USP - Universidade de São Paulo

Instituto de Ciências Matemáticas e Computação

Laboratório de Introdução à Ciência da Computação II

SCC0220 - 2021/2

SEGUNDA IMPLEMENTAÇÃO PRÁTICA

ÜBERSORT

1 Introdução

Zaratustra, um estudante de Ciências da Computação, estava muito animado com os algoritmos de ordenação que aprendia. Toda aula, ele era surpreendido com a introdução de um método mais rápido e mais elegante que o anterior.

Todavia, ao terminar o Módulo 2 de Laboratório de Ciências da Computação e começar o Módulo 3, ele percebeu que os demais algoritmos com complexidade de tempo menor que $O(n \log n)$, além de utilizar memória extra na maioria dos casos, eram extremamente limitados.

O pobre do Zaratustra ficou tão atordoado que decidiu deixar seus outros trabalhos de lado e tirar um tempo para refletir sobre o fato. Demorou pouco até ele ter uma brilhante ideia: fazer um algoritmo que vá **além do quicksort**.

Após ler a mensagem encaminhada para o grupo da sala por Zaratustra, você resolveu implementar todas as sugestões dadas e agrupá-las em um só algoritmo.

2 Sugestões de otimização do quicksort

Mudando a seleção do pivô

Zaratustra percebeu que o desempenho do quicksort muitas vezes acaba sendo prejudicado devido ao método de escolha do pivô. Pesquisando sobre o assunto, ele descobriu dois métodos muito interessantes para resolver esse problema:

1. **Mediana de três**

   Fazemos a mediana entre o primeiro elemento, o elemento do meio e o último elemento de uma partição para decidir o pivô. Esse método resolve 2 dois piores casos do quicksort tradicional (aquele que escolhe o elemento mais a esquerda para ser pivô): quando o vetor já está ordenado e quando o vetor está reversamente ordenado.

2. **Randômico**

   Assim como o nome sugere, este modo implica na escolha de um elemento com índice aleatório para ser o pivô. Por ser randômico, as chances de cairmos no pior caso acaba se tornando mais esparsa.

Lidando com elementos repetidos

Outra culpada pela queda de desempenho do algoritmo é a repetição de elementos na partição. Felizmente, resolver isso é muito simples: em vez de separar o problema em 2 grupos: aqueles que são menores que o pivô e os demais; podemos dividir em 3 grupos: aqueles que são menores, os que são iguais e os que são maiores.

Implementação conjunta com o insertion sort

Zaratustra também lembra do início do semestre, onde aprendeu que o insertion sort possui o melhor desempenho dos algoritmos vistos até então para vetores pequenos. Desse modo, ele pensa em alterar a lógica do quicksort de modo que, dado um limiar específico, a recursão do algoritmo seja finalizada e o insertion sort seja executado.

3 Informações adicionais

Antes de mais nada, vale a pena lembrar que **NÃO** estamos mexendo na complexidade do quicksort em si, apenas diminuindo a chance do nosso algoritmo cair nos piores casos. Em relação as especificações da implementação, considere os seguintes itens:

A. O limiar para a transição do insertion sort deverá ser um array com ≤ 10 elementos.

B. No caso de implementação do pivô randômico, a *seed* para o $rand()$ deverá ser setada utilizando $srand(42)$ antes do início da recursão.

C. Independente do modo, o pivô escolhido em cada iteração deverá ser sempre trocado com o último elemento do vetor. Exemplo:

```
swap(&array[pivoIndex], &array[end])
```

   Assim, quando a função de $partiticao()$ do quicksort for chamada o pivô será igual ao $array[end]$

4 Entrada

A. A primeira linha de entrada contem um inteiro n relativo a quantidade de elementos do array.

B. Na segunda linha temos um char c indicando o método de escolha do pivô:

```
'1' - para MÉDIA DE TRÊS
'2' - para RANDÔMICO.
```

C. As n linhas seguintes contêm um inteiro i_j correspondente a um elemento do array de entrada.

5 Saída

Ao final da execução do algoritmo deverá ser impresso, linha por linha, o vetor resultante.

6 Exemplos de entrada e saída

Entrada

```
10
1
810
451
905
210
653
167
625
838
185
781
```

Saída esperada

```
167
185
210
451
625
653
781
810
838
905
```

Cuidado: alguns dos casos de entrada podem ser grandes!

Bom trabalho :)