Trabalho Prático 2
Sorting Algorithms

Chrystian Paulo Ferreira de Melo

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

chrystian1@ufmg.br
meloo.chrys@gmail.com

1. Introdução

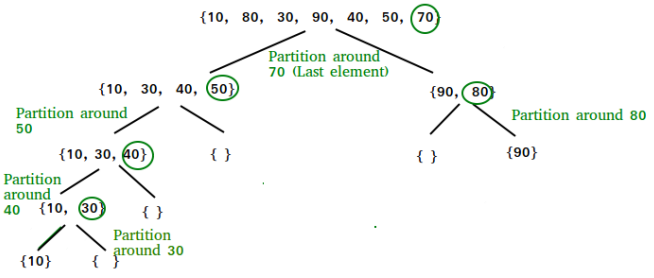
Este trabalho consiste em analisar o desempenho de diferentes algoritmos de ordenação em diferentes cenários, descritos a seguir. Esta análise consiste em comparar os algoritmos considerando três métricas de desempenho: número de comparações de chaves, o número de cópias de registros realizados, e o tempo total gasto para ordenação (tempo de processamento e não o tempo de relógio).

Impacto de variações do Quicksort Neste cenário, você deverá comparar o desempenho de diferentes variações do Quicksort para ordenar um conjunto de N registros armazenados em um vetor. Cada registro contém: um inteiro, que é a chave para ordenação, quinze cadeias de caracteres (strings), cada uma com 200 caracteres e 10 números reais

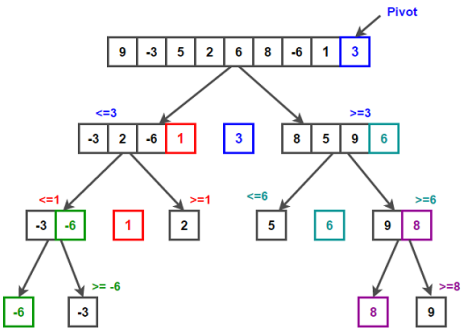
2. Implementação

As variações do Quicksort a serem implementadas e avaliadas são:

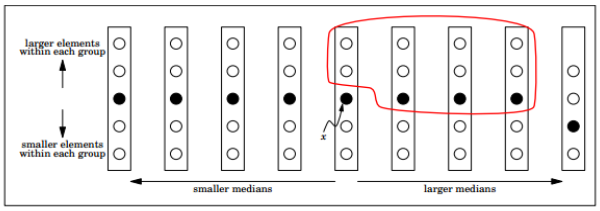
- Quicksort Recursivo: este é o Quicksort recursivo apresentado em sala de aula.



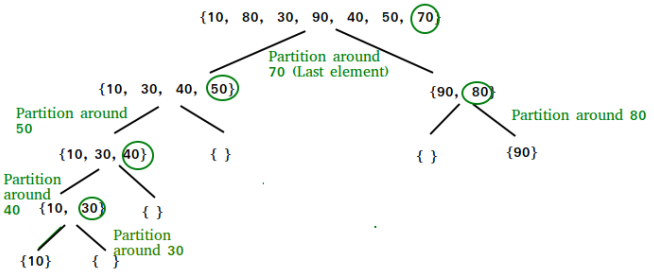
- Quicksort Mediana(k): esta variação do Quicksort recursivo escolhe o pivô para partição como sendo a mediana de k elementos do vetor, aleatoriamente escolhidos. Experimente com k = 3, k = 5, e k = 7.



- Quicksort Seleção(m): esta variação modifica o Quicksort Recursivo para utilizar o algoritmo de Seleção para ordenar partições (isto é, pedaços do vetor) com tamanho menor ou igual a m. Experimente com m = 10 e m = 100.



- Quicksort não Recursivo: esta variação escolhe o pivô como o elemento do meio (como apresentado em sala de aula), mas não é recursiva, utilizando uma pilha para armazenar partições a serem processadas posteriormente.



- Quicksort Empilha Inteligente: esta variação do Quicksort processa primeiro a menor partição. Você deve aplicar esta otimização à versão não recursiva do Quicksort.

2.1. Classes

Para modularizar a implementação foram montadas duas classes principais.

2.1.1 QuickSortUtils

Definição da classe para armazenar métodos uteis para ordenações quicksort.

2.1.2 Sorting

Definição da classe para armazenar demais métodos de ordenação, além de conter o resultado do melhor algoritmo para quicksort, dentre os testes feitos nesse trabalho.

3. Análise de Complexidade

A análise de complexidade dos algoritmos se resume em O(n log n). Com pior caso em O(n^2).

4. Conclusão

O trabalho foi realizado com sucesso, utilizando boas estruturas e mantendo um código coeso.

Ao fim desse trabalho, ficou ainda evidente a importância de uma boa escolha de estrutura de dados. Além disso, pode-se protagonizar dois papéis como programador: aquele que cria a estrutura de dados e aquele que utiliza a estrutura de dados, podendo dessa forma visualizar de duas formas diferentes o resultado e melhorar o código conforme o projeto foi se desenvolvendo.

5. Análise de resultados

Analisando os resultados foi possível perceber que o QuickSort Select e o QuickSort não recursivo se destacaram por se aproximar mais de um algoritmo ótimo.

Obs: O teste sugerido foi realizado até o valor 500.000, pois a partir desse ponto a máquina usada não produz resultados coesos.

Quicksort Recursivo

```
---------- 1° Sort ----------
Coping    : 2165 copies.
Comparing : 1175 comparisons.
Timing    : 301 clocks.
---------- 2° Sort ----------
Coping    : 12927 copies.
Comparing : 7937 comparisons.
Timing    : 2994 clocks.
---------- 3° Sort ----------
Coping    : 35184 copies.
Comparing : 25194 comparisons.
Timing    : 9160 clocks.
---------- 4° Sort ----------
Coping    : 130062 copies.
Comparing : 80072 comparisons.
Timing    : 205665 clocks.
---------- 5° Sort ----------
Coping    : 299974 copies.
Comparing : 199984 comparisons.
Timing    : 816260 clocks.
---------- 6° Sort ----------
Coping    : 1448114 copies.
Comparing : 948124 comparisons.
Timing    : 2.01595e+07 clocks.
```

Quicksort Mediana

```
---------- 1° Sort ----------
Coping    : 2662 copies.
Comparing : 1672 comparisons.
Timing    : 291 clocks.
---------- 2° Sort ----------
Coping    : 15428 copies.
Comparing : 10438 comparisons.
Timing    : 3882 clocks.
---------- 3° Sort ----------
Coping    : 28080 copies.
Comparing : 18090 comparisons.
Timing    : 9304 clocks.
---------- 4° Sort ----------
Coping    : 125165 copies.
Comparing : 75175 comparisons.
Timing    : 203706 clocks.
---------- 5° Sort ----------
Coping    : 389935 copies.
Comparing : 289945 comparisons.
Timing    : 802063 clocks.
---------- 6° Sort ----------
Coping    : 1448204 copies.
Comparing : 948214 comparisons.
Timing    : 1.98997e+07 clocks.
```

QuickSort Select

```
---------- 1° Sort ----------
Coping    : 422 copies.
Comparing : 416 comparisons.
Timing    : 17 clocks.
---------- 2° Sort ----------
Coping    : 7514 copies.
Comparing : 7505 comparisons.
Timing    : 129 clocks.
---------- 3° Sort ----------
Coping    : 27158 copies.
Comparing : 27148 comparisons.S
Timing    : 359 clocks.
---------- 4° Sort ----------
Coping    : 155628 copies.
Comparing : 155617 comparisons.
Timing    : 2081 clocks.
---------- 5° Sort ----------
Coping    : 459617 copies.
Comparing : 459605 comparisons.
Timing    : 3881 clocks.
---------- 6° Sort ----------
Coping    : 1395628 copies.
Comparing : 1395617 comparisons.
Timing    : 10959 clocks.
```

QuickSort Não Recursivo

```
---------- 1° Sort ----------
Coping    : 10089 copies.
Comparing : 3156 comparisons.
Timing    : 158 clocks.
---------- 2° Sort ----------
Coping    : 52851 copies.
Comparing : 17918 comparisons.
Timing    : 2430 clocks.
---------- 3° Sort ----------
Coping    : 115108 copies.
Comparing : 45175 comparisons.
Timing    : 8862 clocks.
---------- 4° Sort ----------
Coping    : 529986 copies.
Comparing : 180053 comparisons.
Timing    : 196373 clocks.
---------- 5° Sort ----------
Coping    : 1099898 copies.
Comparing : 399965 comparisons.
Timing    : 772153 clocks.
---------- 6° Sort ----------
Coping    : 5448038 copies.
Comparing : 1948105 comparisons.
Timing    : 1.97299e+07 clocks.
```

6. Instruções

O trabalho foi criado conforme as instruções indicadas.

1. Recebe a semente do gerador de números aleatórios bem como os demais parâmetros, incluindo os nomes dos arquivos de entrada e de saída. Estes parâmetros devem ser passados pela linha de comando (argc e argv em C). Assim:
   a. Quicksort Recursivo: quicksort -v 1 -s 10 -i entrada.txt -o saida10.txt
   b. Quicksort Mediana (k): quicksort -v 2 -k 3 -s 10 -i entrada.txt -o saida10.txt
   c. Quicksort Seleção (m): quicksort -v 3 -m 100 -s 10 -i entrada.txt -o saida10.txt
   d. Quicksort não Recursivo: quicksort -v 4 -s 10 -i entrada.txt -o saida10.txt
   e. Quicksort Empilha Inteligente: quicksort -v 5 -s 10 -i entrada.txt -o saida10.txt

Logo para a execução será necessário executar um comando similar a esse:

programa.out quicksort -v 1 -s 10 -i input.txt -o output.txt

7. References

Material de estudo no moodle.
Aulas.
Exemplo de uso do analisaMem - TAD Matriz Estático
Projeto com Makefile para Linux (C++)
http://en.wikipedia.org/wiki/Netpbm_format.
http://en.wikipedia.org/
https://en.cppreference.com/w/
https://stackoverflow.com
https://cplusplus.com