Análise do algoritmo Quicksort

[Veja a versão desta página em formato perguntas-e-respostas]

Este página é inspirada no capítulo 8 de CLR. Ela trata do seguinte problema de ordenação de vetor:

> PROBLEMA DA ORDENAÇÃO: rearranjar um vetor $A[p..r]$ de modo que ele fique em ordem crescente.

O algoritmo QUICKSORT, inventado por C.A.R. Hoare, usa o paradigma da divisão e conquista para resolver o problema Ele é muito rápido em média, mas lento no pior caso.

Veja o verbete *Quicksort* na Wikipedia.

O subproblema da separação

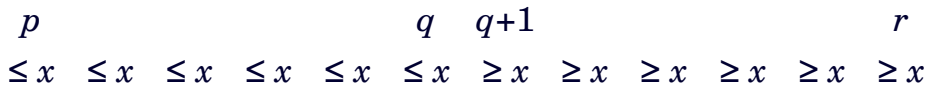
O coração do algoritmo QUICKSORT está no seguinte subproblema, que formularemos de maneira propositalmente vaga:

> rearranjar $A[p..r]$ de modo que todos os elementos pequenos fiquem na parte esquerda do vetor e todos os elementos grandes fiquem na parte direita.

Este é o subproblema da separação. O ponto de partida para a solução do subproblema é a escolha de um "pivô", digamos x: os elementos do vetor que forem maiores que x serão considerados grandes e os demais serão considerados pequenos. É preciso escolher x de tal modo que cada uma das duas partes do vetor rearranjado seja *estritamente menor* que o vetor todo.

O seguinte algoritmo resolve o subproblema da separação da seguinte maneira: supondo $p < r$, o algoritmo rearranja os elementos de $A[p..r]$ e devolve um índice q tal que $p \le q < r$ e, para algum x,

- $A[i] \le x$ para cada i em $p \,..\, q$
- $A[j] \ge x$ para cada j em $q+1 \,..\, r$



A seguinte versão do algoritmo adota como pivô x o valor inicial de $A[p]$.

```
SEPARE (A, p, r)
1   x ← A[p]
2   i ← p−1
3   j ← r+1
4   enquanto 0 = 0 faça
5       repita j ← j−1
6           até que A[j] ≤ x
7       repita i ← i+1
8           até que A[i] ≥ x
9       se i < j
10          então troque A[i] ↔ A[j]
11          senão devolva j
```

Para entender como e por que o algoritmo funciona como deveria, observe que no início de cada iteração do loop que começa na linha 4 temos as seguintes propriedades invariantes:

$$A[p..i] \le x \,, \quad i < j \,, \quad A[j..r] \ge x \,.$$

(Se eu estivesse programando pra valer, escreveria estas relações, como comentário, logo depois da linha 4.)

Na *última* passagem pela linha 4, o vetor $A[i+1 \,..\, j-1]$ consiste em

- zero ou mais elementos $< x$ seguidos de zero ou mais elementos $> x$ ou
- zero ou mais elementos $< x$, seguidos de exatamente um elemento

igual a x , seguido de zero ou mais elementos $> x$.

No primeiro caso, o algoritmo chega à linha 11 com $j = i-1$.

$$\begin{array}{cccccccccccc} p & & & & & j & i & & & & & r \\ \le x & \le x & \le x & \le x & \le x & \le x & \ge x & \ge x & \ge x & \ge x & \ge x & \ge x \end{array}$$

No segundo caso, o algoritmo chega à linha 11 com $j = i$.

$$\begin{array}{cccccccccccc} p & & & & & j{=}i & & & & & & r \\ \le x & \le x & \le x & \le x & \le x & = x & \ge x & \ge x & \ge x & \ge x & \ge x & \ge x \end{array}$$

Não é difícil perceber agora que o algoritmo faz o que prometeu; em particular, que $p \le j < r$ na linha 11.

Exercícios

1. Suponha que todos os elementos do vetor $A[p..r]$ são iguais entre si. Quantas vezes a linha 4 do algoritmo SEPARE é executada? Qual o valor de j que o algoritmo devolve?
2. [Este exercício faz parte da análise do desempenho médio do QUICKSORT.] Suponha que $A[p..r]$ é uma permutação de $1, 2, \ldots, n$. Que índice SEPARE devolve se o valor inicial de $A[p]$ for 1? E se o valor inicial de $A[p]$ for 2? E se o valor inicial de $A[p]$ estiver entre 3 e n?
3. É verdade que a relação $p \le j < r$ vale no início início de cada iteração do SEPARE? Ela vale quando a linha 11 é executada?

O consumo de tempo do algoritmo de separação

Quanto tempo o algoritmo SEPARE consome? Comecemos contando o número total de execuções das linhas 6 e 8. (Note que as execuções do bloco de linhas 5-6 não são, necessariamente, consecutivas. O mesmo vale para o bloco de linhas 7-8.) Se a o número total de execuções da linha 6 e b o número total de execuções da linha 8, então $a+b \le n+2$, sendo $n = r-p+1$.

Analogamente, a soma do número total de execuções da linha 5 e da linha 7 não passa de $n+2$.

Agora considere as demais linhas. O número total de execuções da linha 9 não passa de $n+1$. O número de execuções da linha 10 não passa de n, o número de execuções da linha 4 não passa de $n+1$, a as linhas 1 a 3 são executadas 1 vez cada.

Se a execução de cada linha consome 1 unidade de tempo, o consumo total de tempo não passa de

$$5n + 6 .$$

Se cada execução de cada linha consumisse uma quantidade de tempo diferente de 1 (mas independente de n), o consumo total não passaria de um múltiplo de n, ou seja, o consumo estaria em

$$O(n) .$$

Portanto, o algoritmo é linear.

O algoritmo Quicksort

O algoritmo QUICKSORT recebe um vetor $A[p..r]$ e rearranja o vetor em ordem crescente.

```
QUICKSORT (A, p, r)
1   se  p < r  então
2       q ← SEPARE (A, p, r)
3       QUICKSORT (A, p, q)
4       QUICKSORT (A, q+1, r)
```

Note que $q \geq p$ e $q < r$; portanto os vetores $A[p..q]$ e $A[q+1 .. r]$ são estritamente menores que o vetor original $A[p..r]$. Isso garante que a execução do algoritmo termina, mais cedo ou mais tarde.

(Se você estivesse programando pra valer, que comentários escreveria no seu programa? Eu escreveria só dois comentários. Antes da linha 1 eu diria "estamos supondo $p<r$". Depois da linha 2 eu diria "$p \leq q < r$ e $A[i] \leq A[j]$ para cada i em $p..q$ e cada j em $q+1..r$".)

Exercícios

1. Incorpore o código de SEPARE ao código de QUICKSORT, ou seja, reescreva QUICKSORT substituindo a invocação de SEPARE pelo seu código. Faça os ajustes necessários.

2. Submeta um vetor crescente ao QUICKSORT. Mostre que o algoritmo consome tempo proporcional a n^2.

3. Seja f a função definida pela recorrência $f(n) = 5n + 7 + f(1) + f(n-1)$, com $f(1) = 1$. Mostre que $f(n)$ é $O(n^2)$. Mostre que $f(n)$ é $\Omega(n^2)$.

4. Seja f a função definida pela recorrência $f(n) = 5n + 7 + f(\lceil n/2 \rceil) + f(\lfloor n/2 \rfloor)$, com $f(1) = 1$. Mostre que $f(n)$ é $O(n \lg n)$. Mostre que $f(n)$ é $\Omega(n \lg n)$.

5. Suponha dado um algoritmo SEPAR que faça o seguinte: ao receber um vetor $A[p..r]$, devolve um índice q no intervalo $p..r$ depois de rearranjar o vetor de tal modo que $A[p..q-1] \le A[q] < A[q+1..r]$. Escreva uma versão do QUICKSORT que use o algoritmo SEPAR no lugar de SEPARE.

6. [CLR 8-4] O algoritmo QUICKSORT, tal como apresentado acima, contém duas chamadas recursivas. Escreva uma versão QUICKSORT2 do algoritmo em que a segunda chamada é eliminada e substituída por um "enquanto".
   Durante a execução do QUICKSORT2, o computador usa um pilha para administrar a recursão. Mostre que a pilha pode chegar a ter $\Omega(n)$ elementos quando QUICKSORT2 processa um vetor com n elementos.

   Modifique QUICKSORT2 de modo que a pilha tenha $O(\log n)$ elementos, mesmo no pior caso.

Desempenho do Quicksort no pior caso

Quanto tempo o algoritmo consome no pior caso? Como vimos acima, podemos supor que SEPARE não consome mais que $5n+6$ unidades de tempo, sendo $n = r-p+1$. Então o consumo de tempo do QUICKSORT no pior caso, digamos $T(n)$, satisfaz a seguinte recorrência

$$T(n) = 5n + 7 + \max_{0<k<n} (T(k) + T(n-k)) \qquad (*)$$

onde o máximo é tomado sobre todos os possíveis valores de k no intervalo $1 .. n-1$. Podemos supor $T(1) = 1$. Assim, por exemplo, $T(2) \le 10+7+T(1)+T(1) = 19$. Outro exemplo: $T(3) \le 15+7+T(1)+T(2) = 42$. Mais um exemplo: $T(4) \le 20+7+T(1)+T(3) = 70$, uma vez que $T(1)+T(3) = 43 > 38 = T(2)+T(2)$. Em geral, vamos mostrar que

$$T(n) \le 5n^2$$

para $n = 1, 2, 3, \ldots$ A desigualdade é certamente verdadeira quando $n = 1, 2, 3$. Agora suponha que $n > 3$ e suponha que a desigualdade vale para $T(i)$ sempre que $i < n$. Teremos então

$$\begin{aligned} T(n) &= 5n + 7 + \max_k (T(k) + T(n-k)) \\ &\le 5n + 7 + \max_k (5k^2 + 5(n-k)^2) \\ &= 5n + 7 + 5\,(1 + (n-1)^2) \\ &= 5n^2 - 5n + 17 \\ &\le 5n^2\,. \end{aligned}$$

O terceiro passo da prova segue da seguinte observação: para n fixo e k variando entre 1 e $n-1$, a expressão $k^2 + (n-k)^2$ tem valor máximo nos pontos $k = 1$ e $k = n-1$ (e atinge o mínimo quando k está próximo de $n/2$). O último passo da prova está correto pois quando $n > 3$ tem-se $-5n+17 \le 0$.

A cota superior de $5n^2$ não é exagerada. De fato, é fácil mostrar que $T(n) \ge 2n^2$. Portanto $T(n)$ está em $O(n^2)$ e também em $\Omega(n^2)$.

Se deixarmos de lado a hipótese artificial de que cada linha "simples" do algoritmo consome 1 unidade de tempo, as conclusões ainda serão as mesmas. A recorrência pode ser reescrita como $T(n) = \max_{0<k<n} (T(k) + T(n-k)) + O(n)$ e daí se deduz que $T(n)$ está em $O(n^2)$. Conclusão: no pior caso, o algoritmo QUICKSORT é quadrático. Por que, então, o algoritmo é tão popular?

Exercícios

1. Imite a prova de $T(n) \le 5n^2$ para tentar provar que $T(n) \le 1000n$. Repita com $T(n) \le 100n \lg n$. Repita com $T(n) \le 2n^2$.
2. Seja T a função definida pela recorrência (*). Mostre que $T(n) \ge 2n^2$ para todo número natural $n \ge 1$. Deduza daí que $T(n)$ é $\Omega(n^2)$.
3. Considere a variante do algoritmo QUICKSORT que usa o algoritmo SEPAR no lugar de SEPARE (veja exercício acima). Escreva a recorrência que governa o consumo de tempo de pior caso desta variante do QUICKSORT. Resolva a recorrência.
4. Mostre que $(n!)^2 > n^n$ para todo número natural não nulo n. [Este exercício envolve cálculo semelhante ao que fizemos acima.]

Desempenho típico do Quicksort

O melhor desempenho do QUICKSORT ocorre quando *todas* as invocações de SEPARE dividem o vetor **na proporção (1/2)-para-(1/2)**, ou seja, quando todas as invocações devolvem um índice que está a meio caminho entre p e r. O consumo de tempo do algoritmo nesse caso, digamos $S(n)$, satisfaz a recorrência

$$S(n) = S(\lceil n/2 \rceil) + S(\lfloor n/2 \rfloor) + 5n + 7$$

para todo natural $n \geq 2$ (veja a definição de *teto*). A exemplo do que fizemos em outra ocasião, podemos mostrar que

$$S(n) \leq 12\, n \log_2 n$$

para todo natural $n \geq 3$.

O comportamento não é muito diferente quando o vetor é dividido de maneira menos equilibrada. Suponha, por exemplo, que todas as invocações de SEPARE dividem o vetor **na proporção (1/9)-para-(8/9)**. Nesse caso, o consumo de tempo, digamos $R(n)$, satisfaz recorrência da forma

$$R(n) = R(\lceil n/9 \rceil) + R(\lfloor 8n/9 \rfloor) + 5n + 7$$

para todo natural $n \geq 2$. Mostremos que

$$R(n) \leq 7\, n \log_{9/8} n$$

para todo $n \geq 2$. Supondo $R(1) = 1$, é fácil verificar que a desigualdade vale quando $n = 2, 3$. Agora tome $n \geq 4$. Então, por hipótese de indução, e usando sempre log na base 9/8,

$$\begin{aligned}
R(n) &= 7\,\lceil n/9 \rceil \log \lceil n/9 \rceil + 7\,\lfloor 8n/9 \rfloor \log \lfloor 8n/9 \rfloor + 5n + 7 \\
&\leq 7\,(\lceil n/9 \rceil + \lfloor 8n/9 \rfloor) \log \lfloor 8n/9 \rfloor + 5n + 7 \\
&= 7\, n \log (8n/9) + 5n + 7 \\
&= 7\, n \log (n) + 7\, n \log (8/9) + 5n + 7 \\
&= 7\, n \log n - 7\, n + 5n + 7 \\
&= 7\, n \log n - 2n + 7 \\
&< 7\, n \log n\ .
\end{aligned}$$

(Como seria de se esperar, o coeficiente de n lg n é maior nesse caso que no anterior: $7 \log_{9/8} n > 7{\times}5.884 \log_2 n > 41 \log_2 n > 12 \log_2 n$.)

A análise dos dois casos acima — a divisão na proporção (1/2)-para-(1/2) e a

divisão na proporção (1/9)-para-(8/9) — sugere a seguinte conclusão: se SEPARE sempre divide o vetor em alguma proporção tn-para-$(1–t)n$, o consumo de tempo do QUICKSORT é

$$O(n \lg n).$$

(É claro que a constante escondida sob a notação O é tanto maior quanto mais t se afasta de 1/2.)

As conclusões valem mesmo que o valor de t seja diferente em cada invocação de SEPARE. Essas observações sugerem que o consumo de tempo "típico" do QUICKSORT está em $O(n \lg n)$.

Exercícios

1. Mostre que para todo número natural $n \geq 3$ tem-se $\lceil n/9 \rceil \leq \lfloor 8n/9 \rfloor$.
2. Mostre que para todo número natural $n \geq 2$ tem-se $\lceil n/9 \rceil + \lfloor 8n/9 \rfloor = n$.
3. Invente uma variante de SEPARE que funcione assim: rearranja $A[p..r]$ e devolve q tal que $A[p..q] \leq A[q+1..r]$ e $p+2 \leq q < r-2$. (É claro que isso só faz sentido se o vetor tiver pelo menos 6 elementos; para vetores menores, use o SEPARE usual.) Quanto tempo o seu algoritmo consome? Se QUICKSORT usar o seu algoritmo no lugar de SEPARE, qual será o seu consumo de tempo no pior caso?
4. Seja a um número real no intervalo $(0,½)$. Invente uma variante de SEPARE que funcione assim: rearranja $A[p..r]$ e devolve q tal que $A[p..q] \leq A[q+1..r]$ e $p+an \leq q < r-an$, sendo $n = r-p+1$. Quanto tempo o seu algoritmo consome? Se QUICKSORT usar o seu algoritmo no lugar de SEPARE, qual será o seu consumo de tempo no pior caso?

Quicksort aleatorizado

Se o vetor $A[p..r]$ for crescente ou aproximadamente crescente, o algoritmo QUICKSORT consome $\Omega(n^2)$ unidades de tempo. Para evitar casos desfavoráveis como esses, podemos escolher o pivô aleatoriamente, como faremos a seguir.

Suponha dado um algoritmo auxiliar RANDOM que funciona assim: ao receber um par $p \leq r$ de números naturais, devolve um número aleatório no intervalo $p..r$, sendo todos os números do intervalo igualmente prováveis. Existem implementações (aproximadas) de RANDOM que consomem uma quantidade de tempo constante, ou seja, independente de p e r. Esse algoritmo auxiliar pode ser usado para construir versões aleatorizadas

(= *randomized*) de SEPARE e QUICKSORT:

SEPARE-ALEATORIZADO (A, p, r)

1 $i \leftarrow$ RANDOM(p, r)
2 troque $A[p] \leftrightarrow A[i]$
3 SEPARE (A, p, r)

QUICKSORT-ALEATORIZADO (A, p, r)

1 se $p < r$
2 então $q \leftarrow$ SEPARE-ALEATORIZADO (A, p, r)
3 QUICKSORT-ALEATORIZADO (A, p, q)
4 QUICKSORT-ALEATORIZADO $(A, q+1, r)$

No pior caso, QUICKSORT-ALEATORIZADO consome tanto tempo quanto QUICKSORT. Em média, entretanto, QUICKSORT-ALEATORIZADO é muito mais rápido: seu consumo médio é de $O(n \lg n)$ unidades de tempo, como mostraremos a seguir.

A aplicação do subalgoritmo SEPARE-ALEATORIZADO ao vetor $A[p..r]$ determina dois subvetores. Diremos que o SEPARE-ALEATORIZADO produz resultado $(i, n-i)$ se os dois vetores resultantes tiverem tamanhos i e $n-i$, onde $n = r-p+1$. Os possíveis resultados são

$$(1, n-1), \quad (2, n-2), \quad \ldots, \quad (n-2, 2), \quad (n-1, 1) .$$

Ao todo, temos $n-1$ possíveis resultados. Resta saber qual a *probabilidade* de cada um.

Trataremos apenas do caso em que o vetor *não tem elementos repetidos*. (Acho que os resultados não valem sem essa hipótese.) Sob essa hipótese, a probabilidade de cada caso *só depende do valor de* $A[p]$ antes no fim da linha 2 de SEPARE-ALEATORIZADO. Não é difícil verificar que

- se $A[p]$ é o menor elemento do vetor então SEPARE-ALEATORIZADO produz resultado $(1,n-1)$ e
- se $A[p]$ é o i-ésimo menor elemento do vetor, para qualquer i em $\{2,3, \ldots,n\}$, então SEPARE-ALEATORIZADO produz resultado $(i-1,n-i+1)$.

(Note que SEPARE-ALEATORIZADO produz o mesmo resultado quer $A[p]$ seja o menor ou o segundo menor elemento do vetor.)

Em virtude da aleatorização, $A[p]$ tem igual probabilidade de ser o menor, o segundo menor, etc., o n-ésimo menor elemento do vetor. Concluímos que o resultado $(1,n-1)$ tem probabilidade $2/n$ e cada um dos demais resultados

tem probabilidade $1/n$.

Denotemos por $E(n)$ o valor esperado do consumo de tempo do QUICKSORT-ALEATORIZADO. Então

$$E(n) \;=\; 5n + 9 \;+\; \frac{s_1 + s_1 + s_2 + \ldots + s_{n-1}}{n}\,,$$

onde $s_i = E(i)+E(n-i)$ para $i = 1,2,3,\ldots,n-1$. (O termo s_1 aparece duas vezes, mas nossos cálculos mostrarão que isso não faz diferença para a solução assintótica da recorrência.) Se agruparmos termos iguais teremos a recorrência

$$E(n) \;=\; 5n + 9 \;+\; \frac{E(1)+E(n-1) + 2\,\sum_{0<i<n} E(i)}{n}\,. \qquad (**)$$

A análise de uma recorrência mais simples (veja exercício abaixo) sugere que a solução está em $O(n \lg n)$. Alguns cálculos confirmam esta suspeita:

$$E(n) \;\le\; 20\, n \lg n$$

para $n = 2, 3, \ldots$

Exercícios

1. Seja F a função sobre os naturais definida pela recorrência $F(n) = n + 2\,\sum_{0<i<n} F(i) \,/\, n$ com $F(1) = 1$. Mostre que $F(n) \le 10\, n \lg n$ para $n = 2, 3, \ldots$ (Dica: $\lg i \le \lg(n/2)$ quando $i \le n/2$ e $\lg i \le \lg n$ quando $i < n$.) [Solução]
2. Verifique que a função E definida pela recorrência (**) é tal que $E(n) \le 20\, n \lg n$ para $n = 2, 3, \ldots$

Aula do Leiserson (em video) sobre o Quicksort
Veja aulas de Análise de Algoritmos no MIT

URL of this site: http://www.ime.usp.br/~pf/analise_de_algoritmos/
Last modified: Thu Sep 5 08:13:43 BRT 2013
Paulo Feofiloff
Departamento de Ciência da Computação
Instituto de Matemática e Estatística da USP


W3C HTML 4.01
W3C CSS