Universidade de São Paulo – USP

Instituto de Ciências Matemáticas e de Computação – ICMC

# Projeto 2 – Estruturas de Dados II

Gabriel de Andrade Dezan
10525706

Ivan Mateus de Lima Azevedo
10525602

*Professor:*
Robson Leonardo Ferreira Cordeiro

29 de Junho de 2019

# 1 Exercício 1

## 1.1 Exercício 1a

Neste exercício foi implementada a busca sequencial simples. Para tanto, foram utilizados dois loops. O primeiro é responsável por pegar os valores que serão buscados no vetor *consultas*. O segundo, por fazer a busca em si percorrendo o vetor *entradas*.

Para a busca de cada elemento no vetor *consultas* é utilizada uma flag *achou*, que é setada quando o elemento é encontrado, fazendo com que o loop interno pare e o próximo elemento seja procurado. Além disso, a cada elemento encontrado, uma variável *encontrados*, que, claramente, conta o número de elementos encontrados, é incrementada.

## 1.2 Exercício 1b

Neste segundo exercício, foi implementada a busca sequencial com realocação por meio do método mover-para-frente. Esse método funciona como uma busca sequencial simples. A diferença é que além de setar a flag *achou* e incrementar a variável *encontrados*, também se move o valor encontrado para a primeira posição do vetor.

## 1.3 Exercício 1c

No terceiro, o método implementado foi o da transposição. Esse método funciona de maneira muito similar ao método do exercício anterior. A única diferença é que o valor encontrado é movido para a posição anterior ao invés da primeira posição do vetor.

## 1.4 Exercício 1d

O método utilizado neste exercício foi o de busca indexada. O primeiro a se fazer foi criar a tabela de índices. Como especificado no relatório, ela tem tamanho $T = \frac{N}{S}$, onde $N$ é o número de elementos no vetor de entrada e $S$, o número de elementos que serão indexados por cada índice. Como $N = 50k$ e $S = 10k$, o tamanho da tabela de

índices é igual a 5 e seus valores são os elementos do vetor *entradas* das posições 0, 9.999, 19.999, 29.999 e 39.999, respectivamente.

Com a tabela de índices criada, o próximo passo foi fazer a busca na tabela de índices. Pata tanto, foi utilizado um loop que corre a tabela. Enquanto o índice $j$ for menor que o tamanho da tabela e o valor a ser buscado for maior ou igual que o valor indicado pelo índice, a busca continua. Ou seja, o loop só para se chegar ao fim ou se o número buscado for menor que o valor da posição $entradas[tabela[j]]$. Dessa forma, como o valor é menor que $entradas[tabela[j]]$, a função retorna $tabela[j-1]$.

Após fazer a busca na tabela e ter o valor do índice, utiliza-se o mesmo algoritmo de busca sequencial simples, mas iniciando-se a partir desse índice.

# 2 Exercício 2

## 2.1 Exercício 2a

Neste primeiro exercício, foi implementado uma busca por hashing estático e fechado rehash com overflow progressivo para tratar das colisões. Foram utilizadas duas funções hash: uma por divisão (parâmetro $B = 150.001$) e outra por multiplicação (parâmetros $B = 150.001$ e $A = 0.6180$). As funções são:

$$h_{div}(x) = ((x\%B) + i)\%B$$
$$h_{mul}(x) = ((fmod(x * A, 1) * B) + i)\%B$$

A estratégia utilizada foi a mesma nos dois casos. Primeiro inicializou-se a tabela com strings vazias (). Após isso, inseriu-se os elementos. Para a inserção, converte-se a string de entrada utilizando a função dada e calcula-se a posição de inserção através da função hash. Se a posição estiver vazia, insere a string; senão, recalcula o valor da posição utilizando o overflow progressivo. Isso se repete até que se ache uma posição vazia. Quando acontecer de haver uma colisão, ela é contabilizada. Quando a string for inserida, uma flag *inseriu* é setada e o loop é encerrado. Também se considera o caso de haver palavras repetidas (elas não são inseridas).

Para a consulta dos dados a estratégia foi a mesma. A diferença é que se chegar em uma posição vazia, como é utilizado overflow progressivo, o loop já pode ser encerrado, pois a string não está na tabela.

## 2.2 Exercício 2b

No segundo exercício, foi utilizado o método de rehash de hash duplo em um conjunto limitado. A função de hash inicial escolhida foi a de multiplicação. Se ocorresse uma colisão após que o método de multiplicação tivesse sido aplicado, usaríamos o segundo hash para as tentativas seguintes. O mesmo possui a fórmula:

$$h(x) = (h_{mul}(x) + i \cdot h_{div}(x))\%B$$

## 2.3 Exercício 2c

Como o conjunto de informações desse exercício deve ser ilimitado, utilizamos a busca por hashing estático e aberto. Para isso, foi inicializado um vetor de lista encadeadas. A posição em que o elemento deve ser inserido é determinada pelas mesmas duas funções de hashing do exercício 2a.
A diferença do hashing aberto é que no momento em que ocorria alguma colisão, o elemento era inserido na lista que se encontrava na posição do vetor.
Como a ordem da lista não é importante, o item era inserido logo no começo da lista, para garantir tempo constante de inserção.
A busca era feita acessando o vetor na posição determinada pela função de hashing e percorrendo a lista que se encontrava nessa posição até que o elemento fosse encontrado.

# 3 Resultados

## 3.1 Exercício 1

Para o exercício 1, foram reportados os tempos de três execuções dos métodos. Na tabela abaixo podemos ver o resultado desse teste empírico:

| Busca | | | | |
|---|---|---|---|---|
| | Sequencial Simples | Mover-para-frente | Transposição | Índice Primário |
| Tempo 1 (s) | 3,218750 | 3,640625 | 3,375000 | 0,656250 |
| Tempo 2 (s) | 3,234375 | 3,406250 | 3,390625 | 0,656250 |
| Tempo 3 (s) | 3,312500 | 3,359375 | 3,406250 | 0,625000 |
| Média | 3,255208333 | 3,46875 | 3,390625 | 0,645833333 |
| Desvio Padrão | 0,050227 | 0,150682 | 0,015625 | 0,018042 |

Figura 1: Tabela com resultados dos métodos de buscas

Para melhor visualização, foi feito um gráfico de barras com as médias dos tempos de cada método

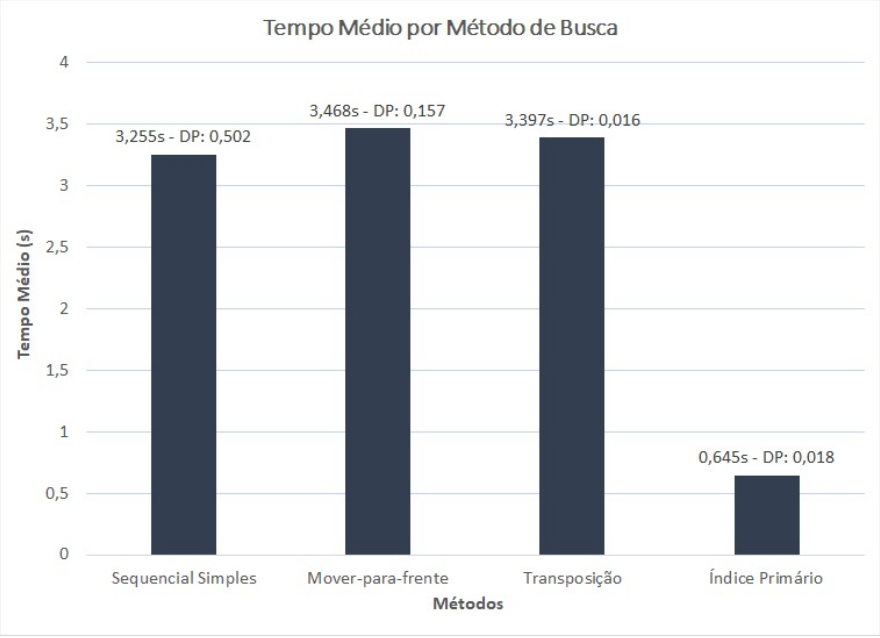


Figura 2: Média de tempo por método (DP = Desvio Padrão)

## 3.2 Exercicio 2

No exercício 2, assim como no exercício 1, foram reportados os tempos em uma tabela. A diferença que temos é que como são métodos de busca por hashing, os tempos de inserção e busca foram contabilizados, assim como o número de colisões.

| Hashing | | | | | |
|---|---|---|---|---|---|
| | Fechado - Multiplicação (Overflow Progressivo) | Fechado - Divisão (Overflow Progressivo) | Fechado (Segundo Hash) | Aberto - Multiplicação (Encadeamento) | Aberto - Divisão (Encadeamento) |
| Tempo de Inserção 1 (s) | 0,234375 | 0,078125 | 0,109375 | 0,015625 | 0,015625 |
| Tempo de Busca 1 (s) | 0,890625 | 0,328125 | 0,406250 | 0,140625 | 0,046875 |
| Tempo de Inserção 2 (s) | 0,234375 | 0,078125 | 0,125000 | 0,000000 | 0,015625 |
| Tempo de Busca 2 (s) | 0,875000 | 0,312500 | 0,359375 | 0,140625 | 0,046875 |
| Tempo de Inserção 3 (s) | 0,234375 | 0,781250 | 0,125000 | 0,015625 | 0,015625 |
| Tempo de Busca 3 (s) | 0,796875 | 0,328150 | 0,359375 | 0,125000 | 0,046875 |
| Média de Inserções | 0,234375 | 0,312500 | 0,119792 | 0,010417 | 0,015625 |
| Média de Buscas | 0,854167 | 0,322925 | 0,375000 | 0,135417 | 0,046875 |
| Desvio Padrão de Inserções | 0,000000 | 0,405949 | 0,009021 | 0,009021 | 0,000000 |
| Desvio Padrão de Buscas | 0,050227 | 0,009028 | 0,027063 | 0,009021 | 0,000000 |
| Colisões | 33959 | 25461 | 34232 | 49495 | 39582 |

Figura 3: Resultados de buscas com hashing

Temos também gráficos para melhor visualização dos tempos e número de colisões dos métodos.

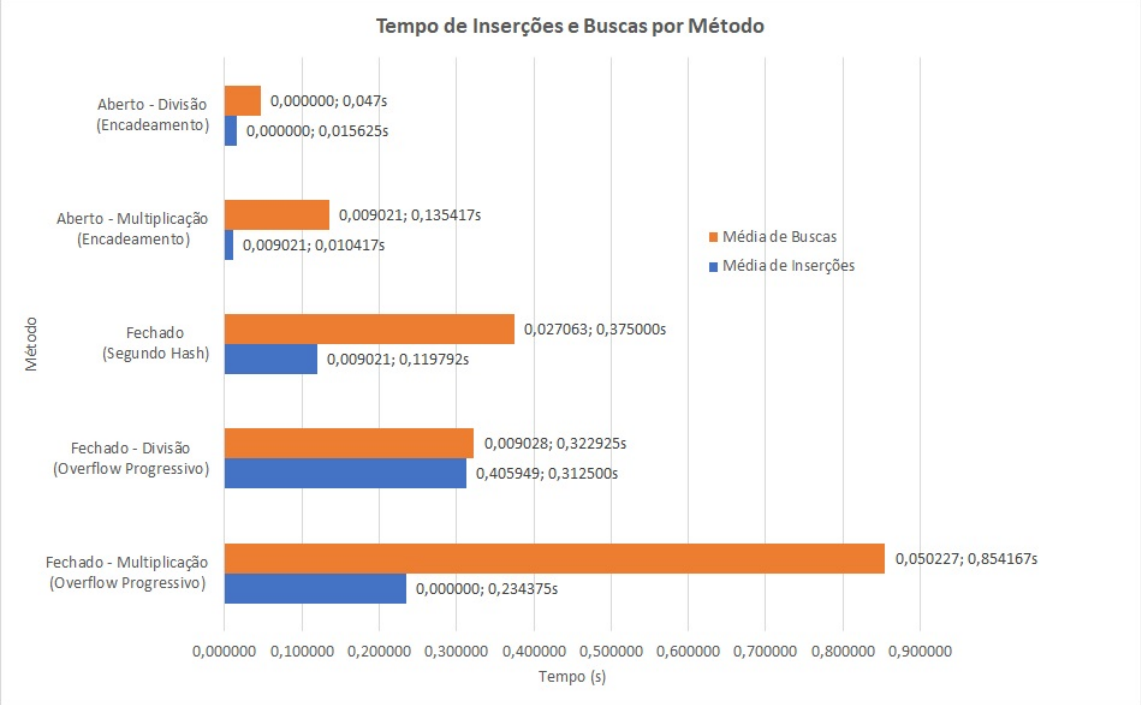


Figura 4: Médias de tempos de inserção e busca por método - Os valores em cada barra são respectivamente o Desvio Padrão e o valor da média do Tempo

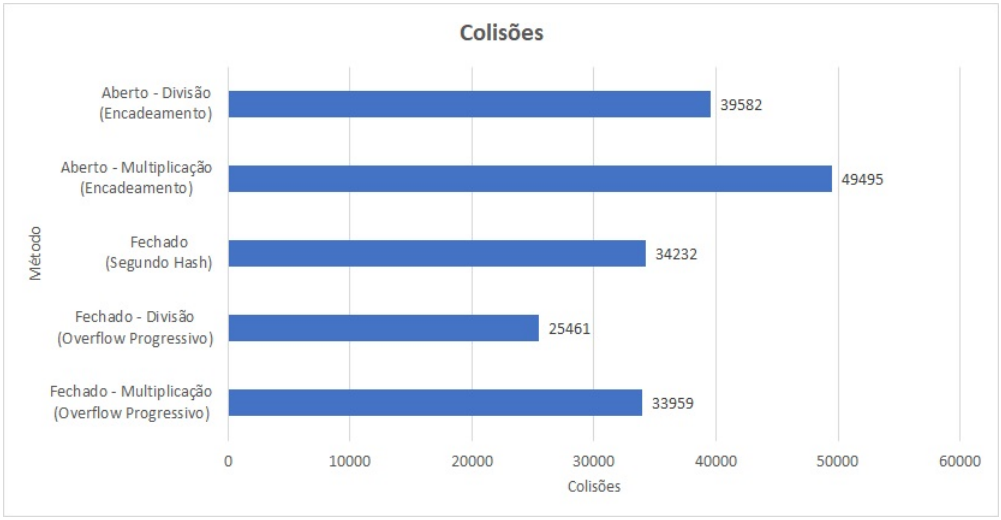


Figura 5: Número de colisões por método

# 4 Conclusão

## 4.1 Exercício 1

A partir dos dados da tabela da figura 1 e do gráfico da figura 2, podemos perceber que, de todos, a busca sequencial indexada é o melhor dos três algoritmos de busca sequencial, seguido por sequencial simples, mover-pra-frente e transposição.

## 4.2 Exercício 2

A partir dos dados da tabela da figura 3 e do gráfico da figura 4, podemos perceber que dentre todas as estratégias de hash utilizadas, a melhor delas é **de longe** a do hashing aberto, em particular o por multiplicação, por conta da busca ser mais rápida. Em segundo lugar vem o hashing fechado com hash duplo e por último o hashing fechado com overflow progressivo.