☰   `>_` educative(/learn)   ⚙   📋

Happy Number (medium)

We'll cover the following ⌃

- Problem Statement

- Try it yourself

- Solution

  - Code

- Time Complexity

- Space Complexity

Problem Statement#

Any number will be called a happy number if, after repeatedly replacing it with a number equal to the **sum of the square of all of its digits, leads us to number '1'**. All other (not-happy) numbers will never reach '1'. Instead, they will be stuck in a cycle of numbers which does not include '1'.

Example 1:

```
Input: 23
Output: true (23 is a happy number)
Explanations: Here are the steps to find out that 23 is a happy
number:
```

1. $2^2 + 3^2$ = 4 + 9 = 13
2. $1^2 + 3^2$ = 1 + 9 = 10
3. $1^2 + 0^2$ = 1 + 0 = 1

Example 2:

```
Input: 12
Output: false (12 is not a happy number)
Explanations: Here are the steps to find out that 12 is not a ha
ppy number:
```

1. $1^2 + 2^2$ = 1 + 4 = 5
2. 5^2 = 25
3. $2^2 + 5^2$ = 4 + 25 = 29
4. $2^2 + 9^2$ = 4 + 81 = 85
5. $8^2 + 5^2$ = 64 + 25 = 89
6. $8^2 + 9^2$ = 64 + 81 = 145
7. $1^2 + 4^2 + 5^2$ = 1 + 16 + 25 = 42
8. $4^2 + 2^2$ = 16 + 4 = 20
9. $2^2 + 0^2$ = 4 + 0 = 4
10. 4^2 = 16
11. $1^2 + 6^2$ = 1 + 36 = 37
12. $3^2 + 7^2$ = 9 + 49 = 58

13. $5^2 + 8^2$ = 25 + 64 = 89

Step '13' leads us back to step '5' as the number becomes equal to '89', this means that we can never reach '1', therefore, '12' is not a happy number.

Try it yourself#

Try solving this question here:

| Java	Python3	JS JS	C++

```java
 1   class HappyNumber {
 2
 3     public static boolean find(int num) {
 4       // // TODO: Write your code here
 5       return false;
 6     }
 7
 8     public static void main(String[] args) {
 9       System.out.println(HappyNumber.find(23));
10       System.out.println(HappyNumber.find(12));
11     }
12   }
```

Run   Save   Reset   []

Solution#

The process, defined above, to find out if a number is a happy number or not, always ends in a cycle. If the number is a happy number, the process will be stuck in a cycle on number '1,' and if the number is not a happy

number then the process will be stuck in a cycle with a set of numbers. As we saw in Example-2 while determining if '12' is a happy number or not, our process will get stuck in a cycle with the following numbers: 89 -> 145 -> 42 -> 20 -> 4 -> 16 -> 37 -> 58 -> 89

We saw in the LinkedList Cycle (https://www.educative.io/collection/page/5668639101419520/56714648543 55968/6556337280385024/) problem that we can use the **Fast & Slow pointers** method to find a cycle among a set of elements. As we have described above, each number will definitely have a cycle. Therefore, we will use the same fast & slow pointer strategy to find the cycle and once the cycle is found, we will see if the cycle is stuck on number '1' to find out if the number is happy or not.

Code#

Here is what our algorithm will look like:

☕ Java	🐍 Python3	⚙ C++	JS JS

```java
class HappyNumber {

  public static boolean find(int num) {
    int slow = num, fast = num;
    do {
      slow = findSquareSum(slow); // move one step
      fast = findSquareSum(findSquareSum(fast)); // move two steps
    } while (slow != fast); // found the cycle

    return slow == 1; // see if the cycle is stuck on the number '1'
  }

  private static int findSquareSum(int num) {
```

```
14        int sum = 0, digit;
15        while (num > 0) {
16          digit = num % 10;
17          sum += digit * digit;
18          num /= 10;
19        }
20        return sum;
21      }
22
23      public static void main(String[] args) {
24        System.out.println(HappyNumber.find(23));
25        System.out.println(HappyNumber.find(12));
26      }
27    }
```

Run   **Save**   **Reset**   ⌗

Time Complexity#

The time complexity of the algorithm is difficult to determine. However we know the fact that all unhappy numbers (https://en.wikipedia.org/wiki/Happy_number#Sequence_behavio) eventually get stuck in the cycle: 4 -> 16 -> 37 -> 58 -> 89 -> 145 -> 42 -> 20 -> 4

This sequence behavior (https://en.wikipedia.org/wiki/Happy_number#Specific_%7F'%22%60UNIQ --postMath-00000027-QINU%60%22'%7F-happy_numbers) tells us two things:

1. If the number N is less than or equal to 1000, then we reach the cycle or '1' in at most 1001 steps.

2. For $N > 1000$, suppose the number has 'M' digits and the next number is 'N1'. From the above Wikipedia link, we know that the sum of the squares of the digits of 'N' is at most 9^2M, or $81M$ (this will happen when all digits of 'N' are '9').

This means:

1. $N1 < 81M$
2. As we know $M = log(N + 1)$
3. Therefore: $N1 < 81 * log(N + 1)$ => $N1 = O(logN)$

This concludes that the above algorithm will have a time complexity of $O(logN)$.

Space Complexity#

The algorithm runs in constant space $O(1)$.

← **Back**

Next →

Start of LinkedList Cycle (medium)

Middle of the LinkedList (easy)

✅ Mark as Completed

⚠ Report an Issue

? Ask a Question
(https://discuss.educative.io/tag/happy-number-medium__pattern-fast-slow-pointers__grokking-the-coding-interview-patterns-for-coding-questions?open=true&ctag=grokking-the-coding-interview-patterns-for-coding-questions__design-gurus&aid=5668639101419520&cid=5671464854355968&pid=5315930441646080)