CS102L Data Structures & Algorithms
Lab 9

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

Spring 2024

Contents

Introduction

1.1 Instructions

- This lab will contribute 1% towards your final grade.
- The deadline for submission of this lab is at the end of lab time.
- The lab must be submitted online via CANVAS. You are required to submit a zip file that contains all the *.py* files.
- The zip file should be named as *Lab_09_aa1234.zip* where *aa1234* will be replaced with your student id.
- **Files that don't follow the appropriate naming convention will not be graded.**
- **Your final grade will comprise of both your submission and your lab performance.**

1.2 Marking scheme

This lab will be marked out of 100.

- 50 Marks are for the completion of the lab.
- 50 Marks are for progress and attendance during the lab.

1.3 Lab Objectives

- To understand and implement hashing in Python
- To use hashing in different exercises

1.4 Late submission policy

There is no late submission policy for the lab.

1.5 Use of AI

Taking help from any AI-based tools such as ChatGPT is strictly prohibited and will be considered plagiarism.

1.6 Viva

Course staff may call any student for Viva to provide an explanation for their submission.

Prerequisites

2.1 Pytest

Pytest is a popular testing framework in Python known for its simplicity and power in facilitating efficient and scalable testing for applications and software. In order to use Pytest, one first needs to install it and create test functions prefixed with *test_* in a Python file. To install Pytest, you can write the following command on the terminal:

```
pip install pytest
```

After installing Pytest, one can execute all the tests by typing the following command on the terminal:

```
pytest
```

If you want to test any particular file, then you will have to specify the filename as follows:

```
pytest <filename>.py
```

In case the above mentioned pytest commands do not work on your system, even though pytest is installed, try using the following commands:

```
py -m pytest
```

```
py -m pytest <filename>.py
```

If all the tests pass successfully, then you should receive a similar output.

```
PS C:\Work\Spring2024\DSA\Lab1> pytest test_Question1.py
============================= test session starts ==============================
platform win32 -- Python 3.11.2, pytest-7.4.4, pluggy-1.3.0
rootdir: C:\Work\Spring2024\DSA\Lab1
collected 10 items

test_Question1.py ..........

============================== 10 passed in 0.06s ==============================
```

However, if all the tests fail, then you should receive a similar output.

```
PS C:\Work\Spring2024\DSA\Lab1> pytest test_Question1.py
============================= test session starts ==============================
platform win32 -- Python 3.11.2, pytest-7.4.4, pluggy-1.3.0
rootdir: C:\Work\Spring2024\DSA\Lab1
collected 10 items

test_Question1.py FFFFFFFFFF

================================== FAILURES ===================================
_____________________________ test_question1[1st10-1st20-e54dc15ccafa608142ffa6340b035c327f972e24882052cfc3345f240f4ee237] ___
```

Exercises

3.1 Creating a Hash Table

Write a function called create_hashtable that creates a hash table and returns it. You will represent your hash table as a tuple with two lists in it. The first list represents the keys and the second list represents their associated values. Keys and values should be stored at the same index in both of the lists i.e., a key at index 1 in the first list will have its associated value at index 1 in the second list. This function takes a variable called size which would be the size of the hash table i.e., the size of both of the lists inside the tuple.

3.1.1 Examples

```
>>> size = 4
>>> print(create_hashtable(size))
([None, None, None, None], [None, None, None, None])
```

3.1.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question1.py
```

```
py -m pytest test_Question1.py
```

3.2 Creating a Hash Function

A hash function is a function that gives the hash of a given key. This hash is used as an index in hash tables. You will create the following two functions:

- hash_function(key,size)
  - Here, the key represents the key out of a key-value pair and size is the size of the hash table.

  - If the key is of type integer, use the Divide Modulo function, i.e. key % size to generate and return the index where the key and its value should be placed.
  - If the key is of type string, then the index will be calculated by the sum of the Unicode/ASCII values of each letter in the string. Don't forget to get the modulus value so that it remains within the range of the hash table size.

- collision_resolver(key,size, iteration)
  - Use Linear Probing to resolve collisions
  - Here, the key represents the previous hashed value, size is the size of the hash table, and iteration is the iteration number to get the next interval in case of collisions
  - Get the new hash value by adding *iteration* to the previous value and taking a mod of it with size.

3.2.1 Examples

```
>>> key = "Hello"
>>> size = 11
>>> iteration = 2
>>> print(hash_function(key,size))
>>> print(collision\_resolver(key,size,iteration))
5
7
```

3.2.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question2.py
```

```
py -m pytest test_Question2.py
```

3.3 Get and Put

A hash table would need a getter and setter function which would place the key-value pair in their appropriate location and get the value of a key when called. Create the following two functions to implement the primary getter and setter operations of hashing:

3.3.1 Put

Algorihtm:

```
function put(hashtable, key, value, size)
    hashvalue = hash of key
    if hashtable[hashvalue] is nil
        place key at index hashvalue in keys list in hashtable
        place value at index hashvalue in values list in
   hashtable
    else
        if hashtable[hashvalue] = key
            place value at index hashvalue in values list in
   hashtable
        else
            newhash = resolve collision using iteration
            while hashtable[pos] is not empty and hashtable[pos
   ] is not equals to key
                newhash = resolve collision using iteration
            if hashtable[newhash] is nil
                place key at index hashvalue in keys list in
   hashtable
                place value at index hashvalue in values list
   in hashtable
            else
                if hashtable[newhash] = key
                    place value at index hashvalue in values
   list in hashtable
```

3.3.2 Get

Algorithm:

```
function get(hashtable, key, size)
    start = hash of key
    pos = start
    while hashtable[pos] is not empty
        if key = hashtable[pos] and hashtable[pos] is not
   deleted then
            return hashtable[pos]
        else
            pos = resolve collision using iteration
            if pos == start
                break
    return nil
```

3.3.3 Examples

```
>>> H = create_hashtable(5)
>>> put(H,"Hello","World",5)
>>> print(get(H,"Hello",5))
World
```

3.3.4 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question3.py
```

```
py -m pytest test_Question3.py
```

3.4 Delete

Deleting a key value pair from a hash table is a critical part in the representation of a hash table. Create the following function which remove a key value pair from a hash table and in place of it places a tombstone (the hashtag symbol: #) which signifies that something has been deleted and this index was not originally empty. Algorithm:

```
function delete(hashtable, key, size)
    hashvalue = hash of key
    if hashtable[hashvalue] is not key
        while hashtable[hashvalue] is not key
            newhash = resolve collision using iteration
    place # at index newhash in keys list in hashtable
    place # at index newhash in values list in hashtable
```

3.4.1 Examples

```
>>> H = create_hashtable(5)
>>> put(H,"Hello","World",5)
>>> delete(H,"Hello",5)
>>> print(H)
(['#', None, None, None, None], ['#', None, None, None, None
   ])
```

3.4.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question4.py
```

```
py -m pytest test_Question4.py
```

3.5 Practically Checking the Hash Table

Create a function called main that simulates how a hash table works. It will firstly take as a parameter the size of a hash table. It would also take the following 3 lists as parameters:

- to_put - This is a list of tuples where each tuple contains a key value pair. These are to be added to a hash table.
- to_delete - This is a list of keys whose associated key value pair has to be deleted from the hash table.
- to_get - This is a list of keys whose values are to be retrieved and stored in a list.

You will first create a hash table based on the given size and then do the necessary operations based on the available data. This function should return a list of values which are associated with the keys in the to_get list. It should also return the hash table itself as well. These should be in a tuple that would look like: (values to get, hash table).

3.5.1 Examples

```
>>> size = 5
>>> to_put = [(1,2),("key","value")]
>>> to_delete = [1]
>>> to_get = ["key"]
>>> print(main(to_put, to_delete, to_get, size))
(['value'], ([None, '#', None, 'key', None], [None, '#',
   None, 'value', None]))
```

3.5.2 Testing

To test your function, type one of the following commands in the terminal:

```
pytest test_Question5.py
```

```
py -m pytest test_Question5.py
```