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.

$$T(1) = 1$$
$$T(n) = 2T\left(\frac{n}{2}\right) + c_1 + c_2$$
$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + c_1 + c_2$$
$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + c_1 + c_2$$

Substituindo recursivamente

$$T(n) = 2T\left(\frac{n}{2}\right) + c_1 + c_2$$
$$T(n) = 2[2T\left(\frac{n}{2^2}\right) + c_1 + c_2] + c_1 + c_2$$
$$T(n) = 2[2[2T\left(\frac{n}{2^3}\right) + c_1 + c_2] + c_1 + c_2] + c_1 + c_2$$
$$T(n) = 2[2^2T\left(\frac{n}{2^3}\right) + 2c_1 + 2c_2 + c_1 + c_2] + c_1 + c_2$$
$$T(n) = 2^3T\left(\frac{n}{2^3}\right) + 2^2c_1 + 2c_2 + c_1 + 2^2c_2 + 2c_1 + c_2$$

Generalizando T(n) = $2^iT\left(\frac{n}{2^i}\right) + (c_1 + c_2)[\sum_{i=1}^{i}(2)^{k-1}]$

Resolvendo o somatório, que representa uma progressão geométrica.

$$S_i = \frac{2^i - 1}{2 - 1}$$
$$S_i = 2^i - 1$$

Substituindo o somatório na fórmula:

$$T(n) = 2^iT\left(\frac{n}{2^i}\right) + (c_1 + c_2)(2^i - 1)$$
$$T(n) = 2^iT\left(\frac{n}{2^i}\right) + c_12^i - c_1 + c_22^i - c_2$$
$$T(n) = 2^iT\left(\frac{n}{2^i}\right) + 2^i(c_1 - c_2) - (c_1 + c_2)$$

Considerando $n = 2^i$
$T(n) = nT\left(\frac{n}{n}\right) + n(c_1 - c_2) - (c_1 + c_2)$
$T(n) = nT(1) + n(c_1 - c_2) - (c_1 + c_2)$

Substituindo T(1)=1
$T(n) = n + n(c_1 - c_2) - (c_1 + c_2)$

Prova por indução,
Passo base: para n = 1, o resultado esperado é 0
$T(n) = n + n(c_1 - c_2) - (c_1 + c_2)$
$T(1) = 1 + 1(c_1 - c_2) - (c_1 + c_2)$
T(1) = 1 (correto)

Para n=2k
$T(2k) = 2k + 2k(c_1 - c_2) - (c_1 + c_2)$

Passo indutivo: por hipótese de indução, assumimos que a fórmula está correta para n=2k, isto é, $T(2n) = 2T\left(\frac{2k}{2}\right) + c_1 + c_2$ é igual $T(k) = 2T(k) + c_1 + c_2$
Então, temos que verificar se $T(n) = n + n(c_1 - c_2) - (c_1 + c_2)$, sabendo-se que T(n) = $2T\left(\frac{n}{2}\right) + c_1 + c_2$ e partindo da H.I. que T(2k) = $2T\left(\frac{2k}{2}\right) + c_1 + c_2$
T(n) = $2T\left(\frac{n}{2}\right) + c_1 + c_2$
T(n) = $2[k + k(c_1 + c_2) - (c_1 + c_2)] + c_1 + c_2$
T(n) = $2k + 2k(c_1 + c_2) - 2(c_1 + c_2) + c_1 + c_2$
T(n) = $2k + 2k(c_1 + c_2) - (c_1 + c_2)$ passo indutivo provado)

Comprovado o resultado com a hipótese.
Demonstrado que $2T\left(\frac{n}{2}\right) + c_1 + c_2 =$ $T(n) = n + n(c_1 - c_2) - (c_1 + c_2)$
para n >= 1

Outra forma de resolução

T(n) = 2T(n/2) + c
log_2 2 = 1 = n1 > c

c2: custo de impressão

Logo, pelo teorema mestre a complexidade da recorrência é igual a $\Theta(n)$.
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)
se n = 0 então                       O(1)
    CAIXA-PRETA(n, r)                O(1)
    devolve 1                        O(1)
k   <- r x ALGO(n - 1, r)            n-1
para i  <- 1 até k faça              n
    CAIXA-PRETA(n, i)                O(1)
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
$T(0) = 1$
$T(n) = T(n-1) + \mathrm{c}_1\mathrm{k} + \mathrm{c}_2$

Pode-se observar no algoritmo que k está em função r. Mas precisamente
$k = r^n$

Então, a fórmula original é
$T(n) = T(n-1) + \mathrm{c}_1\mathrm{r}^n + \mathrm{c}_2$

T(n) está escrito em função de $T(n-1)$ e $T(n-2)$ (Duas expansões)
$T(n) = T(n-3) + \mathrm{c}_1\left(\mathrm{r}^n + \mathrm{r}^{n-1} + \mathrm{r}^{n-2}\right) + 3\mathrm{c}_2$

Forma Geral (i-ésimo valor)
$T(n) = T(n-i) + \mathrm{c}_1\left(\mathrm{r}^n + \mathrm{r}^{n-1} + \ldots + \mathrm{r}^{n-(i-1)}\right) + i\mathrm{c}_2$

Considerando i=n:

$T(n) = T(0) + c_1(r^n + r^{n-1} + ... + r^1) + nc_2$

Aqui temos uma soma de PG

$T(n) = 1 + c_1\left(\frac{r(r^n-1)}{r-1}\right) + nc_2$

$T(n) = 1 + c_1\left(\frac{r^{n+1}-r}{r-1}\right) + nc_2$

Fórmula fechada, portanto, o algoritmo é:

$O(r^n)$

Prova por indução:

$T(n) = 1 + c_1\left(\frac{r-r}{r-1}\right) + 0$ //Correto

Considerando $n - 1$ na formula fechada:

$T(n-1) = 1 + c_1\left(\frac{r^n-r}{r-1}\right) + (n-1)c_2$

Substituindo $T(n-1)$na formula original e por HI:

$T(n) = 1 + c_1\left(\frac{r^n-r}{r-1}\right) + (n-1)c_2 + c_1 r^n + c_2$

$T(n) = 1 + c_1\left(\frac{r^n-r}{r-1} + r^n\right) + (n-1+1) + c_2$

$T(n) = 1 + c_1\left(\frac{r^{n+1}-r}{r-1}\right) + nc_2$ //Correto

Como nosso objetivo e determinar apenas a quantidade de vezes que CAIXA-PRETA é chamada, podemos considerar $c_1 = 1$ e $c_2 = 0$. Portanto, esta contagem e dada por

$contagem = 1 + \frac{r^{n+1}-r}{r-1}$ //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 = $\sum_{i=1}^{k} r^i + 1$

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

$$Sn = \frac{a1(q^n-1)}{q-1} = \frac{r(r^n-r)}{q-1}$$

Substituindo o cálculo do somatório em = $\sum_{i=1}^{k} r^i + 1$ para determinar a fórmula geral.

Total de Chamadas = $\frac{r(r^n-r)}{q-1} + 1$

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

$\frac{r(r^n-r)}{q-1} + 1 => \frac{4(4^5-1)}{4-1} => 1365 =>$ = $\frac{4096-4}{3} => 1365$ //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