Segundo Projeto

SCC 0224/0606 – Estruturas de Dados II

Prof. Robson L. F. Cordeiro

5 de maio de 2023

1 Introdução

O projeto está dividido em 2 partes: (I) com foco na análise de algoritmos de busca sequencial, e; (II) com foco na análise de algoritmos de busca com espalhamento (hashing). Para cada parte do projeto serão disponibilizados:

- Dois arquivos de entrada: um que deverá ser utilizado para inserção dos dados e outro para realizar buscas nos dados inseridos;
- Um template para cada implementação, com exemplos de leitura da entrada dos dados e formato de saída será disponibilizado. Este template serve apenas como um guia para implementação das soluções, ele poderá ser modificado (adicionar variáveis, funções, etc.) caso necessário, respeitando o formato de entrada e saída dos dados.

Note que não será necessário implementar a leitura nos arquivos de teste e nem imprimir a saída, pois este trecho de código já está implementado no arquivo de template. Para a saída, basta que seu algoritmo realize as atribuições nas variáveis de resposta.

Um dos requisitos para as duas partes do projeto é realizar a contagem de itens encontrados. No caso específico de itens duplicados, por exemplo, se a palavra "abc"existir duas vezes no conjunto de consultas e ela existir no conjunto de inserção, ela deverá ser contabilizada duas vezes. Essa regra se aplica para as duas partes do projeto.

Outro requisito é o calculo da média e desvio padrão dos tempos de inserção e busca. Esses valores podem ser calculados por meio de ferramentas auxiliares (por exemplo, Excel). Dessa forma, não é necessário que seja implementado código para calcular esses valores. O código que será submetido poderá conter somente instruções para uma única execução de cada exercício. Porém, no relatório que será submetido, deverão ser apresentados os

resultados de média e desvio padrão em forma de gráficos e tabelas obtidos por meio de, no mínimo, 3 execuções independentes.

Caso não esteja usando uma IDE como o NetBeans, a seguir se encontram alguns comandos para compilar e testar seu código em ambiente Linux. Nos exemplos, substitua o X pelo número do exercício. Por exemplo, para compilar o exercício 1a digite no terminal: `gcc exercicio1a.c -lm -o exercicio1a`.

Note que o parâmetro `-lm` é utilizado para a compilação. Este comando é necessário para que o seu programa consiga utilizar a biblioteca com funções matemáticas. Algumas implementações podem requerer que estas funções estejam disponíveis.

- **Como compilar o programa de um exercício?** `gcc exercicioX.c -lm -o exercicioX`
- **Como executar o programa de um exercício?** `./exercicioX`
- **Como compilar e executar o programa de um exercício?** `gcc exercicioX.c -lm -o exercicioX && ./exercicioX`

2 Relatório do Projeto

Deve ser redigido um relatório para o projeto contendo uma descrição do que foi desenvolvido. O relatório deve ter no máximo 5 páginas de texto, **não incluindo** nesta limitação de páginas possíveis tabelas, gráficos e lista de referências. O relatório deve conter as seguintes informações:

- Uma breve introdução apresentando a organização do relatório;
- Descrições pertinentes à implementação do trabalho. Por exemplo, escolha por determinadas estruturas de dados, bibliotecas, tipos de variáveis, assim como eventuais modificações feitas nos templates sugeridos;
- Tabelas apresentando as medições de tempo realizadas para cada exercício;
- Análise dos resultados empíricos para cada exercício;
- Citação de eventuais dificuldades enfrentadas durante o trabalho;
- Uma seção de conclusão em que o aluno apresenta, de forma breve, sua opinião sobre os algoritmos implementados, embasada nos resultados apresentados.

3 Parte I - Análise de Algoritmos de Busca Sequencial

Esta primeira parte do projeto consiste em implementar quatro variações de buscas sequenciais discutidas em sala de aula. Serão disponibilizados dois arquivos de entrada [`inteiros_entrada.txt` e `inteiros_busca.txt`], que deverão ser utilizados, respectivamente, para inicialização e busca de dados. O arquivo de entrada contém 50.000 inteiros com valores entre 0 e 50.000. O arquivo de busca contém 50.000 inteiros com valores entre 0 e 70.000. O arquivo de entrada não contém inteiros duplicados. No entanto, o arquivo de busca contém inteiros duplicados e também inteiros que não existem no arquivo de entrada.

Para este exercício são propostas quatro variações de busca sequencial. Deve haver um único arquivo para cada variação implementada. Existe um *template* para cada variação contendo funções auxiliares que não precisam ser alteradas. Mas, novas funções devem ser implementadas conforme necessário.

Os resultados referentes ao tempo de busca deverão ser inclusos no relatório, contendo 1 gráfico de barras, onde o *eixo x* representa cada uma das variações implementadas e o *eixo y* representa o tempo médio das buscas. O gráfico deve contar com informações de desvio padrão. Para extrair o tempo médio e o desvio padrão as implementações deverão ser executadas pelo menos 3 vezes. Também deverá ser elaborada 1 tabela com os valores apresentados no gráfico. As variações propostas estão descritas a seguir:

a) Implemente um algoritmo que realize uma busca sequencial simples;

b) Implemente um algoritmo que realize uma busca sequencial com realocação por meio do método *mover-para-frente*, movendo o item encontrado para a primeira posição. Caso o item seja encontrado na primeira posição, nada precisa ser feito;

c) Implemente um algoritmo de busca sequencial com realocação por meio do método da *transposição*, movendo o item encontrado para uma posição anterior à posição em que o mesmo foi encontrado. Caso o item seja encontrado na primeira posição, nada precisa ser feito;

d) Implemente um algoritmo de busca sequencial que utilize índice primário; a busca sequencial simples deve ser utilizada tanto na tabela de índices quanto no vetor de elementos. Para isso, utilize uma estrutura de vetor sequencial "secionado"(lista não encadeada) para os elementos, onde a posição de cada "seção"do vetor sequencial é calculada por um indicador e o indicador, na tabela de índices, é uma variável inteira.

Defina o tamanho da tabela de índices (T) igual a $T = \frac{N}{S}$, onde N é o número total de inteiros da entrada (50.000) e S é o número total de inteiros que serão indexados por cada índice (10.000).

4 Parte II - Análise de Algoritmos de Busca com Espalhamento

Esta segunda parte do projeto consiste em implementar três variações de busca com espalhamento (*hashing*). Serão disponibilizados dois arquivos de entrada [`strings_entrada.txt` e `strings_busca.txt`], que deverão, respectivamente, ser utilizados para inserção e busca na tabela *hash*. O arquivo de inserção contém 50.000 palavras e o arquivo de busca contém 70.000 palavras. Em ambos os arquivos existem palavras duplicadas, sem acento ou caracteres especiais. Porém, o arquivo de busca contém palavras que não existem no arquivo de inserção. O tamanho máximo de uma palavra é de 20 caracteres.

Para este exercício são propostas três variações de busca com espalhamento. Deve haver um único arquivo para cada variação implementada. Existe um *template* para cada variação contendo funções auxiliares que não precisam ser alteradas. Mas, novas funções devem ser implementadas conforme necessário. Considere a função `converte` disponível nos *templates* para converter uma palavra para sua representação na forma de um inteiro. Com a representação inteira da palavra, utilize as funções de espalhamento por divisão (`h_div`) e por multiplicação (`h_mul`) para implementar as variações requisitadas.

Os resultados referentes ao número de colisões na inserção, tempo de inserção, tempo de busca e número de palavras encontradas durante a busca deverão ser inclusos no relatório para cada variação deste problema. Para cada variação, reporte os valores encontrados em 1 gráfico de barras, onde o *eixo x* representa as implementações realizadas e o *eixo y* representa o tempo médio com informação de desvio padrão. Para extrair a média e desvio padrão, as implementações deverão ser executadas pelo menos 3 vezes. Também deverá ser elaborada 1 tabela com os valores apresentados nos gráficos. As variações propostas estão descritas a seguir:

a) Implemente um algoritmo de busca com espalhamento estático que permita armazenar um conjunto de informações de tamanho limitado (fechado). Para tratar as colisões, considere a estratégia de *rehash* implementando o algoritmo *overflow progressivo*. Compare as duas

funções distintas de *hash* do *template*: função de divisão `h_div` (parâmetro $B = 150.001$), e; função de multiplicação `h_mul` (parâmetros $B = 150.001$ e $A = 0,6180$);

b) Implemente um algoritmo de busca com espalhamento estático que permita armazenar um conjunto de informações de tamanho limitado (fechado). Para tratar as colisões, considere a estratégia de *rehash* implementando o algoritmo *hash duplo*. Para isso, utilize as duas funções de *hash* especificadas na questão anterior de modo que sua função hash seja $h(x) = (h\_mul(x) + i * h\_div(x))\%B$ (parâmetros $B = 150.001$ e $A = 0,6180$).

c) Implemente um algoritmo de busca com espalhamento estático que permita armazenar um conjunto de informações de tamanho ilimitado (aberto). Para tratar as colisões, considere a estratégia de encadeamento em lista linear não ordenada. Compare as duas funções distintas de *hash* do *template*: função de divisão `h_div` (parâmetro $B = 150.001$), e; função de multiplicação `h_mul` (parâmetros $B = 150.001$ e $A = 0,6180$).

5 Conteúdo e data de entrega

O trabalho deve ser realizado em **duplas** ou **trios**, e a data de entrega é **02/06/2023** (serão tolerados atrasos até às 8h do dia posterior). Trabalhos com maior atraso não serão aceitos, recebendo nota **zero**. Quaisquer projetos similares terão nota **zero** independente de qual for o original e qual for a cópia. Será utilizada **ferramenta automatizada** para a detecção de **plágio**, com conferência manual de casos suspeitos. Os projetos devem ser entregues via Atividades do Tidia-ae (`https://ae4.tidia-ae.usp.br/portal`).

O formato da entrega deve ser um arquivo *.ZIP contendo:

- Um relatório em um arquivo de formato *.pdf* descrevendo o que foi desenvolvido no trabalho;
- Um projeto NetBeans contendo todo o código desenvolvido, incluindo todos os arquivos de extensão *.h* e *.c*. Para projetos desenvolvidos em outros ambientes, devem ser entregues os arquivos de extensão *.h* e *.c* juntamente com um `makefile` que permita a compilação e execução.

Observações importantes:

- A pasta a ser **zipada** deve ter por nome **só os números USP** dos alunos, separados por traços. Trabalhos sem esse padrão de nome de arquivo não serão corrigidos e valerão **zero**;
- É recomendado que os nomes e números USP dos integrantes do grupo sejam incluídos em todos os arquivos da entrega;
- O trabalho deve ser desenvolvido utilizando a linguagem C, em qualquer versão;
- **Todo o código** do projeto deve ter sido desenvolvido pelos integrantes do grupo. A utilização de qualquer código criado por terceiros, inclusive o código desenvolvido pelo docente em aula, será considerada como **plágio**;
- Para cada exercício, somente um único arquivo *.c* como solução é permitido. O arquivo deverá incluir a função *main*. A implementação dos arquivos *headers* (*.h*) é opcional;
- Em caso de dúvidas, envie email para `ilidio@alumni.usp.br` ou `fernanda.marana@usp.br`, ou simplesmente compareça no horário de atendimento.

6 Critérios de avaliação

- **7,0 pontos** pela implementação dos algoritmos de forma correta, legível, organizada e bem comentada. Incluindo tanto o código-fonte referente aos algoritmos, quanto o código referente às medições de tempo (3,0 pontos Parte I - 4,0 pontos Parte II);
- **3,0 pontos** pelo relatório, considerando estruturação, boa escrita e qualidade de apresentação de resultados.