Problem 14. The following iterative sequence is defined for the set of positive integers:

$$n \to n/2 \text{ (n is even)}$$
$$n \to 3n + 1 \text{ (n is odd)}$$

Using the rule above and starting with 13, we generate the following sequence:

$$13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1. Which starting number, under one million, produces the longest chain?
NOTE: Once the chain starts the terms are allowed to go above one million.

Knowledge Required How to cache up a simple recursive function.

Solution Outline We start off by implementing a recursive function `collatz_chain_len` which takes in a number and returns the length of the chain (i.e the number of steps taken to reach 1 from the number). Now the brute force way would be to search for all numbers less than one million and return the number which would produce the longest chain. But this solution is inefficient as it recomputes the length of longest chain for smaller sub-problems of the bigger problems. For example in the given question, we found out `collatz_chain_len(5)` is 6. Now when finding `collatz_chain_len(13)` we do not need to find `collatz_chain_len(5)` again as it was computed previously, so we need to cache the results of smaller sub problems. This technique is called memoization. A simple way to memoize the results is to use a hashmap(dict in python).

Non-Memoized Solution

```python
def collatz_chain_len(num):
    if num == 1:
        return 1
    if num % 2 == 0:
        return collatz_chain_len(num//2) + 1
    return collatz_chain_len(3*num+1)

iter = 1
max_chain_len = 0
max_chain_num = 0
while iter < int(1e6):
    curr_chain_len = collatz_chain_len(iter)
    if curr_chain_len > max_chain_len:
        max_chain_len = curr_chain_len
        max_chain_num = iter
    iter += 1

print(max_chain_num)
```

Memoized Solution

```python
cache = dict()

def collatz_chain_len(num):
    if num == 1:
        return 1

    # already found the chain length
    if num in cache:
        return cache[num]

    # else now solve this problem
    if num % 2 == 0:
        cache[num] = collatz_chain_len(num//2) + 1
    else:
        cache[num] = collatz_chain_len(3*num + 1) + 1

    # return the result
    return cache[num]

iter = 1
max_chain_len = 0
max_chain_num = 0
while iter < int(1e6):
    curr_chain_len = collatz_chain_len(iter)

    if curr_chain_len > max_chain_len:
        max_chain_len = curr_chain_len
        max_chain_num = iter

    iter += 1

print(max_chain_num)
```