
Assignment 2: Tracking Flagged Cars

7% of overall grade
Due Friday 28 September 2018, 11:55pm

1 Overview

1.1 Introduction

Thanks to advances in computer and network technologies we live in an age of increasingly pervasive surveillance. As with most things, there are pros and cons to such widespread surveillance but we will leave that philosophical discussion to your Philosophy courses and/or COSC101. This assignment isn't going to take sides in the debate—our aim is simply to give you some insight into how data from such surveillance systems might be used efficiently to meet whatever good/evil ends that one might have for the data.

1.2 Due date

Your assignment code should be submitted by Friday 28 September 2018, 11:55pm. The drop dead date is 7 days later, and any late assignments will incur a 15% absolute penalty. That is your mark will be the raw mark less 15 marks (given the assignment is marked out of 100).

1.3 Submission

Submit via the quiz server. The submission page will be open closer to the due date.

1.4 Implementation

All the files you need for this assignment are on the quiz server. You must not import any additional standard libraries unless explicitly given permission within the task outline. For each section there is a file with a function for you to fill in. You need to complete these functions, but you can also write other functions that these functions call if you wish. All submitted code needs to pass `PYLINT` program checking (it will make you proud of your stylish code).

1.5 Getting help

The work in this assignment is to be carried out individually, and what you submit needs to be your own work. You must not discuss code-level details with anyone other than the course tutors and lecturers. You are, however, permitted to discuss high-level details of the program with other students. If you get stuck on a programming issue, you are encouraged to ask your tutor or the lecturer for help. You may not copy material from books, the internet, or other students. We will be checking carefully for copied work. If you have a general assignment question or need clarification on how something should work, please use the class forums on Learn, as this enables everyone to see responses to common issues, but **never** post your own program code to Learn. Remember that the main point of this assignment is for you to exercise what you have learned from lectures and labs, so make sure you do the relevant labs first, so don't cheat yourself of the learning by having someone else write material for you.

2 Tracking Flagged cars

This assignment follows on from the first assignment and will be primarily focused on the use of hash tables to provide fast look-up and storage of data. You will again be working with the data from a roadside camera based Automatic Number Plate Recogniser (ANPR) but in this assignment each camera sighting will come with a time stamp and you will be aiming to build up a table containing the list of times that each *flagged* vehicle was seen by the camera. A database of number plates, with *flags* will be provided and you will need to store the database in a hash table for quick look-up. Vehicles of interest will be flagged with such things as *stolen*, *outstanding fines*, etc.... When processing the sightings from the camera you will build up another hash table that maps from number plates to lists of time stamps.

2.1 General Overview

- Building linear probing and chaining hash tables that operate in a similar way to Python dictionaries. For example you can do things like `my_table[plate] = flag`, `flag = my_table[plate]` and `plate in my_table`.
- Write code for storing the plate database, ie, storing flags for plates so that the flags can be accessed using `flag = database_table[plate]`
- Write code that builds a chaining hash table containing the time stamps for sightings of vehicles that have flags (ie, are interesting). Using a chaining hash table will allow the table to handle as many plates as you want to store (without getting full and needing to be resized). Once built looking up the list of sightings for a given number plate will be quick, eg,
`sighting_list = results_table[NumberPlate]` will be fast.

3 Supplied Code and Data

3.1 Provided Skeleton Code

In this assignment you will be completing the implementation of two hash table classes and two functions (plus a little bit of counting in the `ListTable` class). The

skeleton code for these is provided in the `hashing.py` module. More details on what you need to do is provided in the task outlines below (and in the docstrings).

We also provide a class called `ListTable` that gives you an example of how your hash tables should be working. It uses a list to store items and searches the list with a sequential algorithm, rather than using hashing. This class will work for small data sets but it's main value is that it demonstrates the horrible slowness of such sequential/linear searching!

Note: you will need to add the code for plate comparison counting to the `ListTable` class.

3.2 Provided classes


The `classes2.py` module contains the definition for the `NumberPlate` class. The `NumberPlate` class is basically a wrapper around the existing Python `str` class, but with the added benefit of being able to keep track of every `NumberPlate` comparison that is made. See below for more details on how the `NumberPlate` class differs from that used in assignment 1.

NumberPlate : The most fundamental class is the `NumberPlate`. For now, you may consider a `NumberPlate` to be a `str`, in that all the common comparisons (`==`, `<`, `>`, `<=`, `>=`, `!=`) are available (eg, if `a` and `b` are `NumberPlates` then expressions like `a < b` and `a == b`, etc, will work as expected). The difference is that we keep track of these comparisons in the background, via the `StatCounter` class. You will now also be able to hash number plates using `hash(my_plate)`. The hash method that is provided will use a small fast hashing function that will give the same value across different runs of your program (unlike the built-in Python hash function, which randomises hash values between runs). You should be counting how many `NumberPlate` comparisons and hashes that your hash tables use — via their `self.n_plate_comparisons` and `self.n_plate_hashes` instance variables. Note, this means your functions don't need to return the counts.

Note: For the purposes of this assignment, time stamps will simply be ISO8601 formatted strings, eg, `'2017-05-22T19:51:16'`.

The `stats.py` module contains another class that you may find useful: `StatCounter`. This class holds all the comparison information that you might need to check your code against. We also use it to verify that you do perform the correct number of comparisons.

StatCounter : In order to count how many comparisons your code makes, we provide a `StatCounter` class. Note that because of the way it is implemented, it will not behave like a regular class. This class is only for testing purposes, and not in your final code. You should only use the `StatCounter.get_count(counter_name)` method for testing and should remove all calls to it from the code you submit to the Quiz Server. You will, of course, need to remove any imports related to `stats` from your code—otherwise *pylint* might get angry with an unused import.

 **Important:** You cannot rely on `StatCounter` (or more specifically, the methods and instance variables in `StatCounter`) being available to you on the quiz server at the time of submission, so do not use it in your final code.

3.3 Testing your code

Off-line tests are available in the `tests.py` module and you should use them to check your implemented code before submission (the submission quiz won't fully mark your code until submissions close). The provided tests use the Python `unittest` framework. You are not expected to know how these tests work; you just need to know that the tests will check that your code finds the list of stolen number plates that were sighted, and that the number of comparisons that your code makes is appropriate.

You should also be doing your own testing, eg, of various edge cases such as when an input list is empty or when no number plates are sighted etc...

3.3.1 Provided tools and test data

The `utilities.py` module contains functions for reading test data from test files. Test files and expected output files are in the folder `test_data` to make it easier to test your own code.

The most useful function in `utilities.py` is obviously `read_dataset`, it reads the contents of a test file and returns three lists.

The test data files are named `db[s]-sighted-matches-seed.txt`, where *db* is the number of plates in the database [followed by an *s* to indicate that they are in sorted order]; *sighted* is the number of plate sightings by the camera (note each plate may be sighted more than once); *matches* is the number of sightings that were flagged plates (for you to check that you get the right ones) and, finally, the *seed* is the seed value that was used when generating the test file.

For example `10-20-5-a.txt` was generated with `a` as the random key and has a database of 10 plates with 20 plate being sighted by the camera and 5 of the plate sightings were flagged vehicles (note there may have been less than 5 vehicles sighted as this is the number of time stamped sightings and each vehicle can be sighted more than once). The list of sighted flagged plates is sorted by plate and then by time stamp.

If the file name was `10s-20-5-a.txt` then the database would be provided in sorted order — most of the test data files will use sorted database entries (as this is likely the way that you would receive a database file). You should open some of the test data files and make sure you understand the format—remember to open them in Wing or a suitably smart text editor. If you are using windows then try Notepad++ because the normal Notepad is pretty horrible ¹. If you are using Linux (or a Mac) then almost any text editor will do.

The test files are generated in a quasi-random fashion and the seed suffix indicates the random seed that was used when generating the data—you don't need to worry about this. But, you do need to worry about the fact that we can easily generate different random files to test with.

The expected output files contain string representations of the hash tables expected from various test files. The specifics of these expected files will be explained in the relevant task sections below.

¹There will soon be an update to Windoze 10 that finally gives Notepad the ability to deal with line endings properly, like every other text editor on the planet. Yay!

The following example shows how to use the `read_dataset` function from the `utilities` module:

```
# from utilities import read_dataset # if you need it for the example
# note you can ignore the indent below they are just typesetting in this document
filename = './test_data/5s-5-2-a.txt'
db_list, sighted_list, matches_list = read_dataset(filename)
print(db_list)
[('DZ4997', ''), ('HQ9423', ''), ('LA5930', 'Crusaders supporter'), ('RC3494', ''),
 ('SH5242', '')]
print(sighted_list)
[('SH5242', '2017-01-01T00:03:17'), ('HQ9423', '2017-01-01T00:03:52'), ('LA5930',
 '2017-01-01T00:04:11'), ('HN8360', '2017-01-01T00:04:34'), ('LA5930',
 '2017-01-01T00:04:42')]
print(matches_list)
[('LA5930', '2017-01-01T00:04:11'), ('LA5930', '2017-01-01T00:04:42')]
print(type(db_list[0]))
<class 'classes.NumberPlate'>
```

3.3.2 Provided tests

The `tests.py` provides a number of tests to perform on your code. You can adjust which tests you are running by commenting/uncommenting the lines that add test cases to the test suite (much the same as in assignment 1). Each test has a name indicating what test data it is using, what it is testing for, and a contained class indicating which algorithm is being tested. In the case of a test case failing, the test case will print which assertion failed or what exception was thrown. The `all_tests_suite()` function has a number of lines commented out indicating that the commented out tests will be skipped; uncomment these lines out as you progress through the assignment tasks to run subsequent tests.

As with the first assignment, these tests aren't going to be very helpful for debugging your code. You will want to write your own tests using small hash tables and small amounts of data. This will let you get a feel for how your hash tables are storing data and whether or not they are operating as expected (because you will be able to work out how they should be working by hand).

Important: In addition to the provided tests you are expected to do your own testing of your code. This should include testing the trivial cases such as empty lists and lists of differing sizes.

4 Tasks [100 Marks Total]

4.1 Linear Probing Hash Table [30 Marks]

To get warmed up you should check out the `ListTable` class to get a feel for the worst case way of making a look-up table. The code is working but you must add in some code to keep track of any plate comparisons that are made (eg, insert some `self.n_plate_comparisons += 1` lines where appropriate).

Once warmed up you can move onto the main task which requires you to complete the `LinearHashTable` class so that it implements a linear probing hash table. The data should be stored in the `table_list` instance variable as setup in the provided `__init__` method.

The main work here will be in completing the `__set_item__`, `__get_item__`, and `__contains__` methods. You should count all number plate comparisons in the `n_plate_comparisons` instance variable (eg, do a `self.n_plate_comparisons += 1`

each time number plates are compared. You should also count the number of times number plates are hashed using the `n_plate_hashes` instance variable. It will also be important to keep track of the number of items stored in the hash table (via `n_items` as this instance variable can then be used to tell when the table is full.

The methods that you define will mean that the hash table can be used in much the same way as a Python dictionary. For example,

```
plate = NumberPlate('ABC231')
flag = 'Overdue fines'
table = LinearHashTable(11)
table[plate] = flag      # translates to table.__set_item__(flag)
print(plate in table)    # translates to table.__contains__(flag)
print(table[plate])      # translates to table.__get_item__(plate)
table[plate] = 'new_flag' # translates to table.__set_item__(plate)
plate2 = NumberPlate('BBB222')
table[plate2] = 'BBB222-flag'
#
print(hash(plate)%11) # gives 7
print(hash(plate2)%11) # gives 9
# so we expect the plates to be in indexes/slots 7 and 9
print(table)
# Printing the table should give the following
Linear Hash Table:
0-> None
1-> None
2-> None
3-> None
4-> None
5-> None
6-> None
7-> ('ABC231', 'new_flag')
8-> None
9-> ('BBB222', 'BBB222-flag')
10-> None
```

Some examples of usage are provided in the example functions in the hashing module. Feel free to play around with them to help you get a feel for how your tables are operating.

Now might also be a good time to complete the code for the `make_db_hash_table` function as it will help you test your hash table. See topic [4.3](#) below.

4.2 Chaining Hash Table [30 Marks]

In this task you need to complete the implementation of the `ChainingHashTable` class. Its interface should be the same as the `LinearHashTable` class but under the bonnet it will use chaining to resolve hash collisions. To store the *chain* of items in each slot it uses a Python list. The initialiser method will setup the `table_list` with an empty Python list in each slot (ie, [] in each slot).

Because chaining is being used the table can contain more items than slots. That is, `n_items` can be greater than `n_slots` or in terms of the load factor, having $\lambda > 1$ is possible.

The example below shows what a table will look like after adding a few flags.

```
plate1 = NumberPlate('BAA754')
plate2 = NumberPlate('M00123')
plate3 = NumberPlate('W0F833')
plate4 = NumberPlate('EEK001')

table = ChainingHashTable(5)
table[plate1] = 'Sheep'
table[plate2] = 'Cow'
table[plate3] = 'Dog'
```

```

table[plate4] = 'Mouse'
print(table)
# gives
0-> [('WOF833', 'Dog')]
1-> [('BAA754', 'Sheep'), ('M00123', 'Cow')]
2-> []
3-> []
4-> [('EEK001', 'Mouse')]

```

Note: Our testing will include using a ChainingHashTable to store database data but the final usage of the ChainingHashTable will use NumberPlates as the key and lists of time stamps as the values. See the next section for details and examples of this.

If you haven't done so already, now would be a good time to complete the code for the `make_db_hash_table` function. Then you can try building a chaining hash table with the database data. See 4.3 below.

4.3 Processing Camera Sightings [30 Marks]

Your task here is to complete the `process_camera_stream` function. The `process_camera_stream` function takes a database list and a sighted list, followed by the size of the table to use for the database and the size of the table to use for storing the timestamps of sighted flagged vehicles. The `process_camera_stream` function should return a tuple containing the resulting database table and the resulting sighted flagged vehicles table. The database table should be a LinearHashTable and the results table should be a ChainingHashTable.

The database table is generated by going through all the (plate, flag) pairs in the database list and adding/storing them to the database table. You should complete the `make_db_hash_table` function and use it for doing this. You should test building database tables with both Linear Probing and Chaining hash tables. Expected output for each is given in the files prefixed with `expected_db_linear_` and `expected_db_chaining_`. Have a look at a few of those files, in the `test_data` folder, to see what the resulting database tables look like.

The results table is generated by going through all the plates in the sighted list and recording the time stamp for each plate that has a flag in the database (the contents of the flag doesn't matter, just the fact that it has one). If you think of the results table as a dictionary then the keys are plates and the value associated with each key will be a list of time stamps.

The example results table output files should give you a good idea of what the final results tables should look like. The contents of one of these files is given below.

4.3.1 Expected output files

For this task the expected results table files have the obvious prefix (`expected_results_table`) followed by the following values:

```
{n_db}[s]-{n_sighted}-{n_flagged_stamps}-{seed}-{db_table_size}-{results_table_size}
```

For example `expected_results_table_10s-10000-10-a-20-5.txt` gives the expected results table when using the input from the `10s-10000-10` file with a database table of size 20 and a results table of size 5. Notice that each slot contains a list of tuples of the form (plate, time_stamp_list).

```

Chaining Hash Table:
0-> [('BI7323', ['2017-05-22T16:54:23', '2017-05-23T10:33:45', '2017-05-25T12:34:24'])]
1-> []
2-> [('KA2801', ['2017-05-22T15:24:38', '2017-05-23T06:13:42'])]

```

```
3-> [('TQ5014', ['2017-05-23T11:03:17']), ('JI6816', ['2017-05-24T16:00:52'])]  
4-> [('HT9954', ['2017-05-22T19:51:16', '2017-05-23T21:16:52', '2017-05-25T00:39:27'])]
```

4.4 General Hash Table questions [10 marks]

Once you have all your code working and have had some time to experiment with various test data files. You can answer a few short answer questions about hash tables and hashing. You will need to provide answers in the appropriate text boxes in the submission quiz. The questions you will need to answer are list below.

- The hash function we have been using meets the requirements of a “good” hash function. What are these requirements?
- A linear hash table seems to perform poorly when the table has a high load factor. Why might this be?
- What happens when the linear probing hash table has a load factor greater than one? And what happens when the chaining hash table has a load factor greater than one?
- When deciding on the size of a hash table, what trade-offs are we forced to make?
- Chaining and linear probing are both methods to handle collisions in a hash table. What other methods are there?

5 Updates and clarifications

Please keep an eye on the new forums for any updates or clarifications that might come out during the course of the assignment.

— Have fun —