<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:

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

**Example 2:**

Input: n = 2

Output: false

**Constraints:**

1 <= n <= 231 - 1

**Attempt 1: 2023-09-08**

**Solution 1:  Hash Table (10 min)**

Style 1:

class Solution {

public boolean isHappy(int n) {

Set<Integer> set = new HashSet<Integer>();

while(n != 1) {

int sum = 0;

while(n != 0) {

sum += (n % 10) \* (n % 10);

n /= 10;

}

n = sum;

if(set.contains(sum)) {

break;

}

set.add(n);

}

return n == 1;

}

}

======================================================================

Style 2:

class Solution {

public boolean isHappy(int n) {

if(n == 1) {

return true;

}

Set<Integer> set = new HashSet<Integer>();

while(n != 1) {

int sum = 0;

while(n != 0) {

sum += (n % 10) \* (n % 10);

n /= 10;

}

// If sum up as 1 return true

if(sum == 1) {

return true;

}

// If looply find same number means circle

if(set.contains(sum)) {

break;

}

set.add(sum);

n = sum;

}

return false;

}

}

**Refer to**

<https://grandyang.com/leetcode/202/>

这道题定义了一种快乐数，就是说对于某一个正整数，如果对其各个位上的数字分别平方，然后再加起来得到一个新的数字，再进行同样的操作，如果最终结果变成了1，则说明是快乐数，如果一直循环但不是1的话，就不是快乐数，那么现在任意给我们一个正整数，让我们判断这个数是不是快乐数，题目中给的例子19是快乐数，那么我们来看一个不是快乐数的情况，比如数字11有如下的计算过程：

1^2 + 1^2 = 2

2^2 = 4

4^2 = 16

1^2 + 6^2 = 37

3^2 + 7^2 = 58

5^2 + 8^2 = 89

8^2 + 9^2 = 145

1^2 + 4^2 + 5^2 = 42

4^2 + 2^2 = 20

2^2 + 0^2 = 4

我们发现在算到最后时数字4又出现了，那么之后的数字又都会重复之前的顺序，这个循环中不包含1，那么数字11不是一个快乐数，发现了规律后就要考虑怎么用代码来实现，我们可以用 HashSet 来记录所有出现过的数字，然后每出现一个新数字，在 HashSet 中查找看是否存在，若不存在则加入表中，若存在则跳出循环，并且判断此数是否为1，若为1返回true，不为1返回false，代码如下：

class Solution {

public:

bool isHappy(int n) {

unordered\_set<int> st;

while (n != 1) {

int sum = 0;

while (n) {

sum += (n % 10) \* (n % 10);

n /= 10;

}

n = sum;

if (st.count(n)) break;

st.insert(n);

}

return n == 1;

}

};

**Solution 2: Two Pointers (10 min, fast + slow)**

class Solution {

public boolean isHappy(int n) {

if(n == 1) {

return true;

}

int slow = n;

int fast = n;

while(true) {

slow = transfer(slow);

fast = transfer(fast);

fast = transfer(fast);

if(slow == fast) {

break;

}

}

return slow == 1;

}

private int transfer(int n) {

int sum = 0;

while(n != 0) {

sum += (n % 10) \* (n % 10);

n /= 10;

}

return sum;

}

}

==========================================================

'fast' will always earlier to get 1 if exist

class Solution {

public boolean isHappy(int n) {

if(n == 1) {

return true;

}

int slow = n;

int fast = n;

while(true) {

slow = transfer(slow);

fast = transfer(fast);

fast = transfer(fast);

// 'fast' will always earlier to get 1 if exist

if(fast == 1) {

return true;

}

if(slow == fast) {

break;

}

}

return false;

}

private int transfer(int n) {

int sum = 0;

while(n != 0) {

sum += (n % 10) \* (n % 10);

n /= 10;

}

return sum;

}

}

**Refer to**

<https://leetcode.com/problems/happy-number/solutions/56917/my-solution-in-c-o-1-space-and-no-magic-math-property-involved/>

I see the majority of those posts use hashset to record values. Actually, we can simply adapt the Floyd Cycle detection algorithm. I believe that many people have seen this in the Linked List Cycle detection problem. The following is my code:

int digitSquareSum(int n) {

int sum = 0, tmp;

while (n) {

tmp = n % 10;

sum += tmp \* tmp;

n /= 10;

}

return sum;

}

bool isHappy(int n) {

int slow, fast;

slow = fast = n;

do {

slow = digitSquareSum(slow);

fast = digitSquareSum(fast);

fast = digitSquareSum(fast);

} while(slow != fast);

if (slow == 1) return 1;

else return 0;

}

**Detailed explanation for those who still don't get the logic/idea behind this algorithm**

Surprisingly, we can apply the Floyd Cycle Detection (the one we used in Detect Linked List Cycle) on this problem: think of what is a **cycle** in this case:

from a number A, we can get to another B using the ways given in this case from number B, when we doing the transformation, we will eventually get back to B again ---> this forms a **cycle** (infinite loop)

for example:

1^2 + 1^2 = 2

2^2 = 4 ------> notice that from here we are starting with 4

4^2 = 16

1^2 + 6^2 = 37

3^2 + 7^2 = 58

5^2 + 8^2 = 89

8^2 + 9^2 = 145

1^2 + 4^2 + 5^2 = 42

4^2 + 2^2 = 20

2^2 + 0^2 = 4 -------> notice that we just get back to 4 again

Using Floyd Cycle Detection algorithm (fast and slow pointer), we will be able to actually get the value of B. Then the rest of task would be very simple, we simply check whether this value will be 1 or not.

You may ask: what if value "1" also appears in the cycle and we are skipping over it. Well, in that case, the value that slow and fast are equal to will be 1, as transformation of 1 is still 1, so we still cover this case.

Nice idea, but it seems that if n is a happy number, definitely fast will turn to 1 first, which means in this case, your code will take one time more loops than it should be, because you have to wait the slow turning to 1 as well. So I made a little simplification.

int digitSquareSum(int n) {

int sum = 0, tmp;

while (n) {

tmp = n % 10;

sum += tmp \* tmp;

n /= 10;

}

return sum;

}

bool isHappy(int n) {

int slow, fast;

slow = fast = n;

do {

slow = digitSquareSum(slow);

fast = digitSquareSum(fast);

fast = digitSquareSum(fast);

if(fast == 1) return 1;

} while(slow != fast);

return 0;

}

**Refer to**

<https://grandyang.com/leetcode/202/>

这道题还有一种快慢指针的解法，由热心网友喵团团提供，跟之前那道 [Linked List Cycle](http://www.cnblogs.com/grandyang/p/4137187.html) 检测环的方法类似，不同的是这道题环一定存在，不过有的环不符合题意，只有最后 slow 停在了1的位置，才表明是一个快乐数。而且这里每次慢指针走一步，快指针走两步，不是简单的指向next，而是要调用子函数计算各位上数字的平方和，当快慢指针相等时，跳出循环，并且判断慢指针是否为1即可，参见代码如下：

class Solution {

public:

bool isHappy(int n) {

int slow = n, fast = n;

while (true) {

slow = findNext(slow);

fast = findNext(fast);

fast = findNext(fast);

if (slow == fast) break;

}

return slow == 1;

}

int findNext(int n) {

int res = 0;

while (n > 0) {

res += (n % 10) \* (n % 10);

n /= 10;

}

return res;

}

};

**Refer to**

<https://grandyang.com/leetcode/202/>

其实这道题也可以不用 HashSet 来做，我们并不需要太多的额外空间，关于非快乐数有个特点，循环的数字中必定会有4，这里就不做证明了，我也不会证明，就是利用这个性质，就可以不用set了，参见代码如下：

class Solution {

public:

bool isHappy(int n) {

while (n != 1 && n != 4) {

int sum = 0;

while (n) {

sum += (n % 10) \* (n % 10);

n /= 10;

}

n = sum;

}

return n == 1;

}

};