MAC 338 - Análise de Algoritmos

Departamento de Ciência da Computação

Primeiro semestre de 2003

Limite inferior para ordenação

Vimos em aula vários algoritmos de ordenação. Alguns tinham complexidade de pior caso $\Theta(n^2)$, outros tinham complexidade de pior caso $\Theta(n \log n)$. Será que existem algoritmos de ordenação com complexidade de pior caso melhor que estes?

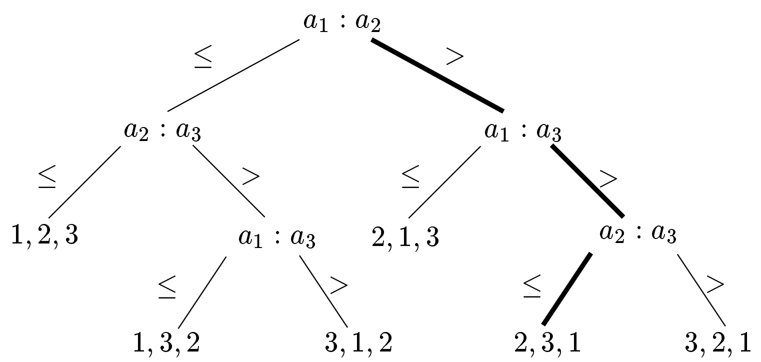
Vamos mostrar que a resposta a esta pergunta é "não", pelo menos para algoritmos como os que vimos até agora, baseados em comparações.

Dizemos que um algoritmo de ordenação *baseia-se em comparações* se, para seqüências de entrada de tamanho n, o fluxo do algoritmo depende apenas do resultado de comparações entre elementos da seqüência.

Considere um algoritmo de ordenação arbitrário que baseie-se em comparações e denote por $a_1, \ldots, a_n$ os elementos de uma seqüência arbitrária de entrada de tamanho n. Suponha, sem perda de generalidade, que os elementos da seqüência são todos distintos. Neste caso, podemos supor, também sem perda de generalidade, que todas as comparações feitas pelo algoritmo são do tipo "$a_i \leq a_j$".

A *árvore de decisão* de um tal algoritmo (definida para cada valor de n) é uma árvore binária cujos vértices internos estão rotulados por pares de elementos da seqüência de entrada, denotados por exemplo por $a_i : a_j$. Cada vértice interno tem dois filhos. As arestas de um vértice interno para os seus filhos estão rotuladas uma por $\leq$ e a outra por $>$. Para facilitar a exposição, digamos que o rótulo $\leq$ leva ao filho esquerdo e que o rótulo $>$ leva ao filho direito. Cada folha está rotulada por uma permutação de $1 \mathinner{..} n$. A árvore de decisão está associada ao algoritmo da seguinte maneira. O rótulo da raiz da árvore corresponde à primeira comparação efetuada pelo algoritmo quando a entrada é uma seqüência $a_1, \ldots, a_n$. A subárvore esquerda da raiz descreve as comparações subseqüentes, caso o resultado desta primeira comparação, digamos, $a_i \leq a_j$, seja verdadeiro. Já a subárvore direita descreve as comparações subseqüentes caso o resultado da comparação $a_i \leq a_j$ seja falso. Com isso, cada seqüência $a_1, \ldots, a_n$ corresponde a um caminho da raiz até uma folha da árvore de decisão. A permutação que rotula essa folha é exatamente a permutação que deixa $a_1, \ldots, a_n$ ordenada.

Exemplo: A árvore de decisão do algoritmo `inserção` para $n = 3$ é



Por exemplo, se $a_1 = 10$, $a_2 = 5$ e $a_3 = 7$, o "caminho" do algoritmo na árvore de decisão acima é o caminho marcado em negrito.

Note que cada uma das 3! permutações aparece nas folhas. Isso é de se esperar mesmo para um n arbitrário. Ou seja, cada uma das $n!$ permutações deve aparecer como rótulo de uma das folhas na árvore de decisão. Afinal, para cada permutação específica, há pelo menos uma seqüência de entrada que é ordenada apenas por aquela permutação. Se uma permutação não aparecesse, então o algoritmo não ordenaria a correspondente seqüência de entrada.

Observe também que o número de comparações que o algoritmo faz no pior caso é exatamente a altura da árvore de decisão. Portanto, se calcularmos uma delimitação inferior para a altura da árvore de decisão do algoritmo, temos uma delimitação inferior para a complexidade de pior caso do algoritmo. Isso é o que vamos fazer.

Existem $n!$ permutações de $1\,..\,n$. Cada uma delas deve aparecer em alguma das folhas da árvore de decisão. Portanto, a árvore de decisão deve ter pelo menos $n!$ folhas. Por outro lado, uma árvore binária de altura h tem no máximo 2^h folhas. Assim, se h é a altura da árvore de decisão de um algoritmo de ordenação baseado em comparações, então $2^h \geq n!$. Sabemos, pela fórmula de Stirling, que

$$n! \;\geq\; \left(\frac{n}{e}\right)^n.$$

Então, podemos concluir que

$$h \;\geq\; \log n! \;\geq\; \log\left(\frac{n}{e}\right)^n \;=\; n\,\log\frac{n}{e} \;=\; n(\log n - \log e) \;=\; \Omega(n\log n).$$

Portanto, de fato, qualquer algoritmo de ordenação baseado em comparações tem complexidade de pior caso $\Omega(n\log n)$.

Exercícios

1. Construa a árvore de decisão para o algoritmo Mergesort para $n = 4$. Quais as permutações de pior caso?

2. Construa a árvore de decisão do algoritmo Quicksort (versão não-probabilística) para $n = 3$. Quais as permutações de pior caso?

3. Por que a hipótese de que os elementos são todos distintos na análise acima não é restritiva? (Ou seja, por que ela não impede que concluamos que o algoritmo, para instâncias arbitrárias, consome tempo de pior caso $\Omega(n\log n)$?)

4. Mostre que qualquer algoritmo de busca baseado em comparações faz no pior caso pelo menos $\lceil\log n\rceil$ comparações. Um algoritmo de busca é um algoritmo que determina uma posição em que um elemento x aparece em um dado vetor $v[0..n-1]$. Um algoritmo de busca baseia-se em comparações se o seu fluxo de controle depende apenas de comparações envolvendo elementos do vetor e x.