Homework 3

ECE567: Software Engineering 1

Tanya Sharma
tds104
15 October 2023

Design

- The homework assignment defines a class ArrayStats that extends the built-in Array class. This custom class adds methods for calculating the average and standard deviation of the elements in an array.
- It also includes a custom implementation of the reduce function using map. The average method calculates the average of the array elements, and the stdev method calculates the standard deviation.
- In the average method, it first checks if the array is empty and returns NaN in that case. It then uses the custom customReduce method to sum up the array elements and calculates the average. It also defines a non-writable property avgVal to store the calculated average.
- The mapperVariance function is used by the stdev method to calculate the variance of each element with respect to the average value. The stdev method calculates the standard deviation using the variance and defines a non-writable property sdevVal to store the calculated standard deviation.
- After defining the ArrayStats class, it creates an instance of it with some initial values, calculates the average and standard deviation, and logs the results to the console.

Commands

In order to execute the both the program as well as test cases the following instructions can be followed:

1. **npm install**
Install all necessary dependencies

2. **node main.js**
This command will execute the main function where all our logic takes place

3. **jasmine**
This command will execute the test suite

Code Snippets

Main Program: main.js

```
class ArrayStats extends Array{

    //implementing reduce using map for use in the code
    customReduce(reducer, initialValue){
        let accumulator=initialValue;
        this.map((currentValue,currentIndex)=>{
```

```
accumulator=reducer(accumulator,currentValue,currentIndex,th
is);
        });
        return accumulator;
    }

    //function to calculate average
    average(){
        //handle case when array is empty
        if(this.length===0)
        {
            return NaN;
        }
        const averageValue=this.customReduce((a,b)=>a+b,0)/
this.length;
        Object.defineProperty(this,"avgVal",{
            value:averageValue,
            writable:false,
        });
        return averageValue;
    }

    mapperVariance(currentValue){
        const averageValue=this.avgVal;
        return (currentValue-averageValue)**2;
    }

    stdev(){
        //handle empty array
        if(this.length<1){
            return NaN;
        }
        const
varianceArray=this.map(this.mapperVariance,this);
        const
varianceVal=varianceArray.customReduce((a,b)=>a+b,0);
        const stdevValue=Math.sqrt(varianceVal/
(this.length-1));
        Object.defineProperty(this,"sdevVal",{
            value:stdevValue,
            writable:false,
        });
        return stdevValue;
    }
}

let myArray=new ArrayStats(7,11,5,14);
const avgResult = myArray.average();
```

```
const stdevResult = myArray.stdev();
console.log("Average:",avgResult);
console.log("Standard Deviation:",stdevResult);

module.exports={
    ArrayStats
};
```

Test: test.spec.js

```
const {ArrayStats,average,stdev,mapperVariance}=require('../
main');

describe('ArrayStats',function(){

    let myArray;

    beforeEach(function(){
        myArray=new ArrayStats(7,11,5,14);
    });

    it('should create an instance of ArrayStats with initial
values',function(){
        const myArray=new ArrayStats(7,11,5,14);
        expect(myArray instanceof ArrayStats).toBe(true);
    });

    it('should handle empty arrays for average',function(){
        myArray=new ArrayStats();
        const avgResult=myArray.average();
        expect(isNaN(avgResult)).toBe(true);
    });

    it('should handle empty arrays for stdev',function(){
        myArray=new ArrayStats();
        const stdevResult=myArray.stdev();
        expect(isNaN(stdevResult)).toBe(true);
    });

    it('should calculate the correct average',function(){
        const avgResult=myArray.average();
        expect(avgResult).toBe(9.25);
    });

    it('should set avgVal property correctly',function(){
        myArray.average();
        expect(myArray.avgVal).toBe(9.25);
```

```
    });

    it('should calculate the correct standard
deviation',function(){
        myArray.average();
        const stdevResult=myArray.stdev();
        expect(stdevResult).toBeCloseTo(4.031128874149275);
    });

    it('should set sdev property correctly',function(){
        myArray.average();
        myArray.stdev();
        expect(myArray.sdevVal).toBe(4.031128874149275);
    });

    it('should be able to handle average of a single element
as 0',function(){
        const singleArray=new ArrayStats(5);
        const avgResult=singleArray.average();
        expect(avgResult).toBe(0);
    });

    it('should be able to handle standard deviations of a
single element as 0',function(){
        const singleArray=new ArrayStats(5);
        singleArray.average();
        const stdevResult=singleArray.stdev();
        expect(stdevResult).toBe(0);
    });

    it('should handle an array with negative values for
average',function(){
        const negativeArray=new ArrayStats(-3,-7,-1,-5);
        const avgResult=negativeArray.average();
        expect(avgResult).toBe(-4);
    });

    it('should handle an array with negative values for
standard deviation',function(){
        const negativeArray=new ArrayStats(-3,-7,-1,-5);
        negativeArray.average();
        const stdevResult=negativeArray.stdev();
        expect(stdevResult).toBeCloseTo(2.5819888974716);
    });

    it('should be able to handle a large array in a
reasonable time for average',function(){
        jasmine.DEFAULT_TIMEOUT_INTERVAL=10000;
```

```
        const largeArray = new
ArrayStats(...Array(10000).fill(1));
        const startAverage=new Date().getTime();
        const avgResult=largeArray.average();
        const endAverage=new Date().getTime();
        const maxExecutionTime=5000;
        expect(endAverage-
startAverage).toBeLessThan(maxExecutionTime);
        expect(avgResult).toBe(1);
    });

    it('should be able to handle a large array in a
reasonable time for standard deviation',function(){
        jasmine.DEFAULT_TIMEOUT_INTERVAL=10000;
        const largeArray = new
ArrayStats(...Array(10000).fill(1));
        largeArray.average();
        const startStdev=new Date().getTime();
        const stdevResult=largeArray.stdev();
        const endStdev=new Date().getTime();
        const maxExecutionTime=5000;
        expect(endStdev-
startStdev).toBeLessThan(maxExecutionTime);
        expect(stdevResult).toBe(0);
    });

    it('should calculate the squared difference from the
average',function(){
        myArray.average();
        const varianceResult=myArray.mapperVariance(11);
        expect(varianceResult).toBe(3.0625);
    });

    it('should work correctly for multiple
instances',function(){
        const a1=new ArrayStats(7,11,5,14);
        const a2=new ArrayStats(3,8,2,9);

        const avg1=a1.average();
        const avg2=a2.average();

        expect(avg1).toBe(9.25);
        expect(avg2).toBe(5.5);
        expect(a1.avgVal).toBe(9.25);
        expect(a2.avgVal).toBe(5.5);
    });

    it('should handle arrays with very large values (edge
cases)',function(){
```

```
        const largeArray = new ArrayStats(1e20, 2e20, 3e20,
4e20);
        const smallArray = new ArrayStats(1e-20, 2e-20,
3e-20, 4e-20);

        const avgResultLarge = largeArray.average();
        const stdevResultLarge = largeArray.stdev();
        const avgResultSmall = smallArray.average();
        const stdevResultSmall = smallArray.stdev();

        expect(avgResultLarge).toBe(2.5e20);

expect(stdevResultLarge).toBeCloseTo(1.2909944487358056e20);
        expect(avgResultSmall).toBe(2.5e-20);

expect(stdevResultSmall).toBeCloseTo(1.2909944487358056e-20)
;
    });

    it('should handle non-numeric values gracefully',
function() {
        const mixedArray = new ArrayStats('string', 7,
{ key: 'value' }, [1, 2, 3]);

        const avgResult = mixedArray.average();
        const stdevResult = mixedArray.stdev();

        expect(isNaN(avgResult)).toBe(true);
        expect(isNaN(stdevResult)).toBe(true);
    });

});
```