Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estruturas de Dados 2019/1
Prof. Luiz Chaimowicz e Raquel Prates
Monitores: Matheus Nunes e Ronaldo Vieira

Trabalho Prático 2: Biblioteca Digital de Arendelle

Valor: 10 pontos
Data de entrega: 10 de Junho de 2019

Introdução

A fim de manter os altos índices de crescimento econômico de seu reino proporcionados pela ascensão do mercado de gelo na última década, Elsa, rainha de Arendelle, lançou no início deste ano o Plano de Democratização do Acesso à Educação (PDAE). O projeto possui como um de seus objetivos garantir aos arendellenses o direito de acesso à informação via Internet, ao invés de apenas via pombo-correio ou pessoalmente, de forma a adequar-se aos padrões do século XXI.

Devido aos recentes sucessos de cooperação dos alunos de Estrutura de Dados da UFMG com o reino, você foi contratado(a) novamente para ajudar em um dos vários projetos previstos no PDAE: a criação da Biblioteca Digital de Arendelle. Uma vez inaugurada, a Biblioteca disponibilizará online de forma gratuita parte do acervo de livros e pergaminhos que o reino possui em suas bibliotecas físicas. Em razão da quantidade massiva de itens a ser disponibilizada, o uso de um algoritmo de ordenação eficiente é essencial para a tarefa de organização do acervo. Por ser especialista no assunto, você foi encarregado(a) de comparar o desempenho de vários algoritmos de ordenação candidatos. Em particular, variações do Quicksort.

Detalhes do problema

O objetivo deste trabalho é praticar os conceitos de ordenação e desenvolver a habilidade de análise comparativa de algoritmos. Você deverá implementar as variações do Quicksort descritas abaixo e testá-las com diversos vetores de entrada (vetores de números inteiros), contabilizando o **número de comparações de chaves**, o **número de movimentações de registros** e o **tempo de execução**. Para isso você deverá colocar contadores em seu código e instrumentá-lo de forma a obter o tempo de execução (o código de funções de "timer" se encontra no apêndice A, ao fim deste documento).

As variações do Quicksort a serem consideradas são:

1. **Quicksort clássico**. Seleção de pivô usando o elemento central.
2. **Quicksort mediana de três**. Seleção do pivô usando a "mediana de três" elementos, em que o pivô é escolhido usando a mediana entre a chave mais à esquerda, a chave mais à direita e a chave central (como no algoritmo clássico).
3. **Quicksort primeiro elemento**. Seleção do pivô como sendo o primeiro elemento do subconjunto.
4. **Quicksort inserção 1%**. O processo de partição é interrompido quando o subvetor tiver menos de $k = 1\%$ chaves. A partição então deve ser ordenada usando uma implementação especial do algoritmo de ordenação por inserção, preparada para ordenar um subvetor. Seleção de pivô usando a "mediana de três" elementos, descrita acima.
5. **Quicksort inserção 5%**. Mesmo que o anterior, com $k = 5\%$.
6. **Quicksort inserção 10%**. Mesmo que o anterior, com $k = 10\%$.

7. **Quicksort não recursivo**. Implementação que não usa recursividade. Utiliza pilha para simular as chamadas de função recursivas e identificar os intervalos a serem ordenados a cada momento.

Especificamente, deverão ser realizados testes com vetores de **tamanho** diferentes, variando entre 50 mil e 500 mil elementos, em intervalos de 50 mil elementos. Além disso, três tipos diferentes de vetores deverão ser utilizados: **aleatórios**, **ordenados de forma crescente** e **ordenados de forma decrescente**. **Seu próprio programa deve gerar os vetores dos três tipos**. Em outras palavras, para cada uma das sete variações do Quicksort listadas a seguir, você realizará testes com dez tamanhos e três tipos de vetores diferentes ($7 \times 10 \times 3 = 210$ testes no total).

Você deverá realizar um estudo comparativo entre os algoritmos, utilizando os resultados dos testes realizados e com base nas métricas coletadas, destacando as vantagens e desvantagens de cada variação do Quicksort, seus melhores e piores casos e quando vale a pena ou não utilizá-los e por quê. Para isso, você deverá construir tabelas e gráficos mostrando o desempenho de cada um em diferentes circunstâncias para sustentar seus argumentos.

É importante repetir cada teste algumas dezenas de vezes (sugestão: no mínimo vinte) a fim de obter a **mediana** do tempo de execução, uma vez que este pode variar e a média pode ser fortemente enviesada por uma execução muito ruim. No caso dos vetores aleatórios, as repetições são necessárias para obter uma **média** das comparações e movimentações, visto que estes números irão divergir para cada entrada aleatória.

Entrada e saída

O formato de entrada e saída desejados são:

Entrada. Nenhum dado precisa ser lido da entrada padrão ou arquivo. No entanto, seu programa deverá aceitar quatro **parâmetros**[1]:

- Variação. Qual variação do Quicksort será avaliada. As alternativas possíveis devem ser `QC` (clássico), `QM3` (mediana de três), `QPE` (primeiro elemento), `QI1` (inserção 1%), `QI5` (inserção 5%), `QI10` (inserção 10%) e `QNR` (não recursivo).
- Tipo do vetor. Qual tipo de vetor será ordenado com a variação escolhida. As alternativas possíveis devem ser `Ale` (aleatório), `OrdC` (ordem crescente) e `OrdD` (ordem decrescente).
- Número de itens. Qual o número de itens dos vetores. As alternativas possíveis devem ser o conjunto dos números inteiros não negativos.
- Exibir vetores utilizados. Parâmetro opcional. Se estiver presente, exibe na saída os vetores que foram gerados e utilizados na análise.

Dessa forma, após compilado, seu programa poderá ser executado em um computador Linux da seguinte maneira:

`./nomedoprograma <variacao> <tipo> <tamanho> [-p].`

Por exemplo:

`./nomedoprograma QPE Ale 50000 -p`

ou

`./nomedoprograma QI10 OrdD 100000 > qi10_ordd_100000.txt.`

No primeiro caso, o programa avaliaria o Quicksort primeiro elemento em vetores aleatórios de 50 mil itens, e exibiria os vetores utilizados na saída. Já no segundo, o o Quicksort inserção 10% seria avaliado com vetores de 100 mil itens, e a saída seria gravada no arquivo `qi10_ordd_100000.txt`.

[1] `http://linguagemc.com.br/argumentos-em-linha-de-comando/`

Saída. O resultado de seus testes deverão ser impressos na saída padrão (`stdout`). A saída deve conter a variação do Quicksort avaliada, o tipo e tamanho de vetor, assim como sua mediana de tempo de execução em microssegundos e média de comparações e de movimentações efetuadas, de acordo com a seguinte formatação:

`<variacao> <tipo> <tamanho> <n_comp> <n_mov> <tempo>`,

onde `variacao`, `tipo` e `tamanho` são os parâmetros recebidos na entrada, `n_comp` e `n_mov` se referem ao número médio de comparações de chaves e de movimentações de registros efetuadas e `tempo` ao tempo mediano de execução, em microssegundos.

Se o parâmetro `-p` foi encontrado na entrada, as próximas linhas da saída deverão conter, para cada execução do teste, o vetor utilizado.

Por exemplo, a saída

```
QC Ale 10 34 10 15681
9 3 10 2 6 8 1 9 9 3
1 1 2 9 7 5 4 9 2 4
8 1 9 8 4 4 7 10 3 4
5 2 1 10 2 6 9 10 5 9
4 7 6 7 3 9 4 7 1 6
⋮
```

mostra que as execuções do Quicksort clássico em um vetor aleatório de tamanho 10 resultaram, em média, em 34 comparações e 10 movimentações, e que a execução mediana levou 15681 microssegundos. O parâmetro `-p` foi usado, portanto, as linhas restantes mostram os vetores utilizados nas execuções.

Entregáveis

Código-fonte. A implementação poderá ser feita utilizando as linguagens C ou C++. Não será permitido o uso da *Standard Library* do C++ ou de bibliotecas externas que implementem as estruturas de dados ou os algoritmos. **A implementação das estruturas e algoritmos utilizados neste trabalho deve ser sua**. Os códigos devem ser executáveis em um computador com *Linux*. Caso não possua um computador com *Linux*, teste seu trabalho em um dos computadores do laboratório de graduação do CRC[2].

O código-fonte terá diversas partes, incluindo porém não limitado a: leitura e interpretação dos parâmetros, implementação de cada variante do Quicksort, implementação de pilha, implementação especial do algoritmo de ordenação por inserção, geração de vetores de cada tipo, captação das métricas, captação do tempo de execução e impressão da saída. Aplique boas práticas de programação e organize seu código-fonte em arquivos, classes e funções de acordo com o significado de cada parte. A utilização de *Makefile*[3] é recomendada para este trabalho.

Documentação. Neste trabalho a documentação corresponderá **a 60% da nota**. A documentação de seu programa **deverá** estar em formato **PDF**, **seguir** o modelo de trabalhos acadêmicos da SBC (que pode ser encontrado online[4]) e ser **sucinta**. Deverá conter **todos** os seguintes tópicos:

- Cabeçalho. Título do trabalho, nome e número de matrícula do autor.

[2] `https://crc.dcc.ufmgdeboas.br/infraestrutura/laboratorios/linux`

[3] `https://opensource.com/article/18/8/what-how-makefile`

[4] `http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros/878-modelosparapublicaodeartigos`

- Introdução. Apresentação do problema abordado e visão geral sobre o funcionamento do programa. O funcionamento do Quicksort e suas variações devem ser explicados.
- Implementação. Descrição sobre a implementação do programa. Devem ser detalhados o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes que porventura estejam omissos no enunciado.
- Instruções de compilação e execução. Instruções de como compilar e executar o programa.
- Análise experimental. Descrição da metodologia da análise, especificações básicas do computador usado para os testes e estudo comparativo dos algoritmos, usando tabelas e gráficos e fazendo uma análise crítica sobre o desempenho obtido.
- Conclusão. Resumo do trabalho realizado, conclusões gerais sobre os resultados e eventuais dificuldades ou considerações sobre seu desenvolvimento.
- Bibliografia. Fontes consultadas para realização do trabalho.

O código-fonte e a documentação devem ser organizados como demonstrado pela árvore 1. O diretório raiz deve ser nomeado de acordo com o **nome e último sobrenome** da(o) aluna(o), separado por *underscore*, por exemplo, o trabalho de "Kristoff das Neves Björgman" seria entregue em um diretório chamado `kristoff_bjorgman`. Este diretório principal deverá conter um subdiretório chamado `src`, que por sua vez conterá os códigos (`.cpp, .c, .h, .hpp`) na estrutura de diretórios desejada. A documentação **em formato PDF** deverá ser incluída no diretório raiz do trabalho. Favor evitar o uso de caracteres especiais, acentos e espaços na nomeação de arquivos e diretórios.

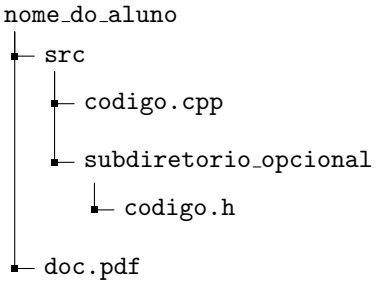

Figura 1: Estrutura de diretórios do entregável do TP2

O diretório deverá ser submetido em um único arquivo **'Nome_Sobrenome.zip'**, (onde Nome e Sobrenome seguem as mesmas diretrizes para o nome do diretório, explicado acima) através do Moodle da disciplina até as **23:59** do dia **10 de Junho de 2019**.

Considerações Finais

Algumas considerações finais importantes:

- **Preste bastante atenção nos detalhes da especificação**. Cada detalhe ignorado acarretará em perda de pontos.
- O que será avaliado no trabalho:

  Boas práticas de programação: se o está código bem organizado e indentado, com comentários explicativos, possui variáveis com nomes intuitivos, modularizado, etc.

Implementação correta dos algoritmos: se as variações do Quicksort foram implementadas de forma correta.

Conteúdo da documentação: se todo o conteúdo necessário está presente, reflete o que foi implementado e está escrito de forma coerente e coesa.

- Após submeter no Moodle seu arquivo **'.zip'**, faça o download dele e certifique-se que não está corrompido. Não será dada segunda chance de submissão para arquivos corrompidos.
- Em caso de dúvidas, **não hesite em perguntar** no Fórum de Discussão no Moodle ou procurar os monitores da disciplina – estamos aqui para ajudar!
- **PLÁGIO É CRIME**: caso utilize códigos disponíveis online ou em livros, **referencie** (inclua comentários no código fonte e descreva a situação na documentação). Trabalhos onde o plágio for identificado serão devidamente penalizados: o aluno terá seu trabalho anulado e as devidas providências administrativas serão tomadas. Discussões sobre o trabalho entre colegas são encorajadas, porém compartilhamento de código ou texto é plágio e as regras acima também se aplicam.
- Em caso de atraso na entrega, serão descontados $2^d - 1$ pontos, onde d é o número de dias de atraso arredondado para cima.
- Comece o trabalho o mais cedo possível. A data de entrega está tão longe quanto poderia estar.

Bom trabalho!

Apêndices

A Medição de tempo em C/C++

Os códigos abaixo descrevem como utilizar as bibliotecas `time.h` (C) e `std::chrono` (C++) para medir o tempo de execução de funções em sistemas Unix.

```c
#include <time.h>
#include <stdio.h>

int main() {
    struct timespec start, end;

    // Pega o horário do sistema antes da execução do código
    clock_gettime(CLOCK_REALTIME, &start);

    // Código que você quer medir o tempo
    for (int i = 0; i < 1000000; i++) {
        char* queen = "Elsa";
    }

    // Pega o horário do sistema depois da execução do código
    clock_gettime(CLOCK_REALTIME, &end);

    // Obtém a diferença entre o horário de fim e o de início
    long elapsed_time = 1.e+6 * (double) (end.tv_sec - start.tv_sec)
                        + 1.e-3 * (double) (end.tv_nsec - start.tv_nsec);

    printf("Tempo de execução: %ld microssegundos\n", elapsed_time);

    return 0;
}
```

```cpp
#include <iostream>
#include <chrono>
#include <string>

int main() {
  using namespace std::chrono;

  // Pega o horário do sistema antes da execução do código
  high_resolution_clock::time_point t1 = high_resolution_clock::now();

  // Código que você quer medir o tempo
    for (int i = 0; i < 1000000; i++) {
        string queen = "Elsa";
    }

  // Pega o horário do sistema depois da execução do código
  high_resolution_clock::time_point t2 = high_resolution_clock::now();

  // Obtém a diferença entre o horário de fim e o de início
```

```
  duration<double, std::micro> elapsed_time = duration_cast<duration<double>>(t2 - t1);

  std::cout << "Tempo de execução: " << elapsed_time.count()
            << " microssegundos" << std::endl;

  return 0;
}
```

B Dicas para a documentação

O objetivo desta seção é apresentar algumas dicas para auxiliar na redação da documentação do trabalho prático.

1. **Sobre *Screenshots***: ao incluir *screenshots* (imagens da tela) em sua documentação, evite utilizar o fundo escuro. Muitas pessoas preferem imprimir documentos para lê-los e imagens com fundo preto dificultam a impressão e visualização. Recomenda-se o uso de fundo branco com caracteres pretos, para *screenshots*. Evite incluir trechos de código e/ou pseudocódigos utilizando *screenshots*. Leia abaixo algumas dicas de como incluir código e pseudocódigo em seu texto.

2. **Sobre códigos e pseudocódigos**: Ao incluir este tipo de texto em sua documentação procure usar a ferramenta adequada para que a formatação fique a melhor possível. Existem várias ferramentas para LaTeX, como o `minted`[5], o `lstlisting`[6], e o `algorithm2e`[7] (para pseudocódigos), e algumas para Google Docs (`Code Blocks`[8], Code Pretty[9]).

3. **Sobre URLs e referências**: evite utilizar URLs da internet como referências. Geralmente URLs são incluídas como notas de rodapé. Para isto basta utilizar o comando `\footnote{\url{}}` no LaTeX, ou ativar a opção nota de rodapé[10] no Google Docs/MS Word.

4. **Evite o Ctrl+C Ctrl+V**: encoraja-se a modularização de código, porém a documentação é única e só serve para um trabalho prático. Reuso de documentação é auto-plágio[11]!

[5] https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted
[6] https://www.overleaf.com/learn/latex/Code_listing
[7] https://en.wikibooks.org/wiki/LaTeX/Algorithms
[8] https://gsuite.google.com/marketplace/app/code_blocks/100740430168
[9] https://chrome.google.com/webstore/detail/code-pretty/igjbncgfgnfpbnifnnlcmjfbnidkndnh?hl=en
[10] https://support.office.com/en-ie/article/insert-footnotes-and-endnotes-61f3fb1a-4717-414c-9a8f-015a5f3ff4cb
[11] https://blog.scielo.org/blog/2013/11/11/etica-editorial-e-o-problema-do-autoplagio/#.XORgbdtKg5k