Project 12.2 Economic mobility

In Section 12.5, we designed a `Dictionary` class that assumes that no collisions occur. In this project, you will complete the design of the class so that it properly (and transparently) handles collisions. Then you will use your finished class to write a program that allows one to query data on upward income mobility in the United States.

To deal with collisions, we will use a technique called ***chaining*** in which each slot consists of a *list* of (key, value) pairs instead of a single pair. In this way, we can place as many items in a slot as we need.

Question 12.2.1 *How do the implementations of the* insert, delete, *and* lookup *functions need to change to implement chaining?*

Question 12.2.2 *With your answer to the previous question in mind, what is the worst case time complexity of each of these operations if there are n items in the hash table?*

Part 1: Implement chaining

First, modify the `Dictionary` class from Section 12.5 so that the underlying hash table uses chaining. The constructor should initialize the hash table to be a list of empty lists. Each `__getitem__`, `__setitem__`, and `__delitem__` method will need to be modified. Be sure to raise an appropriate exception when warranted. You should notice that these three methods share some common code that you might want to place in a private method that the three methods can call.

In addition, implement the methods named `_printTable`, `__str__`, `__contains__`, `items`, `keys`, and `values` described in Exercises 12.5.1–12.5.4.

Test your implementation by writing a short program that inserts, deletes, and looks up several entries with integer keys. Also test your class with different values of `self._size`.

Part 2: Hash functions

In the next part of the project, you will implement a searchable database of income mobility data for each of 741 commuting zones that cover the United States. A commuting zone is an area in which the residents tend to commute to the same city for work, and is named for the largest city in the zone. This city name will be the key for your database, so you will need a hash function that maps strings to hash table indices. Exercise 12.5.8 suggested one simple way to do this. Do some independent research to discover at least one additional hash function that is effective for general strings. Implement this new hash function.

Question 12.2.3 *According to your research, why is the hash function you discovered better than the one from Exercise 12.5.8?*

Part 3: A searchable database

On the book website is a tab-separated data file named `mobility_by_cz.txt` that contains information about the expected upward income mobility of children in each of the 741 commuting zones. This file is based on data from The Equality of Opportunity Project (`http://www.equality-of-opportunity.org`), based at Harvard and the University of California, Berkeley. The researchers measured potential income mobility in several ways, but the one we will use is the probability that a child raised by parents in the bottom 20% (or "bottom quintile") of income level will rise to the top 20% (or "top quintile") as an adult. This value is contained in the seventh column of the data file (labeled `"P(Child in Q5 | Parent in Q1), 80-85 Cohort"`).

Write a program that reads this data file and returns a `Dictionary` object in which the keys are names of commuting zones and the values are the probabilities described above. Because some of the commuting zone names are identical, you will need to concatenate the commuting zone name and state abbreviation to make unique keys. For example, there are five commuting zones named "Columbus," but your keys should designate Columbus, GA, Columbus, OH, etc. Once the data is in a `Dictionary` object, your program should repeatedly prompt for the name of a commuting zone and print the associated probability. For example, your output might look like this:

```
Enter the name of a commuting zone to find the chance that the
income of a child raised in that commuting zone will rise to
the top quintile if his or her parents are in the bottom quintile.
Commuting zone names have the form "Columbus, OH".

Commuting zone (or q to quit): Columbus, OH
Percentage is 4.9%.
Commuting zone (or q to quit): Columbus
Commuting zone was not found.
Commuting zone (or q to quit): Los Angeles, CA
Percentage is 9.6%.
Commuting zone (or q to quit): q
```

Part 4: State analyses

Finally, enhance your program so that it produces the following output, organized by state. You should create additional `Dictionary` objects to produce these results. (Do not use any built-in Python dictionary objects!)

1. Print a table like the following of all commuting zone data, alphabetized by state then by commuting zone name. (Hints: (a) create another `Dictionary` object as you read the data file; (b) the `sort` method sorts a list of tuples by the first element in the tuple.)

```
AK
  Anchorage: 13.4%
  Barrow: 10.0%
  Bethel: 5.2%
  Dillingham: 11.8%
  Fairbanks: 16.0%
  Juneau: 12.6%
  Ketchikan: 12.0%
  Kodiak: 14.7%
  Kotzebue: 6.5%
  Nome: 4.7%
  Sitka: 7.1%
  Unalaska: 13.0%
  Valdez: 15.4%
AL
  Atmore: 4.8%
  Auburn: 3.5%
  ⋮
```

2. Print a table, like the following, alphabetized by state, of the average probability for each state. (Hint: use another `Dictionary` object.)

```
State  Percent
-----  -------
 AK     11.0%
 AL      5.4%
 AR      7.2%
 ⋮
```

3. Print a table, formatted like that above, of the states with the five lowest and five highest average probabilities. To do this, it may be helpful to know about the following trick with the built-in `sort` method. When the `sort` method sorts a list of tuples or lists, it compares the first elements in the tuples or lists. For example, if `values = [(0, 2), (2, 1), (1, 0)]`, then `values.sort()` will reorder the list to be `[((0, 2), (1, 0), (2, 1)]`. To have the `sort` method use another element as the key on which to sort, you can define a simple function like this:

```
def getSecond(item):
    return item[1]

values.sort(key = getSecond)
```

When the list named `values` is sorted above, the function named `getSecond` is called for each item in the list and the return value is used as the key to use when sorting the item. For example, suppose `values = [(0, 2), (2, 1), (1, 0)]`. Then the keys used to sort the three items will be `2`, `1`, and `0`, respectively, and the final sorted list will be `[(1, 0), (2, 1), (0, 2)]`.