**Nome: João Pedro Silveira e Silva**

**Matrícula: 00303397**

**Questão 1: (2,5 pontos)** Prove que a união de uma quantidade infinita, porém contável de conjuntos contáveis produz um conjunto que é contável. Mais formalmente, demonstre que se A0, A1, A2, A3, A4 são todos conjuntos contáveis, então o conjunto ∪i∈N Ai é contável.

**Resolução:**

Dado Ai conjuntos contáveis, então há uma função injetora que mapeia cada Ai no conjuntos dos números naturais.

Sabendo da seguinte propriedade da decomposição em fatores primos:

* Todo o número natural **maior do que 1** pode ser escrito como um produto onde todos os fatores são primos. Retirando o número **1** do conjunto dos primos, esta fatoração se torna única.

E também do fato de que o Teorema de Euclides garante a existência de uma infinidade de números primos.

Podemos mapear os i conjuntos contáveis com suas i funções sobrejetoras **Fi** da seguinte forma:

Dados i conjuntos **Ai com n elementos, todos contáveis com funções injetoras Fi que mapeiam cada elementos de cada conjunto A nos naturais**. Dado um conjunto **P contendo todos os infinitos números primos, excluindo o 1**. E dado que **o número de conjuntos mapeados A é contável** podendo ser estabelecida uma relação entre cada número natural, de 0 a infinito.

Primeiro definimos a operação P(n) que retornará o enésimo número primo em ordem crescente. Agora definimos uma função injetora que mapeará todos os elementos dos conjuntos Ai nos números naturais.

**f(x) = P(i)^(Fi(x))**, onde x é um valor dado de um conjunto Ai, i é um número dado ao conjunto Ai, P(i) é o enésimo número primo baseado no número dado ao conjunto Ai e Fi é a função injetora que mapeia cada elemento de um conjunto Ai nos naturais. Assim cada elemento de cada conjunto irá gerar um número natural diferente baseado nos números fatoráveis apenas pelo enésimo número primo.

**Questão 2: (2,5 pontos)** Vimos em sala a famosa codificação de Godel para pares de naturais, a saber a codificação **(x, y) → (2^x)\*(3^y)**. Nesta questão, vamos introduzir uma nova codificação de pares de naturais chamada de codificação por entrelaçamento de dígitos. Sejam x e y naturais quaisquer. Suponha que dk^x \* dk-1^x . . . d1^x \* d0^x e dk^y \* dk-1^y . . . d1^y \* d0^y sejam as respectivas representações dos números x e y na base 10, onde di^x e di^y, para todo i ∈ {0, . . . , k}, são os (i + 1)-ésimos dígitos decimais de x e de y, respectivamente, da direita para esquerda (completando, se necessário, o menor dos números com 0’s à esquerda para igualar as quantidades de dígitos). A codificação por entrelaçamento de dígitos do par (x, y) é definida como descrito abaixo:

(x, y) = (dk^x \* dk-1^x \* . . . \* d1^x \* d0^x, dk^y \* dk-1^y \* . . . \* d1^y \* d0^y) →

dk^y \* dk^x \* dk-1^y \* dk-1^x \* . . . \* d1^y \* d1^x \* d0^y \* d0^x

Seguem alguns exemplos:

• (8, 19) = (08, 19) → 1098

• (19, 8) = (19, 08) → 0189 = 189

• (25, 13) → 1235

• (0, 0 → 00 = 0

• (99, 9999) = (0099, 9999) → 90909999

Implemente para a máquina NORMA uma sub-rotina com a sintaxe

**C := pair\_entrelaça(A, B)**

que receba de entrada as componentes do par (x, y), armazenadas respectivamente nos registradores A e B, e “retorne” o valor da codificação por entrelaçamento de dígitos do par (x, y) no registrador C preservando os valores iniciais contidos em A e B. Explique como seu código funciona e indique claramente nele os registradores auxiliares. A implementação deve ser feita em uma das estruturações vistas em aula (monolítica, iterativa ou recursiva; consulte os slides para mais detalhes). Para facilitar, você pode utilizar livremente qualquer uma das sub-rotinas que foram apresentadas em sala nos slides. Entretanto, quaisquer sub-rotinas usadas no seu código que não se encontram definidas nos slides deverão ser explicitamente implementadas.

**Exemplo de teste para o código da resolução no simulador de norma (**<http://www.inf.ufrgs.br/~rma/simuladores/norma.html>)**:**

**Dado o par (2,11) a resposta deverá ser 1012. Para dar a entrada na máquina é utilizada a codificação vista em aula. Então para (2,11) entre com (2^2\*(2\*11 + 1) – 1) que é igual a 91.**

**A saída em Y deverá ser 1012. Evite casos de teste que gerem uma entrada muito grande, como (8, 19) que irá gerar 9983.**

**Seguem link de vídeos que fiz com o código rodando nas resoluções de teste passadas utilizando “força bruta”, sem codificação, inserindo diretamente no código, para a entrada de alguns pares.**

**Código da resolução: (Feito para o simulador de Norma)**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Zera o registrador recebido por parâmetro

//

// Variáveis:

// Zera\_A: Armazena o valor de entrada e o valor de saída

//---------------------------------------------------------------------

operation zera(Zera\_A){

1: if zero Zera\_A then goto 0 else goto 2

2: do sub Zera\_A 999999999999 goto 1

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Soma Soma\_A com Soma\_B, salvando o resultado em Soma\_A

//

// Variáveis:

// Soma\_A: Armazena o valor de entrada e saída

// Soma\_B: Armazena o valor de entrada

// Soma\_C: Armazena o valor de C usado para manter o valor de Soma\_B

//---------------------------------------------------------------------

operation soma(Soma\_A,Soma\_B){

1: if zero Soma\_B then goto 0 else goto 2

2: if zero Soma\_B then goto 6 else goto 3

3: do dec Soma\_B goto 4

4: do inc Soma\_A goto 5

5: do inc Soma\_C goto 2

6: if zero Soma\_C then goto 0 else goto 7

7: do dec Soma\_C goto 8

8: do inc Soma\_B goto 6

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Soma Somad\_A com Somad\_B, salvando o resultado em Somad\_A e

// destruindo Somad\_B

//

// Variáveis:

// Somad\_A: Armazena o valor de entrada e saída

// Somad\_B: Armazena o valor de entrada

//---------------------------------------------------------------------

operation soma\_destrutiva(Somad\_A,Somad\_B){

1: if zero Somad\_B then goto 0 else goto 2

2: do dec Somad\_B goto 3

3: do inc Somad\_A goto 1

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Subtrai SubtraiS\_A com SubtraiS\_B, salvando o resultado em

// SubtraiS\_A

//

// Variáveis:

// SubtraiS\_A: Armazena o valor de entrada e saída

// SubtraiS\_B: Armazena o valor de entrada, preserva este valor

// SubtraiS\_C: Armazena o valor de B para evitar destruição

//---------------------------------------------------------------------e

operation subtracao\_parcial\_destrutiva(SubtraiS\_A,SubtraiS\_B){

1: do zera(SubtraiS\_C) goto 2

2: do soma(SubtraiS\_C,SubtraiS\_B) goto 3

3: if zero SubtraiS\_A then goto 0 else goto 4

4: if zero SubtraiS\_C then goto 0 else goto 5

5: do dec SubtraiS\_C goto 6

6: do dec SubtraiS\_A goto 3

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Subtrai Subtrai\_A com Subtrai\_B, salvando o resultado em Subtrai\_A

//

// Variáveis:

// Subtrai\_A: Armazena o valor de entrada

// Subtrai\_B: Armazena o valor de entrada

// Subtrai\_C: Armazena o valor de Subtrai\_C usado para manter Subtrai\_B

// Subtrai\_R: Armazena o valor de saída

// de Subtrai\_B

//---------------------------------------------------------------------e

operation subtrai(Subtrai\_A,Subtrai\_B,Subtrai\_R){

1: do zera(Subtrai\_R) goto 2

2: do zera(Subtrai\_C) goto 3

3: do soma(Subtrai\_R,Subtrai\_A) goto 4

4: if zero Subtrai\_R then goto 9 else goto 5

5: if zero Subtrai\_B then goto 9 else goto 6

6: do dec Subtrai\_B goto 7

7: do dec Subtrai\_R goto 8

8: do inc Subtrai\_C goto 4

9: if zero Subtrai\_C then goto 0 else goto 10

10: do dec Subtrai\_C goto 11

11: do inc Subtrai\_B goto 9

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Dado dois registradores, compara e retorna se o primeiro é menor

// Variáveis:

// Menor\_A: Armazena o valor de entrada

// Menor\_B: Armazena o valor de entrada

// Menor\_C: Armazena o resultado da subtração

//---------------------------------------------------------------------

test eh\_menor(Menor\_A,Menor\_B){

1: do subtrai(Menor\_A,Menor\_B,Menor\_C) goto 2

2: if zero Menor\_C then goto 3 else goto false

3: do subtrai(Menor\_B,Menor\_A,Menor\_C) goto 4

4: if zero Menor\_C then goto false else goto true

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Descrição: Retorna o maior entre dois números

//

// Variáveis:

// MAX\_A: Entrada A

// MAX\_B: Entrada B

// MAX\_R: Saída, resultado

//---------------------------------------------------------------------

operation max(MAX\_A, MAX\_B, MAX\_R){

1: do zera(MAX\_R) goto 2

2: if eh\_menor(MAX\_A,MAX\_B) then goto 3 else goto 4

3: do soma(MAX\_R, MAX\_B) goto 0

4: do soma(MAX\_R, MAX\_A) goto 0

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Descrição: Dado duas entradas, subtrai sucessivamente até antes de

// zero e retorna o número de subtrações

//

// Variáveis:

// SUBS\_A: Entrada, destruida

// SUBS\_B: Entrada, número a se subtrair de A até SUBS\_A < SUBS\_B

// SUBS\_C: Saída, contagem de subtrações

//---------------------------------------------------------------------

operation sub\_sucessiva(SUBS\_A,SUBS\_B,SUBS\_C){

1: if zero SUBS\_B then goto 0 else goto 2

2: if zero SUBS\_A then goto 0 else goto 3

3: do zera(SUBS\_C) goto 4

4: if eh\_menor(SUBS\_A,SUBS\_B) then goto 0 else goto 5

5: do subtracao\_parcial\_destrutiva(SUBS\_A,SUBS\_B) goto 6

6: do inc SUBS\_C goto 4

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Divide um número por dois e retorna o quociente e o resto

//

// Variaveis:

// DP2\_A: Armazena o valor de entrada a ser dividido

// DP2\_Div: Armazena o dividendo de entrada A para cálculos

// DP2\_Qo: Armazena o quociente resultante da divisão

// DP2\_Re: Armazena o resto resultante da divisão

//---------------------------------------------------------------------

operation divide\_por\_dois(DP2\_A,DP2\_Qo,DP2\_Re){

//Zera valores iniciais

1: do zera(DP2\_Div) goto 2

2: do zera(DP2\_Qo) goto 3

3: do zera(DP2\_Re) goto 4

//Salva A em DP2\_Div

4: do soma(DP2\_Div,DP2\_A) goto 5

//Caso DP2\_Div== 0 => Fim

5: if zero DP2\_Div then goto 0 else goto 6

6: do dec DP2\_Div goto 7

//Caso DP2\_Div== 1 => DP2\_Re

7: if zero DP2\_Div then goto 8 else goto 9

//Salva o DP2\_Re

8: do inc DP2\_Re goto 0

//Soma DP2\_Qo e continua subtração sucessiva

9: do dec DP2\_Div goto 10

10: do inc DP2\_Qo goto 5

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Realiza a divisão sucessiva de um número por 2 até alcançar um resultado

// ímpar

//

// Variáveis:

// DP2AI\_A: Armazena o valor de entrada a ser dividido

// DP2AI\_TA: Armazena o número que esta sendo dividido temporariamente

// DP2AI\_Re: Armazena o resto da divisão

// DP2AI\_Cd: Armazena o contador de divisão por 2 com resultado par

// DP2AI\_Ri: Armazena o último resto e por fim o resto ímpar

// DP2AI\_Qo: Armazena o resultado de cada divisão

//---------------------------------------------------------------------

operation dividir\_por\_2\_ate\_impar(DP2AI\_A,DP2AI\_Qo,DP2AI\_Ri,DP2AI\_Cd){

//Zera valores iniciais

1: do zera(DP2AI\_TA) goto 2

2: do zera(DP2AI\_Re) goto 3

3: do zera(DP2AI\_Cd) goto 4

//Salva A em DP2AI\_TA

4: do soma(DP2AI\_TA, DP2AI\_A) goto 5

//Inicia looping zerando o último resto e inicializando com DP2AI\_TA

5: do zera(DP2AI\_Ri) goto 6

6: do soma(DP2AI\_Ri, DP2AI\_TA) goto 7

//Realiza a divisão

7: do divide\_por\_dois(DP2AI\_TA,DP2AI\_Qo,DP2AI\_Re) goto 8

//Caso o resto seja 0, então continua as divisões sucessivas com o resultado

8: if zero DP2AI\_Re then goto 9 else goto 0

//Salva o resultado da divisão em DP2AI\_TA para nova divisão

9: do zera(DP2AI\_TA) goto 10

10: do soma(DP2AI\_TA, DP2AI\_Qo) goto 11

//Incrementa o contador de divisões com resultado par

11: do inc DP2AI\_Cd goto 5

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Dada uma entrada, calcula o valor de Xo e Yo baseado na equação:

// X := (2^Xo) \* (2\*Yo + 1) - 1

//

// Variaveis:

// CXOEYO\_X: armazena o valor de entrada, X

// CXOEYO\_Xo: armazena o resultado de Xo

// CXOEYO\_Yo: armazena o resultado de Yo

// CXOEYO\_Ri: armazena o resto ímpar da operação de divisões sucessivas

// CXOEYO\_Re: armazena o resto da divisão da última equação

//---------------------------------------------------------------------

operation calcular\_par\_entrada(CXOEYO\_X,CXOEYO\_Xo,CXOEYO\_Yo){

//Realiza a subtração inicial (-1) da equação em X

1:do inc CXOEYO\_X goto 2

//Divide X por 2 sucessivamente até

//alcançar um resultado ímpar (resto = 1)

//Ao encontrar este resultado, Xo irá ser o

//número de vezes que se dividiu

//x para chegar a este resultado.

2:do dividir\_por\_2\_ate\_impar(CXOEYO\_X,CXOEYO\_Yo,CXOEYO\_Ri,CXOEYO\_Xo) goto 3

//Decrementa o resultado ímpar para dividir por 2

//Esta operação se refere ao cálculo:

// 2\*Yo + 1 = Resto ímpar

3:do dec CXOEYO\_Ri goto 4

4:do divide\_por\_dois(CXOEYO\_Ri,CXOEYO\_Yo,CXOEYO\_Re) goto 0

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Multiplica a entrada por 10 e salva o resultado na mesma

//

// Variaveis:

// MLT\_A: Entrada e Saída, multiplicando

//---------------------------------------------------------------------

operation multiplica10(MLT\_A){

//Zera o contador

1: do zera(MLT\_C) goto 2

//Soma A ao Contador

2: do soma(MLT\_C,MLT\_A) goto 3

3: do zera(MLT\_A) goto 4

//Enquando C não for Zero soma 10 em A

4: if zero MLT\_C then goto 0 else goto 5

5: do add MLT\_A 10 goto 6

6: do dec MLT\_C goto 4

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Descrição: Retorna o valor de 10^DEZNA\_X

//

// Variáveis:

// DEZNA\_X: Entrada

// DEZNA\_R: Saída, resultado

// DEZNA\_X\_TEMP: Temporária, armazena o valor de DEZNA\_X

//---------------------------------------------------------------------

operation dez\_elevado\_a\_x(DEZNA\_X,DEZNA\_R){

1: do zera(DEZNA\_X\_TEMP) goto 3

3: do soma(DEZNA\_X\_TEMP, DEZNA\_X) goto 4

4: do zera(DEZNA\_R) goto 5

5: do inc DEZNA\_R goto 6

6: if zero DEZNA\_X\_TEMP then goto 0 else goto 7

7: do dec DEZNA\_X\_TEMP goto 8

8: do multiplica10(DEZNA\_R) goto 6

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Descrição: Conta quantos dígitos um número possui

//

// Variáveis:

// CONDIG\_X: entrada

// CONDIG\_COUNT: saída

// CONDIG\_X\_TEMP: variável para armazenar o valor de CONDIG\_X

//---------------------------------------------------------------------

operation contador\_de\_digitos(CONDIG\_X,CONDIG\_COUNT){

1:do zera(CONDIG\_COUNT) goto 2

2:if zero CONDIG\_X then goto 0 else goto 3

3:do inc CONDIG\_COUNT goto 4

4:do zera(CONDIG\_TEMP) goto 5

5:do add CONDIG\_TEMP 9 goto 6

6:do zera(CONDIG\_X\_TEMP) goto 7

7:do soma(CONDIG\_X\_TEMP,CONDIG\_X) goto 8

8:do subtracao\_parcial\_destrutiva(CONDIG\_X\_TEMP,CONDIG\_TEMP) goto 9

9:if zero CONDIG\_X\_TEMP then goto 0 else goto 10

10:do inc CONDIG\_COUNT goto 11

11:do multiplica10(CONDIG\_TEMP) goto 8

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Descrição: Pega o valor do digito mais significativo dado a sua

// posicao. Tolera tentativas de pegar digito maior que o msd,

// retornando ZERO.

// Variáveis:

// PEGENE\_X: Entrada, número para pegar o digito

// PEGENE\_DC: Entrada, posição do dígito msd

// PEGENE\_A: Saída, valor do dígito msd

// PEGENE\_P10: Temporária, potencia de 10 utilizada no cálculo

//---------------------------------------------------------------------

operation pegar\_enesimo\_digito\_se\_msd(PEGENE\_X,PEGENE\_DC,PEGENE\_A){

1:do zera(PEGENE\_A) goto 2

2:if zero PEGENE\_X then goto 0 else goto 3

3:if zero PEGENE\_DC then goto 0 else goto 4

4:do zera(PEGENE\_P10) goto 5

5:do dec PEGENE\_DC goto 6

6:do dez\_elevado\_a\_x(PEGENE\_DC,PEGENE\_P10) goto 7

7:do sub\_sucessiva(PEGENE\_X,PEGENE\_P10,PEGENE\_A) goto 8

8:do inc PEGENE\_DC goto 0

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

main{

//Zera as variáveis da main

1: do zera(Xo) goto 2

2: do zera(Yo) goto 3

3: do zera(C\_Xo) goto 4

4: do zera(C\_Yo) goto 5

5: do zera(Contador) goto 6

6: do zera(Contador\_Digito) goto 7

7: do zera(Contador\_Digito\_Temp) goto 8

8: do zera(A) goto 9

9: do zera(B) goto 20

//Calcula a entrada baseado na codificação

// -> 2^y\*(2\*x + 1) - 1

20: do calcular\_par\_entrada(X,A,B) goto 21

21: do soma(Xo,A) goto 22

22: do soma(Yo,B) goto 30

//Busca o número dígitos do maior valor, (12,9) = 2

30: do contador\_de\_digitos(Xo, C\_Xo) goto 31

31: do contador\_de\_digitos(Yo, C\_Yo) goto 32

32: do max(C\_Xo,C\_Yo,Contador) goto 33

//Verifica se há um número de dígitos > 0

33: if zero Contador then goto 0 else goto 34

//Cria um contador para os dígitos do resultado final

//Exemplo: A = 12 e B = 08, o resultado final terá 4 dígitos

34: do soma(Contador\_Digito,Contador) goto 35

35: do soma(Contador\_Digito,Contador) goto 36

//Subtrai 1 do resultado devido a lógica do sistema, para pegar o

//quarto digito é necessário usar o número 3, inicia por 0.

36: do dec Contador\_Digito goto 37

//Pega o enésimo digito de Y(B)

37: do pegar\_enesimo\_digito\_se\_msd(Yo,Contador,Digito) goto 38

//Verifica se não é zero, caso for zero pula está etapa

38: if zero Digito then goto 44 else goto 39

//Multiplica o enésimo digito por sua posição no valor final

//Ex: se o quarto digito for 1, multiplica até obter o valor 1000

39: do soma(Contador\_Digito\_Temp, Contador\_Digito) goto 40

40: if zero Contador\_Digito\_Temp then goto 43 else goto 41

41: do multiplica10(Digito) goto 42

//Decrementa os contadores e transfere o valor do Digito para o Y final

42: do dec Contador\_Digito\_Temp goto 40

43: do soma\_destrutiva(Y,Digito) goto 44

44: do dec Contador\_Digito goto 45

//Realiza o mesmo processo para X(A)

45: do pegar\_enesimo\_digito\_se\_msd(Xo,Contador,Digito) goto 46

46: if zero Digito then goto 52 else goto 47

47: do soma(Contador\_Digito\_Temp, Contador\_Digito) goto 48

48: if zero Contador\_Digito\_Temp then goto 51 else goto 49

49: do multiplica10(Digito) goto 50

50: do dec Contador\_Digito\_Temp goto 48

51: do soma\_destrutiva(Y,Digito) goto 52

52: do dec Contador\_Digito goto 53

//Decrementa o contador principal e verifica se ainda há mais dígitos para

//serem contados, se não finaliza

53: do dec Contador goto 54

54: if zero Contador then goto 0 else goto 37

}

**Questão 3:** (2,5 pontos) Para cada um dos termos lambda mostrados abaixo, faça o que é pedido nos itens (a)–(e) a seguir:

• (λx.xy) (x λy.yx) (λyz.zy)

• (λz.z (λy.yzx) y) (λxz.(λy.zxy) x)

(a) (0,25 ponto) Explicite os parênteses e os λ’s ausentes no termo lambda em questão.

•(((λx.(xy)) (x λy.(yx))) (λyλz.(zy)))

• ((λz.((z (λy.((yz)x))) y)) (λxλz.((λy.((zx)y)) x)))

(b) (0,25 ponto) Identifique todas as ocorrências livres e todas as ocorrências ligadas das variáveis presentes no termo lambda em questão.

Ocorrências livres em vermelho e ligadas em verde

• (λx.xy) (x λy.yx) (λyz.zy)

• (λz.z (λy.yzx) y) (λxz.(λy.zxy) x)

(c) (0,5 ponto) Identifique todos os redexes presentes no termo lambda em questão.

• (λx.xy) (x λy.yx) (λyz.zy) => 1 Redex

(λx.xy) (x (λy.yx)) = P

(λyz.zy) = Q

• (λz.z (λy.yzx) y) (λxz.(λy.zxy) x) => 2 Redex

1: (λz.z (λy.yzx) y) = P

(λxz.( λy.zxy) x) = Q

2: (λy.zxy) = P

x = Q

(d) (0,75 ponto) O termo lambda em questão possui forma normal? Se sim, mostre uma sequência de β-reduções do referido termo lambda até sua forma normal e, em cada uma das reduções da referida sequência, explicite o redex que está sendo contraído. Se não, justifique.

• (λx.xy) (x λy.yx) (λyz.zy) => Possui!

Dado (λx.xy) (x (λy.yx)) = P e (λyz.zy) = Q sua forma normal é:

xy (λyz.zy)

• (λz.z (λy.yzx) y) (λxz.(λy.zxy) x) => Possui!

Dado (λz.z (λy.yzx) y) = P e (λxz.( λy.zxy) x) = Q reduzindo temos:

(λxz.(λy.zxy) x) (λy.yzx) y

Dado que (λxz.(λy.zxy) x) (λy.yzx) = P e y = Q reduzindo temos:

(λy.zxy) x y

Dado que (λy.zxy) x = P e y = Q reduzindo temos:

zxy y

E chegamos a forma normal.

(e) (0,75 ponto) Usando exclusivamente a estratégia de avaliação aplicativa, mostre (se possível) uma sequência de três β-reduções partindo do termo lambda em questão e, em cada uma das reduções da referida sequência, explicite o redex que está sendo contraído.

• (λx.xy) (x λy.yx) (λyz.zy)

Dado (λx.xy) = P e (x λy.yx) = Q temos:

((x λy.yx) y) (λyz.zy) => Forma normal.

• (λz.z (λy.yzx) y) (λxz.(λy.zxy) x)

Dado (λy.zxy) = P e x = Q temos:

(λz.z (λy.yzx) y) (λxz.(zxx))

Dado (λz.z (λy.yzx) y) = P e (λxz.(zxx)) = Q temos:

(λz. (λxz.(zxx)) (λy.yzx) y)

Dado (λxz.(zxx)) = P e (λy.yzx) = Q temos:

zxx y => Forma normal.

**Questão 4:** (2,5 pontos) Escreva, sem usar nenhum combinador de ponto fixo, um termo lambda chamado fibonacci que compute a função Cn → CFn. Em outras palavras, o termo lambda fibonacci, ao ser aplicado a um numeral de Church Cn, retorna o n-ésimo termo Fn da sequência de Fibonacci 0, 1, 1, 2, 3, 5, 8, . . . .

**Resolução:**

let

true = \a b. a ;

false = \a b. b ;

if = \c a b. c a b ;

succ = \n p q. p (n p q) ;

add = \m n. m succ n ;

isZero = \n . n (\x.false) true ;

pair = \m n b. b m n ;

fst = \p. p true ;

snd = \p. p false ;

shiftInc = \p. pair (snd p) (succ (snd p)) ;

pred = \n. fst (n shiftInc (pair 0 0)) ;

Y = \f.(\x.f(x x))(\x.f(x x)) ;

fibonacci\_rec = \M.\n.if (isZero n) 0 (if (isZero(pred n)) 1 (add (M (pred n)) (M (pred (pred n)))));

fibonacci = Y fibonacci\_rec;

in

(fibonacci 5)

**Infelizmente não consegui alcançar sozinho uma forma de realizar o Fibonacci sem ponto fixo. Tentei utilizando tripla e operações de add, mult e exp porém não obtive sucesso em nenhuma tentativa.**