Quick Sort

SCC0201 - Introdução à Ciência de Computação II

Clausius G. Reis Leandro A. Amaral Tiago S. Nazaré
Vanessa Q. Marinho

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

21 de junho de 2015

Agenda

Agenda

Ideia Básica

Passos:

- Escolher um elemento do *array*, chamado de pivô;
- Colocar o pivô na sua posição correta (Partição);
  - Elementos menores ou iguais ao pivô ficam à sua direita;
  - Elementos maiores que o pivô ficam à sua esquerda;
- Aplicar os passos nos *subarrays* à esquerda e à direita do pivô.

Quick Sort - Exemplo

5	2	8	3	7	1	0
0	2	3	1	5	8	7
0	**2**	3	1	5	8	7
0	1	2	3	5	8	7
0	1	2	3	5	**8**	7
0	1	2	3	5	7	8

Agenda

Particionamento - Exemplo

5	2	8	3	7	1	0
0	2	8	3	7	1	**5**
0	2	8	3	7	1	**5**
0	2	8	3	7	1	**5**
0	2	8	3	7	1	**5**
0	2	3	8	7	1	**5**
0	2	3	8	7	1	**5**
0	2	3	1	7	8	**5**
0	2	3	1 |	**5** |	8	7

Particionamento

```c
div = first;

for(i = first; i < last; ++i){
    if(a[i] <= pivot_val){
            aux = a[i];
            a[i] = a[div];
            a[div] = aux;

            ++div;
    }
}

a[last] = a[div];
a[div] = pivot_val;
```

Código

```c
void quick_sort_rec(int *a, int first, int last){
        if(last - first <= 0) return;

        int pivot_idx = first;
        int pivot_val = a[pivot_idx];
        int aux, i, div = first;

        a[pivot_idx] = a[last];
        a[last] = pivot_val;

        for(i = first; i < last; ++i){
                if(a[i] <= pivot_val){
                        aux = a[i];
                        a[i] = a[div];
                        a[div] = aux;

                        ++div;
                }
        }

        a[last] = a[div];
        a[div] = pivot_val;

        quick_sort_rec(a, first, div - 1);
        quick_sort_rec(a, div + 1, last);
}
```

Agenda

1. Quick Sort
   - Introdução
   - Particionamento
   - Análise de Complexidade

Análise de Complexidade

O particionamento é uma operação $\mathcal{O}(n)$:

- Seleção **aleatória** do pivô: $\mathcal{O}(1)$;
- Dividir o vetor em maiores e menores que o pivô: $\mathcal{O}(n)$;

O tempo $T(n)$ para ordenar um vetor é dado por:

$$T(n) = T(k) + T(n-k) + \underbrace{\alpha n}_{\mathcal{O}(n)},$$

onde:

- k é a quantidade de elementos menores ou iguais ao pivô;
- $an + c$ é o tempo para se fazer o particionamento;

www.cise.ufl.edu/class/cot3100fa07/quicksort_analysis.pdf

Análise de Complexidade - Pior Caso ($\mathcal{O}(n^2)$)

O pivô é sempre o primeiro ou o último elemento ($k = 1$ ou n):

$$T(n) = T(n-1) + T(1) + \alpha n$$

Desenvolvendo a recursividade tem-se:

$$T(n) = [T(n-2) + T(1) + \alpha(n-1)] + T(1) + \alpha n$$
$$T(n) = T(n-2) + 2T(1) + \alpha(n + (n-1))$$
$$T(n) = T(n-3) + 3T(1) + \alpha(n + (n-1) + (n-2))$$
$$T(n) = T(n-i) + iT(1) + \alpha \sum_{j=0}^{i-1} (n-j)$$

Assim, para $i = n - 1$ tem-se:

$$T(n) = nT(1) + \alpha \left(\frac{n^2 + n}{2} - 2 \right)$$

Análise de Complexidade - Melhor Caso ($\mathcal{O}(n \log_2 n)$)

O pivô dividirá sempre o vetor ao meio ($k = \frac{n}{2}$):

$$T(n) = 2T\left(\frac{n}{2}\right) + \underbrace{\alpha n}_{\mathcal{O}(n)}$$

Desenvolvendo a recursividade tem-se:

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{\alpha n}{2}\right] + \alpha n = 2^2 T\left(\frac{n}{4}\right) + 2\alpha n$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + k\alpha n$$

O processo continua até $k = \log_2 n$, onde tem-se:

$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + \alpha n \log_2 n = nT(1) + \alpha n \log_2 n$$

Evitando o pior caso

A escolha aleatória de um pivô reduz as chances do pior caso ocorrer.

A mediana do vetor pode ser encontrada em tempo linear.

Tarefa - Para a Próxima Aula

Entregar uma pesquisa sobre a importância da escolha do pivô.

Implementar uma versão diferente da implementada em sala (diferente do ponto de vista da escolha do pivô).

Assistir o seguinte vídeo:

- `https://www.youtube.com/watch?v=aXXWXz5rF64`