Happy Number

Mark Cabanero

September 23, 2016

Program Design

Description

happyNumber tests positive integers and then either prints out a number if the sum of the squares converges to 1, or nothing if the sequence of the sum of the squares loops back onto itself. It achieves this in three steps:

1. Convert the number into an integer array through helper functions.
2. Iterate over the array and sum the squares of the digits.
3. Compare to see if the resulting sum is equal to 1 or a number already seen. If neither case is true, run the loop again on the new sum.

Tradeoffs

From observation:

- large happy numbers quickly converge to smaller integers when summing the squares.
  - A 32 digit number results in a 3-4 digit number for squared sums.
  - A 64 digit number results in a 4 digit number for squared sums.
  - A 128 digit number results in a 4-5 digit number for squared sums.
- happy numbers only occur with positive integers.

Since the input doesn't grow insanely large, a combination of integer arrays, strings, and a singly linked list are fine for the size of the numbers being dealt with. There shouldn't be concern with backtracking through the chain of squared sums, since the order just needs to be maintained. Therefore, a singly linked list was chosen over a doubly linked list.

Extensions

- More robust exceptions: Currently, the error cases are handled by rudamentary if-else statements. Better implementation would be through exceptions, catching them, and dealing with them appropriately.
- Better handling of large strings: Currently, it's assumed that a number passed as a string is not a trivial large happy number (1,000,000,000,000 for example), which is clearly happy. This will process twice: once as a string of length 13, and then once again as an integer of 1. Testing if the first digit is one and then the rest are 0s would be naive solution.

Test Cases

Since happy numbers only occur with positive integers, any non-positive number or any decimal should result in an error. Also, since the user can test versus large numbers, malformed string input (empty or any characters) should not work, in addition to the conditions held by the integers (no negative, no zero).

- Negative integer -18174: Negative numbers do not work while making happy numbers.
- Non-positive integer 0: 0 would not result in a happy number.
- Real number 4.2: 4.2 is not a positive integer, so it is not a happy number.
- String input "00000000000000000000000000000": It's firstly a non-existant integer, and secondly, should not result in a happy number.
- String input "abcd": Characters should not result in a happy number.
- String input "": Input cannot be empty, and should not result in a happy number.

```
False inputs that should not work:
Number primatives:
Testing -18174 (negative integer).
ERROR: Happy numbers can only be positive integers.
Testing 0 (non-positive integer).
ERROR: Happy numbers can only be positive integers.
Testing 4.2 (non-integer).
ERROR: Happy numbers can only be positive integers, not a decimal.

String inputs:
Testing 00000000000000000000000000000 (invalid large input).
ERROR: Happy numbers can only be positive integers.
Testing abcd (invalid input).
ERROR: Cannot have characters in input.
Testing "" (empty input).
ERROR: Input passed is empty.
```

Figure 1: Invalid input tests.