Exercícios 2

Davi Juliano Ferreira Alves - RA: 133595
Projeto e Análise de Algoritmo

7 de junho de 2021

1 Resolução dos exercícios

Exercício 1 Considere o algoritmo implementado para resolver o problema 1 (trabalho prático).

a)Faça uma análise da complexidade assintótica desse algoritmo. Forneça o algoritmo a ser analisado.

Resolução: Primeiramente definiremos nosso Input. Sejam dois inteiros x e y tais que $x, y \in [0, 10^{16}]$. Se x e y são inteiros, eles possuem representação binária, ou seja, na base 2. A distância $d(x, y)$ definida nos inteiros pode ser dada por:

$$d(x, y) = |x - y|$$

Ainda que haja mudanca de bases, essa distância só seria posta em uma base distinta, mas não se alteraria o valor. Se x e y são um inteiros que possuem representação binária, então eles possuem n e m algarismos em binário, respectivamente. Como $y > x$, portanto $m > n$.

A ideia do algoritmo é utilizar de uma fórmula (linear) para conseguir calcular a quantidade de 1's dentro do intervalo $[n + 1; m - 1]$. Como x pertence ao conjunto de números de algarismos de n, mas ele pode não ser necessariamente o maior ou o menor, então, calcularemos utilizando a força bruta até a fronteira do conjunto de n, e tomaremos o conjunto de $n+1$ para aplicar na fórmula, pois lá sabemos que estão todos os valores do conjunto de $n+1$ algarismos. Semelhantemente faremos o processo para y, mas nivelaremos por baixo, pois enquanto queremos a distância de x à y, os valores de x são crescentes e os de y, decrescentes, portanto, tomaremos o conjunto $m - 1$.

A fórmula proposta visa contar a quantidade de 1's que existem entre uma quantidade z de algarismos até z+1. Note que, para uma quantidade z de algarismos, teremos:

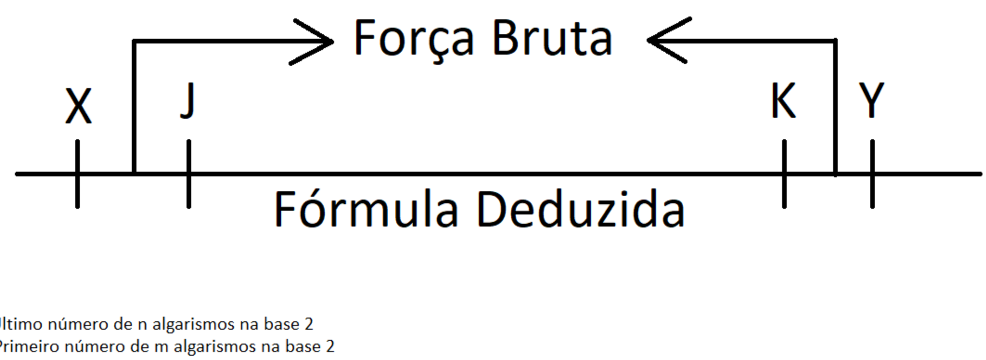
10...000
10...001
10...010
10...100

.
.
.
11...000
11...001
11...010
11...100
.
.
.

Note que 1 do primeiro algarismo aparece 2^{z-1} vezes, e que a quantidade de 1 e 0, excluindo o primeiro algarismo 1, é igual, ou seja, para sabermos a quantidade de 1 de um z até $z+1$, precisamos contar o primeiro algarismo 2^{z-1} vezes mais a metade (quantidade de 1) do produto de 2^{z-1} por (z-1) (quantidade de algarismos que existem excluindo o primeiro que já foi contado). Portanto, a fórmula que descreve esse pensamento é dada por:

$$F(z) = 2^{z-1} + \frac{2^{z-1}.(z-1)}{2}$$

Definido o pensamento, segue a imagem abaixo com o as distâncias ditas e o que o algoritmo executa:



Seguindo o pensamento acima, o processo de força bruta seria reduzido de uma boa maneira e a fórmula deduzida iria rodar em uma quantidade limitada, ou seja, não rodaria em uma quantidade n de vezes. Portanto, será analisada a complexidade da primeira e da segunda numa primeira análise, pois ambas são análogas, e da terceira parte de maneira separada.

- Para a primeira, temos um while com n instruções e posteriormente, um while que roda n vezes encadeado de outro com consequentes n/2 instruções, portanto, temos uma complexidade dada por:

$$f_1(n) = log(n) + n.log(n) = log(n).(n+1)$$
$$= log(n) + n.log(n) \leq n.log(n) + n.log(n) = 2.n.log(n)$$

Ou seja, tomando $c = 2$, temos que $f_1(n) = O(n.log(n))$.

- Para a segunda, temos uma instrução somada a apenas um while com uma quantidade n de iterações, portanto, temos uma complexidade dada por:

$$f_2(n) = 1 + n \leq n + n = 2n$$

Ou seja, tomando $c = 2$, temos que $f_2(n) = O(n)$.

Portanto, a complexidade do algoritmo pode ser dada da soma entre as três partes:

$$f(n) = f_1(n) + f_2(n) = log(n) + n.log(n) + n + 1$$
$$= log(n) + n.log(n) + n + 1 \leq 4.n.log(n)$$

Tomando $c = 4$, temos que $f(n) = O(n.log(n))$. Portanto, temos que o algoritmo, ainda que separado em intervalos menores, possui uma complexidade $O(n.log(n))$.

b)Crie um vídeo com a sua explicação da lógica de como o algoritmo resolve o problema.

Resolução: Link juntamente com essa resolução.

Exercício 2 Suponha que você tem 3 algoritmos que resolvem um certo problema:

- Algoritmo I resolve o problema dividindo-o em 5 subproblemas de metade do tamanho cada, resolvendo-os recursivamente, e combinando suas soluções em tempo linear;
- Algoritmo II resolve o problema de tamanho n resolvendo dois subproblemas de tamanho $n - 1$ e combinando as soluções em tempo constante;
- Algoritmo III resolve o problema de tamanho n dividindo-o em nove subproblemas de tamanho $\frac{n}{3}$ cada, resolvendo-os recursivamente, e combinando as soluções em tempo $\Theta(n^2)$.

Determine os tempos de execução desses algoritmos utilizando a notação O. Qual algoritmo você escolheria?

Resolução: Primeiro, iremos determinar o tempo computacional de cada algoritmo para, posteriormente, escolher e determinar o algoritmo mais eficiente.

- Suponha que o algoritmo I tenha de resolver um problema de tamanho n. O enunciado diz que ele divide o problema de tamanho n em 5 subproblemas de tamanho $n/2$.

  Sabendo que a combinação das soluções é em tempo linear, temos que terão de ser combinados 5 subproblemas de tamanho $n/2$, ou seja:

$$T(n) = 5.T(n/2) + O(n)$$

Pois deve-se resolver recursivamente os 5 subproblemas de tamanho $n/2$, e logo após combinar com uma estratégia $O(n)$. Utilizando do Teorema Mestre, podemos encontrar para $T(n)$:

$$a = 5,\ b = 2 \text{ e } f(n) = O(n)$$
$$log_b(a) = 2.32 \Rightarrow n^{2.32} > O(n) \Rightarrow T(n) = O(n^{2.32})$$

Portanto, o primeiro algoritmo é da ordem $O(n^{2.32})$.

- Suponha que o algoritmo II tenha de resolver um problema de tamanho n. Sabendo que o algoritmo dividiu em 2 subproblemas de $n-1$ cada, combinando em tempo constante, temos a seguinte função:

$$T(n) = 2.T(n-1) + O(1) = 2.T(n-1) + 1$$

Primeiramente, usaremos a recursão para tentar desvendar qual é a ordem da função, para, posteriormente, usar o método da substituição:

$$T(n) = 2T(n-1) + 1$$
$$T(n) = 2.(2.T(n-2) + 1]) + 1 = 4.T(n-2) + 3$$
$$T(n) = 4.(2.T(n-3) - 1) + 3 = 8.T(n-3) + 7$$
$$T(n) = 8.(2.T(n-4) - 1) + 7 = 16.T(n-4) + 15$$
$$\vdots$$
$$T(n) = 2^n.T(1) - (2^n - 1)$$

Temos um palpite de que $T(n)$ pode ser $O(2^n)$. De fato, suponha que $T(n) = O(2^n)$, usando o método da substituição, teremos que:

$$m = n-1,\ T(n-1) \leq c.2^{(n-1)}$$
$$\Rightarrow T(n) = 2.T(n-1) + 1 \leq 2.c.2^{(n-1)} + 1 = c.2^{(n-1)+1} = c.2^n, \text{ para qualquer } c \in \mathbb{Z}$$

De fato, pelo método da substituição, fica evidente que $T(n) = O(2^n)$.

- Suponha que o algoritmo III tenha de resolver um problema de tamanho n. Sabendo que o algoritmo dividiu o problema em 9 subproblemas de tamanho $n/3$ cada e combinando-os em um método $\Theta(n^2)$, temos a seguinte função:

$$\text{T(n)} = 9.\text{T(n/3)} + \Theta(n^2).$$

Pelo Teorema Mestre, encontramos, pra T(n),

$$a = 9,\ b = 3 \text{ e } f(n) = \Theta(n^2)$$
$$log_b(a) = 2 \Rightarrow n^2 = \Theta(n) \Rightarrow T(n) = O(log(n).n^2)$$

Sabendo o tempo de execução de cada algoritmo na notação O, teremos de decidir qual algoritmo possui a maior eficiência. Note que $O(2^n) > O(n^{2.32})$ e $O(log(n).n^2)$. Portanto, devemos decidir entre os algoritmos com tempo computacional $O(n^{2.32})$ e $O(log(n).n^2)$.

$n^2.n^{0.32}$ e $n^2.log(n)$, portanto, queremos provar a relação entre:
$n^{0.32}$ e $log(n)$.

Para demonstrar, tome uma $f(n) = \frac{n^{0.32}}{log(n)}$. Se $n^{0.32} > log(n)$, $f(n) \to \infty$, enquanto se $n^{0.32} < log(n)$, $f(n) \to 0$, para n suficientemente grande. Portanto, teremos:

$\lim_{x\to\infty} \frac{n^{0.32}}{log(n)} = \frac{\infty}{\infty}$, uma indeterminacao, usaremos L'Hospital:
$$\lim_{x\to\infty} \frac{(n^{0.32})'}{(log(n))'} = \lim_{x\to\infty} \frac{0.32.x.ln(10)}{x^0.68} = \lim_{x\to\infty} 0.32.ln(10).x^{0.32} \to \infty$$

Portanto, como $f(n) \to \infty$, temos que $n^{0.32} > log(n)$, e portanto, o algoritmo que deve ser escolhido é o de natureza $O(log(n).x^2)$, ou seja, o Algoritmo III.

Exercício 3 Considere a seguinte demonstração por indução:

Desmonstrar que $\sum_{i=1}^{n} i = O(n)$

Base: para $n = 1, \sum_{i=1}^{1} i = O(1)$

Passo: $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + n + 1 = O(n) + n + 1 = O(n+1)!$

a)Qual o erro no processo de demonstração?

Resolução: Primeiramente, a hipótese de indução está incorreta, visto que:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2} \leq \frac{n^2+n^2}{2} = n^2$$

Portanto, como n^2 é $O(n^2)$, temos que $\sum_{i=1}^{n} i = O(n^2)$.

O caso base está incorreto, pois a hipótese de indução compromete o caso base, mesmo ele fazendo sentido.

Em segundo lugar, a conta do passo de indução está incorreto na seguinte passagem:

$$\sum_{i=1}^{n} i + n + 1 = O(n) + n + 1 = O(n+1)!$$

Uma função f(n) é $O(n)$ quando f(n) < c.n, e portanto, ela é linear. No caso acima, a hipótese de indução comprometeu os resultados, e também a função ser o fatorial de uma classifição de uma certa função não faz sentido.

b)Faça uma demonstração de um limite correto usando indução.

Resolução: Seja uma certa fórmula, queremos demonstrar que $\sum_{i=1}^{n} i = O(n^2)$, pois

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2} \leq \frac{n^2+n^2}{2} = n^2$$

Portanto, seguindo corretamente a hipótese de indução, devemos:

Desmonstrar que $\sum_{i=1}^{n} i = O(n^2)$

Base: para $n = 1, \sum_{i=1}^{1} i = O(1)$

Passo: $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + n + 1 \leq c.n^2 + n + 1 \leq c.n^2 + 2.c.n + c = c.(n^2 + 2.n + 1) = c.(n+1)^2$

Como $\sum_{i=1}^{n+1} i \leq c.(n+1)^2$, temos que $\sum_{i=1}^{n+1} i = O((n+1)^2)$, e portanto está provado, pela indução.

Exercício 4 Considere a recorrência $T(n) = 4.T(n/2) + c.n$.

a) Encontre um bom limite superior assintótico para a recorrência utilizando o método de árvore de recursão.

Resolução:
Seguindo a ordem de recorrência, em cada nível da árvore, teremos uma tabela com informações sobre a dita ordem de recorrência:

Nível	Tamanho	Qnt Nós	Tempo por Nó
0	n	1	$c.n$
1	$n/2$	4	$c.n/2$
2	$n/4$	16	$c.n/4$
3	$n/8$	64	$c.n/8$
4	$n/16$	256	$c.n/16$
i	$n/2^i$	4^i	$c.n/2^i$
$log_2 n$	1	n^2	1

Portanto, utilizaremos a forma da soma para desvendar a ordem:

$$\text{Tempo: } \sum_{i=0}^{\log_2 n} 2^{2i}.\frac{c.n}{2^i} = c.n.\sum_{i=0}^{\log_2 n} 2^i = c.n.\frac{(2^{\log_2 n+1}-1)}{2-1}$$
$$= c.n.(2n^{\log_2 2} - 1) = 2.c.n^2 - c.n = T(n)$$

Como queremos um limite assintótico superior, e como c é uma constante real, ou seja, não podemos limitá-la por um $k \in \mathbb{R}$, tal que $T(n) \leq k.n^2$, portanto, temos que $T(n) \in O(n^3)$. De fato, $T(n) \in O(n^3)$, pois, suponha $n > 2$:

$$T(n) = 2.c.n^2 - c.n \leq 2.c.n^2 < c.n^3$$

Portanto, temos que $T(n) \in O(n^3)$.

b) Mostre que o limite encontrado é válido utilizando o método da substituição.

Resolução: Utilizando o método da árvore de recursão, temos que nossa $T(n) = 2.c.n^2 - c.n$ pertence à classe de limite superior assintótico dos $O(n^3)$. De fato, pois:

Hipótese da Substituição: Seja $m = n/2, T(n/2) \leq c.\frac{n^3}{8}$.

Passo da Substiuição: $T(n) = 4.T(n/2) + c.n \leq 4.c.\frac{n^3}{8} + c.n = c.\frac{n^3}{2} + c.n = c.n^3 - (c.\frac{n^3}{2} - c.n) \leq c.n^3$.

Portanto, está provado pelo método da substiuição que $T(n) = O(n^3)$.

2 Algoritmo do Problema 1

Algorithm 1: Nivelamento do x com a cota superior na base 2

```
Input  : Dois inteiros x e y (0 < x ≤ x ≤ 10^16)
Output: Resultado do cálculo
Inicializacao;
Numaux = x;
while Numaux > 0 do
    if Numaux%2 == 1 then
        Soma++;
    end
    Numaux = Numaux/2;
    j++;
end
Iteracao1 = j;
Numero = x+1;
while j == Iteracao1 do
    Numaux = Numero;
    j=0;
    while Numaux > 0 do
        if Numaux%2 == 1 then
            Soma++;
        end
        Numaux = Numaux/2;
        j++;
    end
    Numero++;
end
Iteracao1++;
```

Algorithm 2: Nivelamento do y com a cota inferior na base 2

```
Inicializizacao;
j=0;
numaux=y;
while Numaux > 0 do
    if Numaux%2 == 1 then
        Soma++;
    end
    Numaux = Numaux/2;
    j++;
end
Iteração2 = j;
Número = y-1;
while j == Iteracao2 do
    Numaux = Numero;
    j=0;
    while Numaux > 0 do
        if Numaux%2 == 1 then
            Soma++;
        end
        Numaux = Numaux/2;
        j++;
    end
    Numero−−;
end
Iteracao2−−;
```

Algorithm 3: Utilização da fórmula para o cálculo de 1's

Soma = soma - (Iteracao2+1);
for $i \in [Iteracao1, Iteracao2]$ **do**
    $Formula = 2^{i-1} + \frac{2^{i-1}.(i-1)}{2}$;
    Soma = Soma + Formula;
end