Projeto e Análise de Algoritmos

Flávio L. C. de Moura

04 de maio de 2023

O algoritmo *quicksort*, assim como *merge sort*, utiliza o paradigma de divisão e conquista para ordenar um vetor $A[1..n]$, mas diferentemente de *mergesort*, seu foco está no particionamento e não na combinação das soluções. Este algoritmo, que foi inventado pelo cientista da computação britânico Charles Antony Richard Hoare em 1960, utiliza as seguintes etapas para ordenar um subvetor $A[p..r]$ $(1 \leq p \leq r \leq n)$:

1. **Dividir**: Esta etapa consiste em particionar o vetor $A[p..r]$ em dois subvetores (possivelmente vazios) $A[p..q-1]$ e $A[q+1..r]$ tais que cada elemento do subvetor $A[p..q-1]$ é menor ou igual a $A[q]$, que por sua vez também é menor ou igual do que cada elemento do subvetor $A[q+1..r]$. Esta etapa também computa o índice q do particionamento.

2. **Conquistar**: Esta etapa consiste em recursivamente ordenar os subvetores $A[p..q-1]$ e $A[q+1..r]$.

Assim, a ideia do algoritmo pode ser apresentada a partir do pseudocódigo a seguir[1]:

```
if p < r then
    q ← partition(A, p, r);
    quicksort(A, p, q − 1);
    quicksort(A, q + 1, r);
end
```

Algoritmo 1: $quicksort(A, p, r)$

A ordenação do vetor $A[1..n]$ é, então, obtida pela chamada $quicksort(A, 1, n)$.
O particionamento do vetor consiste na principal etapa do algoritmo *quicksort*:

```
x ← A[r] ;                                                    // o pivô
i ← p − 1 ;                          // o maior índice do subvetor a ser ordenado
for j = p to r − 1 ;                 // percorre todos os elementos exceto o pivô
 do
    if A[j] ≤ x ;                      // este elemento é menor ou igual ao pivô?
     then
        i ← i + 1 ;          // em caso afirmativo, incrementa o tamanho do subvetor com
         elementos menores ou iguais ao pivô
        troca A[i] com A[j] ;                  // coloca este elemento neste subvetor
    end
end
troca A[i + 1] com A[r] ;                        // o pivô vai para a posição correta
retorna i + 1 ;                                        // retorna a posição do pivô
```

Algoritmo 2: $partition(A, p, r)$

O algoritmo *partition* reorganiza o subvetor $A[p..r]$ *in place*, ou seja, sem alocar espaço adicional, e retorna o índice da posição que divide o subvetor $A[p..r]$ entre os elementos que são menores ou iguais ao pivô, e maiores do que o pivô.

Exercício 1. *Prove a seguinte invariante de laço, e conclua que o algoritmo* partition *é correto:*

Antes de cada iteração do laço ***for*** *(linhas 3-10), para todo k, temos:*

1. *Se $p \leq k \leq i$, então $A[k] \leq x$;*
2. *Se $i+1 \leq k \leq j-1$, então $A[k] > x$;*
3. *Se $k = r$, então $A[k] = x$.*

O número de comparações (linha 5) $T^p(n)$ feitas por *partition* em um vetor com n elementos é $T^p(n) = \sum_{j=1}^{n-1} 1 = n - 1 = \Theta(n)$. Qual seria, então, o tempo de execução de *quicksort*? O pior caso de *quicksort* ocorre quando o particionamento gera um vetor vazio, e outro com $n-1$ elementos. Neste caso, dizemos que o particionamento é *desbalanceado*. Se assumirmos que este desbalanceamento ocorre em cada chamada recursiva, podemos modelar o tempo de execução $T_w^{qs}(n)$ pela seguinte equação de recorrência:

$$T_w^{qs}(n) = T_w^{qs}(n-1) + T_w^{qs}(0) + T^p(n)$$

ou seja,

$$T_w^{qs}(n) = T_w^{qs}(n-1) + T_w^{qs}(0) + \Theta(n)$$

Como não há trabalho a fazer em um vetor vazio, temos que $T_w^{qs}(0) = \Theta(1)$, e portanto a recorrência acima pode ser reescrita como:

$$T_w^{qs}(n) = T_w^{qs}(n-1) + \Theta(n) \tag{1}$$

Como exercício, mostre que $T_w^{qs}(n) = \Theta(n^2)$, e observe que esta situação ocorre, por exemplo, quando o vetor dado como argumento já está ordenado.

Por outro lado, quando o particionamento é balanceado temos o melhor caso para *quicksort*. Agora, o particionamento divide o problema original em dois subproblemas sendo um de tamanho $\lfloor \frac{n}{2} \rfloor$, e outro de tamanho $\lceil \frac{n}{2} - 1 \rceil$, o que nos dá a recorrência:

$$T_b(n) = T_b(\lfloor \frac{n}{2} \rfloor) + T_b(\lceil \frac{n}{2} - 1 \rceil) + \Theta(n)$$

Se ignorarmos as funções de aproximação e a subtração por 1, temos a recorrência:

$$T_b(n) = 2.T_b(\frac{n}{2}) + \Theta(n)$$

que, pelo Teorema Mestre, tem solução $T_b(n) = \Theta(n. \lg n)$.

Assim, o tempo de execução de *quicksort* depende, por exemplo, do fato do particionamento estar balanceado ou não: Em caso afirmativo, *quicksort* é assintoticamente tão rápido quanto *mergesort*, mas se o particionamento não estiver balanceado, *quicksort* tem o mesmo comportamento assintótico de *Insertion sort* no pior caso.

O que ocorre se o particionamento é sempre da ordem de 9-1, *i.e.* 90% dos elementos são menores ou iguais ao pivô, e apenas 10% são maiores do que o pivô? Neste caso, temos a recorrência

$$T(n) = T(9n/10) + T(n/10) + c.n$$

podemos usar a árvore de recorrência para concluir que $T(n) = O(n. \lg n)$. Esta resposta sugere que a complexidade do caso médio para *quicksort* está mais próxima do melhor caso do que do pior caso.

Por fim, como fica a complexidade assintótica de *quicksort* se tivermos uma combinação de particionamentos? Mostraremos que a complexidade do pior caso é, ainda, quadrática.

Inicialmente estabeleceremos a cota superior, e escreveremos $T(n)$ ao invés de $T_w^{qs}(n)$. Para isto, considere a recorrência

$$T(n) \leq \max_{0 \leq q < n} (T(q) + T(n-q-1)) + \Theta(n) \tag{2}$$

A partir da solução vista anteriormente quando $q = 0$, parece razoável acreditar que $T(n) \leq c.n^2$ para alguma constante positiva c. Substituindo esta possível solução, temos

$$T(n) \leq \max_{0 \leq q < n} (c(q^2 + (n-q-1)^2) + \Theta(n) = c. \max_{0 \leq q < n} (q^2 + (n-q-1)^2) + \Theta(n)$$

e a expressão $q^2 + (n-q-1)^2$ assume valor máximo quando $q = 0$ ou $q = n-1$ (Exercício!), portanto

$$T(n) \leq c. \max_{0 \leq q < n} (q^2 + (n-q-1)^2) + \Theta(n) \leq (n-1)^2$$

Assim, podemos escrever $T(n) \leq c.n^2 - c.(2n-1) + \Theta(n) \leq c.n^2$, já que podemos tomar c grande o suficiente para dominar $\Theta(n)$. Ou seja, $T(n) = O(n^2)$. A prova da cota inferior fica como exercício:

Exercício 2. *Mostre que a recorrência* $T(n) = \max_{0 \leq q < n} (T(q) + T(n-q-1)) + \Theta(n)$ *tem solução* $T(n) = \Theta(n^2)$. *Considerando o que já foi feito anteriormente, falta apenas mostrar que* $T(n) = \Omega(n^2)$.

Exercício 3. *Mostre que a complexidade de* quicksort*, no melhor caso, é* $\Omega(n \lg n)$.

Exercício 4. *Mostre que o algoritmo* quicksort *é correto.*

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.