**UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC- CTC**

**DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**PROJETO E ANÁLISE DE ALGORITMO**

**Prof. Alexandre Gonçalves Silva**

**Aluno: Osmar de Oliveira Braz Junior**

# Questão 8

8. Para os algoritmos abaixo, pede-se:

(a) Quantas linhas, em função de n, em forma de Θ(.), o seguinte programa imprime? Escreva e resolva a recorrência. Considere n como uma potência de 2.

FUNÇÃO *f*(n)

1: **se** n > 1 **então O(1) c1**

2: imprime linha (“ainda rodando”) **O(1) c2**

3: f (n/2) **n/2**

4: f (n/2) **n/2**

**R.**

Substituindo recursivamente

Generalizando T(n) =

Resolvendo o somatório, que representa uma progressão geométrica.

Substituindo o somatório na fórmula:

Considerando

Substituindo T(1)=1

Prova por indução,

**Passo base**: para n = 1, o resultado esperado é 0

T(1) = 1 (correto)

Para n=2k

**Passo indutivo**: por hipótese de indução, assumimos que a fórmula está correta para n=2k, isto é, é igual

Então, temos que verificar se , sabendo-se que T(n) = e partindo da H.I. que T(2k) =

T(n) =

T(n) =

T(n) =

T(n) = passo indutivo provado)

Comprovado o resultado com a hipótese.

Demonstrado que =

para n >= 1

**Outra forma de resolução**

T(n) = 2T(n/2) + c

= 1 = n1 > c

c2: custo de impressão

Logo, pelo teorema mestre a complexidade da recorrência é igual a Θ(n).

Para a impressão:

Dado n = 2.

n =2 Ainda rodando

Imprime 1 vez

Dado n = 4.

n =4 Ainda rodando

n =2 Ainda rodando

n =2 Ainda rodando

Imprime 3 vezes.

Dado n = 8.

n =8 Ainda rodando

n =4 Ainda rodando

n =4 Ainda rodando

n =2 Ainda rodando

n =2 Ainda rodando

n =2 Ainda rodando

n =2 Ainda rodando

Imprime 7 vezes.

Logo o número de impressões é igual a Θ(n-1). Ou simplesmente f(n) ϵ Θ(n).

(b) Considere dois números inteiros n e r como argumentos, sendo n >= 0 e r >= 0. Qual o número de vezes que a subrotina CAIXA-PRETA é chamada pelo algoritmo ALGO? Expresse esse número como função de *n* e *r* e justifique a sua resposta.

ALGO(n, r)

1: **se** n = 0 **então O(1)**

2: CAIXA-PRETA(n, r) **O(1)**

3: **devolve** 1 **O(1)**

4: k <- r x ALGO(n - 1, r) **n-1**

5: **para** i <- 1 **até** k **faça n**

6: CAIXA-PRETA(n, i) **O(1)**

7: **devolve** k **O(1)**

R.

Note que no algoritmo a variável k define o número de chamadas à função **CAIXA-PRETA** e será invocado n+1 vezes pela função “**ALGO**”. Na última iteração, teremos n=0 e k não será chamado pois "**ALGO**" retornará 1 antes do for.

Portanto, temos k = r\*[r\*algo(n-1)]... até k = r\* r \* r ... \* r (n vezes);

Podemos considerar

Pode-se observar no algoritmo que k está em função r. Mas precisamente

Então, a fórmula original é

T(n) está escrito em função de e (Duas expansões)

Forma Geral (i-ésimo valor)

Considerando i=n:

Aqui temos uma soma de PG

Fórmula fechada, portanto, o algoritmo é:

Prova por indução:

//Correto

Considerando na formula fechada:

Substituindo na formula original e por HI:

//Correto

Como nosso objetivo e determinar apenas a quantidade de vezes que CAIXA-PRETA é chamada, podemos considerar e . Portanto, esta contagem e dada por

//Correto

//Discussão da Sala de Aula

**IMPLEMENTAÇÃO EM JAVA:**

public class Principal {

public static void main(String[] args) {

algo(5, 4, 0); //Acrescentei o parâmetro cont para poder contar mais fácil

}

public static int algo(int n, int r, int cont) {

if (n == 0) {

cont++;

System.out.println("n="+n+" r="+r+" cont="+ cont); //CAIXA-PRETA(n,r) --> Onde aparece a chamada a caixa-preta, apenas somo

return 1;

}

int k = r \* algo(n - 1, r, cont);

for (int i = 1; i <= k; i++) {

cont++; // CAIXA-PRETA(n, i) --> mais um para caixa-preta

if (i==k) // Imprimo a última iteração no for para poder ter a parcial de chamadas

System.out.println("n="+n+" r="+r+" cont="+cont);

}

return k;

}

}

Comentários:

Note que no algoritmo a variável k define o número de chamadas à função caixa-preta e será invocado n+1 vezes pela função algo. Na última iteração, teremos n=0 e k não será chamado pois "algo" retornará 1 antes do for.

Portanto, temos k = r\*[r\*algo(n-1)]... até k = r\* r \* r ... \* r (n vezes);

Acrescenta-se a esta contagem a chamada feita na condição de parada da recursão, ou seja, quando n=0.

Total de Chamadas =

Conclui-se que o somatório se trata de uma progressão geométrica de n+1 elementos, a1=r e razão=r.

Substituindo o cálculo do somatório em = para determinar a fórmula geral.

Total de Chamadas =

Calculando o total de chamadas para n=5 e r=4 e então comparando com o resultado obtido na execução do algoritmo:

= //Verdadeiro

Resultado da execução do algoritmo:

Resultado para n=5 e r=4

n=0 r=4 cont=1

n=1 r=4 cont=4

n=2 r=4 cont=16

n=3 r=4 cont=64

n=4 r=4 cont=256

n=5 r=4 cont=1024

Total: 1024 + 256 + 64 + 16+ 4 + 1 = (4^(5+1)-1)/(4-1) = 1365