



Applied Combinatorial Algorithms (5LIG0)

ASSIGNMENT 1: TOOLS AND SCRIPTS DOCUMENT

(v1.0)

2022

1 Introduction

This document is a step-by-step help for using tools and scripts that are provided with Assignment 1. In this text, we have tried to fully explain the use of the tools provided in popular operating systems. If you have any problem or you find a bug in provided tools, please contact with p.gohari.nazari@tue.nl. The specifications of the test system are abstracted in Table 1.

Table 1: Test system specifications

OS	Distribution	Version
Linux	Debian (Ubuntu)	20.04 LTS (Kernel: 5.9.16-generic)
Windows	-	Windows 10 (21H1)
Mac OS	-	11.1


Notice

Before asking questions, please read the document very carefully and try to make sure that you followed the steps correctly.

1.1 Basic requirements

For using provided tools recent versions of python3 is required. For installing python:

Assuming that you are using apt as a package manager, open a command prompt then you can easily install Python3 with the following commands:



```
1 $ sudo apt-get update
2 $ sudo apt-get install python3
```

To verify that everything work fine, execute the following command:

```
1 $ python3 --version
```

For installing python3 in mac, we highly recommend using a package manager like [Homebrew](https://brew.sh) (<https://brew.sh>). In the rest of the text, we will use Homebrew package manager, if you don't want to use package manager, you can download official version from [python site](https://python.org). Open a terminal and install Homebrew using the following command:

```
1 /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```



Next, add Homebrew to your PATH variable at the bottom of your ~/.profile file:

```
1 export PATH="/usr/local/opt/python/libexec/bin:$PATH"
```

Now, easily install python3 with the following command:

```
1 brew install python
```

Then, verify your installation with the following command:

```
1 python3 --version
```

In order to install python in windows simply download the installer from [python site](https://www.python.org/downloads/windows/) (https://www.python.org/downloads/windows/) or use Microsoft store.



After installing python, verify that your environment variables are set correctly. To doing this open CMD or PowerShell then:

```
python3 --version
```

After executing the last command in each OSs you should see an output like Fig. 1-1.

```
[pourya@Pouryas-iPro ~ % python3 --version  
Python 3.8.9
```

Figure 1-1: Desired output of python3 --version

2 Obtaining result for all provided inputs

For running your solution code for all of the provided inputs in Dataset1 we have prepared a script that runs your code for each input system (i.e., the combination of a mission and a configuration file), and gathers the results in the same folder structure as the input files.

First, extract the scripts folder that you downloaded from the CANVAS. Next, go to 1 _gathering_results folder and then linux folder. Now, copy the executable file of your code to this directory. In another word, if you are using C++ you should copy the compiled file of your code with .out format and if you are using python you should copy your code file with .py format.

Notice

If you want to use this script without any change, compiled C++ code should be named main.out and the python code should be named main.py.



Now, open a terminal in the current directory and if you are using python, use the following command:

```
$ ./gather_result_python.sh
```

Similarly, if you are using C++, use the following command:

```
$ ./gather_result_cpp.sh
```

By executing the above mentioned command, the script starts running your code with each provided input and gathers all results in the results folder.

First, extract the scripts folder that you downloaded from the CANVAS. Next, go to 1 _gathering_results folder and then windows folder. Now, copy the executable file of your code to this directory. In another word, if you are using C++ you should copy the compiled file of your code with .exe format and if you are using python you should copy your code file with .py format.

Notice

If you want to use this script without any change, compiled C++ code should be named main.exe and the python code should be named main.py.



Now, open a CMD or powerShell in the current directory and if you are using python, use the following command:

```
1 .\gather_result_python.cmd
```

Similarly, if you are using C++, use the following command:

```
1 .\gather_result_cpp.cmd
```

By executing the above mentioned command, the script start running your code with each provided input and gather all results in the results folder.

First, extract the scripts folder that you downloaded from the CANVAS. Next, go to 1 _gathering_results folder and then mac folder. Now, copy the executable file of your code to this directory. In another word, if you are using C++ you should copy the compiled file of your code with .out format and if you are using python you should copy your code file with .py format.

Notice

If you want to use this script without any change, compiled C++ code should be named main.out and the python code should be named main.py.



Now, open a terminal in the current directory and if you are using python, use the following command:

```
1 bash ./gather_result_python.sh
```

Similarly, if you are using C++, use the following command:

```
1 bash ./gather_result_cpp.sh
```

By executing the above mentioned command, the script start running your code with each provided input and gather all results in the results folder.

Hint

If you have a problem with running this script, you can open the script with a text editor and check the file directories again.

3 Compare results with baseline and sanity check

For comparing your results obtained from the previous step, we provided a script that checks your output and then generates the overall result in the CSV format (that you can open in Microsoft Excel application). For more information about the sanity check and its tests you can read section 4. Before using this script, Please make sure you have installed the required packages mentioned in section 4.1.



Go to 2_compare_results and then linux folder, then paste the results folder that you obtained from previous step (section 2). Now open a terminal in the current directory and use the following command:

```
1 $ ./runme.sh
```



Go to 2_compare_results and then windows folder, then paste the results folder that you obtained from previous step (section 2). Now open a CMD or powerShell in the current directory and use the following command:

```
1 .\runme.cmd
```



Go to 2_compare_results and then mac folder, then paste the results folder that you obtained from previous step (section 2). Now open a terminal in the current directory and use the following command:

```
1 bash ./runme.sh
```

If all goes well, at the end you should have a file named `final_result.csv` in the folder and if you open the file it should look like Fig 3-1.

comment	baseline makespan	your solution makespan	your solution exec. time	initial test	move test	makespan test	obstacle test	hauler test	charging test	mission test
Part1										
0-Easy										
Mission length=4										
1	31	31	10	True	True	True	True	True	True	True
2	26	26	1	True	True	True	True	True	True	True
3	31	31	5	True	True	True	True	True	True	True
4	32	32	7	True	True	True	True	True	True	True
5	63	63	8	True	True	True	True	True	True	True
Mission length=6										
1	51	51	53	True	True	True	True	True	True	True
2	46	46	45	True	True	True	True	True	True	True
3	53	53	34	True	True	True	True	True	True	True
4	46	46	32	True	True	True	True	True	True	True
5	83	83	65	True	True	True	True	True	True	True

Figure 3-1: An example of final output

4 Sanity check

This code is written with *python* and evaluates your code output based on the problem criteria.

4.1 Required packages

Sanity check just needs one extra package to work. To install the following package use the following command in your OS terminal:

```
1 pip3 install docopt
```

Notice

If you want to run the code inside an IDE¹ (like *PyCharm*) please make sure that you have installed all of the above-mentioned packages in its virtual environment.

4.2 Input/output structure

The sanity check gets three `config.txt` (the problem configuration file), `mission.txt` (the missions of each hauler), and `output.txt` (your solution output file) as inputs and generates a result file (default: `result.txt`). In the following you can see an example of result output and its format:

```
1 106      //Makespan
2 240      //Elapsed time
3 True     //Initial test
4 True     //Move test
5 True     //Makespan test
6 True     //Obstacle test
7 True     //Hauler test
8 True     //Charging test
9 True     //Mission test
```

In this format, `True` means that your solution output passed that test and `False` means that your solution output failed that test. Details of each test are given in Table 2.

Table 2: Test system specifications

Test	Details
Initial test	Checking if the haulers started in their initial position
Move test	Checking the movements of each haulers and finding possible jump (moving more than one unit in time)
Makespan test	Checking if the reported makespan is correct or not
Obstacle test	Checking if there is a collision between the haulers' position and obstacles
Hauler test	Checking if there is a collision between two haulers
Charging test	Checking if there is out of charge condition or not, and checking charging time
Mission test	Checking if all mission goals are met or not

¹Integrated development environment

4.3 Command-line arguments

Sanity check code has three optional arguments that are shown in the following:

```

1 Usage:
2   sanity_check.py [-c FILE] [-m FILE] [-o FILE]
3   sanity_check.py -h
4
5 Options:
6   -c FILE, --config FILE      config file [default: config.txt]
7   -m FILE, --mission FILE     mission file [default: mission.txt]
8   -o FILE, --output FILE      output file [default: output.txt]
9   -h, --help                  show this message

```

The arguments are config file, mission file, and output file locations. If you don't pass any argument to the code, it considers that these files are located in the same folder of the code with the default names.

4.4 Running

In this section, we explain how you should run and use sanity check code with two example.

Example 1: Assuming that you have a folder that contain the sanity check code (`sanity_check.py`), and all input files as shown in Fig. 4-1.

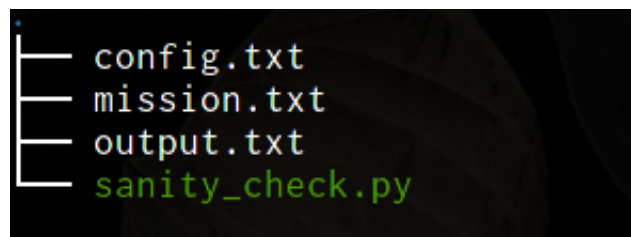


Figure 4-1: Default file structure for running `sanity_check.py`

Now, for running the code, you can simply open a terminal in the code directory and use the following command:

```
1 python3 sanity_check.py
```

Example 2: If you don't want to use the code with default arguments you can choose the location of inputs file using the command-line arguments. For example consider that we have `config.txt` in `~/input1/` directory and `mission.txt` in `~/input2/` directory and finally `output.txt` in `~/output1/` directory. Now for running the code we should pass input files locations to the code with the following command:

```
1 python3 sanity_check.py -c ~/input1/config.txt -m ~/input2/mission.txt -o ~/
  output1/output.txt
```

Notice

If your folders or file-names contain a space character, you should add quotation mark (") before and after each address. For example:

```
1 python3 sanity_check.py -c "~/input1/blah blah/config.txt" -m ~/input2/mission.
  txt -o ~/output1/output.txt
```