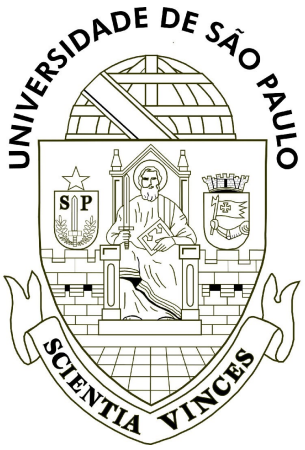
Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

SCC0606 - Estrutura de Dados II



Projeto 2

Alunos:
Lucas Lima Romero (13676325)
Luciano Gonçalves Lopes Filho (13676520)
Marco Antonio Gaspar Garcia (11833581)
Professor: Robson Leonardo Ferreira Cordeiro

Junho

2023

Sumário

1 Introdução

Os Algoritmos de busca desempenham um papel fundamental na era da informação. Com o crescimento exponencial de dados na internet, a capacidade de encontrar e recuperar informações de forma eficiente tornou-se essencial. Nesse contexto, o estudo de algoritmos de busca se torna imprescindível.

Esse relatório está estruturado da seguinte forma:

Após a introdução, tem-se a primeira parte, que consiste na análise de algoritmos de Busca sequencial, na qual serão analisadas quatro variações do algoritmo de busca sequencial por meio de dois arquivos de entrada. O arquivo de entrada possui 50.000 inteiros com valores entre 0 e 50.000, e o arquivo de busca contém 50.000 inteiros com valores entre 0 e 70.000.

Posteriormente se encontra a segunda parte, dedicada à análise de de algoritmos de busca por espalhamento (*hashing*), que consiste na implementação de três tipo de busca por espalhamento. Serão utilizados dois arquivos de entrada que deverão ser utilizados na inserção e busca na tabela *hash*. O arquivo de inserção contém 50.000 palavras e o arquivo de busca contém 70.000 palavras, ambos com palavras duplicadas.

Sobre o trabalho, a priori todo o código foi elaborado pelos participantes, com base nos código discutidos nas aulas. Os comentários do próprio código apresentam explicações detalhadas sobre a estrutura e os algoritmos. Foram testadas todas as funções de busca, até chegar ao código com as saídas enxutas, que se resumem aos tempos de busca e os desvios padrões deles, além do numero de colisões para algoritmos de *hashing*. A partir disso, todo o relatório foi feito, se baseando nos resultados obtidos.

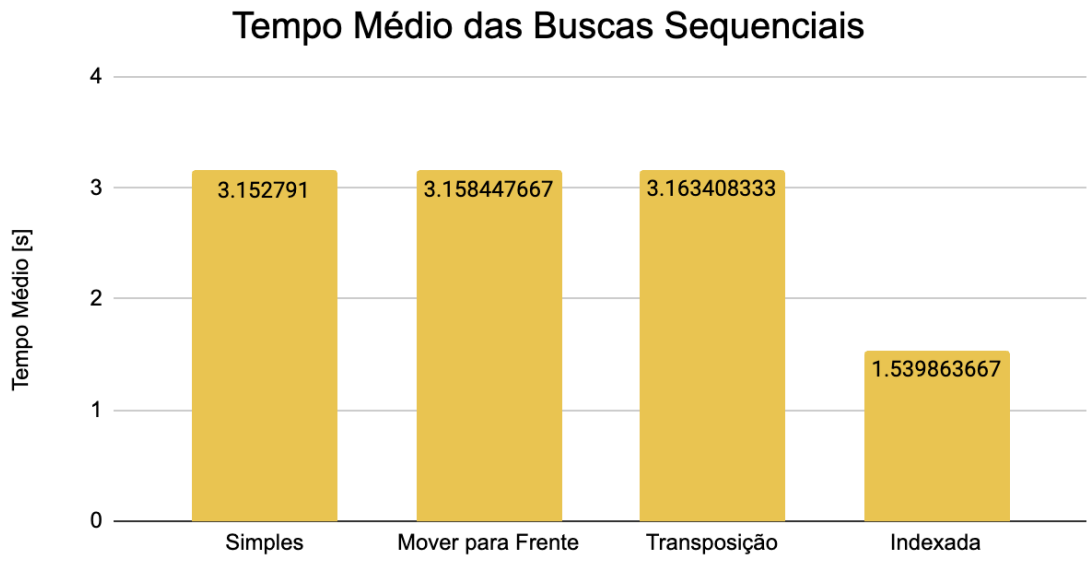
2 Parte I - Análise de Algoritmos de Busca Sequencial

Na Análise Empírica dos algoritmos de busca sequenciais, foram utilizados dois arquivos. O arquivo de entrada possui 50.000 inteiros com valores entre 0 e 50.000, e o arquivo de busca contém 50.000 inteiros com valores entre 0 e 70.000. Então, foram testados quatro tipos de algoritmos de busca sequencial: Busca sequencial Simples, Busca Sequencial "Mover-para-Frente", Busca Sequencial por Transposição e Busca Sequencial de Índice primário.

Em todos os casos, o número de itens encontrados foi de 35557, indicando que todos os códigos foram construídos corretamente. Utilizando-se da biblioteca time.h, foram medidos os tempos de execução, enquanto que suas médias e desvios padrões foram calculados com o auxílio do Excel. Os resultados estão dispostos na tabela e no gráfico abaixo.

Tabela 1: Tempos Medidos das Buscas Sequenciais (s)

Tentativas	Simples	Mover para Frente	Transposição	Indexada
1	3.152813	3.159872	3.152177	1.540169
2	3.150387	3.152007	3.180737	1.536598
3	3.155173	3.163454	3.157311	1.542824
Média	3.152791	3.158448	3.163408	1.539864
Desvio Padrão	0.002393	0.005860	0.015225	0.003124



Com base nos dados obtidos empiricamente, inferiu-se que a Busca Sequencial de Índice Primário foi o método mais eficiente de busca sequencial, Uma vez que seu tempo médio é aproximadamente metade do tempo médio das outras variações, nos casos testados acima.

3 Parte II - Análise de Algoritmos de Busca com Espalhamento

Primeiramente, foi necessário inserir os 50.000 elementos na tabela Hash, por meio das funções Hash caracterizadas pelos exercícios. Então, os 70.000 elementos da tabela de consultas foram buscados na tabela, também por meio da função Hash correspondente.

3.1 Busca com espalhamento estático fechado

Foram realizadas medições para três funções Hash diferentes: por divisão, por multiplicação e um misto entre as duas anteriores. O Hashing fechado necessita de uma estratégia para tratar as colisões, e neste caso, foi utilizado o overflow progressivo. Foi preciso apenas programar as funções de criação, inserção e busca, baseado nas funções Hash já presentes no template.

Nos três casos, o número de itens encontrados foi de 64098, evidenciando que as funções foram programadas corretamente pelo grupo. Os tempos de execução, tanto da inserção quanto da busca, foram contabilizados utilizando a biblioteca time.h, enquanto que as médias e desvios padrões foram calculadas com auxílio do Excel. Estes dados podem ser vistos nas tabelas e no gráfico abaixo.

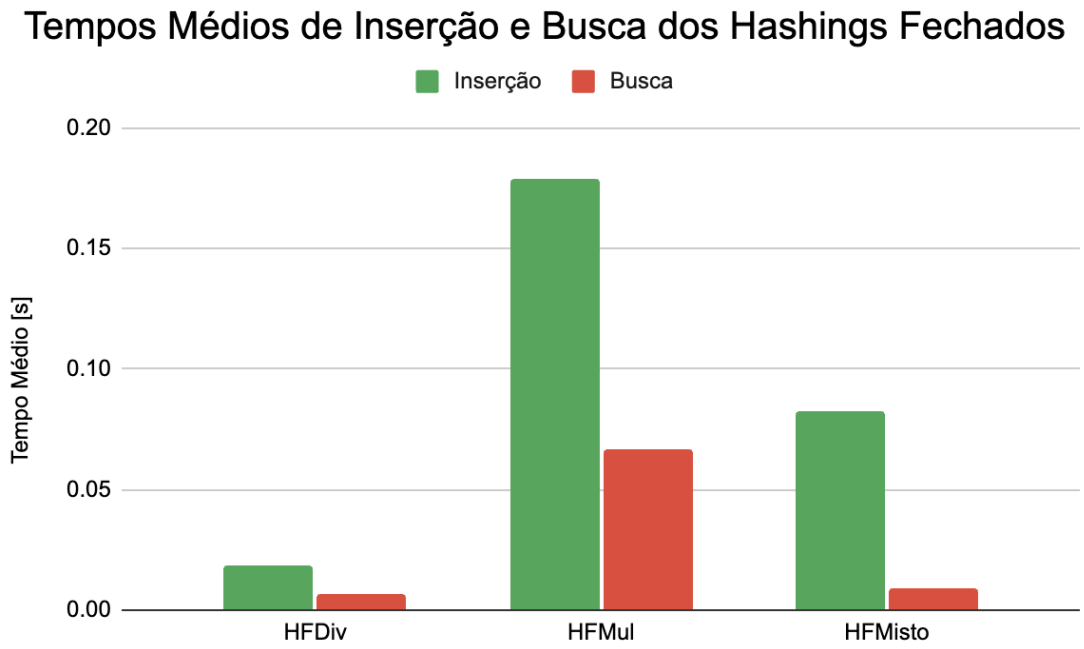
Tabela 2: Tempos de Inserção dos Hashings Fechados

Tentativas	Hash por Divisão	Hash por Multiplicação	Hash Misto
1	0.019244	0.179971	0.087858
2	0.018296	0.178393	0.076796
3	0.018401	0.178464	0.084049
Média	0.018647	0.178942	0.082901
Desvio Padrão	0.000520	0.000891	0.005620

Tabela 3: Tempos de Busca dos Hashings Fechados

Tentativas	Hash por Divisão	Hash por Multiplicação	Hash Misto
1	0.006689	0.066745	0.009000
2	0.006127	0.066306	0.009568
3	0.006186	0.066652	0.009037
Média	0.006334	0.066568	0.009202
Desvio Padrão	0.000309	0.000231	0.000318

Analisando o gráfico de barras, nota-se que o Hashing Fechado por Multiplicação foi o que pior performou dentre os três, enquanto que o mais rápido foi o por Divisão. O misto, apesar de ter uma função Hash mais cara dos que os outros dois, ficou em segundo lugar.



3.2 Busca com espalhamento estático aberto

Nesta seção, também foram utilizados Hash por Divisão e Multiplicação, entretanto, com Hashing Aberto. Com base nos conhecimentos do grupo de Estrutura de Dados I, foi programado um TAD (Tipo Abstrato de Dado), de uma lista encadeada, para tratar as colisões. Em ambos, o número de itens encontrados foi o mesmo, 64098.

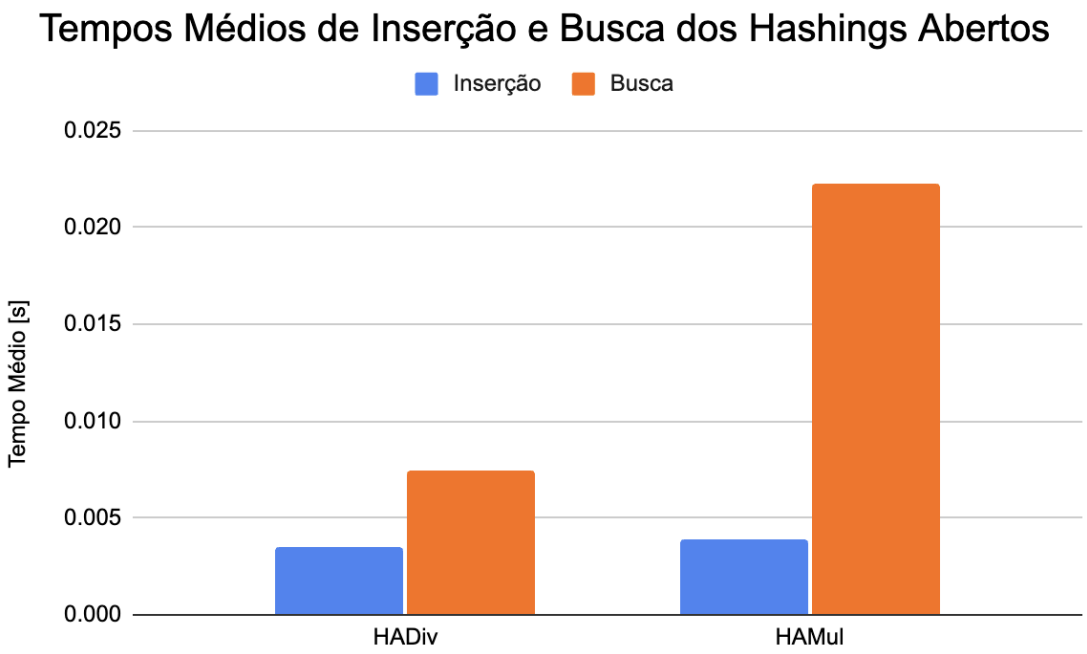
Os tempos de execução, tanto da inserção quanto da busca, foram contabilizados utilizando a biblioteca time.h, enquanto que as médias e desvios padrões foram calculadas com auxílio do Excel. Estes dados podem ser vistos nas tabelas e no gráfico abaixo.

Tabela 4: Tempos de Inserção dos Hashings Abertos

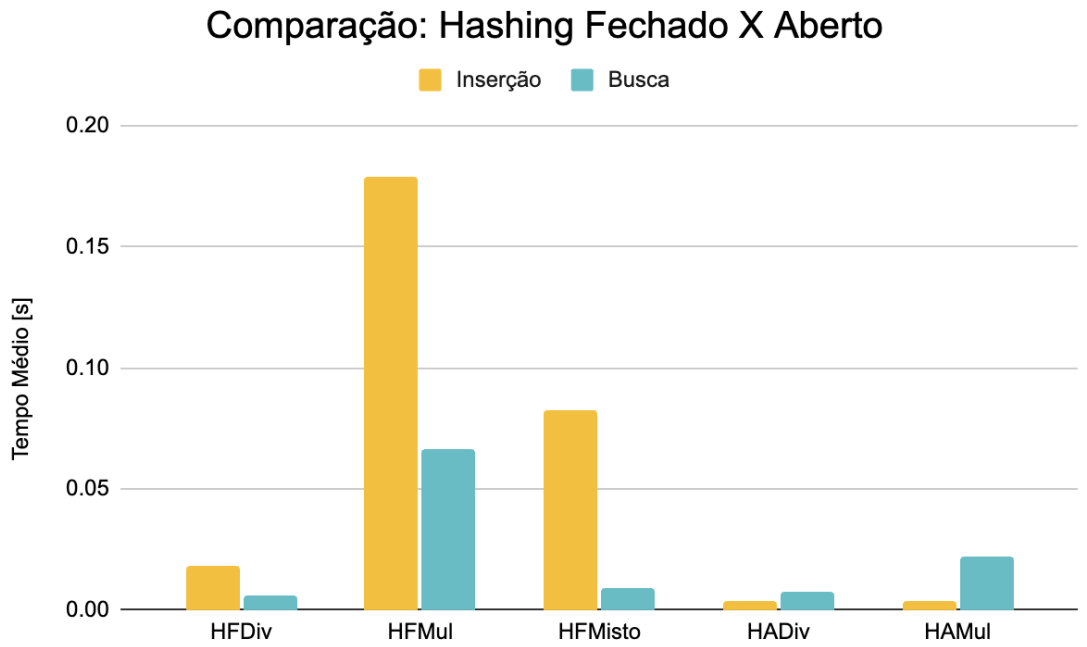
Tentativas	Hash por Divisão	Hash por Multiplicação
1	0.003605	0.007601
2	0.003368	0.007269
3	0.003539	0.007410
Média	0.003504	0.007427
Desvio Padrão	0.000122	0.0001666

Tabela 5: Tempos de Busca dos Hashings Abertos

Tentativas	Hash por Divisão	Hash por Multiplicação
1	0.003871	0.022558
2	0.004032	0.021762
3	0.003872	0.022486
Média	0.003925	0.022269
Desvio Padrão	0.000093	0.000440



O padrão se repetiu, e o Hash por Multiplicação foi mais lento que o por Divisão na busca, entretanto, é preciso estabelecer uma comparação mais clara com os resultados obtidos na seção anterior. Para isso, foi plotado um outro gráfico, com os tempos de todos os Hashings construídos durante o trabalho, obtendo uma maior clareza dos resultados.



Plotando-os lado-a-lado, é possível ter uma dimensão maior da diferença dos tempos de execução. O hashing aberto mostrou-se superior que o fechado no quesito velocidade de execução.

4 Conclusões

Sobre a primeira parte do trabalho, é possível concluir que a busca sequencial indexada reduz drasticamente o tempo de busca, e que é preferível sempre que possível, utilizá-la em detrimento da sequencial simples. As outras buscas obtiveram resultados muito próximos, evidenciando que as buscas Mover-para-frente e Transposição não são mais eficientes do que a simples em arranjos uniformes.

Já na segunda parte, concluiu-se que o Hash por divisão é mais rápido que o Hash por Multiplicação, e também que o Hashing aberto é superior ao Hashing fechado. Sempre que possível fazer uso de um TAD, deve-se utilizá-lo se o intuito for eficiência.

A única alteração que o grupo fez no template foi de pegar as variáveis B, M e N e defini-las como constantes, para que pudéssemos reduzir um argumento na maioria das funções, tornando o código mais enxuto.

Uma dificuldade enfrentada pelo grupo foi a da implementação do TAD, uma vez que foi necessário revisar conceitos da matéria Estrutura de Dados I, cursada no semestre passado. Além de programar as funções, foi necessário adaptá-las a realidade do problema de busca em uma Hash.

Outra dificuldade foi a conciliação dos códigos dos estudantes. Para contornar esse problema, utilizamos a tecnologia Git, por meio do Github, para ter um repositório desse trabalho, de modo que todos os integrantes fossem colaboradores ativos, permitindo que todos trabalhassem sobre o mesmo código durante todo o processo de estruturação.

Bibliografia

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, MIT Press & McGraw-Hill, 2001.

Tabelas: elaboradas pelos compiladores. **Gráficos:** elaborados pelos compiladores.