IT-309 - Assignment 6: Hashing Analysis

Assignment Given: 10/27/2020
Assignment Due: 11/10/2020, 11:59 pm

In this assignment you are to create code to build a hash table using information about students as input. The goal is to learn how to set up a basic hash table, create a hash function, implement a collision handling strategy, and experiment with variations and note the impact they have on the resulting table. The assignment is therefore more like the sorting assignment and less about creating a lot of program code.

The input will consist of a stream of up to 95 transactions and data records. Each transaction is one line with comma-separated data as described in the following table:

Field Description	Type	Size (max)
transaction code	alphabetic	1 character "I" = insert "F" = find "R" = remove/delete
surname	alphabetic	15 characters
given name	alphabetic	15 characters
year of birth	numeric	4 digits
gender	Alphabetic	1 character
major	alphabetic	6 characters

Examples:

I, Rizzo, Mike, 1962, M, PhEd
F, Womack, Mary, 1972, F, Math

Each of the above lines will be read, individually hashed, and placed in one of the hash table index locations.

How to Do This Assignment

The hash table to be built consists of 99 entries, with keys ranging from 0 to 98. The table should be defined as a Python list of fixed capacity. Normally Python lists can be extended indefinitely, but to model the way a real hash table works we give it a limited capacity. The code below will do that by defining a variable called "HTcapacity" that can be reset to try different table sizes, and then using it to set up the table "HT". Note that the table sets up "HTcapacity" number of slots, each with value of 'None'. 'None' indicates that the table index location, or "slot", is unoccupied and can have an entry placed there. Code to define the table is below:

```
HTcapacity = 99                # Set the hash table size
HT = [None]*HTcapacity    # Create the hash table as a Python list of size = HTcapacity, indexed 0 to 98
```

To insert input records into the table you will have to create a hash function to generate the keys using information in the records. Recall that creating a hash function is a two step process:

Step H1:
Select some or all of the input data stream and use one of the methods discussed in class to convert it into a number. One method, for example, used a polynomial with each input character converted to its numeric ASCII value

Step H2:
Since the number coming out of H1 is probably larger than the capacity of the hash table it must be compressed so that it results in a number that corresponds to an index in the table. A common method is modulo division of the H1 number by the size of the table. That would be computing the function composition H2(H1), then calculating index =

H2(H1)%HTcapacity. The input data will be store at HT[index].

For H1, you can choose to use all the characters in the input stream or a subset to produce the number. This assignment only uses a short character stream as input, so it is practical to use all the characters. For larger streams you would want to select a subset that represents (or "abstracts") the particular record or object you would want to hash. A good representation would ensure the input records are well separated in the hash table and minimizes collisions.

For example, if using a person's name one way to abstract that would be to select the first four characters of the last name and first two of the given name for use in step H1. Other information used might be the date of birth, g-number (not included in this assignment), or both. While using the g-number might seem like a good choice, keep in mind that many different g-numbers might hash to the same value and produce a lot of collisions. Experimenting with various combinations of user data can reveal which to use. This is as much art as science.

For H2, you need to select a way of reducing or compressing the large H1 number to the range of table indexes. For a table of size 99, for example, H2 should produce a number in the range 0 to 98. The resulting data, or a pointer to it, is stored in the hash table at that index.

The function composition H2(H1)) generates a record key for the data and inserts the data (or pointer) for that line into the table. For this assignment you can just insert the entire data string into the table at the index value. Alternately, you could create a small object, store the data elements in the object as attributes, and store the object's reference value in the table at the has value's index. That is, in fact, the way many hashing schemes operate.

While the table is larger than the number of input records, collisions are likely and will have to be dealt with. A hash function should provide a good abstract representation of the data and minimize collisions. We discussed two general strategies for handling collisions:

1. Probing (linear or quadratic)
2. Chaining

Probing. When a collision occurs, to probe you would look forward in the table and find the next available slot for the input data. Remember that when a table element is deleted you need to keep the table slot occupied (not 'None'), but marked as 'deleted'. Otherwise the probing strategy would break down. Subsequent data hashing to the 'deleted' index slot can be stored there.

Chaining. For this assignment you could chain by storing each input record as a list of strings. When placing string 'indata1' into an open slot, "open" means the tables slot value = None, you would store it as "[indata1]". If a subsequent input record, indata2, hashes to the same table index, creating a collision, you would append it to the list of strings. In that case the table would have "[indata1, indata2]" stored at that index. Additional collisions could be handled in a similar way. This uses the Python list feature as a chaining mechanism.

In creating and coding the program:

1. The input is a single transaction text file, with up to 95 records as described above. A test file will be made available on Blackboard.
2. The transaction code describes what to do with the record – insert, find, or remove
3. 'Insert' means hash the record and place it in the table.
4. 'Find' means hash the record and search the table – indicate 'found' or 'not found'
5. 'Remove' means hash the record and delete the record from the table
6. Choose your own hash function. Using one presented in class, in the textbook, or elsewhere is permitted. Provide attribution if appropriate.
7. **You must create your own hash function and may not use a built-in or imported hash function. And you may not use the Python dictionary class, which uses hashing.**
8. The hash table should be implemented as a 99 element storage array or list consisting of the generated key (0 to 98) and the associated data.
9. Collisions are to be reconciled using one of the methods discussed in class
10. As each transaction is processed, print a copy of the transaction to output, along with what was accomplished (record

inserted, found, etc.), and note whether a collision was encountered
11. Print a copy of the hash table to output at the end of execution

Experimentation and Analysis Report

You are to submit a short report (the equivalent of a page or two of text) that describes your hash function and how you handle collisions. Also include how many collisions were encountered while processing the transaction stream, how many extra table slots needed to be examined when a collision occurred, and the overall load factor of the table at various stages of the run, including the end. Experiment with various hash functions and table sizes. For example:

1. Does selecting only a subset of input characters change the number of collisions or the number of open table slots?
2. Does calculating H1 differently change the above
3. How does reducing the table capacity to 50, or increase it to 200 change the above?

What and where to submit:

All items are to be submitted in a Jupyter Notebook to Blackboard. Include:

1. The Python code, with a comment block that includes your name, date, appropriate code attributions (if any), and a short description of the code's input, processing, and outputs. The code must execute from the cell so the GTA can evaluate it.
2. A short report as described above in with a Markdown cell (or set of such cells) or a pdf file embedded a notebook cell.
3. Include a short descriptive title in a markdown cell for the various code cells describing what the code does and any explanatory notes you think are needed. Markdown cells containing the report or other explanatory text descriptions (i.e. not "title" cells) should also include a descriptive title cell.

How the assignment will be assessed

The code will be visually and executed. The output will be inspected and evaluated with the written report. Your program must read an input file I will provide that contains several student records, perform the hash function, and store the result in a has table.

Item	Assessment Description	Max Value
Jupyter Notebook	Notebook submitted as requested, including Python executable code and written report	5
Execution Output - Basic	Code executes against the transaction file and displays the hash table at the end of the run. GTA assigns partial credit for less than 100% compliance with the assignment specification, using their own judgment.	15
Written report	A short report embedded in the submitted Jupyter notebook as a Markdown cell (or cells) or attached pdf. The report should describe the hash function and collision handling. Including a few examples of hash input and output is encouraged (i.e. show the character string input and the hash value as output). The report should be brief and readable, and address the particular points requested. GTA may assign partial credit.	15
Experimentation Results	In addition to the basic 99 element table, experiment with various table sizes and hash functions, and report on their effects in terms of number of table slots occupied and number of collisions.	5
Total		**40**