Manny

678 Followers · About Follow

How To Solve The Counting Valleys Challenge

How To Solve HackerRank's Counting Valleys Code Challenge With JavaScript

Manny Jan 2, 2019 · 9 min read

How To Solve The Counting Valleys Code Challenge

Problem

The Counting Valleys challenge is counting the number of valleys Gary the hiker goes though:

- Gary = Hiker

- S — Descriptive string that is the path of steps Gary the hiker takes

- U and D are “Up” and “Down” respectively and the direction of Gary’s step
- N = number of steps between 2 and 10^6 (1,000,000)
- AR is a single string of spaced numbers with values ranging between 1 and 100 — *ex 10 11 20 31*
- N is the number of values in steps in the path between 2 and 1,000,000 (which could be useless if we’re just calculating the array length)
- A valley is defined as going lower than sea level and then back to sea level

Example 1:

```
N = 8
S = UDDDUDUU

_/\      _
   \    /
    \/\/

Result: 1 Valley
```

Example 2:

```
N = 10
S = UDDDUDUUDU

_/\      _  _
   \    / \/
    \/\/

Result: 2 Valleys
```

Goal

number of valleys found by traversing the string path (S) of steps

Covering Our Bases

Covering our bases, we need to make sure that:

- Number of item (values) of S is in between 2 and 10^6
- N is an integer, between 2 and 10^6
- N = the total number of values of S

Counting The Values Of The Path (S)

In order to get a better sense of how many values are in S we need to convert the string to an array and count it.

```
function countingValleys(n, s) {
     // setting the constraints
     const min = 2;
     const max = 1000000;

     // if it's a string convert it to an array
     // ex "UDU" = ["U", "D", "U"]
     s = (typeof ar === "string") ? s.split('') : s;

     // check if s meets the requirements
     if (s.length >= min && s.length <= max) {
          // continue
     }
}
```

Validating N

Next we need to make sure that N is an integer and matches the same number of values of the path (S).

```
function countingValleys(n, s) {
     // setting the constraints
     const min = 2;
     const max = 1000000;

     // if it's a string convert it to an array
     // ex "UDU" = ["U", "D", "U"]
```

```
        if (s.length >= min
            && s.length <= max
            && n === parseInt(n, 0)
            && n >= min
            && n <= max
            && n === s.length) {
            // continue
        }
    }
```

Understanding The Problem

In order to get a better idea of directions, another way we could look at things is going Up (U) would +1 and going Down (D) would be -1. If the definition of a valley is to go below sea level and then back to sea level, our goal is come up with a way to only start counting when we meet this condition.

Converting Steps To Integers

Converting the array to integers is the first step to getting the directions.

```
// Example
// n = 8
// s = "UDDDUDUU"

function countingValleys(n, s) {
    // setting the constraints
    const min = 2;
    const max = 1000000;

    // if it's a string convert it to an array
    // ex "UDU" = ["U", "D", "U"]
    s = (typeof s === "string") ? s.split('') : s;
    // ["U", "D", "D", "D", "U", "D", "U", "U"]

    // check if s meets the requirements
    if (s.length >= min
        && s.length <= max
        && n === parseInt(n, 0)
        && n >= min
        && n <= max
        && n === s.length) {

        // converting the array steps to integers
        s = s.map(steps => ((steps === "U") ? 1 : -1));
        // [1, -1, -1, -1, 1, -1, 1, 1]
    }
}
```

```
// Example
// n = 8
// s = "UDDDUDUU"

function countingValleys(n, s) {
     // setting the constraints
     const min = 2;
     const max = 1000000;

     // if it's a string convert it to an array
     // ex "UDU" = ["U", "D", "U"]
     s = (typeof s === "string") ? s.split('') : s;
     // ["U", "D", "D", "D", "U", "D", "U", "U"]

     // check if s meets the requirements
     if (s.length >= min
          && s.length <= max
          && n === parseInt(n, 0)
          && n >= min
          && n <= max
          && n === s.length) {

          // converting the array steps to integers
          s = s.map(steps => ((steps === "U") ? 1 : -1));
          // [1, -1, -1, -1, 1, -1, 1, 1]

          let path = 0;
          for(let i in s) {
               path += s[i];
          }
          //  0 +  1 =  1
          //  1 + -1 =  0
          //  0 + -1 = -1
          // -1 + -1 = -2
          // -2 +  1 = -1
          // -1 + -1 = -2
          // -2 +  1 = -1
          // -1 +  1 =  0
          // initial = 0
          // end = 0
     }
}
```

Defining Initial Conditions For Paths

Next we need to make handle the condition of meeting the valley requirements.

- Below sea level (<0)

```
// Example
// n = 8
// s = "UDDDUDUU"

function countingValleys(n, s) {
     // setting the constraints
     const min = 2;
     const max = 1000000;
     let valleys = 0;

     // if it's a string convert it to an array
     // ex "UDU" = ["U", "D", "U"]
     s = (typeof s === "string") ? s.split('') : s;
     // ["U", "D", "D", "D", "U", "D", "U", "U"]

     // check if s meets the requirements
     if (s.length >= min
          && s.length <= max
          && n === parseInt(n, 0)
          && n >= min
          && n <= max
          && n === s.length) {

          // converting the array steps to integers
          s = s.map(steps => ((steps === "U") ? 1 : -1));
          // [1, -1, -1, -1, 1, -1, 1, 1]

          let path = 0;
          for(let i in s) {
               path += s[i];
               if (path < 0) {
                    // start of a valley
               }
               if (path == 0) {
                    // end of valley, increase count
               }
          }
          //  0 +  1 =  1 (Moved up = valley not started)
          //  1 + -1 =  0 (Back to sea level = valley not started)
          //  0 + -1 = -1 (Below sea level = valley started)
          // -1 + -1 = -2 (Moved lower = still in valley
          // -2 +  1 = -1 (Moved up = still in valley)
          // -1 + -1 = -2 (Moved lower = still in valley)
          // -2 +  1 = -1 (Moved up = still in valley)
          // -1 +  1 =  0 (Back to sea level = 1 valley)
          // initial = 0
          // end = 0
     }
}
```

Accounting For Still Being In The Valley

```
// Example
// n = 8
// s = "UDDDUDUU"

function countingValleys(n, s) {
     // setting the constraints
     const min = 2;
     const max = 1000000;
     let valleys = 0;
     let isInValley = false;

     // if it's a string convert it to an array
     // ex "UDU" = ["U", "D", "U"]
     s = (typeof s === "string") ? s.split('') : s;
     // ["U", "D", "D", "D", "U", "D", "U", "U"]

     // check if s meets the requirements
     if (s.length >= min
          && s.length <= max
          && n === parseInt(n, 0)
          && n >= min
          && n <= max
          && n === s.length) {

          // converting the array steps to integers
          s = s.map(steps => ((steps === "U") ? 1 : -1));
          // [1, -1, -1, -1, 1, -1, 1, 1]

          let path = 0;
          for(let i in s) {
               path += s[i];
               if (path < 0 && !isInValley) {
                    // to check that we're not already in a valley
                    // start of a valley
                    isInValley = true;
               }
               if (path == 0 && isInValley) {
                    // to check if we're just coming out of a valley
                    // end of valley, increase count
                    valleys++; // increase count
                    isInValley = false; // reset isInValley
               }
          }
          //  0 +  1 =  1 (Moved up = valley not started)
          //  1 + -1 =  0 (Back to sea level = valley not started)
          //  0 + -1 = -1 (Below sea level = valley started)
          // -1 + -1 = -2 (Moved lower = still in valley
          // -2 +  1 = -1 (Moved up = still in valley)
          // -1 + -1 = -2 (Moved lower = still in valley)
          // -2 +  1 = -1 (Moved up = still in valley)
          // -1 +  1 =  0 (Back to sea level = 1 valley)
          // initial = 0
```

```
        // to make sure we return even when the req. are not met
        return valleys;
    }
```

Let's run the same values again from above:

```
// Example 1
// n = 8
// s = "UDDDUDUU"

countingValleys(8, "UDDDUDUU");

// path = 1
// isInValley = false
// valleys = 0

// path = 0
// isInValley = false
// valleys = 0

// path = -1
// isInValley = true
// valleys = 0

// path = -2
// isInValley = true
// valleys = 0

// path = -1
// isInValley = true
// valleys = 0

// path = -2
// isInValley = true
// valleys = 0

// path = -1
// isInValley = true
// valleys = 0

// path = 0
// isInValley = false
// valleys = 1

// Solution = 1
```

Here is Example 2 (nearly the same):

```
// s = UDDDUDUDU

// ...

// path = 0
// isInValley = false
// valleys = 1

// path = -1
// isInValley = true
// valleys = 1

// path = 0
// isInValley = false
// valleys = 2

// Solution = 2
```

Refactoring For Performance

Now that we have the solution, let's refactor the for loop, with some more performing method, such as ***map*** and ***reduce***.

Before

```
function countingValleys(n, s) {
     const min = 2;
     const max = 1000000;
     let valleys = 0;
     let isInValley = false;

     s = (typeof s === "string") ? s.split('') : s;

     if (s.length >= min
          && s.length <= max
          && n === parseInt(n, 0)
          && n >= min
          && n <= max
          && n === s.length) {

          s = s.map(steps => ((steps === "U") ? 1 : -1));

          let path = 0;
          for(let i in s) {
               path += s[i];
               if (path < 0 && !isInValley) {
                    isInValley = true;
               }
               if (path == 0 && isInValley) {
```

```
        }
    }

    return valleys;
}
```

After

```
function countingValleys(n, s) {
    const min = 2;
    const max = 1000000;
    let valleys = 0;
    let isInValley = false;

    s = (typeof s === "string") ? s.split('') : s;

    if (s.length >= min
        && s.length <= max
        && n === parseInt(n, 0)
        && n >= min
        && n <= max
        && n === s.length) {

        // remove s = s.map because we're already iterating
        s.map(steps => ((steps === "U") ? 1 : -1))
            .reduce((prev, next) => {
                if (prev < 0 && !isInValley) {
                    isInValley = true;
                }
                if ((prev + next) === 0 && isInValley) {
                    valleys++;
                    isInValley = false;
                }
                // continue incrementing by adding
                return prev + next;
            });
    }

    return valleys;
}
```

Solution

And here is the full solution:

```
function countingValleys(n, s) {
    const min = 2;
```

```
    s = (typeof s === "string") ? s.split('') : s;

    if (s.length >= min
        && s.length <= max
        && n === parseInt(n, 0)
        && n >= min
        && n <= max
        && n === s.length) {

        s.map(steps => ((steps === "U") ? 1 : -1))
            .reduce((prev, next) => {
                if (prev < 0 && !isInValley) {
                    isInValley = true;
                }
                if ((prev + next) === 0 && isInValley) {
                    valleys++;
                    isInValley = false;
                }

                return prev + next;
            });
    }

    return valleys;
}
```

Test Cases

```
// N = 8, S = "UDDDUDUU", Expected 1
// N = 12, S = "DDUUDDUDUUUD", Expected 2
// N = 1, S = "DU", Expected 0
// N = 2, S = "DU", Expected 1
// N = 3, S = "DDU", Expected 0
// N = 1000001, S = "DDU", Expected 0
// N = 20, S = "DDUUDDUUDDUUDDUUDDUU", Expected 5
// N = 10, S = "UUUUUDUUUU", Expected 0

countingValleys(8, "UDDDUDUU"); // 1 ✅
countingValleys(12, "DDUUDDUDUUUD"); // 2 ✅
countingValleys(1, "DU"); // 0 ✅
countingValleys(2, " DU"); // 1 ✅
countingValleys(3, "DDU"); // 0 ✅
countingValleys(100001, "DDU"); // 0 ✅
countingValleys(20, "DDUUDDUUDDUUDDUUDDUU"); // 5 ✅
countingValleys(10, "UUUUUDUUUU"); // 0 ✅
```

Feedback?

If you got value from this, share it on twitter or other social media platforms. Thanks again for reading.

Please also follow me on **twitter**: @codingwithmanny and **instagram** at @codingwithmanny.

How To Coding JavaScript Code Nodejs