Cursos de Verão 2010 - IME/USP
Tópicos de Programação
Quicksort
Prof. Alexandre da Silva Freire

1 Quicksort

Considere o seguinte problema: *dado um vetor $v[p \,..\, r]$, com $p < r$, rearranjar os elementos do vetor e devolver j em $p \,..\, r$ tal que $v[p \,..\, j-1] \leq v[j] < v[j+1 \,..\, r]$*. Repare que não especificamos qual é o índice j. Portanto, podemos escolher qualquer elemento para ser o *pivô* (o elemento que ficará na posição j) e, a partir desse pivô, "descobrir" qual será o índice j. Eis um algoritmo para resolver o problema:

```
    int partition(int p, int r, int v[]) {
       int i = p+1, j = r;
(2)    while (i <= j) {
          if (v[i] <= v[p]) {
             i++;
          } else if (v[p] < v[j]) {
             j--;
          } else {
             troca(v, i, j);
             i++; j--;
          }
       }
       troca(v, p, j);
       return j;
    }
```

O algoritmo acima está correto? Sim! Basta observar que os seguintes invariantes se mantêm válidos ao longo de toda a execução do algoritmo:

(i0) *$v[p \,..\, r]$ é uma permutação do vetor original.*

(i1) *imediatamente antes de qualquer execução da linha (2) vale que $v[p+1 \,..\, i-1] \leq v[p] < v[j+1 \,..\, r]$;*

Qual o consumo de tempo do algoritmo `partition`? O laço da linha (2) é executado $\Theta(r-p+1)$, sendo que cada uma delas executa $\Theta(1)$ operações. As demais linhas (as linhas que estão fora do laço) consomem juntas tempo $\Theta(1)$. Portanto, o consumo de tempo do algoritmo `partition` é $\Theta(r-p+1) \cdot \Theta(1) + \Theta(1) = \Theta(r-p+1)$.

Eis o algoritmo Quicksort:

```
    void quicksort(int p, int r, int v[]) {
       int j;

       if(p < r) {
(3)       j = partition(p, r, v);
(4)       quicksort(p, j-1, v);
(5)       quicksort(j+1, r, v);
       }
    }
```

O algoritmo acima está correto? Sim! Basta verificar, no passo da indução, que após a linha (3) o elemento $v[j]$ está "no seu lugar correto", pois $v[p\,..\,j-1] \leq v[j] < v[j+1\,..\,r]$. Além disso, por hipótese de indução vale que $v[p\,..\,j-1]$ *está ordenado após a linha (4), e* $v[j+1\,..\,r]$ *está ordenado após a linha (5).* Não esqueça de verificar a base da indução!

Qual é o consumo de tempo do algoritmo `quicksort`? É impossível escrever uma recorrência sem saber qual será o resultado devolvido pela função `partition`! Vamos então fazer a análise do melhor caso e depois do pior caso. O melhor caso ocorre quando em todas as chamadas recursivas o pivô é o elemento que está exatamente na posição do meio de $v[p\,..\,r]$, ou seja, $j = \lfloor \frac{n}{2} \rfloor$ (sendo que $n := r - p + 1$). Neste caso podemos definir a seguinte recorrência para representar o consumo de tempo do `quicksort`:

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil - 1) + n, & \text{se } n > 1. \end{cases}$$

Claramente $T(n) < S(n)$, sendo que $S(n)$ é definida da seguinte forma:

$$S(n) = \begin{cases} 1, & \text{se } n = 1 \\ 2S(\frac{n}{2}) + n, & \text{se } n > 1. \end{cases}$$

Vimos na aula sobre o Mergesort que $S(n) = \Theta(n \lg n)$. Logo, $T(n) = O(n \lg n)$. Lembre-se de que $T(n)$ é o consumo de tempo do `quicksort` no **melhor caso**. Analisaremos agora o consumo de tempo do `quicksort` no pior caso. O pior caso ocorre quando o pivô é o elemento que está em uma das extremidades do vetor ($j = p$ ou $j = r$). Um exemplo de um caso no qual isso ocorre é quando o vetor já está ordenado (em ordem crescente ou decrescente). Que ironia! Neste caso podemos definir a seguinte recorrência para representar o consumo de tempo do `quicksort`:

$$T'(n) = \begin{cases} 1, & \text{se } n = 1 \\ T'(n-1) + n, & \text{se } n > 1. \end{cases}$$

Desenhando a árvore de recursão, chega-se à conclusão de que $T'(n) = n + (n-1) + \ldots + 1 = \Theta(n^2)$. Ou seja, o desempenho do **Quick**sort no pior caso é $\Theta(n^2)$. Outra ironia!

Através de uma *análise probabilística* mostra-se que o consumo *esperado* de tempo do quicksort é $\Theta(n \lg n)$. Porém, este tipo de análise está fora do escopo deste curso.