DS2030 Data Structures and Algorithms for Data Science
Lab 7

October 7th, 2025

Lab Instructions

- Create a folder named **"DS2030_<RollNo.>"** (all letters in capital) in **"home"** directory. Eg- **DS2030_142402022**
- Name the script files in the given format **"<your_roll_no>_<Name>_Lab7.py"**
- Make sure the **folder, files, classes, functions and attributes** are named as instructed in the lab sheet.
- We will not be able to evaluate your work if the folder is not properly named or is not located in the home directory.
- Make sure to save your progress before leaving the lab.
- Do not shut down the system after completing the lab.
- You are not allowed to share code with your classmates nor allowed to use code from other sources.

Word Dictionary using Hash Tables with Linear Probing

Problem Statement

You are asked to implement a **Word Dictionary Management System** using a **hash table**. Each word will be stored as a **key** in the hash table, and its **frequency of occurrence** will be stored as the **value**.
To handle collisions, you must use **linear probing**. The system should allow insertion of new words, updating frequency counts for existing words, searching for a word, deleting a word, and displaying all words in the dictionary.

Functionalities to Implement

Your program must support the following operations:

1. **Insert a Word**
   - If the word does not exist, insert it with frequency `1`.
   - If the word already exists, increment its frequency.
   - If the table is **full** (all slots occupied and no deleted slots reusable), your program should print an error message:

     ```
     Hash table is full. Cannot insert: <word>
     ```
   - Do not implement resizing.
2. **Search for a Word**
   - Input: a word.

- Output: frequency of the word, or "Word not found" if it doesnt exist.

3. **Delete a Word**
    - Input: a word.
    - Remove it from the hash table (mark slot as <deleted>).
    - If the word does not exist, print "Word not found".

4. **Display All Words (Raw Table View)**
    - Show the contents of the table **slot by slot**, e.g.:

```
Slot 0: ('data', 3)
Slot 1: <deleted>
Slot 2: None
Slot 3: ('python', 1)
...
```

5. **Display Most Frequent Word(s)**
    - Find the word(s) with the highest frequency.
    - Print them along with their frequency.
    - If multiple words share the same highest frequency, print them all.

Data Structure Requirements

- Implement a hash table using a Python list.
- Use the provided hash function in the starter code.
    - **Use the given formula exactly to implement the hash function**
- Resolve collisions using linear probing.
- Do not use Pythons built-in dict or any external libraries.

Instructions

- Do not add extra functions, attributes, or parameters.
- Strictly follow the starter code given.
- Include appropriate comments to document the code you have implemented.
- **Make sure to save the file before final submission to avoid discrepancies in the file used for evaluation.**

Starter Code

```
# -------- Hash Table Implementation using Linear Probing --------
class HashTable:
    def __init__(self, size):
        # initialize the hash table with given size
        # each slot can either be None, a (key, frequency) tuple, or a deleted marker
        self.size = size
        self.table = [None] * size
        self.deleted = "<deleted>"  # marker for deleted entries
    def hash_function(self, key):
    """
    Hash function design:

    Let n = length of the word
    Let m = size of the hash table

    Formula:
        h(key) = [ (n(n+1)/2) * n ] mod m

    - This means the hash index depends only on the length of the word.
    - Words with the same length will map to the same index.
    - Collisions are expected and will be handled using linear probing.
    """
    pass


    def insert(self, key):
        """
        Insert a word.


        Steps to think about:
        1. Compute the starting index using the hash function.
        2. If slot is empty or marked deleted, place (key, 1) there.
        3. If key already exists, update its frequency (increase by 1).
        4. If the slot is occupied by a different word, move forward one step (linear
probing).
        5. Continue probing until:
           - You find an empty/deleted slot, OR
           - You loop all the way back (table full).
        6. If the table is full, your insert function should print an error message
and skip inserting the word. Do not implement resizing.
        """
        pass

    def search(self, key):
        """
        Search for a word.

        Steps to think about:
        1. Compute index using the hash function.
        2. If slot is empty, the word does not exist.
        3. If the slot contains the key, print its frequency.
        4. If not, move forward (linear probing) and keep checking.
        5. Stop when:
           - The key is found, OR
           - You loop back and d o n  t  find it.
        """
        pass

    def delete(self, key):
        """
        Delete a word.

        Important clarification:
          - This operation removes the word completely from the table,
      regardless of its frequency count.
          - Example: if "data" has frequency 3 and you delete "data",
      the entire entry is removed (not reduced to 2).
          - If the word is later reinserted, its frequency starts again from 1.

        Steps to think about:
        1. Compute index using the hash function.
        2. Probe forward until you either:
           - Find the key, OR
```

```
            - Reach an empty slot (not found).
        3. If found, replace the entry with self.deleted (a tombstone marker).
        4. If not found, print "Word not found".
        """
        pass

    def display(self):
        """
        Display the raw contents of the hash table slot by slot.

        Example:
        Slot 0: ('data', 3)
        Slot 1: ('ml', 1)
        Slot 2: None
        Slot 3: <deleted>
        ...
        """


    def most_frequent(self):
        """
        Display the most frequent word(s).

        Steps to think about:
        1. Traverse the table and collect all active (key, frequency) pairs.
        2. Find the maximum frequency among them.
        3. Collect all words that have this frequency.
        4. Print them along with their frequency.
        5. If no active entries, print "(empty)".
        """
        pass
```

Test Function

```
def test_word_dictionary():
    # Create a hash table of size 15 (enough space for insertions + extra tests)
    ht = HashTable(15)

    print("========== Test: Insert ==========")
    ht.insert("data")
    ht.insert("science")
    ht.insert("data")       # duplicate
    ht.insert("python")
    ht.insert("data")       # duplicate again
    ht.insert("ml")
    ht.insert("ai")
    ht.insert("dl")
    ht.insert("AI")         # different from "ai" (case-sensitivity matters)
    ht.insert("")           # empty string (consider it a valid word)
    ht.insert("a")          # single char
    ht.insert("daTa")       # mixed case

    print("\nRaw hash table contents after insertions:")
    ht.display()

    print("\n========== Test: Search ==========")
    print("Searching for 'data':")
    ht.search("data")       # should print frequency 3
    print("Searching for 'machine':")
    ht.search("machine")    # should print "Word not found"
    print("Searching for empty string:")
    ht.search("")           # should print frequency 1

    print("\n========== Test: Most Frequent ==========")
    ht.most_frequent()      # should show "data     3"

    print("\n========== Test: Delete ==========")
    print("Deleting 'science'...")
    ht.delete("science")
    print("Table after deleting 'science':")
    ht.display()

    print("Deleting non-existing word 'java'...")
    ht.delete("java")       # should print "Word not found"

    print("Deleting 'data' (most frequent word)...")
    ht.delete("data")
    print("Table after deleting 'data':")
    ht.display()

    print("\n========== Test: Reinsertion ==========")  #re-insertion resets the
   frequency to 1
    print("Re-inserting 'science' after deletion:")
    ht.insert("science")
    print("Re-inserting 'data' after deletion:")
    ht.insert("data")
    ht.display()

    print("\n========== Test: Table Full ==========")
    # Fill remaining slots with dummy words to force table towards full
    extra_words = ["alpha", "beta", "gamma", "delta", "epsilon",
                   "zeta", "eta", "theta", "iota", "kappa"]
    for w in extra_words:
        ht.insert(w)

    print("Raw table after adding more words (close to full capacity):")
    ht.display()
    print("\n========== Test: Tie in Most Frequent ==========")
    # Increase frequencies of some words to create a tie
    ht.insert("alpha")
    ht.insert("beta")
    ht.insert("alpha")   # alpha freq = 3
    ht.insert("beta")    # beta freq = 3
    print("Raw table contents (alpha and beta now tied with highest frequency):")
    ht.display()
    print("Most frequent words (should show both alpha and beta):")
    ht.most_frequent()

    print("\n========== Test Completed ==========")


if __name__ == "__main__":
    test_word_dictionary()
```

Sample Output

```
Raw hash table contents after insertions:
Slot 0: ('', 1)
Slot 1: ('science', 1)
Slot 2: ('a', 1)
Slot 3: None
Slot 4: None
Slot 5: None
Slot 6: ('python', 1)
Slot 7: ('ml', 1)
Slot 8: ('ai', 1)
Slot 9: ('dl', 1)
Slot 10: ('data', 3)
Slot 11: ('AI', 1)
Slot 12: ('daTa', 1)
Slot 13: None
Slot 14: None

========== Test: Search ==========
Searching for 'data':
data     3
Searching for 'machine':
Word not found
Searching for empty string:
     1

========== Test: Most Frequent ==========
Most frequent word(s):
data     3

========== Test: Delete ==========
Deleting 'science'...
Deleted: science
Table after deleting 'science':
Slot 0: ('', 1)
Slot 1: <deleted>
Slot 2: ('a', 1)
Slot 3: None
Slot 4: None
Slot 5: None
Slot 6: ('python', 1)
Slot 7: ('ml', 1)
Slot 8: ('ai', 1)
Slot 9: ('dl', 1)
Slot 10: ('data', 3)
Slot 11: ('AI', 1)
Slot 12: ('daTa', 1)
Slot 13: None
Slot 14: None
Deleting non-existing word 'java'...
Word not found
Deleting 'data' (most frequent word)...
Deleted: data
Table after deleting 'data':
Slot 0: ('', 1)
Slot 1: <deleted>
Slot 2: ('a', 1)
Slot 3: None
Slot 4: None
Slot 5: None
Slot 6: ('python', 1)
Slot 7: ('ml', 1)
Slot 8: ('ai', 1)
Slot 9: ('dl', 1)
Slot 10: <deleted>
Slot 11: ('AI', 1)
Slot 12: ('daTa', 1)
Slot 13: None
Slot 14: None

========== Test: Reinsertion ==========
Re-inserting 'science' after deletion:
Re-inserting 'data' after deletion:
Slot 0: ('', 1)
Slot 1: ('science', 1)
Slot 2: ('a', 1)
Slot 3: None
Slot 4: None
Slot 5: None
Slot 6: ('python', 1)
Slot 7: ('ml', 1)
Slot 8: ('ai', 1)
Slot 9: ('dl', 1)
Slot 10: ('data', 1)
Slot 11: ('AI', 1)
Slot 12: ('daTa', 1)
Slot 13: None
Slot 14: None

========== Test: Table Full ==========
Hash table is full. Cannot insert: zeta
Hash table is full. Cannot insert: eta
Hash table is full. Cannot insert: theta
Hash table is full. Cannot insert: iota
Hash table is full. Cannot insert: kappa
Raw table after adding more words (close to full capacity):
Slot 0: ('', 1)
Slot 1: ('science', 1)
Slot 2: ('a', 1)
Slot 3: ('alpha', 1)
Slot 4: ('gamma', 1)
Slot 5: ('delta', 1)
Slot 6: ('python', 1)
Slot 7: ('ml', 1)
Slot 8: ('ai', 1)
Slot 9: ('dl', 1)
Slot 10: ('data', 1)
Slot 11: ('AI', 1)
Slot 12: ('daTa', 1)
Slot 13: ('beta', 1)
Slot 14: ('epsilon', 1)

========== Test: Tie in Most Frequent ==========
Raw table contents (alpha and beta now tied with highest frequency):
Slot 0: ('', 1)
Slot 1: ('science', 1)
Slot 2: ('a', 1)
Slot 3: ('alpha', 3)
Slot 4: ('gamma', 1)
Slot 5: ('delta', 1)
Slot 6: ('python', 1)
Slot 7: ('ml', 1)
Slot 8: ('ai', 1)
Slot 9: ('dl', 1)
Slot 10: ('data', 1)
Slot 11: ('AI', 1)
Slot 12: ('daTa', 1)
Slot 13: ('beta', 3)
Slot 14: ('epsilon', 1)
Most frequent words (should show both alpha and beta):
Most frequent word(s):
alpha     3
beta     3

========== Test Completed ==========
```