ESTÁCIO – CEUT

ERICK DONALDSON FONTES DE SOUSA
GABRIEL FONTENELE DE AZEVEDO

TRABALHO FINAL DE ORGANIZAÇÂO DE COMPUTADORES:

Problema das N Rainhas

TERESINA
2017

ERICK DONALDSON FONTES DE SOUSA

GABRIEL FONTENELE DE AZEVEDO

TRABALHO FINAL DE ORGANIZAÇÃO DE COMPUTADORES:

Problema das N Rainhas

Trabalho em Dupla apresentado à disciplina de Organização de Computadores do curso de Bacharelado em Ciência da Computação como requisito parcial para a obtenção de nota.

ORIENTADOR: Prof.ª Dr. Sekeff

TERESINA

2017

PROBLEMA DAS N RAINHAS

Erick Donaldson Fontes de Sousa[1]
Gabriel Fontenele de Azevedo[2]

RESUMO: O problema das n-Rainhas requer um algoritmo de alta complexidade O(n^3). Este trabalho apresenta uma solução recursiva usando backtracking.
PALAVRAS-CHAVE: n-Rainhas, Programação Paralela, Avaliação de desempenho computacional.

1 INTRODUÇÃO

Neste trabalho abordamos o problema baseado na publicação de Max Bezzel das oito rainhas de 1848. Que foi estendido para N rainhas em um tabuleiro N x N.

Como objetivo tivemos quer criar uma solução para uma linguagem de alto nível e converter para assembly MIPS utilizando registradores de pilha a cada chamada, apresentar uma tabela de símbolos e comparar o desempenho dos códigos.

2 FUNDAMENTAÇÃO

Problema: colocar N rainhas em um tabuleiro N x N de xadrez sem que elas se ataquem.

[1] Graduando do curso de Bacharelado em Ciência da Computação da Faculdade Estácio – Ceut.
[2] Graduando do curso de Bacharelado em Ciência da Computação da Faculdade Estácio – Ceut.

3 DESENVOLVIMENTO

Questão 1 - O problema das oito rainhas _e o problema matemático de dispor oito rainhas em um tabuleiro de xadrez de dimensão 8 * 8, de forma que nenhuma das rainhas seja atacada por outra. Para tanto, _e necessário que duas rainhas quaisquer não estejam numa mesma linha, coluna ou diagonal. Este é um caso espeço do Problema das n rainhas, no qual temos n rainhas e um tabuleiro com n x n casas (para qualquer n >= 4).

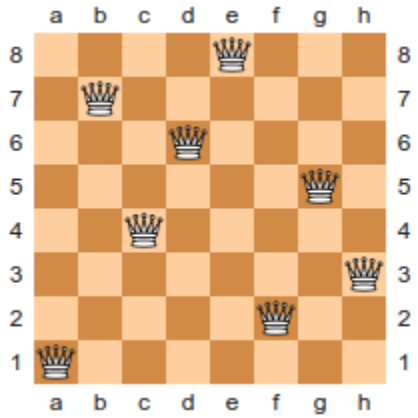


Figure 1: Exemplo de uma das soluções possíveis para o problema das 8 rainhas.

Você deve propor uma solução recursiva, usando *backtracking*, em alguma linguagem de alto nível (Sugiro linguagem C) para qualquer 2 < n < 23. Seu programa deve solicitar o valor de n ao usuário e, assim, recursivamente, tentar dispor n rainhas em um tabuleiro n x n, sem que elas se ataquem.

Questão 2 - Converta seu programa em um código *assembly MIPS* que faça a mesma coisa da questão anterior. Lembre-se que as funções são recursivas. Logo, você dever a usar as instruções jal e jr, além de salvar os dados no registrador de pilha a cada chamada.

Questão 3 - Apresente uma tabela de símbolos para o seu programa em *assembly* da questão anterior, tal qual mostra o livro do *Tannenbaum* no capítulo 7. Explique detalhadamente como você a produziu e todos os problemas de alocação de memória que possam vir a idêntica.

Questão 4 - Ao estudarmos as máquinas multiníveis, sabemos que os códigos em *assembly* devem, a princípio, se mostrar mais eficientes que os códigos escritos em linguagem de alto nível. Isso se deve a alguns fatores, como proximidade da camada de SO e possibilidade de o

desenvolvedor desenvolver algumas estratégias que só são permitidas em códigos dessa camada.

Seria interessante checar se o código da primeira questão (gerado em linguagem C) é mais ou menos performante que o seu gêmeo escrito em *assembly MIPS*. Assim, você deverá rodar os dois códigos (o binário gerado em C e o *assembly MIPS* no *QTSPIM*) e computar, basicamente, 3 variáveis: 1 - consumo de tempo (em milissegundos); 2 - consumo de memória e; 3 - quantidade de interações (chamadas recursivas).

O teste deve ser feito com uma mesma instância do problema (faça n = 15) para os dois algoritmos. Limitamos o n = 15 em razão do algoritmo ter alta complexidade: O (n3).

Ao final, produza um relatório apresentando os resultados e explicando as suas razões. Apresente gráficos plotados com o tempo, consumo de memória e a quantidade de interações.

4 DESCRIÇÃO TECNICA

1 Questão

R: Anexo A

Programa em C

Descrição da Solução:

O código percorre o a coluna a coluna da matriz procurando uma posição sem que conflite com uma das rainhas já colocadas, quando encontra marca no vetor tabuleiro[linha] a posição da coluna (“ANEXO A” linha 63).

Para testar se existe conflito, foi utilizado uma função que percorre todas as linhas anteriores do tabuleiro, verificando se a linha e coluna a serem testadas não estão na mesma coluna (“ANEXO A” linha 48) e se estão na mesma diagonal, usando a heurística de que modulo da diferença entre coluna a ser testada e coluna anterior é igual a modulo da diferença entre linha a ser testada e linha anterior (“ANEXO A” linha 53).

1

2 Questão

R: Anexo B

Programa em Assembly Mips

Descrição da Solução:

Utilizando os métodos e técnicas demonstrados em sala

3 Questão

R:

Tabela de Símbolos

Rótulo	Opcode	1º operando	2º operando	Opcode Hexadecimal	Comprimento da Instrução	Classe da Instrução
main:	-------	-------	-------	-------	0	0
-------	Sw	$t8	($t9)	23	2	0
-------	Move	$a0	$v0	2B	1	2
-------	Jal	-------	rainha	3	5	3
print:	Addi	$sp	4	8	2	8
-------	sw	$a0	0($sp)	23	2	10
-------	Move	$t4	$a0	9	3	12
-------	Lw	$t7	($t9)	2B	4	15
-------	Addi	$t7	1	8	2	19
-------	Sw	$t7	($t9)	23	2	21
-------	Move	$a0	$t7	8	1	23
for1:	Move	$a0	$t0	2B	1	24
-------	Addi	$t0	1	8	2	26
-------	Ble	$t0	$t4	6	2	28
for2:	Move	$a0	$t0	22	1	29
for3:	Mul	$t0	4	0	2	31
-------	Add	$t3	$t5	0	2	32
-------	Lw	$t9	($t3)	23	2	34
-------	Bne	$t9	$t1	5	1	36
imprime:	J	-------	fimfor3	2	5	37
printT:	-------	-------	-------	7	5	42
fimfor3:	Addi	$t1	1	8	2	47
-------	Ble	$t1	$t4	6	2	49
-------	Addi	$t0	1	8	2	51
-------	Ble	$t0	$t4	6	1	53
-------	Lw	$a0	0($sp)	23	2	54
-------	Addi	$sp	4	8	2	56
-------	Jr	$ra	-------	0	5	58

coloca:	-------	-------	-------	-------	7	63
for5:	Bge	$t0	$a1	1	1	70
-------	Mul	$t8	4	0	2	71
-------	Add	$t9	$t8	0	2	73
-------	Lw	$t8	($t9)	23	2	75
-------	Move	$t6	$t8	9	3	77
-------	Sub	$t8	$a2	0	2	80
-------	Sub	$t0	$a1	0	2	82
-------	Bgtz	$t8	saiC1	1	5	84
-------	Mul	$t8	$t9	0	2	89
saiC1:	Bgtz	$t7	saiC2	1	5	91
-------	Mul	$t7	$t9	0	2	96
saiC2:	Beq	$t6	$a2	4	2	98
-------	Beq	$t8	$t7	4	2	100
-------	Addi	$t0	1	8	2	102
-------	J	for	-------	2	5	104
retorna1:	Jr	$ra	-------	0	5	109
retorna2:	Jr	$ra	-------	0	5	114
rainha:	Addi	$sp	-8	8	2	219
-------	Sw	$a2	4($sp)	2B	2	221
-------	Sw	$ra	0($sp)	2B	2	223
for6:	Bgt	$a0	saifor6	1	5	225
-------	Jal	coloca	-------	3	5	230
-------	Beqz	$v1	saiprint1	1	5	235
-------	Mul	$t1	$a1	0	2	240
-------	Add	$t9	$t1	0	2	242
-------	Sw	$a2	($t9)	2B	2	244
-------	Bne	$a1	saiprint2	5	5	246
-------	Jal	print	-------	3	5	251
-------	J	saiprint1	-------	2	5	256
saiprint2:	Addi	$sp	-4	8	2	261
-------	Sw	$a1	0($sp)	2B	2	263

-------	Addi	$a1	1	8	2	265
-------	Jal	rainha	-------	3	5	267
-------	Lw	$a1	0($sp)	23	2	273
-------	Addi	$sp	4	8	2	275
saiprint1:	Addi	$a2	1	8	2	277
-------	J	for6	-------	2	5	279
saifor6:	Lw	$a2	4($sp)	23	2	284
-------	Lw	$ra	0($sp)	23	2	286
-------	Addi	$sp	8	8	2	288
-------	Jr	$ra	-------	0	5	230

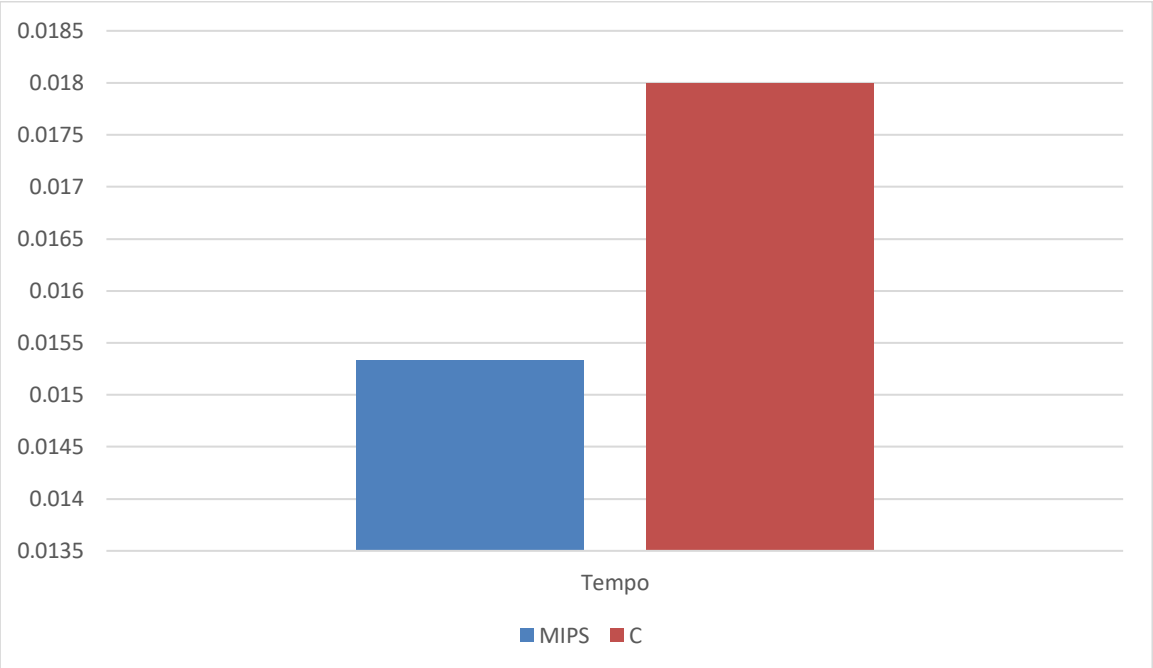
4 Questão

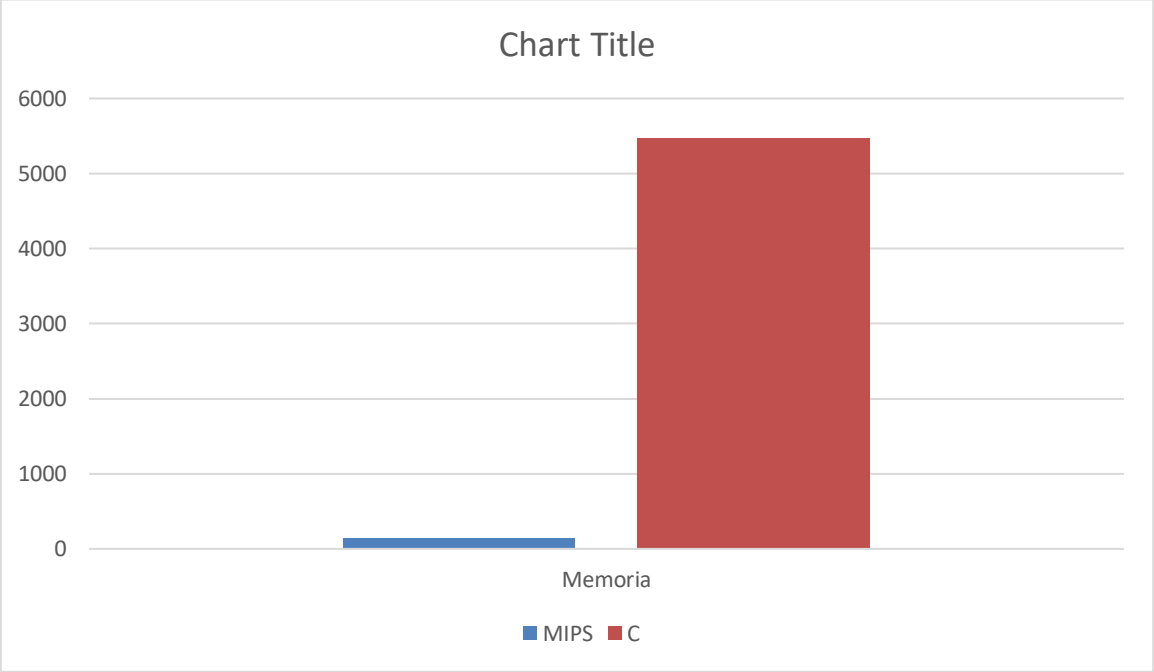
R:

Analise de Desempenhos entre os dois programas com n = 15:

Para computar o desempenho de tempo em Mips utilizamos syscall 30 que é exclusiva da IDE MARS, alterando o código “ANEXO A” função main para o main de “ANEXO C” que inicia chamando a função que retorna para $a0 os bits menos significativos do tempo, o valor de $a0 é movido para $a3 utilizado posteriormente para calcular a diferença de tempo inicial e final do programa em Mips.

A função rainha também foi alterada para (“ANEXO C” linha 33) que o programa encontrasse apenas a primeira solução.





Para computar a memória os códigos em C e Mips foram alterados na função main e função rainha (“ANEXO F” para o C e“ANEXO G” Mips) para encontra apenas a primeira solução para 15 rainhas. Foi utilizado o Data Segment da IDE MARS para calcular a quantidade de memória usada em Mips e o Monitor de Recurso do Windows 10 para computar a memória usada em c.

Para computar a quantidade de interações foram alterados o main em C e Mips para apenas executar n=15 (“ANEXO D” para o C e “ANEXO E” Mips) e a função rainha para adiciona mais um ao contador de interações e mostra apenas a primeira solução resultando em 1359 interações.

REFERÊNCIAS

Preiss, Bruno R. **Estruturas de dados e algoritmos - 2000**. Ed. Campus, Rio de Janeiro.

Gersting, Judth L. **Fundamentos matemáticos para a ciência da computação - 2001**. 4ª Edição. Ed. LTC, Rio de Janeiro.

Wikipédia, **Wikipédia A Enciclopédia Livre Xadrez - 2001**, Disponivel em: <http://pt.wikipedia.org/wiki/Xadrez>. Acesso em: 10 jun. 2017.

Oliveira, Henrique F. **Devmedia - Aplicação da metaheurística em algoritmos genéticos na solução do problema das n-rainhas - 2008**. Disponivel em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=4896&hl=*UML*>. Acesso em: 10 jun. 2017.

ANEXO A – CODIGO NRAINHAS C

```
#include <stdio.h>
#include <stdlib.h>
int tabuleiro[25], n, cont;
int main ()
{
	void rainha(int linha, int n);
	printf("Insira o valor de n:");
	scanf("%d",&n);
	rainha(1,n);
	return 0;
}
void print(int n)
{
	int i, x;
	printf("\n resultado %d:\n", ++cont);
	for(i = 1; i <= n; i++)
{
	printf("\t%d", i);
}
	for(i = 1; i <= n; i++) {
		printf("\n\n%d", i);
		for(x = 1; x <= n; x++) {
			if(tabuleiro[i] == x) {
				printf("\tQ");
			} else {
				printf("\t-");
			}
		}
	}
}
int coloca(int linha, int coluna)
{
	int i;
	for(i = 1; i < linha; ++i) {
		// Diferença entra Coluna de uma
rainha anterior e a Coluna a ser testada
		int x = tabuleiro[i] - coluna;
}}
		// Diferença entra Linha de uma
rainha anterior e a Linha a ser testada
		int y = i - linha;
		// Módulo de y e x
		if(y <= 0) {
			y = y * -1;
		}
		if (x <= 0) {
			x = x * -1;
		}
		if(tabuleiro[i] == coluna) {
	// Teste se existe coluna anterior
conflitante
			return 0;
		} else {
			if(x == y)
				return 0;
		}
	}
return 1;
}
void rainha(int linha, int n)
{
	int coluna;
	for(coluna = 1; coluna <= n; coluna++)
{
		if(coloca(linha, coluna)) {
			tabuleiro[linha] = coluna;
			if(linha == n){
				print(n);	//
Se chego na ultima linha imprime o resultado
			}else{
				rainha(linha + 1,
n);}	// Se não passa para próxima
linha
		}
```

ANEXO B – CODIGO NRAINHAS MIPS

```
		.data
tabuleiro:	.space 100
cont:		.space 4
cham:		.asciiz  "Insira o valor de n:"
str1:		.asciiz "\n resultado "
str2:		.asciiz "\t"
str3:		.asciiz ":\n"
str4:		.asciiz "\n\n"
strQ:		.asciiz "\tQ"
strT:		.asciiz "\t-"
```

```
11
12                  .text
13  .globl main
14  main:
15          la      $t9, cont       # Carrega
16  endereco do contador
17          li      $t8, 0          # Inicia $t8 = 0
18          sw      $t8, ($t9)      # Salva o valor 0
19  no contador
20
21          la      $a0, cham       # Imprime "Insira
22  o valor de n:"
23          li      $v0, 4
24          syscall
25
26          li      $v0, 5          # Recebe n
27          syscall
28
29          move    $a0, $v0        # Move n para $a0
30          li      $a1, 1          # Inicia Linha = 1
31
32          jal     rainha          # Chama função
33  rainha
34
35          li      $v0, 10         # Termina
36  programa
37          syscall
38
39  ####    Print   ####
40  print:
41          addi    $sp, $sp, -4    # Move $sp em -4
42  Bytes
43          sw      $a0, 0($sp)     # Salva $a0 (n)
44
45          move    $t4, $a0        # Move n para $t4
46
47
48          la      $a0, str1       # Imprime "\n
49  resultado "
50          li      $v0, 4
51          syscall
52
53          la      $t9, cont       # Carrega
54  endereço do contador
55          lw      $t7, ($t9)      # Salva o valor do
56  contador em $t0
57          addi    $t7, $t7, 1     # Contador++
58          sw      $t7, ($t9)      # Salva
59  Contador++
60
61          move    $a0, $t7        # Imprime valor
62  do contador
63          li      $v0, 1
64          syscall
65
66          la      $a0, str3       # Imprime ":\n"
67          li      $v0, 4
68          syscall
69
70          li      $t0, 1          # Inicia $t0 = 1
71
72  for1:
73          la      $a0, str2       # Imprime \t
74          li      $v0, 4
75          syscall
76
77          li      $v0, 1          # Imprime valor
78  de $t0 (Coluna)
79          move    $a0, $t0
80          syscall
81
82          addi    $t0, $t0, 1     # $t0++
83          ble     $t0, $t4, for1  # $t0 <= N
84
85          li      $t0, 1          # Inicia $t0 = 1
86  (Linha)
87  for2:
88          la      $a0, str4       # Imprime \n\n
89          li      $v0, 4
90          syscall
91
92          move    $a0, $t0        # Imprime $t0
93  (Linha)
94          li      $v0, 1
95          syscall
96
97          li      $t1, 1          # Inicia $t1 = 1
98  (Coluna)
99  for3:
100         mul     $t5, $t0, 4     # Multiplica $t0
101 por 4
102         la      $t3, tabuleiro  # Chama tabuleiro
103         add     $t3, $t3,$t5    # Move endereço
104 do tabuleiro ate linha $t0
105         lw      $t9, ($t3)      # Carrega do valor
106 de tabuleiro[$t0]
107
108         bne     $t9, $t1, printT  # Se
109 tabuleiro[$t0] diferente de coluna Imprime -
110
111         la      $a0, strQ       # Printa Q
112         li      $v0, 4
113         syscall
114         j       fimfor3
115 printT:
116         la      $a0, strT       # Printa -
117         li      $v0, 4
118         syscall
119 fimfor3:
120         addi    $t1, $t1, 1     # $t1++
```

```
        ble     $t1, $t4, for3  # $t1 <= N

        addi    $t0, $t0, 1     # $t0++
        ble     $t0, $t4, for2  # $t0 <= N

        lw      $a0, 0($sp)     # Carrega $a0 (n)
        addi    $sp, $sp, 4     # Move $sp em 4
Bytes
        jr      $ra             # Volta para o
endereco de $ra

####    Coloca####
coloca:
        li      $t0, 1          # Inicia $t0 = 1
para testar se alguma linha anterior gera conflito
for5:
        bge     $t0, $a1, retorna1 # Se $t0 >=
linha Retorna 1

        mul     $t8, $t0, 4     # Multiplica $t0
por 4
        la      $t9, tabuleiro  # Carrega
endereço de tabuleiro
        add     $t9, $t9, $t8   # Move endereco
do tabuleiro ate linha $t0 * 4bytes
        lw      $t8, ($t9)      # Carrega
tabuleiro[$t0]
        move    $t6, $t8        # Move
tabuleiro[$t0] para $t6

        sub     $t8, $t8, $a2   # t8 =
tabuleiro[$t0] - coluna
        sub     $t7, $t0, $a1   # t7 = t0 - linha


        li      $t9,-1          # Carrega -1 para
fazer o modulo t8 e t7 se necessario

        bgtz    $t8, saiC1      # se t8 > 0 sai
        mul     $t8, $t8, $t9   # t8 * -1
saiC1:
        bgtz    $t7, saiC2      # t7 > 0 sai
        mul     $t7, $t7, $t9   # t7 * -1
saiC2:
        beq     $t6, $a2, retorna0 # se
tabuleiro[$t0] = coluna retorna 0
        beq     $t8, $t7, retorna0 # se
|(tabuleiro[$t0] - coluna)| == |($t0 - linha)|
retorna 0

        addi    $t0, $t0, 1     # $t0++
        j       for5
retorna1:
        li      $v1, 1          # Retorna 1

        jr      $ra             # Volta para o
endereco de $ra
retorna0:
        li      $v1, 0          # Retorna 0
        jr      $ra             # Volta para o
endereco de $ra

####    Rainha####
rainha:
        addi    $sp, $sp, -8    # Move sp em -8
Bytes
        sw      $a2, 4($sp)     # Salva Coluna
        sw      $ra, 0($sp)     # Salva endereco
de retorno

        li      $a2, 1          # Inicia coluna
        = 1
for6:
        bgt     $a2, $a0, saifor6 # Teste de
Coluna <= n

        jal     coloca          # Chama Colocar
(linha, coluna)
        beqz    $v1, saiprint1  # Teste se retorno
= 0

        li      $t1, 4          # Inicia $t1 = 4
        la      $t9, tabuleiro  # Carrega array
tabuleiro
        mul     $t1, $t1, $a1   # Mutiplica linha
por 4 e salva em $t1
        add     $t9, $t9, $t1   # Move endereco
em 4 bytes * linha
        sw      $a2, ($t9)      # Salva
tabuleiro[linha] = coluna

        bne     $a0, $a1, saiprint2 # Se linha = n
        jal     print           # Printa tabela
        j       saiprint1
saiprint2:
        addi    $sp, $sp, -4    # Move sp em -4
Bytes
        sw      $a1, 0($sp)     # Salva linha

        addi    $a1, $a1, 1     # Linha++
        jal     rainha          # Chama
rainha(linha + 1, n)

        lw      $a1, 0($sp)     # Carrega Linha
        addi    $sp, $sp, 4     # Move sp em 4
Bytes
saiprint1:
        addi    $a2, $a2, 1     # Coluna++
        j       for6
saifor6:
```

```
231         lw      $a2, 4($sp)     # Carrega Coluna
232         lw      $ra, 0($sp)     # Carrega
233 endereco de retorno
234         addi    $sp, $sp, 8     # Move sp em 8
235 Bytes
236         jr      $ra             # Volta para o
237 endereco de $ra
```

ANEXO C – CODIGO NRAINHAS MIPS TEMPO

```
  1 main:
  2         li      $v0, 30
  3         syscall
  4         move    $a3, $a0
  5
  6
  7         la      $t9, cont               # Carrega
  8 endereco do contador
  9         li      $t8, 0                  # Inicia $t8
 10 = 0
 11         sw      $t8, ($t9)              # Salva o
 12 valor 0 no contador
 13
 14
 15         li      $a0, 15                 # n = 15
 16         li      $a1, 1                  # Inicia
 17 Linha = 1
 18         jal     rainha                  # Chama
 19 função rainha
 20
 21         li      $v0, 30
 22         syscall
 23
 24         sub     $a0, $a0, $a3
 25
 26         li      $v0, 1
 27         syscall
 28
 29         li      $v0, 10                 # Termina
 30 programa
 31         syscall
 32
 33 rainha:
 34         addi    $sp, $sp, -8    # Move sp em -8
 35 Bytes
 36         sw      $a2, 4($sp)     # Salva Coluna
 37         sw      $ra, 0($sp)     # Salva endereco
 38 de retorno
 39
 40         bnez    $t2, saifor6    # Teste se ja
 41 encontro um resutado
 42         li      $a2, 1          # Inicia coluna
 43         = 1
 44 for6:
 45         bgt     $a2, $a0, saifor6 # Teste de
 46 Coluna <= n
 47
 48         jal     coloca          # Chama Colocar
 49 (linha, coluna)
 50         beqz    $v1, saiprint1  # Teste se retorno
 51 = 0
 52
 53         li      $t1, 4          # Inicia $t1 = 4
 54         la      $t9, tabuleiro  # Carrega array
 55 tabuleiro
 56         mul     $t1, $t1, $a1   # Mutiplica linha
 57 por 4 e salva em $t1
 58         add     $t9, $t9, $t1# Move endereco em
 59 4 bytes * linha
 60         sw      $a2, ($t9)      # Salva
 61 tabuleiro[linha] = coluna
 62
 63         bne     $a0, $a1, saiprint2 # Se linha = n
 64         #jal    print           # Printa tabela
 65         li      $t2, 1          # valor de retorno
 66 para o backtrack
 67         j       saifor6
 68 saiprint2:
 69         addi    $sp, $sp, -4    # Move sp em -4
 70 Bytes
 71         sw      $a1, 0($sp)     # Salva linha
 72
 73         addi    $a1, $a1, 1     # Linha++
 74         jal     rainha          # Chama
 75 rainha(linha + 1, n)
 76
 77         lw      $a1, 0($sp)     # Carrega Linha
 78         addi    $sp, $sp, 4     # Move sp em 4
 79 Bytes
 80 saiprint1:
 81         addi    $a2, $a2, 1     # Coluna++
 82         j       for6
 83 saifor6:
 84         lw      $a2, 4($sp)     # Carrega Coluna
 85         lw      $ra, 0($sp)     # Carrega
 86 endereco de retorno
 87         addi    $sp, $sp, 8     # Move sp em 8
 88 Bytes
 89         jr      $ra             # Volta para o
 90 endereco de $ra
 91
 92 }=
```

ANEXO D – CODIGO NRAINHAS C INTERAÇÕES

```
int main ()
{
	cont_interacoes = 0;
	x = 0;
	void rainha(int linha, int n);
	rainha(1,15);
	printf("\n numero de interacoes
%f\n", cont_interacoes);
	return 0;
}
void rainha(int linha, int n)
{
	cont_interacoes++;
	int coluna;
	for(coluna = 1; coluna <= n;
coluna++) {
		if(coloca(linha, coluna)) {
			tabuleiro[linha] =
coluna;
			if(linha == n){
				//print(n);
				x = 1;
				return 0;

			}else{

	rainha(linha + 1, n);
				if (x==1){

	return 0;
				}
				}
		}
	}

}
```

ANEXO E – CODIGO NRAINHAS MIPS INTERAÇÕES

```
main:
	l.d	$f0, cont_inter # Carrega
contador de interacoes
	la	$t9, cont	# Carrega
endereco do contador
	li	$t8, 0		# Inicia $t8 =
0
	sw	$t8, ($t9)	# Salva o
valor 0 no contador

	la	$a0, cham	# Imprime
"Insira o valor de n:"
	li	$v0, 4
	syscall

	li	$v0, 5		# Recebe n
	syscall

	move	$a0, $v0	# Move n para
$a0
	li	$a1, 1		# Inicia Linha
= 1
	jal	rainha		# Chama
função rainha

	la	$a0, numInte	# Printa
"\n Numero de interacoes: "
	li	$v0, 4
	syscall

	li	$v0, 3		# Printa
	syscall

	li	$v0, 10		#
Termina programa
	syscall

rainha:
	addi	$sp, $sp, -8	# Move
sp em -8 Bytes
	sw	$a2, 4($sp)	# Salva
Coluna
	sw	$ra, 0($sp)	# Salva
endereco de retorno
```

```
45          bnez    $t2, saifor6     # Teste se ja
46  encontro um resutado
47          li      $a2, 1           # Inicia coluna
48          = 1
49          add.d   $f12, $f12, $f0
50  for6:
51          bgt     $a2, $a0, saifor6 # Teste de
52  Coluna <= n
53
54          jal     coloca           # Chama
55  Colocar (linha, coluna)
56          beqz    $v1, saiprint1   # Teste se
57  retorno = 0
58
59          li      $t1, 4           # Inicia $t1 = 4
60          la      $t9, tabuleiro   # Carrega
61  array tabuleiro
62          mul     $t1, $t1, $a1    # Mutiplica
63  linha por 4 e salva em $t1
64          add     $t9, $t9, $t1    # Move
65  endereco em 4 bytes * linha
66          sw      $a2, ($t9)       # Salva
67  tabuleiro[linha] = coluna
68
69          bne     $a0, $a1, saiprint2 # Se linha
70  = n
71          jal     print            # Printa tabela
72          li      $t2, 1           # valor de
73  retorno para o backtrack
74          j       saifor6
75  saiprint2:
76          addi    $sp, $sp, -4     # Move
77  sp em -4 Bytes
78          sw      $a1, 0($sp)      # Salva
79  linha
80
81          addi    $a1, $a1, 1      #
82  Linha++
83          jal     rainha           # Chama
84  rainha(linha + 1, n)
85
86          lw      $a1, 0($sp)      #
87  Carrega Linha
88          addi    $sp, $sp, 4      # Move
89  sp em 4 Bytes
90  saiprint1:
91          addi    $a2, $a2, 1      #
92  Coluna++
93          j       for6
94  saifor6:
95          lw      $a2, 4($sp)      #
96  Carrega Coluna
97          lw      $ra, 0($sp)      #
98  Carrega endereco de retorno
99          addi    $sp, $sp, 8      # Move
100 sp em 8 Bytes
101         jr      $ra              # Volta
102 para o endereco de $ra
```

ANEXO F – CODIGO NRAINHAS C MEMORIA

```
1  int main ()
2  {
3          x = 0;
4          void rainha(int linha, int n);
5          rainha(1,15);
6  }
7          void rainha(int linha, int n)
8  {
9          int coluna;
10         for(coluna = 1; coluna <= n;
11 coluna++) {
12                 if(coloca(linha, coluna)) {
13                         tabuleiro[linha] =
14 coluna;
15                         if(linha == n) {
16                                 //print(n);
17                                 x = 1;
18                                 return 0;
19                         } else {
20
21         rainha(linha + 1, n);
22                                 if (x == 1)
23 {
24
25         return 0;
26                                 }
27                         }
28                 }
29         }
30 }
```

ANEXO G – CODIGO NRAINHAS MIPS MEMORIA

```
main:

        la      $t9, cont       # Carrega endereco do contador
        li      $t8, 0          # Inicia $t8 = 0
        sw      $t8, ($t9)      # Salva o valor 0 no contador

        li      $a0, 15         # n = 15
        li      $a1, 1          # Inicia Linha = 1
        jal     rainha          # Chama função rainha

        li      $v0, 10         # Termina programa
        syscall

rainha:
        addi    $sp, $sp, -8    # Move sp em -8 Bytes
        sw      $a2, 4($sp)     # Salva Coluna
        sw      $ra, 0($sp)     # Salva endereco de retorno

        bnez    $t2, saifor6    # Teste se ja encontro um resutado
        li      $a2, 1          # Inicia coluna = 1
for6:
        bgt     $a2, $a0, saifor6 # Teste de Coluna <= n

        jal     coloca          # Chama Colocar (linha, coluna)
        beqz    $v1, saiprint1  # Teste se retorno = 0

        li      $t1, 4          # Inicia $t1 = 4
        la      $t9, tabuleiro  # Carrega array tabuleiro
        mul     $t1, $t1, $a1   # Mutiplica linha por 4 e salva em $t1
        add     $t9, $t9, $t1   # Move endereco em 4 bytes * linha
        sw      $a2, ($t9)      # Salva tabuleiro[linha] = coluna

        bne     $a0, $a1, saiprint2 # Se linha = n
        ##jal   print   # Printa tabela
        li      $t2, 1          # valor de retorno para o backtrack
        j       saifor6
saiprint2:
        addi    $sp, $sp, -4    # Move sp em -4 Bytes
        sw      $a1, 0($sp)     # Salva linha

        addi    $a1, $a1, 1     # Linha++
        jal     rainha          # Chama rainha(linha + 1, n)

        lw      $a1, 0($sp)     # Carrega Linha
        addi    $sp, $sp, 4     # Move sp em 4 Bytes
saiprint1:
        addi    $a2, $a2, 1     # Coluna++
        j       for6
saifor6:
        lw      $a2, 4($sp)     # Carrega Coluna
        lw      $ra, 0($sp)     # Carrega endereco de retorno
        addi    $sp, $sp, 8     # Move sp em 8 Bytes
        jr      $ra             # Volta para o endereco de $ra

```