Jahongir Hayitov CS-01. 08.02.2023

Problem 1. Consider the following algorithm (see pseudocode conventions in [Cormen, Section 2.1]). The inputs to this algorithm are a map M and a key k. Additionally, assume the following:
(a) The map M uses the same data type both for keys and for values.
(b) The map M is not empty and contains n distinct keys.
(c) The map M is represented as a hashtable with load factor α.
(d) The map M resolves collisions via chaining [Cormen, Section 11.2].
(e) The set of keys stored in M is equal to the set of values stored in M.

```
/* M is a map with a load factor α and size n,
* k is a key that may or may not be in M */
secret(M, k):
    counter := 0
    last_key := k
    repeat
        last_key := M.get(last_key)
        counter := counter + 1
    until last_key = k
    return counter
```

Compute the average case time complexity of secret. The answer must use Θ-notation. For the average case analysis, use independent uniform hashing. For full grade, you may assume the worst case for the arrangement of values stored in M.
For extra credit, assume the average case also for the arrangement of values stored in M.
Optionally, you may provide details for the computation of the running time T(n). Proof for the asymptotic bound is not required for this exercise

ANSWER:
In a hash table in which collisions are resolved by chaining, each unsuccessful and successful search takes Θ(1+α) time on average, under the assumption of independent uniform hashing.

Average time complexity:
Since "The set of keys stored in M is equal to the set of values stored in M", if k exists in our set of keys then our output is valid, unless there is no list element that contains the same key and value which causes infinite loop (worst case). During the search in repeat block, get method can call to different slots of hash-table, because we may have a key contains different value. After that we are assigning this value as a key. So the time complexity depends on the all linked list elements inside the hash-table.
In average case, this would be Θ(n/2*(1+a)) = **Θ(n*(1+a))**

Problem 2. Consider a hash table with 13 slots and the hash function $h(k) = (k^2 + k + 3) \bmod 13$. Show the state of the hash table after inserting the keys (in this order) 5, 28, 19, 15, 20, 33, 12, 17, 10, 13, 3, 33 with collisions resolved by linear probing [Cormen, Section 11.4].

ANSWER:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key	**10**		**3**	**12**	**13**		**19**	**5**	**20**	**28**	**15**	**33**	**17**

Problem 3. In your own words, explain how it is possible to implement deletion of a key-value pair from a hashtable with O(1) worst case time complexity if collision resolution is implemented using chaining?

ANSWER:

In a hash table, the key is first hashed to determine the slot in the table where the key-value pair should be stored and it takes O(1) time to find the slot with hash function. Once the slot is determined, the linked list in that slot is searched to locate the key-value pair.

If we use a singly linked list, the search time would be linear in the worst case, this would result in a time complexity of O(n) for deleting a key-value pair.

But, if we use a doubly linked list, it is possible to delete a node in constant time by updating the previous and next pointers of the nodes before and after the node to be deleted.

So, to delete a key-value pair from the hash table, we need to hash the key to determine the slot in the table, then search the linked list in that slot for the key-value pair, and finally delete the node from the linked list. The time complexity of deletion from a hash table with chaining is O(1) in the worst case, because deletion from a doubly linked list takes constant time,