O método mestre para encontrar a complexidade temporal de algoritmos recursivos

1 Introdução

A recursão é uma das partes essenciais da programação. Muitos algoritmos populares são implementados através desta técnica. Por isso, saber analisá-los também é importante. Aqui vamos apresentar um método bastante utilizado para calcular a complexidade temporal de um algoritmo recursivo: o método mestre.

Uma coisa a ser lembrada é que o método mestre é um método para resolver uma recorrência. Mas antes disso, é necessário obter a expressão de recorrência do algoritmo a ser analisado. Vamos endereçar este problema primeiro.

2 Como obter uma expressão de recorrência a partir de um algoritmo

A forma padrão do método mestre é:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \tag{1}$$

onde:

- `a` é o número de subproblemas que surgem da divisão. Para o algoritmo *mergesort* dividimos o vetor inicial em 2 partes a cada chamada recursiva, então `a`=2;
- `n/b` é o tamanho de cada subproblema. No *mergesort*, a cada vez dividimos o vetor ao meio, de forma que o tamanho de cada subproblema é `n/2`, e portanto `b`=2.
- $f(n) = O(n^d)$ é o custo para criar os subproblemas e combinar os seus resultados.

Para esta demonstração, vamos usar o algoritmo *merge sort*. O pseudocódigo para este algoritmo é mostrado abaixo:

```
merge(v,p,q,r)
    n1 = q-p+1
    n2 = r-q
    para i=0 até n1-1
        L[i] = v[p+i]
    para i=0 até n2-1
        R[i] = v[q+i+1]
    L[n1] = INT_MAX
    R[n2] = INT_MAX
    i = 0
    j = 0
    para k=p até r
        se(L[i]<=R[j])
            v[k] = L[i]
            i++
        senão
            v[k] = R[j]
            j++

mergesort(v,p,r)
    se (p<r)
        q = (p+r)/2
        mergesort(v,p,q)
        mergesort(v,q+1,r)
        merge(v,p,q,r)
```

Seja `n` o comprimento do vetor `v` de entrada, e `T(n)` o tempo de execução do algoritmo. Vamos agora identificar as variáveis do método mestre para este algoritmo:

- `a=2`, pois dividimos o vetor inicial em 2 partes a cada chamada recursiva;
- `n/b = n/2`: a cada vez dividimos o vetor ao meio, de forma que o tamanho de cada subproblema é `n/2`, e portanto `b`=2.
- `d = 1`, pois o custo para fazer a divisão é $O(1)$, e a rotina `merge` tem complexidade $O(n)$, e desta forma, o custo para fazer a divisão e juntar os resultados é $O(n)$.

Assim, a equação de recorrência para o mergesort fica:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \tag{2}$$

3 As fórmulas do método mestre

Existem três soluções para a forma padrão do método mestre:

Caso 1 : $O(n^d \log(n))$ se $a = b^d$

Caso 2 : $O(n^d)$ se $a < b^d$

Caso 3 : $O(n^{\log_b a})$ se $a > b^d$

Para o mergesort, vimos que `a=2`, `b=2` e `d=1`, e desta forma, $a = b^d$, o que nos diz que este algoritmo cai no caso 1. Substituindo os valores de `a`, `b` e `d` na expressão do caso 1, chegamos à conclusão que este algoritmo tem complexidade $O(n \log(n))$.

4 Mais exemplos do método mestre

Vamos ver mais dois exemplos do método mestre

Exemplo 1

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

Para esta recorrência, `a=3`, `b=2` e `d=2`. Assim, $a < b^d$, e estamos diante de um exemplo do caso 2, e a solução é $O(n^2)$.

Exemplo 2

$T(n) = 16T\left(\frac{n}{4}\right) + O(n)$

Neste caso, `a=16`, `b=4` e `d=1`. Assim, $a > b^d$, e estamos diante de um exemplo do caso 3, e a solução é $O(n^{\log_4 16}) = O(n^2)$.