefox

Using this information create a TSL for this part of Firefox called firefox.tsl and generate the firefox.tsl.tsl file.

Question 3:

(a) How many test frames do you have in this file?

(b) Copy the contents of the firefox.tsl to the document. Copy the first and last 5 tests (10 in total of the firefox.tsl.tsl file to this document

2. CIT for Firefox

Create a 2-way sample using the CIT tool from our exercise. You will need to create a model file. Call this firefox.input.txt. Then use the tool to generate a 2-way sample.

Question 4.

(a) Hand in a screen shot of both of these files.
(b) Map the 2-way sample file to the configuration names and hand in the mapped file. This should match the file handed in in part a

(c) Assume you have a base choice

-- *"Bing"*, "check for updates but let you choose to install" ,"don't use performance settings"

(if an option does not appear above it is set to "unchecked")

Create a sample for **Base Choice Criteria**

Hand in the Base Choice criteria