MAC 5711 - Análise de Algoritmos

Departamento de Ciência da Computação

Primeiro semestre de 2004

k-ésimo mínimo

O problema do k-ésimo mínimo consiste no seguinte:

> **Problema:** Dados uma seqüência de n números distintos e um inteiro k entre 1 e n, imprimir o k-ésimo menor número da seqüência.

Suponha que a seqüência é dada em um vetor $v[1\,..\,n]$. Uma solução óbvia para o problema é ordenar o vetor e imprimir o elemento $v[k]$. Essa solução consome no pior caso tempo $\Theta(n \log n)$ (basta usar para ordenar o `mergesort` ou o `heapsort`).

Uma segunda idéia é usar o `particiona` do `quicksort`. A primeira maneira de usá-lo consiste em simplesmente particionar o vetor e procurar recursivamente o k-ésimo mínimo ou numa ou na outra "metade", dependendo da partição devolvida e do valor de k. Em pseudo-código, o algoritmo ficaria assim:

```
int seleciona (int v[], int p, int r, int k) {
  int i;

  if (p == r)
    return v[p];
  else {
    q = particiona(v,p,r);                       (*)
    if (k <= q-p+1)
      return seleciona(v,p,q,k);
    else
      return seleciona(v,q+1,r,k-(q-p+1));
}
```

A análise de pior caso do algoritmo acima é semelhante a do `quicksort` e resulta na recorrência

$$T(n) = T(n-1) + \Theta(n),$$

cuja solução é $\Theta(n^2)$. Se usarmos o `particiona` aleatório, usado na versão aleatória do `quicksort`, podemos fazer uma análise de caso médio. Para tanto, adicione ainda um pré-processamento que rearranja o vetor de entrada de acordo com uma permutação dele escolhida de acordo com a distribuição uniforme. O pior caso desse algoritmo modificado continua, como no caso do `quicksort`, quadrático. Mas, vamos verificar que essa versão aleatória do algoritmo acima, como a do `quicksort`, é em geral bem mais rápida que o seu pior caso indica. Especificamente, sua complexidade de caso médio é $O(n)$.

Lembre-se que o pivô do `particiona` aleatório é um elemento de $v[p\,..\,r]$ escolhido com probabilidade uniforme. Com isso, o valor da variável i na linha `(*)` é p com probabilidade $2/n$ e é um t em $\{p+1,\ldots,q-1\}$ com probabilidade $1/n$, onde $n := q-p+1$, se usarmos a implementação do `particiona` vista em aula. Disso, deduzimos que a complexidade esperada da versão aleatória do algoritmo acima satisfaz a seguinte recorrência:

$$T(n) \;\le\; \frac{2}{n}\max\{T(1),T(n-1)\} + \frac{1}{n}\sum_{i=2}^{n-1}\max\{T(i),T(n-i)\} + \Theta(n).$$

Supondo que a função $T(n)$ é não-decrescente, usando o fato que $T(n) = O(n^2)$ (da análise de pior caso) e substituindo a notação assintótica, obtemos a seguinte recorrência:

$$T(n) \;\leq\; \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} T(i) \;+\; n.$$

(Como sempre, está implícito que $T(1) = 1$.)

Vamos mostrar que $T(n) = O(n)$ ou, mais especificamente, vamos mostrar por indução em n que $T(n) \leq 4n$ para todo $n \geq 1$.

Para $n = 1$, temos que $T(1) = 1 \leq 4$, portanto a hipótese vale. Considere $n \geq 2$ e suponha que $T(m) \leq 4m$ para todo $m < n$. Pela recorrência e pela hipótese de indução, temos que

$$\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} 4i \;+\; n \\
&= \frac{4}{n} \left(n - 1 + \lfloor \frac{n}{2} \rfloor\right) \left(n - 1 - (\lfloor \frac{n}{2} \rfloor - 1)\right) + n \\
&\leq \frac{4}{n} \left(n - 1 + \frac{n}{2}\right) \left(n - \frac{n+1}{2}\right) + n \\
&\leq \frac{4}{n} \left(\frac{3n-2}{2}\right) \frac{n-1}{2} + n \\
&= \frac{1}{n} \left(3n^2 - 3n + 2\right) + n \\
&= 4n - 3 + \frac{2}{n}.
\end{aligned}$$

Como $n \geq 2$, temos que $\frac{2}{n} \leq 1$ e portanto $T(n) \leq 4n$. Isso conclui a prova por indução de que $T(n) \leq 4n$ para todo $n \geq 1$. Ou seja, a complexidade de caso médio do algoritmo acima é $O(n)$. (Supondo que há um pré-processamento que substitui a entrada por uma permutação dela, de acordo com a distribuição uniforme.)

Algoritmo linear para o k-ésimo mínimo

O algoritmo descrito acima consome tempo no pior caso $\Theta(n^2)$ e, em média, consome tempo $O(n)$. Descreveremos agora um algoritmo para o problema do k-ésimo mínimo que, no pior caso, consome tempo $O(n)$.

O algoritmo explora a idéia de encontrar um bom pivô para fazer a partição. O pivô é determinado da seguinte maneira. No que segue, n denota, como sempre, o número de elementos no trecho do vetor em questão, ou seja, $n = r - p + 1$. Primeiramente, o algoritmo divide o vetor $v[p..r]$ em $\lceil \frac{n}{5} \rceil$ grupos de 5 elementos (o último grupo na verdade tem entre 1 e 5 elementos, dependendo do valor de n). Para cada grupo, o algoritmo computa a mediana dos (no máximo) 5 elementos do grupo e inclui tal mediana em um vetor auxiliar. Feito isso, o algoritmo determina a mediana das medianas, através de uma chamada recursiva com o vetor auxiliar como entrada, $p = 1$, $r = \lceil \frac{n}{5} \rceil$ e $k = \lfloor \frac{\lceil \frac{n}{5} \rceil + 1}{2} \rfloor$. Tal chamada recursiva devolve um elemento x, que é a chamada mediana das medianas. O algoritmo prossegue chamando o `particiona` com x como pivô e fazendo o que o algoritmo anterior fazia: recursivamente procura o k-ésimo mínimo no lado esquerdo ou direito da partição, dependendo da relação entre k e o índice devolvido pelo `particiona`.

Pode-se calcular a mediana de 5 (ou menos) elementos em $O(1)$. Com isso, o tempo consumido por este algoritmo para calcular os elementos do vetor auxiliar é $\Theta(n)$, já que há $\lceil \frac{n}{5} \rceil \leq \frac{n+4}{5} = \Theta(n)$ grupos. O `particiona` consome tempo $\Theta(n)$. Assim, fora as chamadas recursivas, o algoritmo consome tempo $\Theta(n)$.

O custo da primeira chamada recursiva é $T(\lceil \frac{n}{5} \rceil)$. Para determinar o custo da segunda chamada recursiva, vamos mostrar que a partição do vetor em relação a x deixa pelo menos $\frac{3n}{10} - 3$ elementos do vetor de cada lado. Vamos na verdade mostrar que o lado esquerdo da partição terá pelo menos $\frac{3n}{10} - 3$ elementos. Fica de exercício verificar que o mesmo vale para o lado direito.

Existem $t := \lfloor \frac{\lceil \frac{n}{5} \rceil + 1}{2} \rfloor - 1 \geq \frac{n}{10} - 1$ medianas menores que x. Ou todas estas determinam, cada uma, três elementos menores que x ou uma delas é a mediana do último grupo, e este tem menos do que 5 elementos. Mas neste caso, considerando os dois elementos do grupo de x menores que x, temos também pelo menos três elementos por cada uma das t medianas. Ou seja, existem pelo menos $3t = \frac{3n}{10} - 3$ elementos menores que x no vetor. Deduze-se disso que a complexidade de pior caso do algoritmo satisfaz a seguinte recorrência:

$$T(n) \;=\; T(\lceil \frac{n}{5} \rceil) + T(\lfloor \frac{7n}{10} \rfloor + 3) + \Theta(n),$$

cuja solução é $O(n)$.

De fato, suponha que, para algum $c > 0$, temos que $T(m) \leq cm$ para todo $1 \leq m < n$. Então

$$\begin{aligned}
T(n) &\leq c\lceil \frac{n}{5} \rceil + c\big(\lfloor \frac{7n}{10} \rfloor + 3\big) + n \\
&\leq c\left(\frac{n}{5} + 1\right) + c\left(\frac{7n}{10} + 3\right) + n \\
&= \frac{cn}{5} + c + \frac{7cn}{10} + 3c + n \\
&= \frac{9cn}{10} + 4c + n \\
&= cn - \frac{cn}{10} + n + 4c.
\end{aligned}$$

Se $c \geq 50$ e $n \geq 50$, podemos concluir que $T(n) \leq cn$. Escolha então um $c \geq 50$ tal que $T(m) \leq cm$ para todo $m < 50$. A prova segue por indução em n. A base da indução é $n < 50$, onde a afirmação vale pela escolha de c, e o passo da indução, que se aplica quando $n \geq 50$, é a dedução acima, que vale pois $c \geq 50$ e $n \geq 50$. Disso concluímos que $T(n) \leq cn$ para todo $n \geq 1$, ou seja, $T(n) = O(n)$.

Exercícios

1. Projete uma versão do `quicksort` que consuma no pior caso $O(n \log n)$.
2. Use a implementação do algoritmo linear do k-ésimo mínimo disponível na página para fazer um estudo experimental que compare esta à segunda idéia para entradas uniformemente distribuídas.