Computer Laboratory 4

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Introduction

Imagine a word in which python does not have a dictionary data structure. The dictionary data structure is core to many problems, so naturally this would be a major issue. Most programmers would react to this situation by writing functions to simulate the behavior of a dictionary. This is the approach we are going to take – we are going to "pretend" that dictionary is not implemented in Python and write functions to implement our own.

In this lab, we will simulate a dictionary using two different versions of what's known as an "association list" structure. In an association list, each key–value pair is stored in a tuple, with the collection of key–value pairs stored in a list. (I.E. the associations are stored in a list, hence, association list).

We will explore two approaches to this:

1. An unsorted association list
2. A sorted association list

In an unsorted association list, associations can be put in any order, and a linear search can be used to search for a key in the list of associations. This leads to very simple code, which runs in $O(n)$ time.

In a sorted association list, the associations are stored sorted by their key. This requires more careful code, but can be much more efficient. In particular, by using a binary search to find key–value pairs we can reduce most operations to a $O(\log(n))$ running time.

This lab will:

- give you more practice working with lists and tuples
- give you a better understanding of dictionaries
- give you practice implementing linear and binary searches.

2 Software environment setup

Like the previous lab, you will be doing python programming. While you are allowed to prepare your python programs in any reasonable programming environment we continue to recommend PyCharm. The basic PyCharm setup instructions can be found in the previous lab. Some notes here:

1. You will want to make a new PyCharm Project for this new lab. Mixing labs in one project can lead to accidentally submitting the wrong file or other "carry-over" from lab to lab that might interfere with grading.

2. If PyCharm asks you to set a project interpreter, and no default shows up, you should be able to add one. On the CSELabs machines "/usr/bin/python3" is the name of the default python3 interpreter, which is a fine choice.

3. Remember to note where PyCharm makes the python project itself, you may even want to make sure the project is placed somewhere designated for this class when you set it up. Eventually you will need to find those files outside of PyCharm.

3 Lab Reminders

- You are allowed to change lab partners each week if you want, you are never locked into one partner, ask the TAs if you want help finding a partner

- Each programmer in a partnership should be equal – try not to run ahead of your partner, make sure they understand what you are doing, and make sure that each programmer spends equal time on the keyboard. (Remember, true pair programming is two programmers, one keyboard, one screen).

- If you do not finish, you and your partner should continue working together outside of lab to finish the lab. I hold both students in a partnership academically responsible for your solution (I expect either student to be able to explain their code to me)

4 Files

This lab will involve the following files

- `assoc_lists.py` You should be writing this module. This is the name the tests will expect.

- `assoc_list_test.py` This file is provided. It has tests for all required functions. It can be found on canvas and will serve as the basis for our grading process, as well as a helpful tool for your personal development.

5 Software Design

We will be representing an association between a key and a value in python using a tuple with two elements, the first being the value, and the second being the key. While this choice is arbitrary (I.E. basically any representation and order could work) this is probably not your first-choice for representations so be careful when coding that you don't get the ordering of these elements confused.

We will store two types of association lists – linear lists (unsorted) and binary lists (sorted by key value).

So the following dictionary:

```
bin_list = {
    "crow" : 3,
    "raven" : 49,
    "fish" : 2,
    "cat" : 13,
}
```

would be represented like this as a linear association list:

```
[(3, 'crow'), (49, 'raven'), (2, 'fish'), (13, 'cat')]
```

and would be represented like this as a binary list:

```
[(13, 'cat'), (3, 'crow'), (2, 'fish'), (49 'raven')]
```

We will place no restrictions on plausible value objects, but, due to the need to sort, we will only use key objects that are comparable data types in python (I.E. data types which allow you to use < and >) This means we can use numbers (int or float) or strings.

The functions we need are based on the core operations in the dictionary class:

- contains(dict, key) returns True or False denoting if the key is in an association in the dictionary or not.
- get(dict, key), returns the value associated with the key in the dictionary (if possible) or the special value `None` in python if key is not in the dictionary (Note, `None` without quotes, it's a special value, not a string)
- put(dict, key, value) modifies the dictionary so that the given key is now associated with the given value. If key was already associated with something the old value should be updated to the new value. If the key was not associated with a value in the dictionary, a new value association should be added to the dictionary.

The test file has examples that might help you understand these functions better if needed.

6 Required Functions

There are six required functions in this lab, three representing the linear search (unsorted) association list, and three representing the binary search (sorted) association list.

- `lin_contains(lin_dict, key)` contains function for use with a linear (unsorted) association lists. Expected to run in $O(n)$ time
- `lin_get(lin_dict, key)` get function for use with linear (unsorted) association lists. Expected to run in $O(n)$ time
- `lin_put(lin_dict, key, value)` put function for use with linear (unsorted) association lists. Expected to run in $O(n)$ time
- `bin_contains(bin_dict, key)` contains function for use with binary (sorted) association lists. Expected to run in $O(log(n))$ time
- `bin_get(bin_dict, key)` get function for use with binary (sorted) association lists. Expected to run in $O(log(n))$ time
- `bin_put(bin_dict, key, value)` put function for use with binary (sorted) association lists. Expected to run in $O(n)$ time. Note, this function is required to maintain the sorted order of `bin_dict`.

The behavior of the three primary functions (contains, get, put) are as described above. The get and contains methods have required return value behavior, but the put functions do not. You can return whatever you want from the put functions, so long as the expected changes to the association list is made.

The big-O runtime behavior listed above are requirements for this code, the three lin_ functions should run in linear $O(n)$ time. The bin_get and bin_contains functions are required to use a binary search algorithm and run in $O(log(n))$ time. (It is not enough to just use a binary search, if you loop over the entire list afterwords for any reason, you will still have $O(n)$ runtime.)

bin_put must maintain sorted order of bin_dict. This means that if it needs to add a new key, it will need to figure out where the key goes in the list. While it is technically possible to do this with a binary search, it requires a modification we did not discuss. Therefore, we do not require using binary search in the bin_put function. This is not a major concern, however, as adding an element into the middle of a list is an $O(n)$ process – therefore this function is not possible to do in $O(log(n))$ anyway.

While you can consult the textbook's code for binary search if needed, you should know in advance that the code will not work without modification due to the fact that associations are stored as tuples ordered value then key. You should try, however, to write this algorithm from memory / first-principals. While it is a bit harder, you stand to learn more from the lab if you do.

7 Deliverable

Before submitting your work, please use the tests to carefully check that your code is bug-free, and has every function named as expected, and returning data types compatible with our expectations.

Name your submission file `assoc_lists.py` and make sure that a comment at the top has your name, and if you worked with a partner, your partner's name. **If you worked without a partner and their name is not on your file they will not get credit, OR, you will be accused of cheating** (depending on if they turn in the file or not)

You should only submit the `assoc_lists.py` file. You will submit this through canvas. If you worked with a lab partner, **BOTH STUDENTS SHOULD SUBMIT A COPY.** This is different than previous weeks, we believe it will be easier to grade submissions quickly if each student turns in their own copy.
EVERY STUDENT IS EXPECTED TO SUBMIT A COPY OF THE LAB! Even if they worked with a partner. It is OK if you and your partner submit the same file. We just want everyone to have a file submitted.

This will be due before the beginning of your next lab next week. The exact time will be based on the official start time of your lab. Canvas will accept re-submissions, and will also accept late submissions. Please be sure you do not turn in the assignment late on accident.

If you finish this lab during the lab time, feel free to notify your lab TAs, and then leave early. If you do not finish this lab during the lab time you are responsible for finishing before the next lab.