Project 3 Hash Tables

CECS 274 Data Structures Fall 2020

Deadline: 11:55pm, October 25th (Sunday), 2020

Note: the sample codes are given in Python format, you can flexibly choose other languages like Java, etc.

1. (20pt) A map data structure can be designed with many different implementations. In this question, please design one table for storing key-values pairs with doubly linked list data structure. The inputs are key-values pairs that can be defined by yourself. You should implement the functions include *insert by index, retrieve the value by key, retrieve value by index, delete value by key, delete value by index, delete item by values, print the entire table*. Since it is a linked list, you should design additional functions such as *insert at the tail and insert at the head*. Name your code to a file as *1_1_doubly_table.py.* Note that you may need to add new functions except the defined functions. (You can either use the given sample code or design by yourself)

2. (20pt) Design a hashtable with fundamental operations by using two types of hashing and collision avoidance. (a) Design two types of hashing, the first is a simple hash that can be completed with O(1), but it may cause collisions; the second hash is a double hash can be completed with O(1). Put the details of your hash function and analysis in codes comments. (b) When your first hash function has collisions, using separate chaining to solve the collision. Name your code file to *1_2_hashtable.py* (or .java, .etc).

   Outputs hints*: Please define your own test cases with the two hashing functions and separate chaining. (1) Your test case should show the collision when you use the first simple hash function and the collision can be handled by separate chaining. (2) Use the same inputs, with double hashing your outputs have no collisions. You can make an option to select one of the hash function or sequentially display these two types of functions.*

   Comments hints*: when you design your own testcase, under the code main function, please give a description of your test case to prove your hash function and hashtable work as your original design*

3. (20pt) Implement a hashtable that consists of two separate tables (arrays). Among this hashtable, you need to design two different hash functions (*h1*, *h2*) for each specific table. Implement it with insert, delete, lookup, etc. When insert a new element, it is always inserted in the first hash table with *h1*. If a collision occurs, the existing element is kicked out and

inserted in the second hash table with *h2*. If in turn cause a collision, the second existing element will be kicked out and inserted in the first hash table with *h1*, and so on. This continues until an empty bucket is found. This case may result to the number of displacements reaches a certain threshold (for instance due to a cycle among the inserted keys), rehashing the table with new hash functions. Further, when the tables are full, you should resize both tables and rehashing all values. *You can use the given sample "1_3_hashtable.py" and test with your test case. (you can also write your own code with same name).*

***Outputs and Comments hints**: when you design your own testcase, under the code main function, please give a description of your test case to prove your hash function and hashtable work as your original design.*

Submission Requirements

You need to strictly follow the instructions listed below:

1) Submit a .zip/.rar file that contains all files.
2) The submission should include your **source code**. Do not submit your binary code.
3) Your code must **be able to compile**; otherwise, you will receive a grade of zero.
4) Your code should not produce anything else other than the required information in the output file.
6) If you code is partially completed, also explain in the begin comment of specific submission what has been completed and the status of the missing parts.
7) Provide **sufficient comments** in your code to help the TA understand your code. This is important for you to get at least partial credit in case your submitted code does not work properly.

Grading criteria:

Details	Points
Submission follows the right formats	10 %
Have a README file shows how to compile and test your submission	5 %
Submitted code has proper comments to show the design details	15 %
Code can be compiled and shows right outputs	70 %

Rubrics for code outputs

	Level 4 (100%)	Level 3 (70%)	Level 2 (40%)	Level 1 (20%)
Code outputs	It is always correct without crashes	It is always correct and eventually it crashes	It is not always correct and eventually it crashes	It is not correct or incomplete