Disciplina **SCC0606**
Estrutura de Dados II

Prof. Robson Leonardo Ferreira Cordeiro

Relatório - Segundo Projeto

Felipe Andrade Garcia Tommaselli
(11800910)
Gianluca Capezzuto Sardinha
(11876933)
São Carlos, Brasil

1. Introdução

Primeiramente, é importante ressaltar a importância dos algoritmos de busca, considerando que é uma tarefa muito utilizada em nosso cotidiano. Além de que, certos métodos de ordenação, os quais foram estudados anteriormente, podem contribuir com o processo de busca, tornando-o mais eficiente. Dessa forma, lembrando que o objetivo principal desses algoritmos é localizar, num conjunto de elementos separados por chaves, o elemento correspondente a uma chave específica. Os algoritmos de busca que foram examinados serviram para o desenvolvimento da identificação de diferentes estilos/métodos que podem ser implementados com um mesmo intuito, a busca; garantindo que seja tarefa do programador decidir qual é o melhor caso para o seu uso desejado. Portanto, há a necessidade de uma prova tanto teórica quanto empírica, provando a complexidade entre elas. Portanto, bastou que fosse feita a implementação da forma variada para cada tipo de busca para a verificação dos resultados, possibilitando uma discussão sobre os dados obtidos tanto em aula quanto ao longo do trabalho.

2. Implementação

O código foi dividido conforme o template fornecido, de forma que apenas houveram alterações dentro de cada código dos exercícios especificamente.

Vale notar que tentou-se manter ao máximo a estrutura do código base, apenas alterando e adicionando funções específicas para cada problema. Além disso, a ideia principal de implementação dos códigos foram extraídas da teoria e dos exercícios trabalhados em aula, de forma a manter uma fidelidade com o conteúdo ministrado ao longo de todo o trabalho.

Na busca sequencial indexada o algoritmo de ordenação quicksort foi escolhido para ordenar o conjunto de dados, visto que ele possui uma boa estabilidade e abrangência para diversas entradas, mantendo ainda um custo razoável de tempo (O(logn)).

Por se tratar de um projeto com uma extensão e complexidade considerável é natural que surjam diversos problemas e erros ao longo das implementações. Contudo, é também natural que algumas dificuldades se sobressaiam, instigando a solução de problemáticas mais complexas, essas merecem maior atenção.

Uma das maiores dificuldades durante o trabalho foi na implementação do hashing, principalmente, no hashing aberto devido a utilização de listas encadeadas e na utilização de diversos ponteiros. Dessa forma, muitas vezes foi necessário parar para pensar tanto num diagrama mental do que estava acontecendo quanto num fluxograma de cada passo que ocorria no código. Porém, vale ressaltar que após a implementação de um dos hashings as coisas ficam mais simples para se trabalhar.

Em uma visão geral, todos os códigos foram implementados de maneira conservadora, sem grandes desvios do proposto. Algumas particularidades de implementação mais evidentes surgiram durante o ashing (tanto fechado quanto aberto), no qual era possível tomar pequenas decisões que não influenciaram o resultado

final. Nesses casos, optou-se sempre pela solução mais clara e otimizada possível.

3. Resultados e Discussão

Nesse momento serão brevemente discutidos alguns dos resultados obtidos, principalmente a análise de complexidade empírica dos algoritmos de busca propostos Em todos os casos, foi feita uma múltipla medição do tempo de busca e inserção para cada algoritmo e tirada a média e o desvio padrão desses dados. De forma que fosse possível no fim reunir os dados em uma comparação de eficiência entre os quatros algoritmos de busca sequencial, e outra entre os três de hashing.

Além disso, outros dados foram obtidos, como o número de elementos buscados, para a busca sequencial e em hashing, e o número de colisões para os algoritmos de hashing em específico. Esses dados todos foram verificados e são coerentes entre si, de tal maneira que fosse possível verificar que mesmo utilizando técnicas de busca diferentes, os elementos buscas eram os mesmo na mesma quantidade. Naturalmente, o número de colisões sofre variações entre os tipos de hashing, porém, verificou-se uma coerência satisfatória entre os resultados obtidos.

3.1. Busca sequencial

A busca sequencial de forma geral procura o elemento sequencialmente, ou seja, de elemento por elemento, com algumas variações que serão apresentadas, porém sob essa mesma ideia principal.

De forma geral, ela possui complexidade O(n) para o pior caso, com algumas possíveis melhorias de otimização que são de grande impacto prático. As diferenças entre as buscas sequenciais internamente com relação ao custo de tempo podem ser analisadas pela figura 1. Antes disso, cabe analisar cada implementação de busca de maneira mais específica.

3.1.1. Busca sequencial simples

Começando pela busca sequencial simples, que justamente leva o nome pela sua implementação extremamente simples e trivial. Esse algoritmo, de forma geral, funciona a partir de busca elemento a elemento no vetor dado, podendo percorrer o vetor todo no pior caso. Os parâmetros específicos do código implementado, como o número de elementos encontrados e o tempo médio, podem ser analisados pela tabela abaixo.

Tabela 1- Busca sequencial simples: Resultados obtidos

BUSCA SEQUENCIAL SIMPLES	
Rodada	**Tempo**
1	3,322
2	3,325
3	3,324
4	3,322
5	3,322
Tempo médio	3,322
Desvio padrão	0,001
Encontrados	35.557

3.1.2. Busca sequencial com realocação do tipo mover-para-frente

Seguindo a mesma linha de raciocínio da busca sequencial simples, a busca com realocação adiciona uma configuração a mais. Nesse caso, a realocação dos elementos mais procurados para o começo da lista, de forma que facilite a busca sequencial desses elementos.

O tipo de realocação mover para frente é o mais intuitivo para essa análise, uma vez que cada vez que o elemento é procurado ele é trocado com o elemento da primeira posição. Essa técnica também busca otimizar o curso O(n) de tempo da busca

sequencial. Pela implementação feita, foi possível avaliar que ele realmente busca o mesmo tanto de elementos que a busca anterior com uma pequena melhora de otimização no tempo médio.

Tabela 2- Busca sequencial com realocação mover para frente: Resultados e obtidos

BUSCA SEQUENCIAL MPF	
Rodada	**Tempo**
1	3,324
2	3,321
3	3,324
4	3,324
5	3,323
Tempo médio	3,324
Desvio padrão	0,002
Encontrados	35.557

3.1.3. Busca sequencial simples com realocação do tipo transposição

Exatamente como o algoritmo acima, a busca sequencial com realocação do tipo transposição busca facilitar a busca de elementos frequentemente procurados. Pensando em um conjunto de busca suficientemente grande, a busca por transposição faz mais sentido, uma vez que cada vez que o elemento é buscado, ele é jogado uma posição para frente (não diretamente para a primeira posição do vetor.

Novamente, tem-se um algoritmo O(n) com pequenas otimizações de custos de tempo. Além disso, o número de elementos buscados permanece o mesmo, mostrando que mesmo por métodos diferentes, os resultados obtidos são coerentes e satisfatórios.

Tabela 3- Busca sequencial com realocação do tipo transposição

BUSCA SEQUENCIAL TRANS	
Rodada	**Tempo**
1	3,330
2	3,317
3	3,367
4	3,324
5	3,337
Tempo médio	3,330
Desvio padrão	0,019
Encontrados	35.557

3.1.4. Busca sequencial indexada

Transitando para uma técnica um pouco distinta do visto até agora, a busca sequencial indexada busca dinamizar drasticamente o custo de tempo da busca sequencial, utilizando um vetor já ordenado. Por meio da criação de uma tabela de índices com posições esparsas do vetor já ordenado é possível diminuir o tamanho útil de busca do vetor consideravelmente.

Pelos resultados da implementação feita, vê-se que o tempo foi expressivamente menor, cerca de 5 vezes menor que os resultados das implementações acima, porém assintoticamente a busca sequencial indexada possui custo O(n) também. Novamente, verifica-se que o número de elementos encontrados é o mesmo.

Tabela 4- Busca sequencial indexada: Resultados obtidos

BUSCA SEQUENCIAL INDEX	
Rodada	**Tempo**
1	0,688
2	0,670
3	0,669
4	0,668
5	0,677
Tempo médio	0,670
Desvio padrão	0,008

Encontrados	35.557

3.2. Busca com Hashing

A busca com hashing por outro lado tem como ideia principal a criação de uma uma tabela hashing, na qual os elementos da entrada serão armazenados a partir do cálculo de sua posição por uma função hash pré definida. Assim, fica evidente que a busca de elementos deve ser extremamente eficiente, uma vez que é possível chegar na posição que o elemento se encontra diretamente pelo cálculo de sua posição com a função hashing.

A complexidade de tempo da busca com hashing não é trivial, pois exige diversos parâmetros e limitações, porém é claro que sua eficiência para os exercícios propostos é extremamente maior. As peculiaridades entre as buscas com hashing relação ao custo de tempo podem ser analisadas pela figura 2 e pela figura 3. Antes disso, cabe analisar cada implementação de busca de maneira mais específica, como feito anteriormente.

3.2.1. Busca com Hashing Fechado (Overflow progressivo)

Utilizando a técnica de hashing fechado (permite armazenar um tamanho de informações limitado) com rehash overflow progressivo, obteve-se os resultados demonstrados nas tabelas abaixo. Essa técnica de rehash possui a premissa de realizar uma pequena alteração na função hash a partir do momento que houver uma colisão em uma colisão, gerando uma nova posição para armazenar a informação.

O tempo médio de busca e de inserção foi extremamente satisfatório, bem como o número de colisões (o que aponta que a função hash e a técnica de rehash escolhidas foram adequadas). A distinção entre as duas funções hash, por multiplicação e por divisão foi notória no âmbito das colisões, o que indica que a função hash por divisão é mais otimizada que a por multiplicação no contexto dado. No caso geral, pela análise dos tempos de busca fica evidente que a busca com Hashing é mais eficiente que qualquer das buscas sequenciais acima.

Tabela 5- Busca com Hashing fechado (Overflow progressivo): Resultados obtidos

BUSCA HASHING FECHADO OP		
Rodada	**Tempo Inserção**	
	Multiplicação	Divisão
1	0,156	0,035
2	0,144	0,039
3	0,148	0,034
4	0,152	0,036
5	0,150	0,035
Tempo médio inserção	0,150	0,035
Desvio padrão	0,005	0,002
Colisões	33.959	25.461

BUSCA HASHING FECHADO OP		
Rodada	**Tempo busca**	
	Multiplicação	Divisão
1	0,344	0,075
2	0,325	0,082
3	0,339	0,074
4	0,332	0,076
5	0,332	0,077
Tempo médio busca	0,332	0,076
Desvio padrão	0,008	0,003
Encontrados	38.344	

3.2.2. Busca com Hashing Fechado (Função Hash duplo)

Seguindo a mesma linha de raciocínio da busca com hashing fechado acima, muda-se a técnica de rehash para a de função

dupla de hash. Nessa técnica, o rehash é feito a partir de uma função primária e uma secundária para caso haja colisões, a qual é ligeiramente diferente da primária, melhorando ainda mais o espalhamento dos valores pela tabela.

Para essa implementação não foram necessárias as duas medições para as duas funções como no rehash por overflow progressivo. A partir dos resultados, percebe-se um número de elementos encontrados igual a da técnica acima é um número satisfatório de colisões. Ainda é possível verificar que tanto o tempo médio de inserção como de busca estão de acordo com o esperado para essa busca com hashing.

Tabela 6- Busca com Hashing fechado (Função Hash duplo): Resultados obtidos

BUSCA HASHING FECHADO HD	
Rodada	**Tempo inserção**
1	0,093
2	0,092
3	0,109
4	0,089
5	0,094
Tempo médio inserção	0,093
Desvio padrão	0,009
Colisões	33.960

BUSCA HASHING FECHADO HD	
Rodada	**Tempo busca**
1	0,185
2	0,179
3	0,183
4	0,179
5	0,178
Tempo médio busca	0,179
Desvio padrão	0,003
Encontrados	38.344

3.2.3. Busca com Hashing Aberto

Seguindo por uma linha um pouco diferente, a busca com hashing aberto permite armazenar um tamanho de informações possivelmente ilimitado. Esse tipo de busca em hashing elimina a necessidade de aplicar uma técnica de rehash.

Para a implementação feita utilizou-se uma lista encadeada como estrutura de dado complementar a tabela de hash. Além disso, a busca em hash aberto foi dividida entre a busca com função hash por multiplicação e por divisão, os resultados obtidos foram tabelados abaixo. Novamente, foi possível analisar que o número de elementos encontrados foi o mesmo e que o hashing aberto por divisão ocasionou em menos colisões que o por multiplicação. Por fim, ainda é possível extrair o fato de que o hashing aberto é ainda mais rápido que as implementações de hash fechado acima.

Tabela 7- Busca com Hashing aberto: Resultados obtidos

BUSCA HASHING ABERTO		
	Tempo inserção	
Rodada	Multiplicação	Divisão
1	0,087	0,037
2	0,099	0,038
3	0,094	0,038
4	0,091	0,039
5	0,088	0,038
Tempo médio inserção	0,091	0,038
Desvio padrão	0,005	0,001
Colisões	33.959	24.046

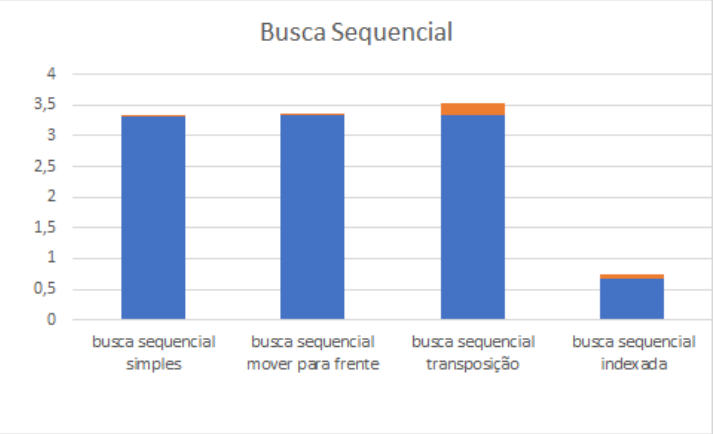
BUSCA HASHING ABERTO		
	Tempo busca	
Rodada	Multiplicação	Divisão
1	0,206	0,061
2	0,199	0,054
3	0,194	0,051
4	0,203	0,053
5	0,195	0,054
Tempo médio busca	0,199	0,054
Desvio padrão	0,005	0,005
Encontrados	38.344	

3.3. Análise geral

Ao avaliar cada implementação individual deve-se, agora, analisar o conjunto de comparativo entre as implementações similares. A priori, analisaremos a busca sequencial de forma geral por meio da figura 1, a qual relaciona o custo em tempo de cada implementação da busca sequencial.

Apenas uma observação a ser feita, quanto ao gráfico de comparação entre as buscas, o desvio padrão (erro) foi multiplicado por um fator de 10 para poder ser visível graficamente. Na inserção da busca com Hashing pelos valores serem muito pequenos o desvio padrão acabou por tomar valores mais expressivos, o que não necessariamente indica resultados imprecisos ou não confiáveis. O erro está representado por uma barra laranja acima da barra azul do tempo médio.

Figura 1- Comparações entre as buscas sequenciais



Por meio da análise dela é possível perceber alguns elementos gerais (válidos tanto para a busca sequencial quanto para o hashing), como o baixo desvio padrão entre os elementos, o que demonstra que os resultados colhidos são consistentes.

Além disso, verifica-se que os tempos de forma geral estão consistentes, onde os primeiros 3 algoritmos apresentam pequenas melhorias de eficiência de um a um. A busca sequencial por outro lado mostra uma melhoria expressiva de velocidade (sem considerar o custo de ordenação do vetor).

Na busca com hashing os resultados exigem uma análise um pouco mais refinada. Percebe-se que os algoritmos menos e mais eficientes são o hashing fechado com overflow progressivo por multiplicação e o hashing aberto por divisão, respectivamente. De forma geral é possível ver que a implementação da função hash por divisão é consideravelmente mais eficiente que por multiplicação.

Pelas figuras 2 e 3 é possível verificar que a análise da busca é válida para a inserção, resultado já esperado, uma vez que os processos são muito similares.

Figura 2- Comparações entre as inserções com Hashing

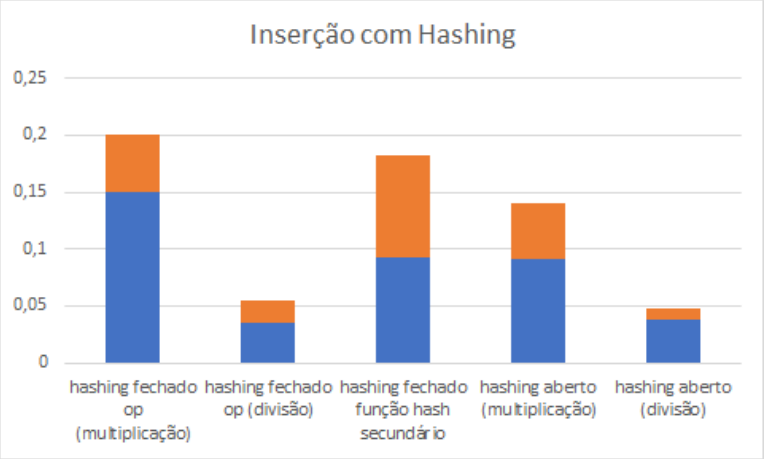
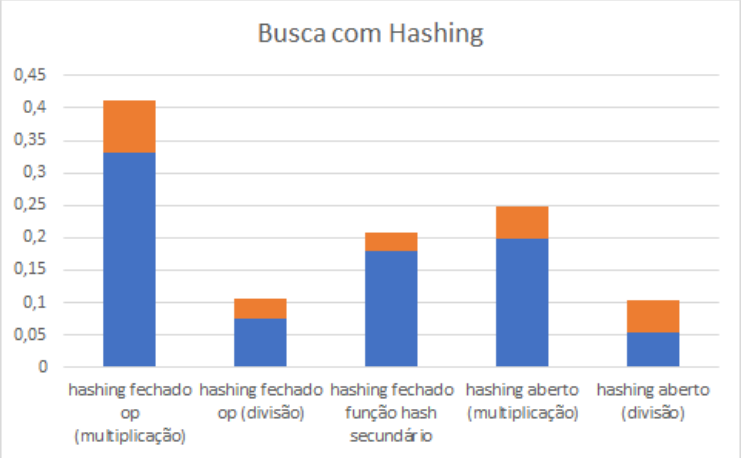


Figura 3- Comparações entre as buscas com Hashing



Além disso, vale ressaltar que a escolha do hashing a ser utilizado depende do contexto de uso, não necessariamente apenas da eficiência, fatores como tamanho da tabela de dados, quantidade de inserções e remoções, simplicidade entre outros também podem entrar na análise.

4. Conclusão

A execução dos diferentes tipos de algoritmos de busca sugeridos possibilitaram construir conhecimentos práticos; tratando-se da arte da busca de diferentes tipos que garantem a existência não só da ciência da computação, como também de diversas situações cotidianas em que devemos utilizá-la.

Dessa forma, é justamente a partir dessa perspectiva que se conclui acerca da potencialidade do estudo de algoritmos de busca. Por mais diversas que tenham sido as análises a cada algoritmo, pode-se concluir que todos possuem uma certa utilidade para cada caso.

Portanto, a partir do analisado, pode-se concluir que para um caso sem restrições o algoritmo de busca que refere-se ao Hashing, tanto aberto quanto fechado, possui grande vantagem no seu custo de tempo. Porém, nem sempre é possível implementá-lo, por isso foi feita a avaliação dos outros algoritmos de busca, como por exemplo os diferentes tipos de busca sequencial, que por sua vez também possuem adversidades.

Referências Bibliográficas

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2009. Disponível em: <https://mitpress.mit.edu/books/introduction-algorithms-third-edition>. Acesso: 21/07/2021

[2] BAASE, Sara (1988). Computer Algorithms. Introduction to Design and Analysis (em inglês) 2ª ed. Reading, Massachusetts: Addison-Wesley. Acesso em: 25/07/2021