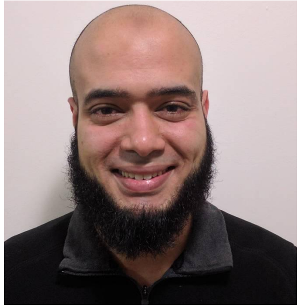
Data Structures

Hashing Homework 2

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Problem #1: Number of Distinct Substrings

- Given a string, count how many unique substrings it contains
  - int count_unique_substrings(const string &str)
- For string aaab, the substrings are:
  - a, aa, aaa, aaab, a, aa, aab, a, ab, b
  - Only 7 distinct substrings
- Input ⇒ Output
  - aaaaa ⇒ 5
  - aaaba ⇒ 11
  - abcdef ⇒ 21
- **def** count_unique_substrings(str):
- Find a hash-based solution. Use built-in. What is your best complexity?
- Do you think we can use another data-structure for a more efficient solution?

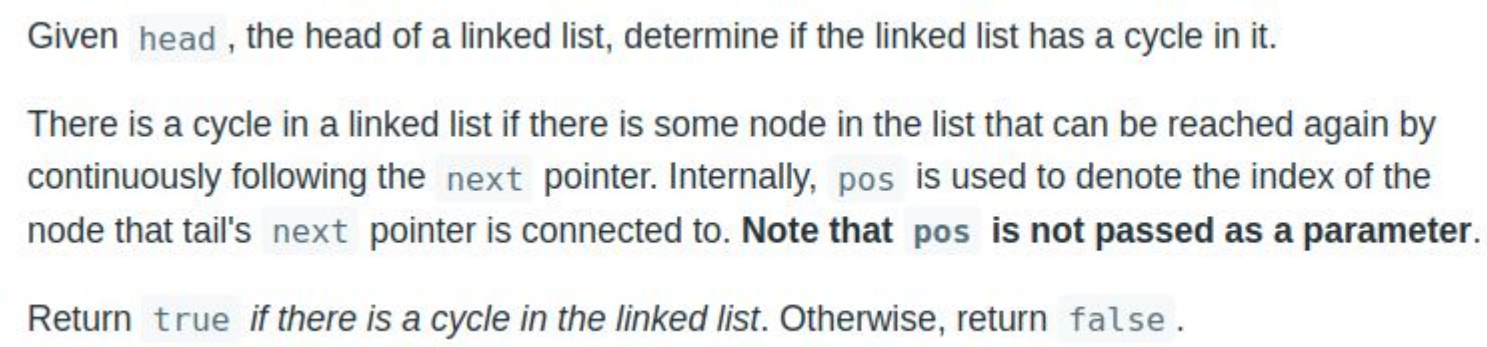
Problem #2: Common substrings

- **def** count_substrings_match(str1, str2):
    - Assume we have set S1 for the unique substrings in str1
    - Assume we have set S2 for the unique substrings in str2
    - Return how many substrings S1 and S2 do have in common
    - Use built-in hash-table
- Input ⇒ Output
    - aaab, aa ⇒ 2 [a, aa]
    - aaab, ab ⇒ 3 [a, b, aa]
    - aaaaa, xy = 0
    - aaaaa, aaaaa ⇒ 5

Problem #3: Unique Anagrams

- An anagram of a string of **lower** letters is another string that contains the **same** characters, although the **order** of characters is different
    - aab and baa are anagrams. bbcde and edcbb are anagrams
    - bbcde and edcb are NOT anagrams (missing b)
- Given a string, find the number of **unique** anagrams among its substrings
    - abba has 10 substrings: a, ab, abb, abba, b, bb, bba, b, ba, a
    - Only 6 are unique anagrams. **Duplicate groups** are (ab, ba), (a, a), (b, b), (abb, bba)
    - Find O(L^3logL) or better O(L^3)
- Input ⇒ Output
    - aaaaa ⇒ 5, abcba ⇒ 9, aabade ⇒ 17
- count_anagram_substrings(str)

Problem #4: LeetCode 141 - Linked List Cycle

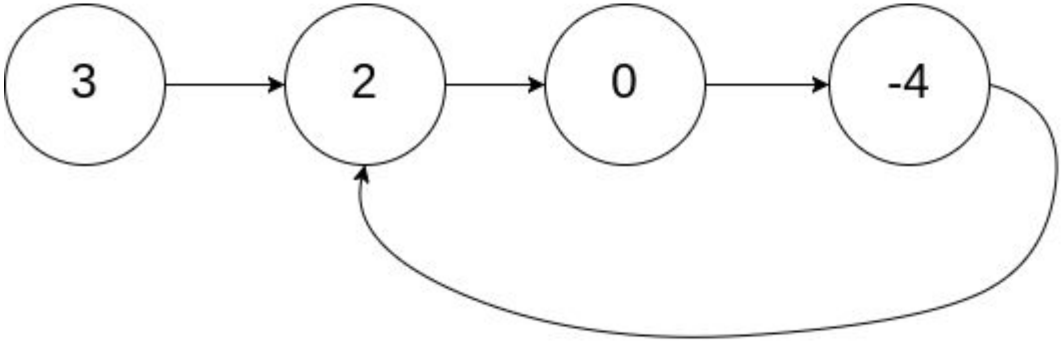


Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter**.

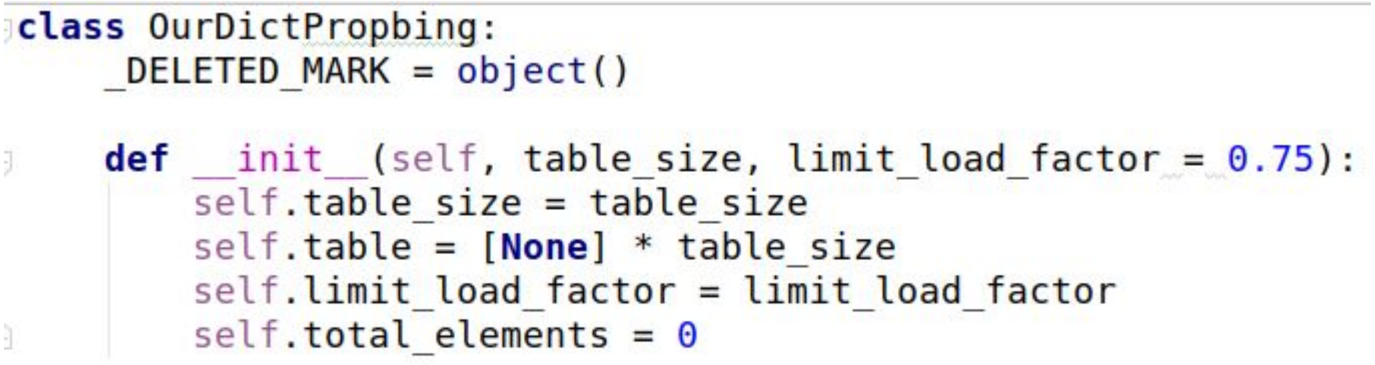
Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

- Develop a simple hash-based solution
  - def hasCycle(self, head)



Problem #5: Quadratic Probing

- Change the lecture code to quadratic probing
  - What are the implication of that on the code compare to the linear probing? Think
  - Also, can we stop earlier than jumping table-size steps?
- Add the rehashing function



```
class OurDictPropbing:
    _DELETED_MARK = object()

    def __init__(self, table_size, limit_load_factor = 0.75):
        self.table_size = table_size
        self.table = [None] * table_size
        self.limit_load_factor = limit_load_factor
        self.total_elements = 0
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."