**Problem Statement: Unique User Identifier Tracker**

In a system that tracks user activity, each user is assigned a unique identifier (UID). The system must efficiently manage and perform operations on these UIDs using a hashset. The operations include adding new UIDs, removing inactive UIDs, and checking if a UID is active in the system.

**Scenario:** Consider a scenario where an online platform needs to manage a large set of active user identifiers. The system should efficiently add new UIDs, remove inactive UIDs, and quickly determine if a UID is active in the set.

**Input Format:**

1. The first line contains an integer n denoting the number of operations.
2. The next n lines contain operations in the format:
   * addUID uid
   * removeUID uid
   * checkUID uid

**Output Format:** For each checkUID operation, print "Active" if the UID exists in the set. Otherwise, print "Inactive".

**Constraints:**

* The number of operations n is in the range [1, 10000].
* uid is a string consisting of lowercase English letters, digits, and hyphens, and its length is in the range [1, 100].
* Each UID is unique in the context of the operations.

**Sample Input:**

6

addUID user123

addUID user456

checkUID user123

removeUID user123

checkUID user123

checkUID user456

**Sample Output:**

Active

Inactive

Active

**Explanation:**

* The first operation adds the UID user123.
* The second operation adds the UID user456.
* The third operation checks if the UID user123 is active, which it is.
* The fourth operation removes the UID user123.
* The fifth operation checks if the UID user123 is active, which it isn't after removal.
* The sixth operation checks if the UID user456 is active, which it is.

**Solution Template:**

python

class UIDTracker:

def \_\_init\_\_(self):

self.uid\_set = set()

def addUID(self, uid: str) -> None:

self.uid\_set.add(uid)

def removeUID(self, uid: str) -> None:

self.uid\_set.discard(uid)

def checkUID(self, uid: str) -> str:

if uid in self.uid\_set:

return "Active"

else:

return "Inactive"

def main():

import sys

input = sys.stdin.read

data = input().strip().split('\n')

n = int(data[0])

tracker = UIDTracker()

results = []

for i in range(1, n + 1):

command = data[i].split()

operation = command[0]

uid = command[1]

if operation == "addUID":

tracker.addUID(uid)

elif operation == "removeUID":

tracker.removeUID(uid)

elif operation == "checkUID":

result = tracker.checkUID(uid)

results.append(result)

for result in results:

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Extra Test Cases:**

**Test Case 1:**

7

addUID john-doe

addUID jane-doe

checkUID john-doe

checkUID jane-doe

removeUID john-doe

checkUID john-doe

checkUID jane-doe

**Output:**

Active

Active

Inactive

Active

**Test Case 2:**

sql

5

addUID user-001

addUID user-002

removeUID user-001

checkUID user-001

checkUID user-002

**Output:**

Inactive

Active

**Test Case 3:**

r

8

addUID alpha-01

addUID beta-02

addUID gamma-03

removeUID beta-02

checkUID alpha-01

checkUID beta-02

checkUID gamma-03

removeUID gamma-03

**Output:**

Active

Inactive

Active

**Test Case 4:**

lua

6

addUID admin-123

addUID guest-456

addUID mod-789

checkUID guest-456

removeUID guest-456

checkUID guest-456

**Output:**

Active

Inactive

**Test Case 5:**

9

addUID u1

addUID u2

addUID u3

checkUID u1

checkUID u4

removeUID u2

checkUID u2

removeUID u1

checkUID u1

**Output:**

Active

Inactive

Inactive

Inactive

This problem tests the implementation of a UID tracking system using hashsets to efficiently manage UID addition, removal, and search operations, covering various scenarios and constraints to ensure the system's robustness and performance