Solution Review: Word Formation Using a Hash Table

This review provides a detailed analysis of the solution to the Word Formation Using a Hash Table Challenge.

We'll cover the following

Solution: Iterative Word Matching

main.py

HashTable.py

HashEntry.py

```
from HashTable import HashTable


def is_formation_possible(lst, word):

    if len(word) < 2 or len(lst) < 2:
        return False

    hash_table = HashTable()
    for elem in lst:
        hash_table.insert(elem, True)

    for i in range(1, len(word)):
```

```
14          # Slice the word into two strings in each iteration
15          first = word[0:i]
16          second = word[i:len(word)]
17          check1 = False
18          check2 = False
19
20          if hash_table.search(first) is not None:
21              check1 = True
22          if hash_table.search(second) is not None:
23              check2 = True
24
25          # Return True If both substrings are present in the hash tabl
26          if check1 and check2:
27              return True
28
```

This is as efficient as the implementation as the trie implementation. We insert all the dictionary words into a hash table.

Just like before, a `for` loop begins and slices the `word` into two substrings in each iteration. Whenever both substrings are found in the hash table, the function returns `True`.

> **Note:** The solution only works for two words and not more.

Time Complexity

We perform the insert operation **m** times for a list of size **m**. After that, we linearly traverse the `word` of size **n** once. Furthermore, we slice strings of size **n** in each iteration. Hence the total time complexity is $O(m + n^2)$.

companies apply to you instead of you applying to them.

how

Back

Challenge 7: Word Formation Using a ...

Next

Challenge 8: Find Two Numbers that ...

Mark as Completed

Report an Issue