Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Escola de Engenharia de São Carlos

Projeto 2 - Algoritmos de busca

João Pedro Gonçalves Ferreira - Nº USP: 12731314

Isaac Santos Soares - Nº USP: 12751713

Nicholas Estevão Pereira de Oliveira Rodrigues Bragança - Nº USP: 12689616

São Carlos, julho de 2022

1. Introdução

Algoritmos de busca estão entre os mais usados e estudados na computação. Eles têm como base a procura de qualquer elemento dentro de um conjunto de elementos com determinada propriedade e podem ser implementados de várias formas. Este segundo projeto visa a implementação e discussão de dois tipos de algoritmos: busca sequencial e de busca com espalhamento (hashing).

Foram implementados 4 algoritmos de busca sequencial, sendo o primeiro tradicional e os outros 3 com técnicas para aumentar a eficiência da busca. Além disso, foram implementadas 3 versões de hashing: duas de hashing fechado com diferentes formas de tratar colisões e uma terceira de hashing aberto com listas encadeadas simples.

As próximas seções deste relatório apresentam uma breve descrição dos códigos desenvolvidos bem como os resultados das análises empíricas desses algoritmos.

2. Busca Sequencial

1.1. Algoritmo de busca 1A

O primeiro algoritmo implementado foi o 1A, de busca sequencial simples, e a função de busca simplesmente percorre o vetor da posição 0 até a posição n - 1 e compara o elemento dessa posição com o elemento buscado. Ou seja, nesse método o vetor dos dados permanece inalterado.

1.2. Algoritmo de busca 1B

O algoritmo de busca sequencial 1B utiliza a técnica de mover-para-frente com o objetivo encontrar mais rapidamente elementos que são buscados mais frequentemente. Toda vez que um elemento buscado é encontrado numa posição i ele é imediatamente colocado na posição 0 do vetor e todos os elementos até a posição i são deslocados para a direita. Esse método faz com que elementos buscados mais frequentemente fiquem concentrados no início do vetor.

1.3. Algoritmo de busca 1C

O algoritmo de busca sequencial 1C utiliza a técnica de transposição também para encontrar mais rapidamente elementos buscados mais frequentemente. Toda vez que um elemento buscado é encontrado numa posição i ele é trocado com o elemento da posição i-1. Dessa forma, aos poucos, os elementos mais buscados vão sendo deslocados para o início do vetor; logo, esse método não é imediato.

1.4. Algoritmo de busca 1D

O algoritmo 1D de busca sequencial consiste em criar uma tabela de índices que é usada para reduzir a área de busca no vetor. Essa tabela de índices é criada com um

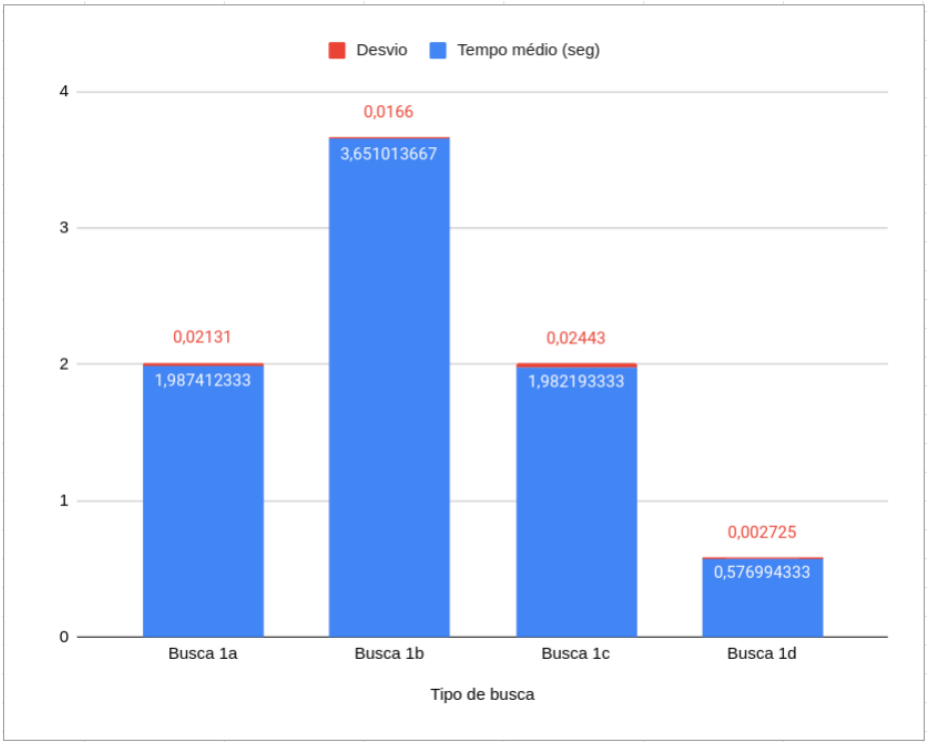
número fixo de índices igualmente espaçados, onde cada índice representa um intervalo da tabela de entrada original.

Ao buscar um elemento procura-o primeiro na tabela de índices para determinar o intervalo, entre dois índices, no qual o elemento buscado deve estar, se existir, no vetor de entradas ordenado.

1.5. Análise Empírica e Análise dos Resultados: Busca Sequencial

A tabela e o gráfico abaixo apresentam os resultados da análise empírica dos algoritmos de busca sequencial.

Tipo de busca	*Tempo médio (segundos)*	*Desvio padrão*
Busca 1a	1,987412333	0,02131
Busca 1b	3,651013667	0,0166
Busca 1c	1,982193333	0,02443
Busca 1d	0,576994333	0,002725



É evidente que o algoritmo mais demorado é o 1B, isso é consequência do método de mover-para-frente, que implica em deslocar parte dos elementos do vetor para a direita sempre que uma busca é realizada, para colocar o elemento buscado na posição 0; o que aumenta significativamente o custo da busca, sobretudo em vetores grandes.

O método que se mostrou mais eficiente foi o 1D, a busca sequencial indexada, pois, como esperado, o uso da tabela de índices reduz significativamente a região de busca no vetor original, acelerando a busca.

A pequena diferença de tempo entre o método da transposição (1C) e a busca sequencial tradicional (1A) evidencia que o método da transposição não é vantajoso para os dados analisados.

3. Busca com Espelhamento

Foram implementados três algoritmos de hashing, usando duas funções hash: por divisão e por multiplicação, para descobrir qual delas resulta em menos colisões e buscas mais rápidas.

3.1. Algoritmo de busca 2A

O primeiro algoritmo implementado foi o hashing fechado com overflow progressivo. A tabela é alocada dinamicamente porém cada posição só pode armazenar um registro por vez. Quando ocorre uma colisão o algoritmo percorre a tabela até encontrar a próxima posição disponível.

3.2. Algoritmo de busca 2B

O segundo algoritmo implementado foi o hashing fechado com função hash duplo. A tabela é alocada dinamicamente porém cada posição só pode armazenar um registro por vez. Para determinar a posição do registro aplica-se a função hashing por

multiplicação. Quando ocorre outra colisão aplica-se a função hash por divisão até que não ocorra mais colisão.

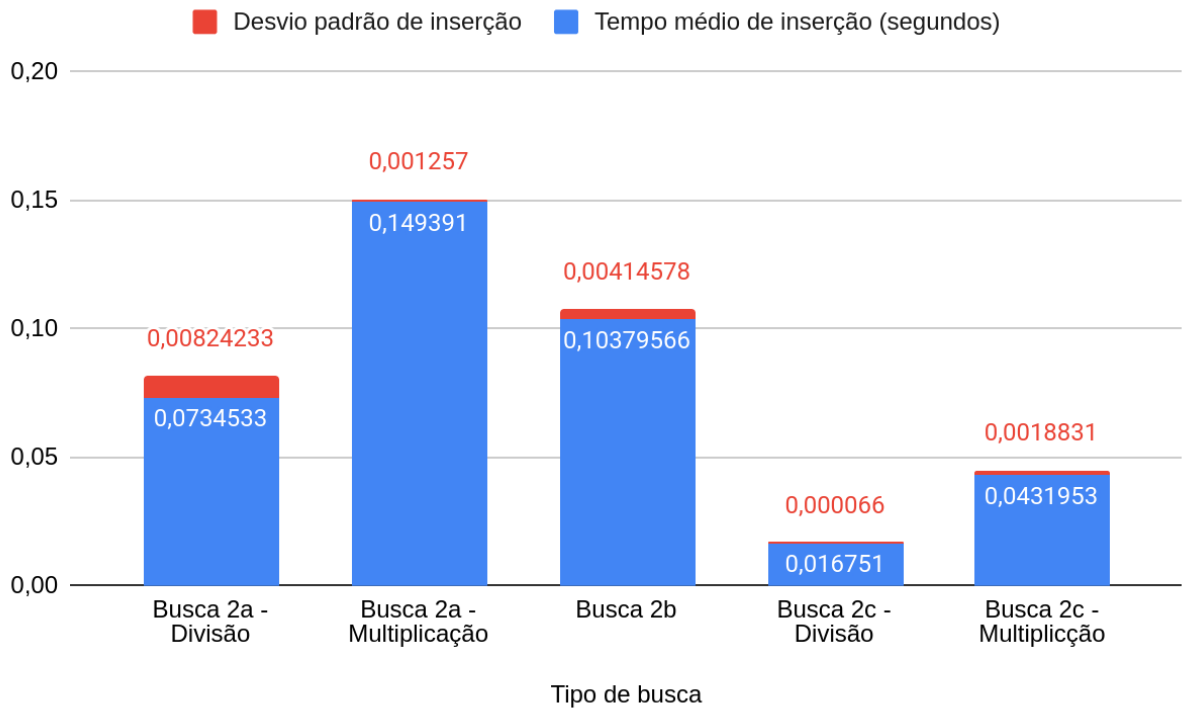
3.3. Algoritmo de busca 2C

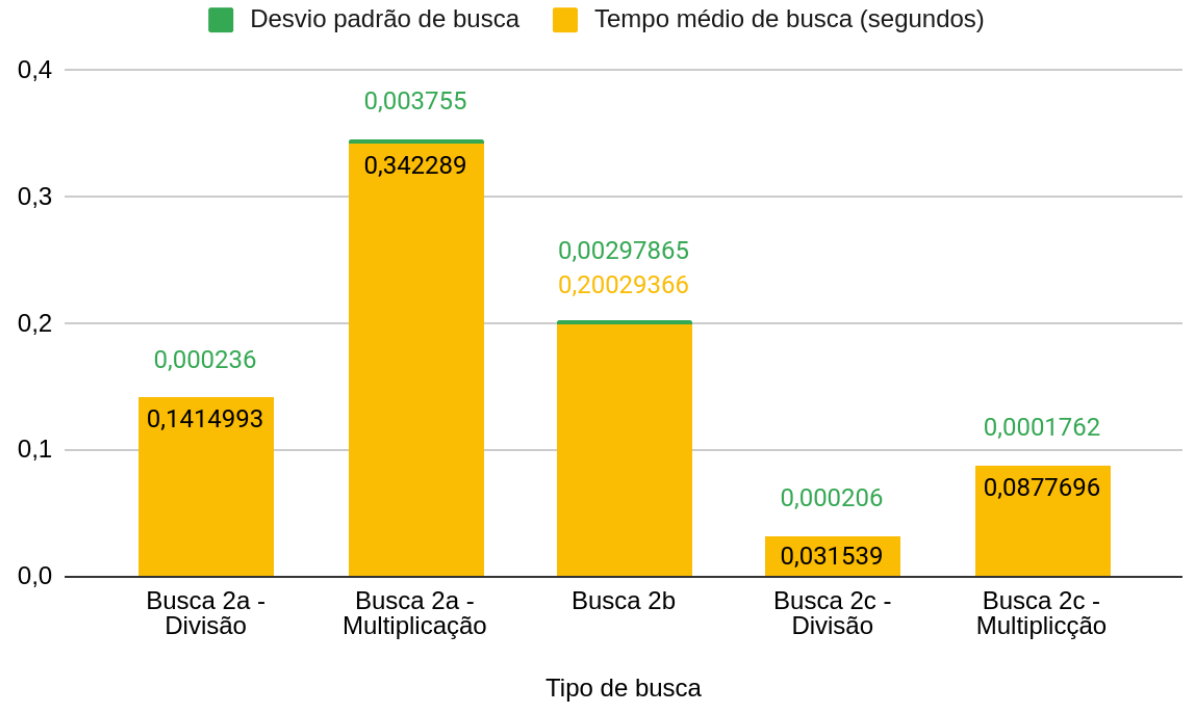
Por fim, foi implementado o hashing aberto com listas encadeadas não ordenadas. Cada elemento da tabela de hash é uma lista. Assim, quando ocorre uma colisão, o elemento é simplesmente adicionado ao fim da lista.

3.4. Análise Empírica e Análise dos Resultados: Hashing

A tabela e o gráfico abaixo apresentam os resultados da análise empírica dos algoritmos de hashing.

Tipo de busca	*Colisões na inserção*	*Tempo médio de inserção (segundos)*	*Desvio padrão de inserção*	*Tempo médio de busca (segundos)*	*Desvio padrão de busca*
Busca 2a - Divisão	1575740	0,0734533	0,00824233	0,1414993	0,000236
Busca 2a - Multiplicação	2825968	0,149391	0,001257	0,342289	0,003755
Busca 2b	939779	0,10379566	0,00414578	0,20029366	0,00297865
Busca 2c - Divisão	627447	0,016751	0,000066	0,031539	0,000206
Busca 2c - Multiplicação	1943609	0,0431953	0,0018831	0,0877696	0,0001762


Desvio padrão de inserção
Tempo médio de inserção (segundos)
0,20
0,15
0,10
0,05
0,00
0,00824233
0,0734533
0,001257
0,149391
0,00414578
0,10379566
0,000066
0,016751
0,0018831
0,0431953
Busca 2a - Divisão
Busca 2a - Multiplicação
Busca 2b
Busca 2c - Divisão
Busca 2c - Multiplicção
Tipo de busca


Desvio padrão de busca
Tempo médio de busca (segundos)
0,4
0,3
0,2
0,1
0,0
0,000236
0,1414993
0,003755
0,342289
0,00297865
0,20029366
0,000206
0,031539
0,0001762
0,0877696
Busca 2a - Divisão
Busca 2a - Multiplicação
Busca 2b
Busca 2c - Divisão
Busca 2c - Multiplicção
Tipo de busca

Com relação à inserção, o algoritmo menos eficiente foi o 2A com a função hash de multiplicação. Isso indica que essa função não é a mais adequada para o conjunto de dados em questão.

O algoritmo 2C, hashing aberto, mostrou-se o mais eficiente, o que indica que ele é o ideal para trabalhar com esse conjunto de dados, pois, apesar do uso de listas encadeadas não ordenadas, que deveriam aumentar a complexidade do algoritmo, ele ainda sim teve o menor tempo de execução.

Com relação às buscas os resultados foram semelhantes, o algoritmo 2A com função de multiplicação foi o menos eficiente de todos enquanto o algoritmo 2C foi o mais rápido apesar do uso de listas encadeadas.

Os algoritmos com a função hash por multiplicação (2A e 2C) são os que possuem menor eficiência na inserção, pois essa função gera mais colisões do que o hash por divisão. Isso também prejudica as buscas pois, no 2A, gera uma tabela com elementos amontoados e, no 2C, gera listas com mais elementos. Assim, a função hash por divisão é a mais adequada para o conjunto de dados analisado.

O método de hashing duplo possui um desempenho intermediário e isso deve-se ao fato desse método usar a multiplicação como hash primário e a divisão para o tratamento para colisões. Assim ocorrem as colisões iniciais do método por multiplicação, mas um novo hash é gerado pelo método de divisão. Isso torna o método mais eficiente do que o hash por multiplicação puro após a primeira colisão.

O algoritmo mais eficiente, tanto na inserção quanto na busca, foi o 2C com hash por divisão pois tem menos colisões e quando elas ocorrem o registro é inserido em uma lista encadeada simples, sem a necessidade de determinação de um novo código hash.

4. Conclusão

O desenvolvimento e análise dos algoritmos apresentados evidenciaram como os métodos de busca podem ter comportamentos diferentes para um mesmo conjunto de dados. Percebeu-se que, apesar de terem a mesma complexidade assintótica, certas variações da busca sequencial e do hashing mostraram-se muito mais eficientes que outras para esse conjunto de dados.

No caso da busca sequencial, o algoritmo mais eficiente foi a busca sequencial indexada, que reduz significativamente a região de busca no vetor por meio de uma tabela de índices. O método mover-para-frente foi o menos eficiente de todos devido à necessidade de grande modificação do vetor de dados a cada busca realizada.

Com relação ao hashing, a função hash por multiplicação mostrou-se inadequada para o conjunto de dados em questão pois gerou muito mais colisões do que o hash por divisão. O hashing aberto mostrou-se mais eficiente do que o fechado tanto na inserção quanto na busca, independentemente da função hash utilizada.

Em geral, o trabalho permitiu entender melhor o funcionamento das diferentes variações da busca sequencial e do hashing. Ficou evidente que, apesar da mesma complexidade assintótica, as variações desses algoritmos podem ter resultados muito diferentes na prática dependendo dos dados analisados.