Software Studio - Project 3

Anastasia Riana 邱慧莉 - 109006234

Installing Valgrind and Generating .dot File

On my device, I installed Valgrind in my ubuntu. Before running any commands, we need to first install the valgrind package. To do that, we need to run *sudo apt-get install valgrind*. Next, we need to change the directory where our input.c file is located. When we are in the right directory, we need to gcc the input file using the command *gcc input.c -o input*. Now, we are ready to run the given command by the TA by filling the executable and param fields with ./input and any number respectively.

Installing and Generating the .dot Graph

Before running the given command by TAs, we need to first install the required Python package. We will first run the command *sudo apt-get install python3 graphviz*. Then, we will also run *pip install gprof2dot*. We then can run the given command to generate the .dot graph. Generating the .png file of the graph can also be done by running the given command. The image will be shown at the end of the report.

Parsing the Input and Setting Up the Right Data Structure

Whenever we want to run the program, we must run this command to ensure that the program will run without any problems. The command is given below:

```
$ python3 109006234_project3.py < 109006234_out.dot
```

In this case, we will be using *sys.stdin.readlines()* to take the .dot file input and will be using the *.strip* and *.append* command to remove all empty spaces and append them to a list. When we print it out, we will get something like this:

```
digraph {

        graph [fontname=Arial, nodesep=0.125, ranksep=0.25];

        node [fontcolor=white, fontname=Arial, height=0, shape=box, style=filled, width=0];

        edge [fontname=Arial];

        "0x00000000000202b0" [color="#ff0000", fontcolor="#ffffff", fontsize="10.00", label="ld-linux-x8
6-64.so.2\n0x00000000000202b0\n100.00%\n(0.00%)\n0×"];

        "0x00000000000202b0" -> "_start" [arrowsize="1.00", color="#ff0000", fontcolor="#ff0000", fontsi
ze="10.00", label="100.00%\n1×", labeldistance="4.00", penwidth="4.00"];
```

However, we only need the last line of information, for example: *Ox00...202b0* as the source node and *_start* node in this assignment. Hence, we need to tidy the data up. To

remove the digraph information, we use *del* command and delete from line 0 to 3. We then use a Python list comprehension method, to only get the information that has ‘ -> ‘. Next, using another list comprehension, we remove all quotations from the string and store them in an array.

We are going to access the elements of the array arr using a for loop to separate the source and destination nodes. For every element in arr, we remove all the spaces and append the first element to the *source_arr* and the third element to the *dest_array*. We then combine these two lists together using the *zip* method and store it in the *temp* variable.

Next, we are setting up the dictionary data structure for us to do the topological sort. We first will convert the list that is stored in the *temp* variable to be a dictionary using a simple for-loop. Then, we are going to create another dictionary and set all destination nodes to have a value of an empty array. After that’s done, we combine those two dictionaries together and perform the topological sort.

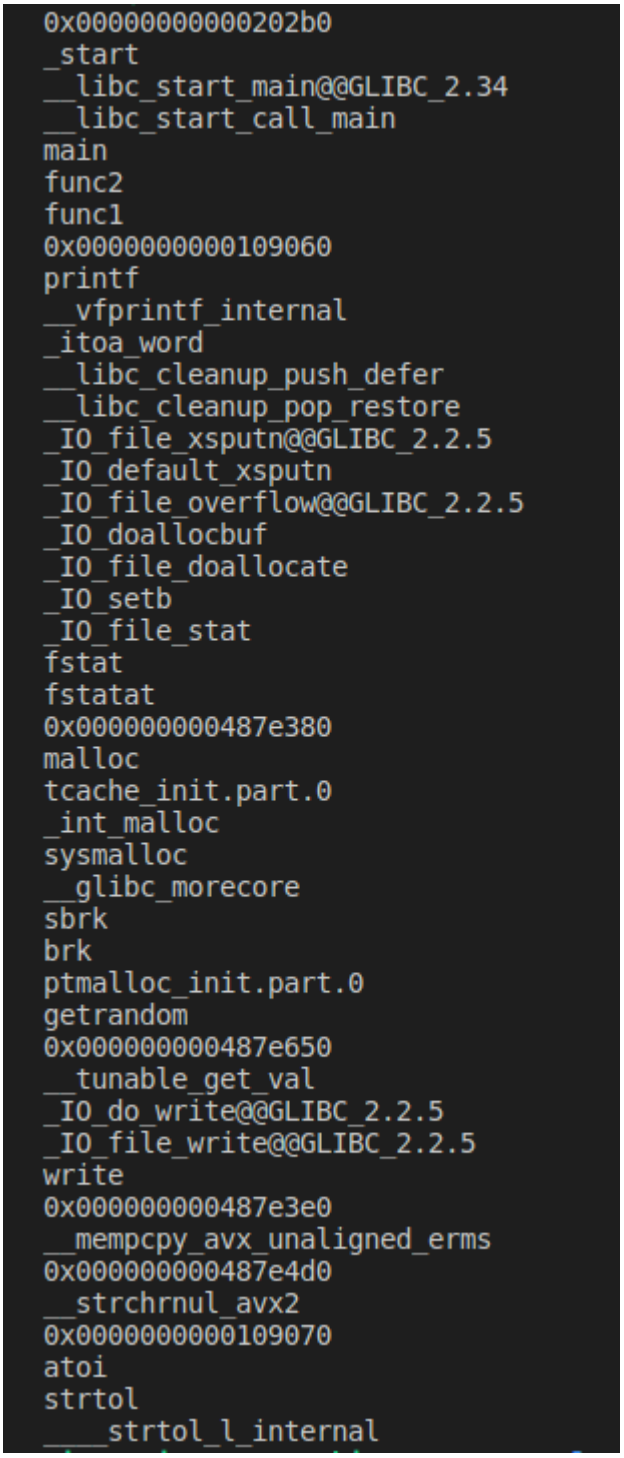
Topological Sort

Topological sort is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Using Python, a dictionary data structure is very suitable for doing a topological sort. In very simple terms, we first need to construct a dictionary with zero as its initial value. We then will iterate through the graph that we constructed earlier and create the adjacency matrix, while also computing the in-degree of each node. Then, we will iterate through the nodes that have no incoming edges, meaning the in-degree is zero.

As long as there are nodes with no incoming edges, we can keep appending it to a list. In that case, we will add the node to the list and also decrement the in-degree. Eventually, we will iterate all the nodes. With this, comes two possible outcomes, that are if we have already added all the nodes or if we encountered a cycle.

In topological sort, if the given graph has a cycle, we will have at least one node which is a parent and a child at the same time, thus this will break the topological order. Therefore, we need to raise an error. In this implementation, the error raised will just be a warning message. Else, it means that we have successfully done the topological sort and the function will return the list of nodes. For printing, we just need to do a for-loop and print each element in that list of nodes that the function had returned.

Result:


0x00000000000202b0
_start
__libc_start_main@@GLIBC_2.34
__libc_start_call_main
main
func2
func1
0x0000000000109060
printf
__vfprintf_internal
_itoa_word
__libc_cleanup_push_defer
__libc_cleanup_pop_restore
_IO_file_xsputn@@GLIBC_2.2.5
_IO_default_xsputn
_IO_file_overflow@@GLIBC_2.2.5
_IO_doallocbuf
_IO_file_doallocate
_IO_setb
_IO_file_stat
fstat
fstatat
0x000000000487e380
malloc
tcache_init.part.0
_int_malloc
sysmalloc
__glibc_morecore
sbrk
brk
ptmalloc_init.part.0
getrandom
0x000000000487e650
__tunable_get_val
_IO_do_write@@GLIBC_2.2.5
_IO_file_write@@GLIBC_2.2.5
write
0x000000000487e3e0
__mempcpy_avx_unaligned_erms
0x000000000487e4d0
__strchrnul_avx2
0x0000000000109070
atoi
strtol
____strtol_l_internal

Graph in .png file