CSDS293 Assignment 6

Harley Phung

October 2023

1 Narrative: Expectation of the Algorithm

- Goal: To find the polynomials with the roots being $x_0, ..., x_{n-1}$ using the polynomial function: $p(x) = (x - x_0) * (x - x_1) * ... * (x - x_n)$

- Requirements for the algorithm: The algorithm is given a set of n distinct values $x_0 < x_1 < ... < x_{n-1}$ and returns a non-zero polynomial p(x) of degree n such that $p(x_0) = p(x_1) = ... = p(x_{n-1}) = 0$ and these are the only points where p(x)=0

2 Diagram Explanation

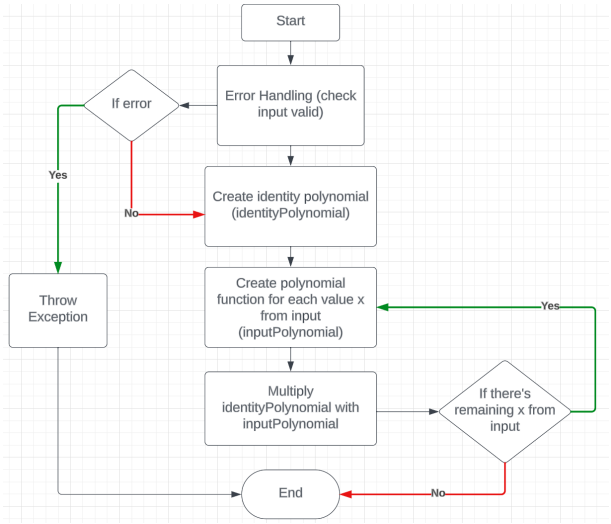

Figure 1: Process Diagram

- **Diagram Explanation**

- Step 1 : Error Handling: Check if the input is null or invalid to the problem (if the input is in different types).
    - If the error is found, immediately handle the error by throwing exception and end the function
    - If the error is not found, proceed to the next step
- Step 2: Initiate identity polynomial, named identityPolynomial. This is constructed to serve as the initial step of interpolation setup.
- Step 3 : Create polynomial function for each value x from the input. This step is to evaluate polynomial to zero when you substitute the x-values of the other data points
- Step 4 : Multiply identityPolynomial with polynomialFunction. This step is to update the identity polynomial by including the terms from the basis polynomial associated with the current data point.
    - If there exists remaining data from input, the return to step 3.
    - If there's out of data from input, terminate
- Step 5: Termination.

3 Pseudo code

1. Class: InterpolatePolynomial
    1. Function name: interpolate
       Input: dataPoints
       Output: resultPolynomial that interpolates with dataPoints
       Pseudo code:

```
function interpolate(dataPoints)
    check if valid, throw exception if needed
    result <- polynomial
    identityPolynomial <- identity()
    for each input in dataPoints:
        basisPolynomial(x) <- basis(dataPoints)
        result <- multiply(identityPolynomial, basisPolynomial)
    endLoop
    return result
endFunction
```

Make sure to have distinct dataPoints

basisPolynomial as input

    2. Function name: identity
       Output: polynomial with one coefficient
       Pseudo code:

```
function identity()
    return new Polynomial;
```

    3. Function name: basis
       Input: dataPoints

Output: basisPolynomial
Pseudo code:

```
function basis(dataPoints)
    basisPolynomial <- polynomial with one coefficient
    for each input in dataPoints:
        for each index in dataPoints.length:
            xj = dataPoints[j].x
            if xj != input:
                basisPolynomial *= Polynomial([-xj, 1]) / (input - xj)
            end if statement
    endLoop
    return basisPolynomial
endFunction
```

4. Function name: multiply
Input: identityPolynomial, basisPolynomial
Output: polynomialList (list of coefficients)
Pseudo code:

```
function multiply(identityPolynomial, basisPolynomial) {
    polynomialList <- result
    size <- identityPolynomial + basisPolynomial - 1
    for index < size - 1:
        count <- 0
        if i >= basisPolynomial.length:
            count <- i - basisPolynomial.length + 1
        end if statement
        if step <= identityPolynomial.length:
            identityPtr <- count
            basisPtr <- i - count
            while identityptr < identityPolynomial.length && basisPtr >= 0:
                polynomialList <- += identityPolynomial[identityPtr] +
                                        basisPolynomial[basisPtr]
            end Loop
        end if statement
    end loop
end function
}
```

Forgot return

2. Nested helper class: ValidDataPointsException
Function name: isValid
Input: dataPoints
Output: boolean (true / false)

3. Class: Polynomial

Function: zero(), identity(), plus(), times(), toString(). Can be reused from Assignment 3.

4 Justification

- This algorithm works because:

- For a polynomial to have a root at a particular value, $(x = x_i)$, it must have $(x - x_i)$ as a factor so that when you plug x_i into the polynomial, that factor becomes zero, making the polynomial function becomes 0.

- If there's a polynomial with multiple roots, it can be product of factors, each corresponding to one of the roots. $(x - x_0) * (x - x_1) * ... * (x - x_n)$ for $x_0, ..., x_{n-1}$. This leads to p(x) $= (x - x_0) * (x - x_1) * ... * (x - x_n)$ where each of the factors is a specific roots

5 Running Time Analysis

- For interpolate() only:

```
function interpolate(dataPoints)
    //Running time: O(1) because only check if it's null
    check if valid, throw exception if needed

    //Running time: O(1) because declare new polynomial
    result <- polynomial

    //Running time: O(1) because assign 1 to identityPolynomial
    identityPolynomial <- identity()

    //Running time: O(n^3) because going through n inputs from dataPoints
    // and multiply() inside.
    for each input in dataPoints:
        basisPolynomial(x) <- basis(dataPoints)
        result <- multiply(basisPolynomial, identityPolynomial)
    endLoop

    return result
endFunction
```

- For multiply():

```
function multiply(identityPolynomial, basisPolynomial) {
    polynomialList <- result
    size <- identityPolynomial + basisPolynomial - 1
//Running time = n^2 since this is for loop times second if statement
```

```
    for index < size - 1:
        count <- 0
        // Running time = 1
        if i >= basisPolynomial.length:
            count <- i - basisPolynomial.length + 1
        end if statement

        //Running time = n (size)
        if step <= identityPolynomial.length:
            identityPtr <- count
            basisPtr <- i - count
            while identityptr < identityPolynomial.length && basisPtr >= 0:
                polynomialList <- += identityPolynomial[identityPtr] +
                                     basisPolynomial[basisPtr]
            end Loop
        end if statement
    end loop
end function
}
```

Therefore, the running time is $O(n^2)$

6 Examples

1. n = 2
Input: [2,4]
Polynomial 1: p(x) = (x - 2)(x - 4)
Output: $p(x) = x^2 - 6x + 8$
p would be [1, -6, 8]
Expected output: $p(x) = x^2 - 6x + 8$

2. n = 4:
Input: [1,2,3]
Output: p(x) = (x-1)(x - 3)(x-2) -¿ $x^3 - 6x^2 + 11x - 6$
p would be [1, -8, 19, 12]

Expected output: p(x) = $x^3 - 6x^2 + 11x - 6$