Module8-HW

July 11, 2024

Problem1-LeetcodeQ 217-Contains Duplicate-Easy

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

- Example 1:
  - Input: nums = [1,2,3,1]
  - Output: true
- Example 2:
  - Input: nums = [1,2,3,4]
  - Output: false
- Example 3:
  - Input: nums = [1,1,1,3,3,4,3,2,4,2]
  - Output: true

1 Q217 pseudocode

[ ]:
```
Initialize an empty dictionary numset.

Iterate through each element i in the list nums.

For each element i:
     If i is not in numset:
       Add i to numset with a value of 1.
     Else:
       Return True (indicating a duplicate is found).

If the loop completes without finding any duplicates, return False.
```

2 Q217 code

[1]:
```
from typing import List

class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        numset = {}
        for i in nums:
```

```
            if i not in numset:
                numset[i] = 1
            else:
                return True
        return False

# Test case
nums = [1, 1, 1, 3, 3, 4, 3, 2, 4, 2]
solution = Solution()
output = solution.containsDuplicate(nums)
print(f"Input: nums = {nums}")
print(f"Output: {output}")
```

```
Input: nums = [1, 1, 1, 3, 3, 4, 3, 2, 4, 2]
Output: True
```

Problem2-Leetcode Q 1-Two Sum-Easy

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

- Example 1:
  - Input: nums = [2,7,11,15], target = 9
  - Output: [0,1]
  - Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
- Example 2:
  - Input: nums = [3,2,4], target = 6
  - Output: [1,2]
- Example 3:
  - Input: nums = [3,3], target = 6
  - Output: [0,1]

3 Q1 pseudocode

[ ]:
```
Initialize an empty dictionary idx.
Iterate through each element x in the list nums with its index j.
For each element x:
    If target - x is in idx: Return the list containing idx[target - x] and j␣
 ↪(the indices of the two numbers that add up to target).
    Else: Add x to idx with the value j (the index of x).
If no such pair is found, return an empty list (though in this problem, a␣
 ↪solution is always assumed to exist).
```

4 Q1 code

[5]:
```python
from typing import List

class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        idx = {}
        for j, x in enumerate(nums):
            if target - x in idx:
                return [idx[target - x], j]
            idx[x] = j

# Test case
nums = [3, 2, 4]
target = 6
solution = Solution()
output = solution.twoSum(nums, target)
print(f"Input: nums = {nums}, target = {target}")
print(f"Output: {output}")
```

```
Input: nums = [3, 2, 4], target = 6
Output: [1, 2]
```

Problem3-LeetcodeQ 146-LRU Cache-Medium

Design a data structure that follows the constraints of a Least Recently Used (LRU) cache.

Implement the LRUCache class:

LRUCache(int capacity) Initialize the LRU cache with positive size capacity.
int get(int key) Return the value of the key if the key exists, otherwise return -1.
void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, evict the least recently used key.

The functions get and put must each run in O(1) average time complexity.

- Example 1:
  - Input [“LRUCache”, “put”, “put”, “get”, “put”, “get”, “put”, “get”, “get”, “get”] [[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
  - Output [null, null, null, 1, null, -1, null, -1, 3, 4]
  - Explanation
    - LRUCache lRUCache = new LRUCache(2);
    - lRUCache.put(1, 1); // cache is {1=1}
    - lRUCache.put(2, 2); // cache is {1=1, 2=2}
    - lRUCache.get(1); // return 1
    - lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
    - lRUCache.get(2); // returns -1 (not found)
    - lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}

* lRUCache.get(1); // return -1 (not found)
* lRUCache.get(3); // return 3
* lRUCache.get(4); // return 4

5 Q146 Pseudocode

[ ]:
```
Initialize ListNode Class
    Initialize key and value attributes.
    Initialize prev and next pointers to None.

LRUCache Class
Initialize the cache with a given capacity.
    Create an empty dictionary hashmap to store key-node pairs.
    Create two dummy nodes head and tail.
    Link head.next to tail and tail.prev to head.

move_node_to_tail Method
    Get the node corresponding to the key.
    Update the previous node's next to point to the current node's next.
    Update the next node's prev to point to the current node's prev.
    Move the node to the position before tail.

get Method
    Check if the key exists in hashmap.
    If it does, move the node to the tail.
    Return the value of the node if found, otherwise return -1.

put Method
    If the key exists in hashmap, update its value and move the node to the␣
 ↪tail.
    If the key doesn't exist:
    If the cache is at full capacity, remove the least recently used item (node␣
 ↪after head).
    Add a new node with the key and value before the tail.
```

6 Q146 code

[2]:
```
class ListNode:
    def __init__(self, key=None, value=None):
        self.key = key
        self.value = value
        self.prev = None
        self.next = None

class LRUCache:
    def __init__(self, capacity: int):
```

```
        self.capacity = capacity
        self.hashmap = {}
        self.head = ListNode()
        self.tail = ListNode()
        self.head.next = self.tail
        self.tail.prev = self.head

    def move_node_to_tail(self, key):
        node = self.hashmap[key]
        node.prev.next = node.next
        node.next.prev = node.prev
        node.prev = self.tail.prev
        node.next = self.tail
        self.tail.prev.next = node
        self.tail.prev = node

    def get(self, key: int) -> int:
        if key in self.hashmap:
            self.move_node_to_tail(key)
            return self.hashmap[key].value
        return -1

    def put(self, key: int, value: int) -> None:
        if key in self.hashmap:
            self.hashmap[key].value = value
            self.move_node_to_tail(key)
        else:
            if len(self.hashmap) == self.capacity:
                lru_node = self.head.next
                self.hashmap.pop(lru_node.key)
                self.head.next = lru_node.next
                lru_node.next.prev = self.head

            new_node = ListNode(key, value)
            self.hashmap[key] = new_node
            new_node.prev = self.tail.prev
            new_node.next = self.tail
            self.tail.prev.next = new_node
            self.tail.prev = new_node

# Test case
commands = ["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get",␣
 ↪"get"]
arguments = [[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
expected_output = [None, None, None, 1, None, -1, None, -1, 3, 4]

cache = None
```

```
results = []

for command, argument in zip(commands, arguments):
    if command == "LRUCache":
        cache = LRUCache(*argument)
        results.append(None)
    elif command == "put":
        cache.put(*argument)
        results.append(None)
    elif command == "get":
        result = cache.get(*argument)
        results.append(result)

print(f"Input: {commands}")
print(f"Arguments: {arguments}")
print(f"Output: {results}")
print(f"Expected Output: {expected_output}")
print(f"Test Passes: {results == expected_output}")
```

```
Input: ['LRUCache', 'put', 'put', 'get', 'put', 'get', 'put', 'get', 'get',
'get']
Arguments: [[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output: [None, None, None, 1, None, -1, None, -1, 3, 4]
Expected Output: [None, None, None, 1, None, -1, None, -1, 3, 4]
Test Passes: True
```

Problem4-Leetcode Q 705-Design HashSet-Easy

Design a HashSet without using any built-in hash table libraries.

Implement MyHashSet class:

void add(key) Inserts the value key into the HashSet.
bool contains(key) Returns whether the value key exists in the HashSet or not.
void remove(key) Removes the value key in the HashSet. If key does not exist in the HashSet, do nothing.

- Example 1:
  - Input[“MyHashSet”, “add”, “add”, “contains”, “contains”, “add”, “contains”, “remove”, “contains”] [[], [1], [2], [1], [3], [2], [2], [2], [2]]
  - Output [null, null, null, true, false, null, true, null, false]
  - Explanation
    - MyHashSet myHashSet = new MyHashSet();
    - myHashSet.add(1); // set = [1]
    - myHashSet.add(2); // set = [1, 2]
    - myHashSet.contains(1); // return True
    - myHashSet.contains(3); // return False, (not found)

* myHashSet.add(2); // set = [1, 2]
* myHashSet.contains(2); // return True
* myHashSet.remove(2); // set = [1]
* myHashSet.contains(2); // return False, (already removed)

7 Q 705 Pseudocode

[4]:
```
MyHashSet Class
Initialize the HashSet with a given number of buckets (default is 1009):

If buckets is 1009, create a nested MyHashSet with 37 buckets for each of the␣
 ↪1009 buckets.
Otherwise, initialize each bucket as an empty list.

Add a key to the HashSet:
Call the append method with the given key.
Remove a key from the HashSet:

Compute the hash of the key using key % buckets.
If the key exists in the bucket at the computed hash index, remove it from the␣
 ↪bucket.
Check if a key is in the HashSet:

Return True if the key is in the HashSet, otherwise return False.
Append a key to the bucket (helper method):

Compute the hash of the key using key % buckets.
If the key is not in the bucket at the computed hash index, add it to the␣
 ↪bucket.
Check if a key is in the HashSet (helper method):

Compute the hash of the key using key % buckets.
Return True if the key is in the bucket at the computed hash index, otherwise␣
 ↪return False.
```

```
  Cell In[4], line 1
    MyHashSet Class
              ^
SyntaxError: invalid syntax
```

8 Q 705 Code

[3]:
```
class MyHashSet:
    def __init__(self, buckets: int = 1009) -> None:
        self.buckets = buckets
        if self.buckets == 1009:
            self.table = [MyHashSet(37) for _ in range(self.buckets)]
        else:
            self.table = [[] for _ in range(self.buckets)]

    def add(self, key: int) -> None:
        self.append(key)

    def remove(self, key: int) -> None:
        hash_key = key % self.buckets
        if key in self.table[hash_key]:
            self.table[hash_key].remove(key)

    def contains(self, key: int) -> bool:
        return key in self

    def append(self, key: int) -> None:
        hash_key = key % self.buckets
        if key not in self.table[hash_key]:
            self.table[hash_key].append(key)

    def __contains__(self, key: int) -> bool:
        hash_key = key % self.buckets
        return key in self.table[hash_key]

# Test case
commands = ["MyHashSet", "add", "add", "contains", "contains", "add",␣
 ↪"contains", "remove", "contains"]
arguments = [[], [1], [2], [1], [3], [2], [2], [2], [2]]
expected_output = [None, None, None, True, False, None, True, None, False]

hash_set = None
results = []

for command, argument in zip(commands, arguments):
    if command == "MyHashSet":
        hash_set = MyHashSet(*argument)
        results.append(None)
    elif command == "add":
        hash_set.add(*argument)
        results.append(None)
    elif command == "remove":
```

```
        hash_set.remove(*argument)
        results.append(None)
    elif command == "contains":
        result = hash_set.contains(*argument)
        results.append(result)

print(f"Input: {commands}")
print(f"Arguments: {arguments}")
print(f"Output: {results}")
print(f"Expected Output: {expected_output}")
print(f"Test Passes: {results == expected_output}")
```

```
Input: ['MyHashSet', 'add', 'add', 'contains', 'contains', 'add', 'contains',
'remove', 'contains']
Arguments: [[], [1], [2], [1], [3], [2], [2], [2], [2]]
Output: [None, None, None, True, False, None, True, None, False]
Expected Output: [None, None, None, True, False, None, True, None, False]
Test Passes: True
```

[ ]:

Problem5-Leetcode Q 706- Design HashMap-Easy

Design a HashMap without using any built-in hash table libraries.

Implement the MyHashMap class:

MyHashMap() initializes the object with an empty map.
void put(int key, int value) inserts a (key, value) pair into the HashMap.
If the key already exists in the map, update the corresponding value. int get(int key) returns the value to which the specified key is mapped,
or -1 if this map contains no mapping for the key.
void remove(key) removes the key and its corresponding value
if the map contains the mapping for the key.

- Example 1:
  - Input [“MyHashMap”, “put”, “put”, “get”, “get”, “put”, “get”, “remove”, “get”]
    [[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
  - Output [null, null, null, 1, -1, null, 1, null, -1]
  - Explanation
    - MyHashMap myHashMap = new MyHashMap();
    - myHashMap.put(1, 1); // The map is now [[1,1]]
    - myHashMap.put(2, 2); // The map is now [[1,1], [2,2]]
    - myHashMap.get(1); // return 1, The map is now [[1,1], [2,2]]
    - myHashMap.get(3); // return -1 (i.e., not found), The map is now [[1,1], [2,2]]
    - myHashMap.put(2, 1); // The map is now [[1,1], [2,1]] (i.e., update the existing value)
    - myHashMap.get(2); // return 1, The map is now [[1,1], [2,1]]

* myHashMap.remove(2); // remove the mapping for 2, The map is now [[1,1]]
* myHashMap.get(2); // return -1 (i.e., not found), The map is now [[1,1]]

9 Q 706 Pseudocode

[ ]:
```
Initialize the HashMap:

Create a list of empty lists called hash with 20011 buckets.
Put a key-value pair into the HashMap:

Compute the hash index t using key % 20011.
Iterate through the bucket at index t to find if the key already exists.
If the key exists, update its value.
If the key does not exist, append the key-value pair to the bucket.
Get the value associated with a key from the HashMap:

Compute the hash index t using key % 20011.
Iterate through the bucket at index t to find the key.
If the key is found, return its value.
If the key is not found, return -1.
Remove a key-value pair from the HashMap:

Compute the hash index t using key % 20011.
Iterate through the bucket at index t to find the key.
If the key is found, remove the key-value pair from the bucket.
```

10 Q 706 Code

[7]:
```
class MyHashMap:
    def __init__(self):
        self.hash = [[] for _ in range(20011)]

    def put(self, key: int, value: int) -> None:
        t = key % 20011
        for item in self.hash[t]:
            if item[0] == key:
                item[1] = value
                return
        self.hash[t].append([key, value])

    def get(self, key: int) -> int:
        t = key % 20011
        for item in self.hash[t]:
            if item[0] == key:
                return item[1]
        return -1
```

```
    def remove(self, key: int) -> None:
        t = key % 20011
        delete = []
        for item in self.hash[t]:
            if item[0] == key:
                delete = item
        if delete:
            self.hash[t].remove(delete)

# Test case
commands = ["MyHashMap", "put", "put", "get", "get", "put", "get", "remove",␣
 ↪"get"]
arguments = [[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
expected_output = [None, None, None, 1, -1, None, 1, None, -1]

hash_map = None
results = []

for command, argument in zip(commands, arguments):
    if command == "MyHashMap":
        hash_map = MyHashMap()
        results.append(None)
    elif command == "put":
        hash_map.put(*argument)
        results.append(None)
    elif command == "get":
        result = hash_map.get(*argument)
        results.append(result)
    elif command == "remove":
        hash_map.remove(*argument)
        results.append(None)

print(f"Input: {commands}")
print(f"Arguments: {arguments}")
print(f"Output: {results}")
print(f"Expected Output: {expected_output}")
print(f"Test Passes: {results == expected_output}")
```

```
Input: ['MyHashMap', 'put', 'put', 'get', 'get', 'put', 'get', 'remove', 'get']
Arguments: [[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
Output: [None, None, None, 1, -1, None, 1, None, -1]
Expected Output: [None, None, None, 1, -1, None, 1, None, -1]
Test Passes: True
```

[ ]: