MAC 338 - Análise de Algoritmos

Departamento de Ciência da Computação

Primeiro semestre de 2003

k-ésimo mínimo

O problema do k-ésimo mínimo consiste no seguinte:

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

Supondo que a seqüência é dada em um vetor v, 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 q, int k) {
  int i;

  if (p == q)
    return v[p];
  else {
    i = particiona(v,p,q);                          (*)
    if (k <= i-p+1)
      return seleciona(v,p,i,k);
    else
      return seleciona(v,i+1,q,k-(i-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 probabilística do `quicksort`, podemos fazer uma análise de caso médio e perceber que a versão probabilística 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 $O(n)$. (Note que, como no `quicksort`, a complexidade de pior caso da versão probabilística continua sendo $\Theta(n^2)$.)

Lembre-se que o pivô do `particiona` aleatório é um elemento de $v[p\,..\,q]$ 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 probabilística do algoritmo acima satisfaz a seguinte recorrência:

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

Assumindo 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, obtermos a seguinte recorrência:

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

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

Vamos mostrar por indução em n que $T(n) = O(n)$ ou, mais especificamente, vamos mostrar que $T(n) \leq 5n$ para todo $n \geq 1$.

Para $n = 1$, temos que $T(1) = 1 \leq 5$, portanto a hipótese vale. Considere $n \geq 2$ e suponha que $T(m) \leq 5m$ para todo $m < n$. Pela recorrência e por indução, temos que

$$\begin{aligned}
T(n) &\leq \frac{2}{n} \sum_{i=\lceil \frac{n}{2} \rceil}^{n-1} 5i \;+\; n \\
&= \frac{5}{n} \left(n - 1 + \lceil \frac{n}{2} \rceil \right) \left(n - \lceil \frac{n}{2} \rceil \right) + n \\
&\leq \frac{5}{n} \left(n - 1 + \frac{n+1}{2} \right) \left(n - \frac{n}{2} \right) + n \\
&\leq \frac{5}{n} \left(\frac{3n-1}{2} \right) \frac{n}{2} + n \\
&= \frac{5}{2} \left(\frac{3n-1}{2} \right) + n \\
&= 5n - \frac{5n}{4} - \frac{5}{4} + n.
\end{aligned}$$

Como $\frac{5n}{4} + \frac{5}{4} - n \geq 0$ se e somente se $n \geq -5$, temos que $T(n) \leq 5n$, já que $n \geq 2$. Isso conclui a prova por indução de que $T(n) \leq 5n$ para todo $n \geq 1$. Ou seja, a complexidade de caso médio da versão probabilística do algoritmo acima é $O(n)$.

Exercícios

1. Mostre um algoritmo que faça no máximo $\lceil 3n/2 \rceil - 2$ comparações para encontrar o máximo e o mínimo de n números.

2. **(Exercício 10.1-2 do CLR)** Mostre que $\lceil 3n/2 \rceil - 2$ comparações são necessárias no pior caso para encontrar o máximo e o mínimo de n números. (Dica: considere quantos números são potencialmente ou o máximo ou o mínimo e investigue como uma comparação afeta esse contador.)

3. **(Exercício 10.1-1 do CLR)** Mostre que o segundo menor elemento dentre n elementos pode ser encontrado em $n + \lceil \lg n \rceil - 2$ comparações no pior caso. (Dica: use uma estratégia como a do `mergesort`.)

4. **(Exercício 10.2-1 do CLR)** Escreva uma versão iterativa (não-recursiva) do `seleciona` probabilístico visto em sala. (Dica: note que a recursão é de cauda.)

Algoritmo linear para o k-ésimo mínimo

O algoritmo descrito acima tem tempo de pior caso $\Theta(n^2)$ e tempo esperado $O(n)$. Descreveremos agora um algoritmo para o problema do k-ésimo mínimo que tem tempo de pior caso $O(n)$.

O algoritmo explora a idéia de encontrar um bom pivô para fazer a partição. O pivô é determinado da seguinte maneira. Primeiramente, o algoritmo divide o vetor $v[1\,..\,n]$ 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 e o valor $k := \lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil$. Denotando a mediana das medianas por x, 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`. Abaixo segue o código do algoritmo.

```
#define TAM_GRUPO 5

void constroi_vetor_das_medianas(int v[], int p, int q, int vaux[], int *m) {
  int aux, i, j, ini = p, fim;
  *m = 0;
  while (ini <= q) {
    fim = ini + TAM_GRUPO - 1;
    if (fim > q) fim = q;
    for (i=fim; i>ini; i--)
      for (j=ini; j<i; j++)
        if (v[j] > v[j+1]) {
          aux = v[j];
          v[j] = v[j+1];
          v[j+1] = aux;
        }
    vaux[*m] = v[(ini+fim)/2];
    (*m)++;
    ini = fim + 1;
  }
}

int seleciona_linear (int v[], int p, int q, int k) {
  int x, i, m, aux, vaux[MAX];
  if (p == q)
    return v[p];
  else {
    constroi_vetor_das_medianas(v, p, q, vaux, &m);
    x = seleciona_linear(vaux, 0, m-1, m/2);
    for (i=p; i<=q; i++)
      if (v[i] == x) v[i] = v[p];
    v[p] = x;
    i = particiona(v, p, q);
    if (k <= i-p+1)
      return seleciona_linear(v, p, i, k);
    else
      return seleciona_linear(v, i+1, q, k-(i-p+1));
  }
}
```

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 $\frac{3n}{10} - 5$ elementos do vetor de cada lado. Vamos na verdade mostrar que o lado direito da partição terá pelo menos $\frac{3n}{10} - 5$ elementos. Fica de exercício verificar que o mesmo vale para o lado esquerdo.

Existem pelo menos $t := \lceil \frac{n}{5} \rceil - \lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil + 1 = \lfloor \frac{1}{2} \lceil \frac{n}{5} \rceil \rfloor + 1$ medianas maiores ou iguais a x. Destas, $t-2$ correspondem a três elementos estritamente maiores que x: todas essas t medianas exceto o próprio x (que corresponde a apenas 2 elementos maiores que ele) e possivelmente a mediana do último grupo, que pode ter menos que 5 elementos. Portanto existem pelo menos

$$3(t-2) \;=\; 3\left(\lfloor \frac{1}{2} \lceil \frac{n}{5} \rceil \rfloor - 1\right) \;\geq\; 3\left(\frac{1}{2}\big(\lceil \frac{n}{5} \rceil - 1\big) - 1\right) \;=\; \frac{3}{2}\lceil \frac{n}{5} \rceil - \frac{3}{2} - 3 \;\geq\; \frac{3n}{10} - 5.$$

Deduze-se disso que a complexidade de pior caso do algoritmo satisfaz a seguinte recorrência:

$$T(n) \;\geq\; T(\lceil \frac{n}{5} \rceil) + T(\lceil \frac{7n}{10} \rceil + 5) + \Theta(n),$$

cuja solução é $O(n)$.

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

$$\begin{aligned} T(n) &\leq c\lceil \frac{n}{5} \rceil + c\big(\lceil \frac{7n}{10} \rceil + 5\big) + n \\ &\leq c\left(\frac{n+4}{5}\right) + c\left(\frac{7n+9}{10} + 5\right) + n \\ &= \frac{cn}{5} + \frac{4c}{5} + \frac{7cn}{10} + \frac{9c}{10} + 5c + n \\ &= cn - \frac{cn}{10} + n + \frac{17c}{10} + 5c. \end{aligned}$$

Assim, se $(c/10 - 1)n - (17/10 + 5)c \geq 0$, que corresponde a

$$n \geq 67 + \frac{670}{c - 10}, \tag{1}$$

podemos concluir que $T(n) \leq cn$. Se $c \geq 77$, a desigualdade (1) vale para todo $n \geq 77$. Seja c' tal que $T(m) \leq c'm$ para todo $m < 77$ e seja $c := \max\{77, c'\}$. Considere como base da indução o caso em que $n < 77$. Evidentemente, para tais valores, $T(n) \leq cn$ (pela escolha de c' e pela definição de c). O passo da indução é aplicado caso $n \geq 77$ e consiste nos cálculos acima. Deles concluímos que $T(n) \leq cn$, já que $n \geq 77$. Assim podemos concluir que $T(n) \leq cn$ para todo $n \geq 1$, ou seja, $T(n) = O(n)$.

Vale ressaltar que, como no caso do `quicksort`, esse segundo algoritmo tem uma performance pior na prática que o primeiro, apesar do tempo de pior caso do primeiro ser quadrático.

Exercícios

1. Reescreva o algoritmo sem usar o vetor auxiliar. Para tanto, acrescente um parâmetro extra, chamado passo, e, ao processar o vetor, pule de passo em passo elementos... (Por exemplo, na primeira chamada recursiva, que calcula a mediana das medianas, ande de 5 em 5 posições no vetor, a partir da posição p+2... Entendeu??????)

2. Por que 5? O que acontece se alterarmos o valor de `TAM_GRUPO` para 3? E se alterarmos para 7? (Dica: refaça a análise usando 3 no lugar de 5 e 7 no lugar de 7 e veja o que consegue concluir em cada caso.)