Happy Number

Question:

https://leetcode.com/problems/happy-number/

Write an algorithm to determine if a number `n` is happy.

A happy number is a number defined by the following process:

- *Starting with any positive integer, replace the number by the sum of the squares of its digits.*
- *Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.*
- *Those numbers for which this process ends in 1 are happy.*

Return `true` if `n` is a happy number, and `false` if not.

Example 1:

```
Input: n = 19
Output: true
Explanation:
```

$1^2 + 9^2 = 82$

$8^2 + 2^2 = 68$

$6^2 + 8^2 = 100$

$1^2 + 0^2 + 0^2 = 1$

Example 2:

Input: n = 2

Output: false

Approach 1:

My approach was to check for a loop. I found this approach by looking at the 2nd Test Case. The second test case moves forward like this: 2 -> 4 -> 16 -> 37 -> 58 -> 89 -> 145 -> 42 -> 20 -> 4 and finally a loop starting from 4.
To check for cycles, I used a hashmap as data fetching takes O(1) time in a hashmap hence we can check in constant time.

Function for finding the sum of squares of the digits.

```python
def check(self,n:int):
    a=0
    while(n>0):
        d=n%10
        a+=d**2
        n=n//10
    return a
```

Solution 1:

https://gist.github.com/vermaayush680/574beb6781d419d027e543215c7ca2dd

```
def isHappy(self, n: int) -> bool:

    d={n}

    while(n!=1):

        n=self.check(n)

        if n in d:

            return False

        d.add(n)

    return True
```

Time Complexity: O(n)

Space Complexity: O(n)

Another approach to finding cycles is Floyd's Cycle-Finding Algorithm.

Approach 2:

Floyd's Cycle-Finding Algorithm:

Floyd's cycle-detection algorithm is a two-pointer algorithm used to find the presence of cycles.
Mostly used in linked lists for cycle detection, this algorithm takes constant time.

Explanation:

https://www.geeksforgeeks.org/detect-loop-in-a-linked-list/

Algorithm:

Use Two Pointers

1. Slow
2. Fast

The fast pointer moves 2 steps at a time while the slow pointer moves 1 step at a time. The idea is that if a cycle is present then at some point in the cycle, the fast pointer will pass the slow pointer and this will prove the existence of a cycle.

If there is no loop then these pointers will never meet and we will reach the end.

Function for finding the sum of squares of the digits. Same as Before.

```
def check(self,n:int):
    a=0
    while(n>0):
        d=n%10
        a+=d**2
        n=n//10
    return a
```

Solution 2:

https://gist.github.com/vermaayush680/36cf1fecb062e2022b7482149d1e7183

```
def isHappy(self, n: int) -> bool:
    slow = n
    fast = self.check(n)
    while fast!=1 and fast!=slow:
        slow=self.check(slow)
        fast=self.check(self.check(fast))
    return fast==1
```

Time Complexity: O(n)

Space Complexity: O(1)

Approach 3:

I found this approach through the discussion forum and think that this is perhaps the best solution among the 3.

After checking various numbers, it was found that there is a single loop that occurs.
4 -> 16 -> 37 -> 58 -> 89 -> 145 -> 42 -> 20 -> 4

Every number that isn't ending at 1 eventually ends in this loop.
The entry of this loop is 4 so we can just check if the sum of square of digits is 4 if so we return False as it will form a loop.

So similar to approach 1, we check for a loop but instead of checking for repeating numbers, we check for 4 as this is the only sum that forms a loop.

Solution 3:

https://gist.github.com/vermaayush680/a710f15143f7143b1b08eee06c63f8c9

```
def isHappy(self, n: int) -> bool:
    while(n!=1) and n!=4:
        n=self.check(n)
        print(n)
    return n==1
```

Time Complexity: O(n)

Space Complexity: O(1)