SOEN 6011: SOFTWARE ENGINEERING PROCESSES

Project Deliverable 1

Himanshu Kohli
Student Id : 40070839
Applied Computer Science
Concordia University

Problem 1 | **Himanshu Kohli**
SOEN 6011 | **40070839**
Prof. Pankaj Kamthan | Due Date: July 19, 2019

1 Function 7 : a^{b^x}

The function a^{b^x} is an exponential function where a and b are real constants and x is a real variable. A normal exponential function is stagnant at first but then increases sharply as the value of the variable increases.

The growth or decay of the function depends upon the sign of the exponent so if the sign is positive the function increases and if the sign is negative the function decreases.

for e.g. if x is positive the function : f(x) = a^{b^x}

and if x is negative the function : f(x) = $\frac{1}{a^{b^x}}$

The domain and co-domain of the function

If we differentiate the function with respect to x

$$\frac{d}{dx}a^{b^x} = a^{b^x}\log a * b^x \log b \tag{1}$$

From equation 1 we can see if the value of either a or b is equal to 1 the $\log b$ or $\log a$ equals to zero because $\log 1 = 0$ which means the exponential function is constant because the derivative is zero.
For a positive value of b and a the function a^{b^x} is monotonically increasing
For a negative value of b and positive value of a the function a^{b^x} is monotonically increasing in negative side i.e. (negative x axis)
For a positive or negative value of b and negative value of a the function a^{b^x} is monotonically decreasing

So the value of the function f(x) = a^{b^x} actually depends upon the b^x and how the slope of the function will be rise or decay.
Properties :

1. The domain is all real numbers
2. The graph is continuous and smooth
3. The graph can be asymptotic or increase without bounds
4. The graph is monotonically increasing or decreasing

2 Requirements Specifications

2.1 Introduction

The implementation of well designed and well defined calculator function specifically a^{b^x} falls in the range of exponential functions which gamuts to alpine values quickly. And within the function lies certain requirements which are to be fulfilled in order to make the entire system tangible.

2.2 Requirements

1. **Title :** Verifying input
   Description : The system should validate the of the inputs by the user where the variables a and b should be real constant and the x must be a real variable whereas any other values rather than real numbers should result in an error.
   Requirement Type : Functional

2. **Title :** Infinity representation
   Description : The system should have a calculation for infinity i.e. for specifically large numbers it should print the result as infinite.
   Requirement Type : Functional, System

3. **Title :** Input Domain
   Description : The system should perform operation on any numbers which falls under the domain of real numbers.
   Requirement Type : Functional, Interface

4. **Title :** Positive large number handling
   Description : The largest value java systems can handle is max double value of 1.8 * 10^{308} so if the function evaluation reaches max value it shall be considered as positive infinity.
   Requirement Type : Functional, System

5. **Title :** Negative large number handling
   Description : The lowest value java systems can handle is a min double value of 4.9 * 10^{-324} so if the function evaluation reach min value it shall be considered as negative infinity.
   Requirement Type : Functional, System

6. **Title :** Rounding input value
   Description : The system shall be able of handle decimal point numbers with a rounding of up to 10 decimal places for the inputs.
   Requirement Type : Functional, Interface

7. **Title :** Valid input and output
   Description : The system shall take the input from command line and for a valid input it should print the valid result to the command line.
   Requirement Type : Functional

8. **Title :** Result Dependency on 'a' value
   Description : The function shall only print negative results when the value of a is negative and positive results when a is positive.

3 Pseudo Code

3.1 Solution

The function is a^{b^x} can be solved using Taylor series and to do that I am breaking and converting the function into two parts which are:

1. Let y = b^x which can be rewritten as $e^{x*log(b)}$
2. And z = a^y which can be rewritten as $e^{y*log(a)}$

The normal Taylor series for

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} ......\frac{x^n}{n!} \tag{2}$$

where n is the precision we are looking for.
And we have to find the algorithm to find the log of a number which has a Taylor series of it's own.

$$\ln\frac{1+x}{1-x} = 2 * (\frac{x}{1} + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7}.....) \tag{3}$$

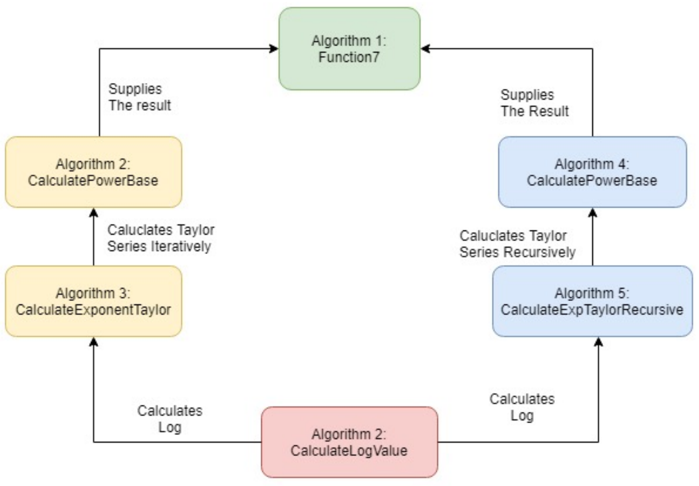
So we have to implement two **subordinate** functions which are

1. Natural Log Function to solve the power Taylor series
2. Taylor series to calculate power

Algorithms:
There are 2 sets of Algorithms written which represents two approaches The approach of Set 1 is iterative whereas the approach of Set 2 is recursive

1. Set 1 includes Algorithm 1,Algorithm 2, Algorithm 3 and Algorithm 4
2. Set 2 includes Algorithm 1,Algorithm 5, Algorithm 6 and Algorithm 4

Figure 1: Representation of the two approaches


Algorithm 1: Function7
Supplies The result
Supplies The Result
Algorithm 2: CalculatePowerBase
Algorithm 4: CalculatePowerBase
Caluclates Taylor Series Iteratively
Caluclates Taylor Series Recursively
Algorithm 3: CalculateExponentTaylor
Algorithm 5: CalculateExpTaylorRecursive
Calculates Log
Calculates Log
Algorithm 2: CalculateLogValue

3.2 Algorithms

ALGORITHM 1: Function7(a,b,x)

```
begin:
1. IF a,b,x are not real numbers THEN throw exception
2. Compute y = CalculatePower(b,x)
3. z = CalculatePower(a,y)
4. PRINT z
5. REPEAT the algorithm for new value of x
end
```

ALGORITHM 2: CalculatePower(base,exponent)

```
begin:
1. SET e = 2.718281828
2. SET LogOfBase = CalculateLog(base)/ CaluclateLog(e)
3. IF exponent $\geq$ 0 THEN
4.         SET exponent = exponent * LogOfBase
5. ELSE
6.         exponent = exponent * (-LogOfBase)
7. SET Result = CalculateExponentTaylor(exponent)
8. RETURN Result
end
```

ALGORITHM 3: CalculateExponentTaylor(exponent)

```
begin:
1. SET Temp = 1
2. SET i = 1
3. SET Result = 0
4. WHILE Temp > 0
5.         SET Result = Result + Temp
6.         SET Temp = Temp * exponent
7.         SET Temp = Temp / i
8.         increment i
9. if exponent < 0 THEN
10.         SET Result = 1 / Result
11. RETURN Result
end
```

ALGORITHM 4: CalculateLog(value)

```
begin:
1. IF value = 1 or value = 0 THEN
2.          RETURN 0
3. IF value == 10 THEN
4.          RETURN 1
5. SET FLAG = FALSE
6. IF value < 0 THEN
7.          SET FLAG = TRUE
8.          SET value = value * (-1)
9. SET LogValue = 2.302585092994046
10. SET BeforeDecimal = 0
11. SET Result = 0
12. SET Temp = 1
13. WHILE value > 0 14.          SET value = value / 10
15.          increment BeforeDecimal
16. SET FractionalValue = (value - 1)/ (value + 1)
17. FOR i = 0 to 20
18.          SET Result = Result + Temp / (2*i + 1)
19.          SET Temp = Temp * FractionalValue * FractionalValue
20. SET Result = Result * 2
21. SET FinalResult = BeforeDecimal + (Result / 2.302585092994046)
22. if FLAG = TRUE
22.          FinalResult = (-1) * FinalResult
23. RETURN FinalResult
end
```

ALGORITHM 5: CalculatePower(base,exponent)

```
begin:
1. SET e = 2.718281828
2. SET LogOfBase = CalculateLog(base)/ CaluclateLog(e)
3. IF exponent ≥ 0
4.          SET exponent = exponent * LogOfBase
5. ELSE
6.          exponent = exponent * (-LogOfBase)
7. SET Result = CalculateExpTaylorRecursive(0,1,exponent,1)
8. IF exponent < 0 THEN
9.          Result = 1 / Result
10. RETURN Result
end
```

ALGORITHM 6: CalculateExpTaylorRecursive(Result,Temp,Exponent,counter)

```
begin:
1. if Temp < 0
2.         RETURN Result
3. ELSE
4.         RETURN CalculateExpTaylorRecursive(Result + Temp,
               ((Temp * Exponent)/counter),Exponent,counter++)
end
```

3.3 Technical Reasons

The upper sections stated two subordinate functions which were used in the solution of the function a^{b^x}

1. It used Taylor series to find the exponent value because the domain of the function is set of real numbers which include decimal point numbers, because it is easier to handle integers but the point value power calculations can be effectively done by Taylor series.
2. The implementation also used odd Taylor series to fine the log value of any number. Without Taylor Series we had to consider all the elements of numbers and construct a small database of log values.
3. The two approaches stated are Recursive and Iterative where recursive approach is seems the logical choice but the consideration of java stack which is holds recursive calls is limited and for large values will run out of memory. So we use iterative approach which lets the JVM automatically handle the memory.
4. Also the domain is -∞ to +∞ so the only and which can be handled by iterative approach.

Taylor series approach
Advantages

1. Easy calculation of complex mathematical functions
2. Goes beyond solving normal arithmetic, it makes computing easy of large functions.

Disadvantages

1. Very time consuming to calculate the values.
2. Recognize patterns to solve complex mathematical equations

1. Easy calculation of complex mathematical functions
2. Goes beyond solving normal arithmetic, it makes computing easy of large functions.

Recursion Approach
Advantages

1. Shorter Code, easier to write,read and debug
2. Advantageous in Parallel programming
3. Does not change the value of variable once its's declared

Disadvantage

1. Trivial Base Case
2. Involves lots of variable declaration and usage of memory
3. Stack which holds recursive call values can fill up fast

Iterative Approach

Advantage

1. Easier to handle and debug and no Trivial Base Case
2. Very less limitation on the code

Disadvantage

1. Large Code, Have to satisfy the conditions of the loops
2. Involves lots of variable declaration and usage of memory
3. Changes the value once it is declared