**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]](#footnote-1)

Gabriel Fontenele de Azevedo[[2]](#footnote-2)

**RESUMO**: O problema das n-Rainhas requer um algoritmo de alta complexidade O(n³). 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.

**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).

**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\*](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-"

.text

.globl main

main:

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

li $v0, 10 # Termina programa

syscall

#### Print ####

print:

addi $sp, $sp, -4 # Move $sp em -4 Bytes

sw $a0, 0($sp) # Salva $a0 (n)

move $t4, $a0 # Move n para $t4

la $a0, str1 # Imprime "\n resultado "

li $v0, 4

syscall

la $t9, cont # Carrega endereço do contador

lw $t7, ($t9) # Salva o valor do contador em $t0

addi $t7, $t7, 1 # Contador++

sw $t7, ($t9) # Salva Contador++

move $a0, $t7 # Imprime valor do contador

li $v0, 1

syscall

la $a0, str3 # Imprime ":\n"

li $v0, 4

syscall

li $t0, 1 # Inicia $t0 = 1

for1:

la $a0, str2 # Imprime \t

li $v0, 4

syscall

li $v0, 1 # Imprime valor de $t0 (Coluna)

move $a0, $t0

syscall

addi $t0, $t0, 1 # $t0++

ble $t0, $t4, for1 # $t0 <= N

li $t0, 1 # Inicia $t0 = 1 (Linha)

for2:

la $a0, str4 # Imprime \n\n

li $v0, 4

syscall

move $a0, $t0 # Imprime $t0 (Linha)

li $v0, 1

syscall

li $t1, 1 # Inicia $t1 = 1 (Coluna)

for3:

mul $t5, $t0, 4 # Multiplica $t0 por 4

la $t3, tabuleiro # Chama tabuleiro

add $t3, $t3,$t5 # Move endereço do tabuleiro ate linha $t0

lw $t9, ($t3) # Carrega do valor de tabuleiro[$t0]

bne $t9, $t1, printT # Se tabuleiro[$t0] diferente de coluna Imprime -

la $a0, strQ # Printa Q

li $v0, 4

syscall

j fimfor3

printT:

la $a0, strT # Printa -

li $v0, 4

syscall

fimfor3:

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:

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

**ANEXO C – CODIGO NRAINHAS MIPS TEMPO**

main:

li $v0, 30

syscall

move $a3, $a0

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, 30

syscall

sub $a0, $a0, $a3

li $v0, 1

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

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

}=

**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

bnez $t2, saifor6 # Teste se ja encontro um resutado

li $a2, 1 # Inicia coluna = 1

add.d $f12, $f12, $f0

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

**ANEXO F – CODIGO NRAINHAS C MEMORIA**

int main ()

{

x = 0;

void rainha(int linha, int n);

rainha(1,15);

}

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);

x = 1;

return 0;

} else {

rainha(linha + 1, n);

if (x == 1) {

return 0;

}

}

}

}

}

**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

1. Graduando do curso de Bacharelado em Ciência da Computação da Faculdade Estácio – Ceut. [↑](#footnote-ref-1)
2. Graduando do curso de Bacharelado em Ciência da Computação da Faculdade Estácio – Ceut. [↑](#footnote-ref-2)