SOEN 6011 SEP
Problem 3
Function 6 : $B(x,y)$ Beta Function

Mahavir Patel
40198619
Date: 05/07/2022

1 Algorithm 1 - Tail Recursive Factorial Function

The gamma function is utilised by the beta function to calculate positive integers. The factorial of the number makes it simple to determine the Gamma Value of a positive integer.

$$B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)} \text{ where } \Gamma x = (x-1)!$$

1.1 Advantages

- The algorithm Gives the accurate answers for the positive integers.
- The execution speed of the tail recursive function compared to recursive function is very fast and memory efficient.
- This algorithm makes it simple to calculate the integer's beta values.

1.2 disadvantages

- Can be used for only Positive numbers.
- Only calculate the Beta Values for the integer
- It does not Consider decimal numbers.

1.3 Why Tail Recursive Function ?

If a function concludes by returning the result of the recursive call, it is considered tail-recursive. It is a waste of memory to keep the caller's frame on the stack after the recursive call returns its value because nothing else has to be done. Therefore, the current frame can be used for the call rather than allocating a new one. And for the small value of the integer sometimes recursive factorial faction can cause StackOverFlow error that's why it's better to use the tail recursive function.

1.4 Algorithm

Algorithm 1 Calculate Beta Function using factorial

Require: value: $x > 0$ & $y > 0$ ▷ where $x, y \in Z^+$

Ensure: $result = Beta(x, y)$

```
procedure BetaFunction(x, y)
    value1 ← CalculateGamma(x)
    value2 ← CalculateGamma(y)
    value3 ← CalculateGamma(x + y)
    betaValue ← (value1*value2) / value3
    return betaValue                                  ▷ It returns the beta value
end procedure

procedure GammaFunction(value)
    value ← value − 1
    return Factorial(value)                           ▷ It returns the gamma value
end procedure

procedure Factorial(value)
    return FactorialTailRecursive(value, 1)           ▷ It returns the factorial of the number
end procedure

procedure FactorialTailRecursive(value, n)
    if value = 0 then
        return n                                      ▷ Return factorial of the number
    else
        return FactorialTailRecursive(value − 1, value ∗ n )   ▷ tail recursive call to function
    end if
end procedure

result ← CalculateBeta(x, y)                          ▷ Final result of Beta(x, y)
```

2 Algorithm 2 - Stirling's approximation

The gamma Stirling's approximation, also referred to as Stirling's formula is a mathematical approximation for factorials for decimal values and can be used to construct the beta function for decimal numbers. Since it is a good approximation, it delivers correct results even for small values of n.

$$B(x,y) = \frac{\Gamma x \Gamma y}{\Gamma(x+y)} \text{ where, } \Gamma n = \sqrt{2\pi n} \cdot (\tfrac{n}{e})^n$$

2.1 Advantages

- The algorithm Gives the answers for the positive real numbers.
- Most values that are available can be computed by the algorithm.
- It is possible to apply this approximation approach in place of the integration function to reduce the complexity of the code

2.2 disadvantages

- The algorithm is unable to produce reliable answers.
- Although the complexity is decreased, the earlier algorithm is still substantially more difficult.
- The code cannot be easily debugged.
- The mismatch between the necessary and actual answers is much different for smaller values. However, the difference gets smaller as the size of the numbers grows.

2.3 Why to use Stirling's Approximation ?

To calculate the Beta value of the decimal number, it's hard to compute using the gamma integral function. However, Stirlings' approximation can be applied to reduce the complexity. Even for small numbers, this is a good approximation procedure that yields accurate answers. The best outcome is always obtained through approximation. The comparison of Stirling's approximation and the factorial is provided below.

2.4 Algorithm

Algorithm 2 Calculate Beta Function using Stirlings's approximation

Require: value: $x > 0$ & $y > 0$ ▷ where $x, y \in R^+$
Ensure: $result = Beta(x, y)$

procedure CALCULATEBETA(x, y)
  $value1 \leftarrow$ CALCULATEGAMMA(x)
  $value2 \leftarrow$ CALCULATEGAMMA(y)
  $value3 \leftarrow$ CALCULATEGAMMA$(x + y)$
  $beta \leftarrow \frac{value1 * value2}{value3}$
  return $beta$ ▷ It returns the beta value
end procedure

procedure CALCULATEGAMMA$(value)$
  $firstPart \leftarrow 2 \cdot \pi \cdot value$
  $secondPart \leftarrow (\frac{value}{e})$
  $gamma \leftarrow$ CALCULATEPOWER$(firstPart, \frac{1}{2})$CALCULATEPOWER$(secondPart, value)$
  return $gamma$ ▷ It returns the gamma value
end procedure

procedure CALCULATEPOWER$(value1,value2)$
  $power \leftarrow math.power(value1, value2)$
  return $power$ ▷ It returns the base to the power
end procedure

procedure CALCULATELOG$(value)$
  $answer \leftarrow 0$
  $base \leftarrow \frac{value-1}{value+1}$
  for $i \leftarrow 1, 125$ **do**
    $exponent \leftarrow 2 * i - 1$
    $answer \leftarrow answer + \frac{1}{exponent} *$ CALCULATEPOWER$(base, exponent)$
  end for
  return $2 * answer$ ▷ It returns the value of Log n
end procedure

$result \leftarrow$ CALCULATEBETA(x, y) ▷ Final result of $Beta(x, y)$

Algorithm 3 Calculate the power(x,y)

Require: value: x & y ▷ where $x, y \in R$

Ensure: $result = Power(x, y)$

```
procedure CalculatePower(base,exponent)
    Convert the exponent into String
    exponentArray ← Split the exponent into integer and fractional part
    if exponetArray[1] > 0 then
        return calculateFractionPower(base,exponent)
    end if
    if exponent < 0 then
        base ← 1/base
        exponent ← (−1) · exponent
    end if
    if exponent ← 0 then
        return 1
    end if
    if exponent%2 ← 0 then
        base ← base ∗ base
        exponent ← exponent/2
        return calculatePower(base,exponent)
    else
        exponent ← (exponent−1)/2
        return base * calculatePower(base ∗ base, exponent)
    end if
end procedure

procedure CalculateFractionPower(base,exponent)
    answer ← 0
    logvalue ← 0
    if exponet ← 0 then
        return 1
    end if
    if base < 0 then
        logvalue ← calculateLog(base ∗ (−1))
    else
        logvalue ← calculateLog(base)
    end if
    if exponent ← 0 ∧ exponent > 0 then
        return answer
    end if
    for i ← 0, 125 do
        numerator ← calculatePower(exponent ∗ logvalue, i)
        denominator ← Factorial(i)
        answer ← answer + numerator/denominator
    end for
    if base < 0 ∧ exponent%2 ≠ 0 then
        return answer ∗ (−1)
    else
        return answer
    end if
end procedure
```

References

[1] Beta Function WikiPedia
https://en.wikipedia.org/wiki/Beta_function

[2] Stirling's Approximation
https://en.wikipedia.org/wiki/Stirling%27s_approximation

[3] Tail Recursion
https://www.geeksforgeeks.org/tail-recursion/