Escritos de Algoritmos e Estruturas de Dados

Rodrigo de Souza

Dezembro 2020

1 Logaritmos na base 2

> *On the most primitive level, a logarithm is an expoent. Thus, the fact that* $100 = 10^2$ *says that* 2 *is the logarithm of* 100 *to the base* 10 *(written* $2 = \log_2 100$*); and* $4 = 64^{1/3}$ *says that* $\frac{1}{3}$ *is the logarithm of* 4 *to the base* 64 *(*$\frac{1}{3} = \log_{64} 4$*).*
>
> *(G. Simmons, Calculus With Analytic Geometry, 1996)*

Um logaritmo é um expoente. Essa é uma maneira eficiente de gravar na cabeça o que é um logaritmo. Parece que muita gente faz cálculos e computações com logaritmos sem saber o que realmente eles são!

Logaritmos foram muito importantes ao longo da História como ferramenta para se realizar cálculos complicados. Você pode ler essa história no excelente livrinho *Logaritmos* de Elon Lages Lima [1]. Nossa intenção nestas notas é simpesmente dar um significado para os logaritmos em uma base particular, porque, nessa base, eles aparecem com frequência na Computação: a base 2. Essa base é tão comum que inventou-se uma notação específica para ela: denotamos o *logaritmo do número* n *na base* 2 *por*

$$\lg n$$

Vemos o gráfico de $\lg n$ na Figura 1. O primeiro detalhe importante a observar nesse gráfico é para que valores de n (eixo horizontal) o valor de $\lg n$ (eixo vertical) é um número natural. Pense um pouco e volte para cá.

Observamos na figura que isso acontece para $n = 1, 2, 4, 8$. Isso sugere que $\lg n$ é natural para as potências de 2: $n = 2^0, 2^1, 2^2, 2^3, \ldots$ e para números que não são potência de 2, $\lg n$ é um número real que não é inteiro.

Para tentar justificar essa sugestão, voltemos ao nosso mnemônico para os logaritmos: um logaritmo é um expoente. Então, uma maneira de ler $\lg n$ é

> $\lg n$ *é o expoente ao qual temos que elevar o número* 2 *para que o resultado seja* n*.*

Vamos dizer a mesma coisa mas com um pouco de formalismo matemático:

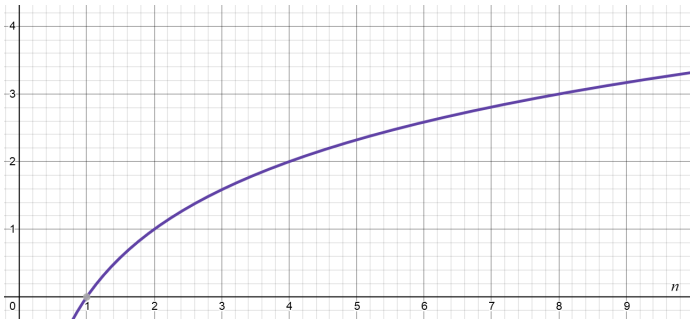

Figura 1: Gráfico de $\lg n$.

Chamando $\lg n$ *de* k*, temos que* $n = 2^k$*.*

Ora, se k é um número natural, então n é uma potência de 2.

O segundo ponto importante a perceber nesse gráfico é como a função $\lg n$ cresce devagar. De fato, para passar de um inteiro k a um inteiro $k+1$ no eixo vertical (ou seja, dar um salto de uma unidade), temos que passar de $n = 2^k$ para $n = 2^{k+1}$ no eixo horizontal (um salto enorme, passar de uma potência de 2 para a próxima). Por exemplo, $\lg 8 = 3$, e para chegar no valor 4, temos que pular para 16 no eixo horizontal: $\lg 16 = 4$. A lição a tirar aqui é

A função $\lg n$ *cresce muito devagar.* [1]

Vamos tabular valores de $\lg n$ para potências de 2; note que a cada potência de 2 (valores de n), $\lg n$ só aumenta uma unidade.

n	$\lg n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6

Às vezes na Computação nos interessamos nos valores de $\lg n$ quando n não é uma potência de 2. Nesses casos, estamos na verdade (quase sempre) interessados no *piso* ou no *teto* de $\lg n$. O piso é o *maior inteiro menor ou igual a* $\lg n$. Em outras palavras, é $\lg n$ arredondado para baixo. É denotado assim:

$$\lfloor \lg n \rfloor$$

[1] Uma outra maneira de ver isso é observando que a função logarítmica é a inversa da função exponencial. E a função exponencial cresce muito rápido.

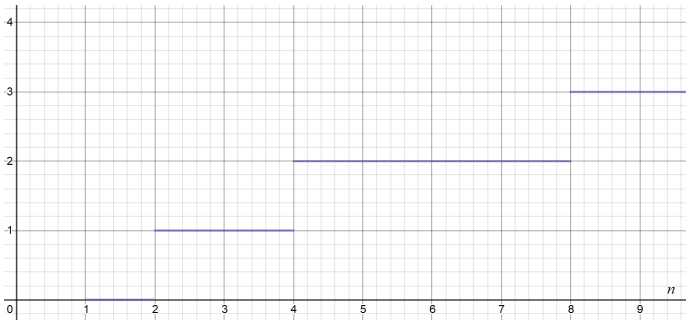

Figura 2: Gráfico de $\lfloor \lg n \rfloor$.

O teto é o menor inteiro maior ou igual a $\lg n$, ou ainda, $\lg n$ arredondado para cima; é denotado por

$$\lceil \lg n \rceil$$

Uma boa maneira de entender $\lfloor \lg n \rfloor$ (para $\lceil \lg n \rceil$ a história é parecida) é como uma função *entre números naturais* que, informalmente, mantém o mesmo valor até que n atinja uma nova potência de 2. A tabela a seguir torna essa descrição mais clara:

n	$\lfloor \lg n \rfloor$
1	0
2	1
3	1
4	2
5	2
6	2
7	2
8	3
9	3
10	3

Também é instrutivo olhar o gráfico de $\lfloor \lg n \rfloor$, que está ilustrado na Figura 2.

Por que os logaritmos na base 2 aparecem na Computação? Por motivos diversos. Um deles, importante na Análise de Algoritmos, está conectado com uma outra maneira de interpretar o valor $\lfloor \lg n \rfloor$:

> $\lfloor \lg n \rfloor$ é o número de vezes que podemos fazer divisão inteira de n por dois até chegar em 1.

Vejamos dois exemplos. Abaixo mostramos a sequência de divisões inteiras por 2 de $n = 18$. Por *divisão inteira* entenda-se: *divide por* 2 *e arredonda para baixo*:

$$18, 9, 4, 2, 1$$

Fizemos aqui 4 divisões por 2: a primeira levou 18 a 9, a segunda 9 a 4, depois 4 a 2, e finalmente 2 a 1. E de fato o valor de $\lfloor \lg 18 \rfloor$ é 4. Nosso segundo exemplo é com $n = 327$:

$$327, 163, 81, 40, 20, 10, 5, 2, 1$$

São 8 divisões, e de fato $\lfloor \lg 327 \rfloor$ vale 8.

Esse comportamento fica mais evidente quando n é uma potência de 2: se $n = 2^k$, então são k divisões por 2 até se chegar em 1. Além disso, todas as divisões são exatas (em cada uma, diminui-se o expoente). Somente quando n é uma potência de 2 as divisões são todas exatas. Isso faz com que, geralmente, quando estamos estudando a função $\lg n$, seja mais fácil observar o que acontece quando n é uma potência de 2.

Se n não é uma potência de 2, então a quantidade de divisões é a mesma que a da maior potência de 2 menor ou igual do que n. Por exemplo, 256 é uma potência de 2: 2^8, e $\lg 256 = 8$. A próxima potência de 2 é 512. Tudo o que está de 256 a 511 (inclusive) exige 8 divisões para se chegar em 1. Uma maneira formal de se dizer isso é: para todo natural $n > 0$,

$$\lfloor \lg n \rfloor \text{ é o único inteiro } k \text{ tal que } 2^k \leq n < 2^{k+1}.$$

Com essa interpretação, fica fácil imaginar que logaritmos na base 2 aparecem na análise de algoritmos que dividem o problema sucessivamente por 2 (por exemplo, o Mergesort). Na análise desses algoritmos, tipicamente pergunta-se: quantas divisões do problema, de tamanho original n, tem que ser feitas até que se atinja um problema de tamanho 1 ou algo próximo disso? A resposta é $\lg n$ (ou o piso, ou o teto disso).

Para concluir, segue uma função iterativa que recebe $n > 0$ e devolve $\lfloor \lg n \rfloor$. Ela fiz isso contando o número de vezes que podemos fazer essa divisão enquanto $n > 1$:

```
int lg (int N)
{
   int i = 0;
   int n = N;
   while (n > 1) {
      n = n / 2;
      i += 1;
   }
   return i;
}
```

Essa função foi tirada de `https://www.ime.usp.br/~pf/algoritmos/aulas/floor-lg.html`.

Referências

[1] Elon Lages Lima. *Logaritmos*. Sociedade Brasileira de Matemática, 2016.