Universidade Federal de Santa Catarina

Departamento de Engenharia Elétrica e Eletrônica

Centro Tecnológico

PROJETO FINAL:

APLICAÇÃO DE MÓDULOS PSEUDO-MÓDULOS PARA EXPONENCIAÇÃO MODULAR

Disciplina: Tópico Avançado em Sistemas Digitais

Alunos: Guilherme Henrique Paggi Daros

Luís Gustavo Piva Machado

Professor: Héctor Pettenghi Roldán

Florianópolis, 19 de Maio de 2021

1 INTRODUÇÃO

Iremos analisar os ganhos que a utilização de pseudo-módulos pode trazer para a realização de operações modulares complexas, como a exponenciação modular.

2 DESENVOLVIMENTO

2.1 Pseudo-módulo para 341

Suponhamos a exponenciação modular $b^a(modm_0)$ tal que $b = 4$, $a = 13$ e $m_0 = 341$.

Decompondo-se o expoente em binário tem-se:

$$\left(4^{2^3} \cdot 4^{2^2} \cdot 4^{2^0}\right)mod_{341}$$

$$(65536)mod_{341} \cdot 256 \cdot 4$$

$$(64 \cdot 256)mod_{341} \cdot 4$$

$$(16 \cdot 4)mod_{341} = 64$$

Fazendo o uso do pseudo-módulo para 341:

Módulo original:

$mod(2^9, 341) = 171$

$m_0 = 2^9 - 171$, onde $k_{(171)} = 010101011$

Recodificando k:

$k_{(171)}$ = 0 1 0 -1 0 -1 0 -1 0 -1

Há 5 ‘1’s na representação

Pseudo-módulo m_1:

$mod(2^{10}, 341) = 1$

$m_1 = 2^{10} - 1$, onde $k_{(1)} = 000000001$

Há 1 ‘1’ na representação

2.2 Pseudo-módulo para 409

Suponhamos a exponenciação modular $b^{a}(mod m_0)$ tal que $b = 4$, $a = 13$ e $m_0 = 409$.

Decompondo-se o expoente em binário tem-se:

$$\left(4^{2^3} \cdot 4^{2^2} \cdot 4^{2^0}\right) mod_{409}$$

$$(65536) mod_{409} \cdot 256 \cdot 4$$

$$(96 \cdot 256) mod_{409} \cdot 4$$

$$(36 \cdot 4) mod_{409} = 144$$

Fazendo o uso do pseudo-módulo para 409:

Módulo original:

$mod(2^9, 409) = 103$

$m_0 = 2^9 - 103$, onde $k_{(103)} = 001100111$

Recodificando k:

$k_{(103)}$ = 0 1 0 -1 0 1 0 0 -1

Há 4 '1's na representação

Pseudo-módulo m_1:

$mod(2^{10}, 409) = 206$

$m_1 = 2^{10} - 206$, onde $k_{(206)} = 011001110$

Recodificando k:

$k_{(206)}$ = 0 1 0 -1 0 1 0 0 -1 0

Há 4 '1's na representação

Pseudo-módulo m_2:

$mod(2^{11}, 409) = 3$

$m_2 = 2^{11} - 3$, onde $k_{(3)} = 000000011$

Há 2 ‘1’s na representação

2.3 Pseudo-módulo para 5461

Suponhamos a exponenciação modular $b^a(modm_0)$ tal que $b = 4$, $a = 13$ e $m_0 = 5461$.

Decompondo-se o expoente em binário tem-se:

$$\left(4^{2^3} \cdot 4^{2^2} \cdot 4^{2^0}\right)mod_{5461}$$

$$(65536)mod_{5461} \cdot 256 \cdot 4$$

$$(4 \cdot 256)mod_{5461} \cdot 4$$

$$(1024 \cdot 4)mod_{5461} = 4096$$

Fazendo o uso do pseudo-módulo para 5461:

Módulo original:

$mod(2^{13}, 5461) = 2731$

$m_0 = 2^{13} - 2731$, onde $k_{(2731)} = 0101010101011$

Recodificando k:

$k_{(2731)}$ = 0 1 0 -1 0 -1 0 -1 0 -1 0 -1 0 -1

Há 7 ‘1’s na representação

Pseudo-módulo m_1:

$mod(2^{14}, 5461) = 1$

$m_1 = 2^{14} - 1$, onde $k_{(1)} = 000000001$

Há 1 ‘1’ na representação

2.4 Pseudo-módulo para 4681

Suponhamos a exponenciação modular $b^a(mod m_0)$ tal que $b = 4$, $a = 13$ e $m_0 = 4681.$

Decompondo-se o expoente em binário tem-se:

$$\left(4^{2^3} \cdot 4^{2^2} \cdot 4^{2^0}\right) mod_{4681}$$

$$(65536) mod_{4681} \cdot 256 \cdot 4$$

$$(2 \cdot 256) mod_{4681} \cdot 4$$

$$(512 \cdot 4) mod_{4681} = 2048$$

Fazendo o uso do pseudo-módulo para 4681:

Módulo original:

$mod(2^{13}, 4681) = 3511$

$m_0 = 2^{13} - 3511$, onde $k_{(3511)} = 0110110110111$

Recodificando k:

$k_{(3511)}$ = 0 1 0 0 -1 0 0 -1 0 0 -1 0 0 -1

Há 5 '1's na representação

Pseudo-módulo m_1:

$mod(2^{14}, 4681) = 2341$

$m_1 = 2^{14} - 2341$, onde $k_{(2341)} = 0100100100101$

Não há ganho recodificando $k_{(2341)}$ pois não há cadeia de '1's

Há 5 '1's na representação

Pseudo-módulo m_2:

$mod(2^{15}, 4681) = 1$

$m_2 = 2^{15} - 1$, onde $k_{(1)} = 000000001$

Há 1 '1' na representação

2.5 Pseudo-módulo para 21845

Suponhamos a exponenciação modular $b^a (mod m_0)$ tal que $b = 4$, $a = 13$ e $m_0 = 21845$.

Decompondo-se o expoente em binário tem-se:

$$\left(4^{2^3} \cdot 4^{2^2} \cdot 4^{2^0}\right) mod_{21845}$$

$$(65536) mod_{21845} \cdot 256 \cdot 4$$

$$(1 \cdot 256) mod_{21845} \cdot 4$$

$$(256 \cdot 4) mod_{21845} = 1024$$

Fazendo o uso do pseudo-módulo para 21845:

Módulo original:

$mod(2^{15}, 21845) = 10923$

$m_0 = 2^{15} - 10923$, onde $k_{(10923)} = 010101010101011$

Recodificando k:

$k_{(10923)}$ = 0 1 0 -1 0 -1 0 -1 0 -1 0 -1 0 -1 0 -1

Há 8 '1's na representação

Pseudo-módulo m_1:

$mod(2^{16}, 21845) = 1$

$m_1 = 2^{16} - 1$, onde $k_{(1)} = 000000001$

Há 1 '1' na representação

2.6 Pseudo-módulo para 6553

Suponhamos a exponenciação modular $b^a (mod m_0)$ tal que $b = 4$, $a = 13$ e $m_0 = 6553$.

Decompondo-se o expoente em binário tem-se:

$$\left(4^{2^3} \cdot 4^{2^2} \cdot 4^{2^0}\right) mod_{6553}$$

$$(65536) mod_{6553} \cdot 256 \cdot 4$$

$$(6 \cdot 256) mod_{6553} \cdot 4$$

$$(1536 \cdot 4) mod_{6553} = 6144$$

Fazendo o uso do pseudo-módulo para 6553:

Módulo original:

$mod(2^{13}, 6553) = 1639$

$m_0 = 2^{13} - 1639$, onde $k_{(1639)} = 011001100111$

Recodificando k:

$k_{(1639)}$ = 0 1 0 -1 0 1 0 -1 0 1 0 0 -1

Há 6 ‘1’s na representação

Pseudo-módulo m_1:

$mod(2^{14}, 6553) = 3278$

$m_1 = 2^{14} - 3278$, onde $k_{(3278)} = 0110011001110$

Recodificando k:

$k_{(3278)}$ = 0 1 0 -1 0 1 0 -1 0 1 0 0 -1 0

Há 6 ‘1’s na representação

Pseudo-módulo m_2:

$mod(2^{15}, 6553) = 3$

$m_2 = 2^{15} - 3$, onde $k_{(3)} = 000000011$

Há 2 ‘1’s na representação

2.7 Pseudo-módulo para 1363

Suponhamos a exponenciação modular $b^a(mod m_0)$ tal que $b = 4$, $a = 13$ e $m_0 = 1363$.

Decompondo-se o expoente em binário tem-se:

$$\left(4^{2^3} \cdot 4^{2^2} \cdot 4^{2^0}\right) mod_{1363}$$

$$(65536) mod_{1363} \cdot 256 \cdot 4$$

$$(112 \cdot 256) mod_{1363} \cdot 4$$

$$(49 \cdot 4) mod_{1363} = 196$$

Fazendo o uso do pseudo-módulo para 1363:

Módulo original:

$mod(2^{11}, 1363) = 685$

$m_0 = 2^{11} - 685$, onde $k_{(685)} = 01010101101$

Recodificando k:

$k_{(685)}$ = 1 0 -1 0 -1 0 -1 0 0 -1 -1

Há 6 ‘1’s na representação

Pseudo-módulo m_1:

$mod(2^{12}, 1363) = 7$

$m_1 = 2^{12} - 7$, onde $k_{(7)} = 000000111$

Recodificando k:

$k_{(7)}$ = 0 0 0 0 0 1 0 0 -1

Há 2 ‘1’s na representação

2.8 Resultados obtidos

Através das operações realizadas, obtêve-se a seguinte tabela de atrasos:

Módulo	n	k	1's em k	Delay
341	9	171	5	3.50 ns
	10	1	1	1.60 ns
409	9	103	4	3.00 ns
	10	206	4	3.10 ns
	11	3	2	2.20 ns
5461	13	2731	7	4.90 ns
	14	1	1	2.00 ns
4681	13	3511	5	3.90 ns
	14	2341	5	4.00 ns
	15	1	1	2.10 ns
21845	15	10923	8	5.60 ns
	16	1	1	2.20 ns
6553	13	1639	6	4.40 ns
	14	3278	6	4.50 ns
	15	3	2	2.60 ns
1363	11	685	6	4.20 ns
	12	7	2	2.30 ns

Tabela 1 – Atraso dos pseudo-módulos

2.9 Aplicação de Pseudo-módulos em números RSA

Iremos analisar teoricamente se a aplicação de Pseudo-módulos para os números RSA-100, RSA-110, RSA-120, RSA-129, RSA-130, RSA-140, RSA-150 e RSA-155.

Para isso foi usado um programa desenvolvido em Python que pudesse fornecer a quantidade de ‘1’s no “k” correspondente a cada um dos módulos e pseudomódulos de cada um dos números RSA. Os trechos utilizados pordem ser visualizados a seguir:

```python
def ModuleCalc(number):
    r = []
    bitLength = len(bin(number)[2:])
    for i in range(10):
        r.append(2**(bitLength + i) - number)
    return r

def onesCounter(x):
    i = 0
    y = "".join(x)
    for j in y:
        if j == "1":
            i+=1
    return i
```

Figura 1 – Descrição das funções para determinação dos módulos/pseudo-módulos e contar ‘1’s em cada “k”.

```python
def FirstOrderBoothRecoding(n):
    n_ = [int(x) for x in list(n + "0")]
    r = []
    for i in range(len(n)):
        x = n_[i+1] - n_[i]
        if x == -1: r.append("-1")
        else:       r.append(str(x))
    return r

def SecondOrderBoothRecoding(x):
    for i in range(len(x) - 1):
        if (f'{x[i]}{x[i+1]}') == "-11":
            x[i] = "0"
            x[i+1] = "-1"
        elif(f'{x[i]}{x[i+1]}') == "1-1":
            x[i] = "0"
            x[i+1] = "1"
    return x

def ThirdOrderBoothRecoding(x):
    for i in range(len(x) - 1):
        if (f'{x[i]}{x[i+1]}') == "11":
            x[i-1] = "1"
            x[i] = "0"
            x[i+1] = "-1"
    return x
```

Figura 2 – Descrição das funções que realizam a codificação de Booth

```python
RSA = {
    'RSA100': 1522605027922533360535618378132637429718068114961380688657908494580122963258952897654000350692006139,
    'RSA110': 35794234179725868774991807832568455403003778024228226193532908190484670252364677411513516111204504060317568667,
    'RSA120': 227010481295437363334259960947493668895875336466084780038173258247009162675779735389791151574049166747880487470296548479,
    'RSA129': 114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541,
    'RSA130': 1807082088687404805951656164405905566278102516769401349170127021450056662540244048387341127590812303371781887966563182013214880557,
    'RSA140': 21290246318258757547497882016271517497806703963277216278233383215381949984056495911366573853021918316783107387995317230889569230873441936471,
    'RSA150': 155089812478348440509606754370011861770654545830995430655466945774312632703463465954363335027577729025391453996787414027003501631772186840890795964683,
    'RSA155': 10941738641570527421809707322040357612003732945449205990913842131476349984288934784717997257891267332497625752899781833797076537244027146743531593354333897
    }
```

Figura 3 – Declaração de cada um dos RSA Numbers.

```python
with open("out.csv","w") as file:

    for Number in RSA:
        file.write(f"{Number}\n")
        for item in ModuleCalc(RSA[Number]):
            x = FirstOrderBoothRecoding(bin(item)[2:])
            y = SecondOrderBoothRecoding(x)
            z = ThirdOrderBoothRecoding(y)
            file.write(str(onesCounter(z)))
            file.write("\n")
```

Figura 4 – Algoritmo que reúne as funções e fornece a quantidade de ‘1’s em cada “k” para cada RSA Number.

Com isso, chegamos aos seguintes resultados:

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA100	m0	108
RSA100	m1	109
RSA100	m2	109
RSA100	m3	109
RSA100	m4	109
RSA100	m5	109
RSA100	m6	109
RSA100	m7	109
RSA100	m8	109
RSA100	m9	109

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA110	m0	119
RSA110	m1	121
RSA110	m2	121
RSA110	m3	121
RSA110	m4	121
RSA110	m5	121
RSA110	m6	121
RSA110	m7	121
RSA110	m8	121
RSA110	m9	121

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA120	m0	128
RSA120	m1	129
RSA120	m2	129
RSA120	m3	129
RSA120	m4	129
RSA120	m5	129
RSA120	m6	129
RSA120	m7	129
RSA120	m8	129
RSA120	m9	129

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA129	m0	153
	m1	154
	m2	154
	m3	154
	m4	154
	m5	154
	m6	154
	m7	154
	m8	154
	m9	154

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA130	m0	144
	m1	145
	m2	145
	m3	145
	m4	145
	m5	145
	m6	145
	m7	145
	m8	145
	m9	145

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA140	m0	164
	m1	166
	m2	166
	m3	166
	m4	166
	m5	166
	m6	166
	m7	166
	m8	166
	m9	166

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA150	m0	152
	m1	154
	m2	154
	m3	154
	m4	154
	m5	154
	m6	154
	m7	154
	m8	154
	m9	154

RSA Number	Módulo/Pseudo-módulo	1's em k
RSA155	m0	162
	m1	164
	m2	164
	m3	164
	m4	164
	m5	164
	m6	164
	m7	164
	m8	164
	m9	164

3 CONCLUSÃO

Com os dados obtidos, concluímos que, para os valores fornecidos inicialmente (341, 409, 5461, 4681, 21845, 6553 e 1363) há ganhos consideráveis no tempo computacional ao utilizar os pseudo-módulos ao invés do módulo inicial m_0, como no caso de 21845 que tem uma redução de tempo computacional de 60,71% na utilização de m_2 em comparação com m_0.

Para os números RSA, no entanto, não obteve-se ganho computacional algum, visto que além de aumentar o expoente "n", o que isoladamente aumenta o tempo computacional, aumenta-se também o valor de '1's em "k", o que também aumenta o tempo computacional.