Trabalho Prático - Hashing Extensível

Isabela Aguilar, Lucas Milard

[1]Pontificia Universidade Católica de Minas Gerais (PUC MG)
Rua Cláudio Manoel, 1.162 – Funcionários – Belo Horizonte – MG – Brasil

1. Acessos

Todo o codigo pode ser acessado no repositório: `https://github.com/Kronomant/CRUD-arquivo-AED-III`

Planilhas e gráficos com as análises estão disponiveis em: `https://1drv.ms/x/s!AkOwS9prkWArgZlbuGhNtoStbwNasw?e=q3UzTa`

2. Introdução

Por meio dos conhecimentos adquiridos ao longo das aulas de Algoritmos e Estrutura de Dados III foi possível a implementação de um sistema de prontuários baseado em hashing dinâmico. Sendo assim, essa documentação abordará todos os aspectos que foram considerados durante a criação dessa solução, além de exemplificar o seu funcionamento por meio de diversos testes. Além disso, foi realizada uma comparação entre os tempos dos processos de inserção e busca de prontuários. Essa análise é importante para medir a eficácia e aplicabilidade da solução que foi codificada.

3. Descrição do problema

O sistema deve criar três arquivos, o arquivo mestre que contém todos os prontuários, que são compostos pelos dados dos pacientes. Um arquivo diretório que possui a profundidade global, e um vetor onde cada índice tem o endereço de um bucket. E por fim um arquivo índice que é composto por uma série de Buckets que tem seus tamanhos determinados pelo usuário e carregam o cpf e o endereço desse cpf no arquivo mestre.Assim que um novo prontuário é criado é necessário inserir o cpf correspondente no arquivo de índice, se um bucket está cheio é necessário inserir algum registro um split deve ocorrer. O arquivo do diretório e do índice formam o hashing extensível, que cresce conforme a necessidade e deixa a busca muito mais rápida no arquivo indexado, nesse caso o arquivo mestre de prontuários.

3.1. Modelagem e técnicas utilizadas

A linguagem escolhida para desenvolver a aplicação foi a linguagem Java. Essa escolha foi tomada devido ao extenso número de recursos que ela oferece, principalmente por ser uma linguagem orientada a objetos. Além disso, durante grande parte do curso de Ciência da Computação estamos utilizando Java para fins acadêmicos e isso contribuiu para um melhor entendimento de como a solução desse trabalho seria executada.

Foram criadas cinco classes para conseguir resolver o problema sendo duas classes, ArquivoMestre.java e Diretorio.java as principais classes para inserir nos três arquivos. A classe ArquivoMestre ficou responsável por inserir no arquivo prontuario.db que é o arquivo mestre com todos os prontuários. Além disso, essa classe chama a Diretorio sempre que insere ou busca algum registro. A classe Diretorio gerencia tanto o

arquivo diretorio.db quanto o indice.db, ela é reponsavel por inserir ou buscar um cpf fornecido pela ArquivoMestre dentro do índice e também realizar a expansão do diretório e do índice quando necessário. As outras 2 classes são Bucket.java e Registro.java que são estruturas de dados que vão ser inseridas nos arquivos.

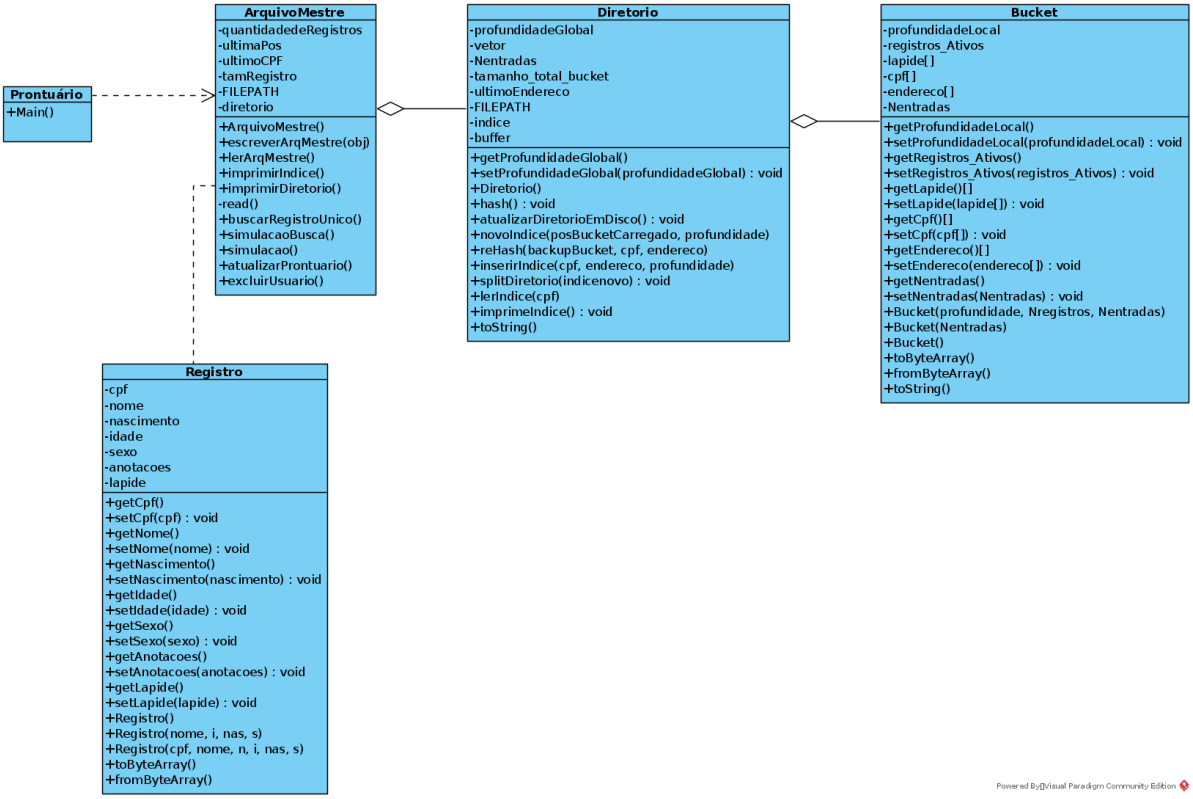

Figure 1. Diagrama de Classes

4. Testes

Foram realizados testes com inúmeras combinações distintas de quantidade de registros, profundidades globais, tamanhos de bucket e tamanhos de campos de anotações. Todos os testes foram realizados em ambos os computadores para permitirem uma comparação posterior entre as máquinas.

4.1. Máquina 1

Aluno: Lucas Milard
Sistema Operacional: Linux Mint
Memória RAM: 4GB
Processador: Intel Core i3-6100U
Disco Rígido: 500GB

Simulação - Lucas Milard							
Teste	Quantidade de Registros	Tempo de Inserção (seg)	Tempo de Busca (seg)	Pronfundidade Global Inicial	Pronfundidade Global Final	Anotações	Buckets
1	5000	3,395985	0,623432	2	3	1000	1000
2	5000	3,27038	0,558797	2	4	1000	500
3	5000	1,626695	0,427824	5	7	1000	100
4	10000	4,879229	0,845884	2	5	1000	500
5	10000	12,25656328	65,67350156	8	8	5000	5000
6	20000	14,473353	2,211835	2	5	1000	1000
7	30000	38,13987963	90,39667684	8	8	5000	5000
8	100000	70,66495812	9,990745086	2	7	1000	1000
9	200000	191,8310184	22,06419479	2	8	1000	1000
10	500000	279,0998554	276,2958789	2	8	1000	5000
11	900000	438,54552	65,05892002	2	7	120	1000

Figure 2. Simulação - Lucas Milard

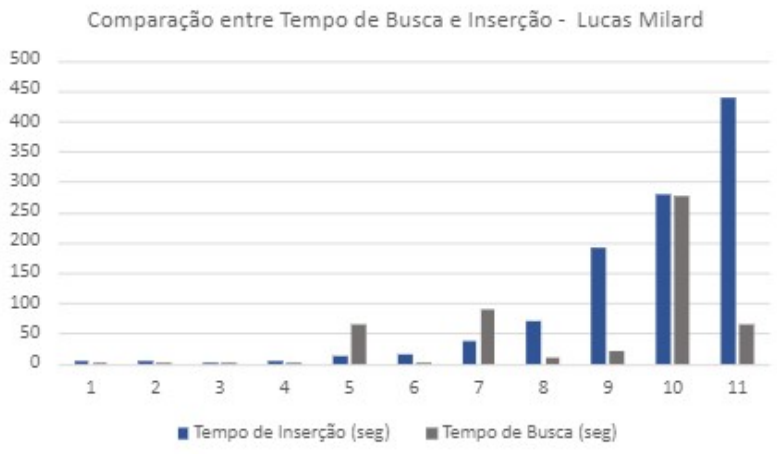

Figure 3. Comparação entre Tempo de Inserção e de Busca - Lucas Milard

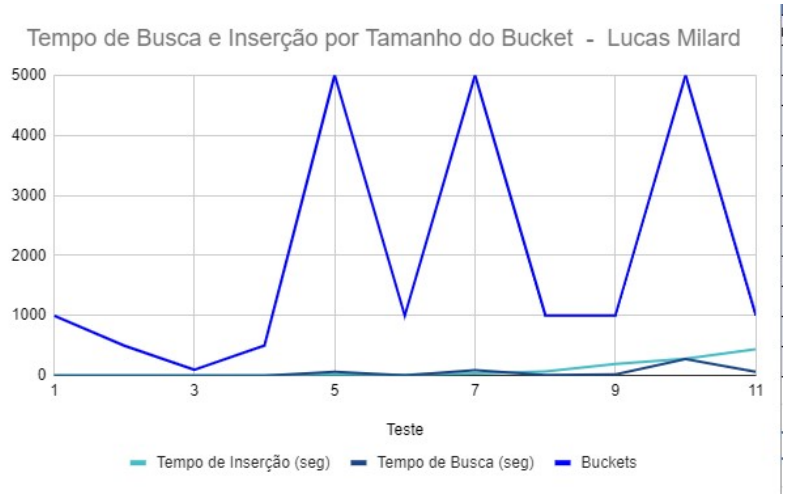

Figure 4. Tempo de Inserção e de Busca por Tamanho do Bucket - Lucas Milard

4.2. Máquina 2

Aluna: Isabela Aguilar
Sistema Operacional: Windows 10
Memória RAM: 8GB
Processador: i5 8th Gen
Disco Rígido: 2TB

Simulação - Isabela Aguilar							
Teste	Quantidade de Registros	Tempo de Inserção (seg)	Tempo de Busca (seg)	Pronfundidade Global Inicial	Pronfundidade Global Final	Anotações	Buckets
1	5000	6,2295937	0,9664638	2	3	1000	1000
2	5000	6,8510956	0,8639571	2	4	1000	500
3	5000	5,5067719	0,776642	5	7	1000	100
4	10000	13,1833521	1,7894728	2	6	1000	500
5	10000	13,4941622	3,4470372	8	8	5000	5000
6	20000	28,9299075	4,9060673	2	5	1000	1000
7	30000	75,1826041	17,4271411	8	8	5000	5000
8	100000	191,0842035	135,6147608	2	7	1000	1000
9	200000	430,7150418	331,1946835	2	8	1000	1000
10	500000	1537,837618	867,3257942	2	7	1000	5000
11	900000	2923,029009	1928,406709	2	10	120	1000

Figure 5. Simulação - Isabela Aguilar

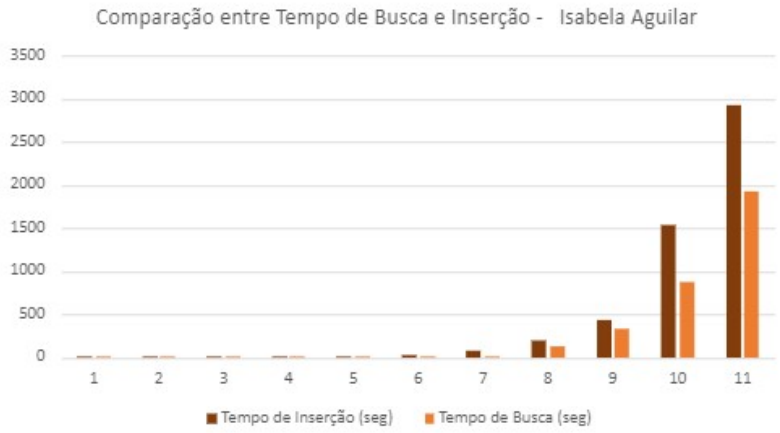

Figure 6. Comparação entre Tempo de Inserção e de Busca - Isabela Aguilar

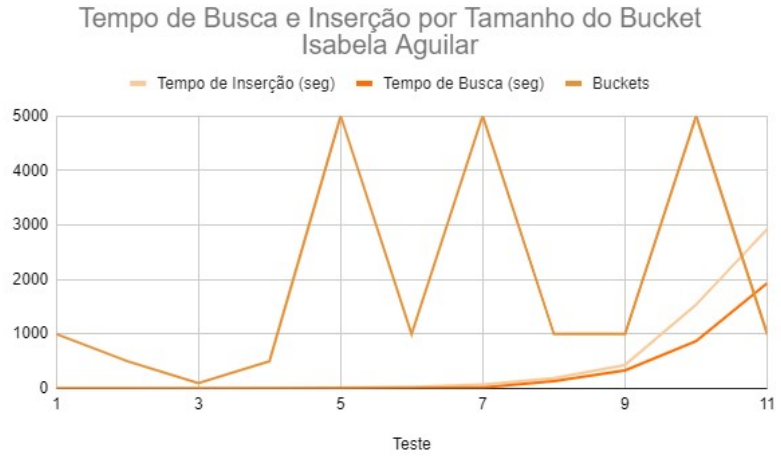

Figure 7. Tempo de Inserção e de Busca por Tamanho do Bucket - Isabela Aguilar

5. Testes com pen-drive

Simulação - Pen-drive							
Teste	Quantidade de Registros	Tempo de Inserção (seg)	Tempo de Busca (seg)	Pronfundidade Global Inicial	Pronfundidade Global Final	Anotações	Buckets
1	5000	222,81987	0,7593431	2	3	1000	1000
2	5000	312,2840928	1,7588923	2	4	1000	500
3	5000	178,1279302	1,429103	5	7	1000	100
4	10000	379,3563231	3,584213	2	6	1000	500
5	10000	572,6488944	4,7894728	8	8	5000	5000
6	10000	332,5482044	2,2092156	2	5	1000	1000

Figure 8. Simulação - Pen-drive

6. Análise de Resultados

Por meio de todos os testes realizados podemos perceber a nítida diferença entre as operações realizadas em sistemas operacionais diferentes. O sistema operacional Linux se apresentou muito mais eficiente durante as ações de inserção e busca da aplicação codificada. Além dsso, em um dos testes realizados foram inseridos muitos registros de uma só vez o que contribuiu para que o tempo de busca fosse muito mais elevado do que nos outros testes. Esse comportamento pode ser explicado pela quantidade de registros que precisam ser percorridos durante a busca.

Outro ponto observado foi quanto aos testes realizados com arquivos alocados em um pen-drive. Foi notória a diferença entre os tempos de inserção e busca, onde foi percebido que o tempo de inserção foi bem maior que o tempo de busca para a grande maioria dos testes realizados. Nota-se também que, devido ao fato de que os arquivos Diretorio, Indice e Prontuario não se encontravam na mesma pasta que o código fonte, o tempo tanto de busca quanto de inserção foi extremamente elevado em comparação aos tempos obtidos nas duas máquinas.

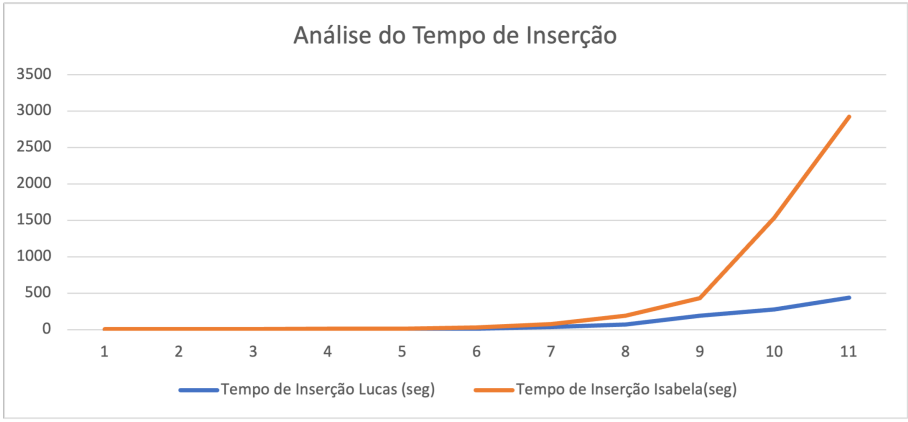

Figure 9. Comparação entre as duas máquinas - Inserção

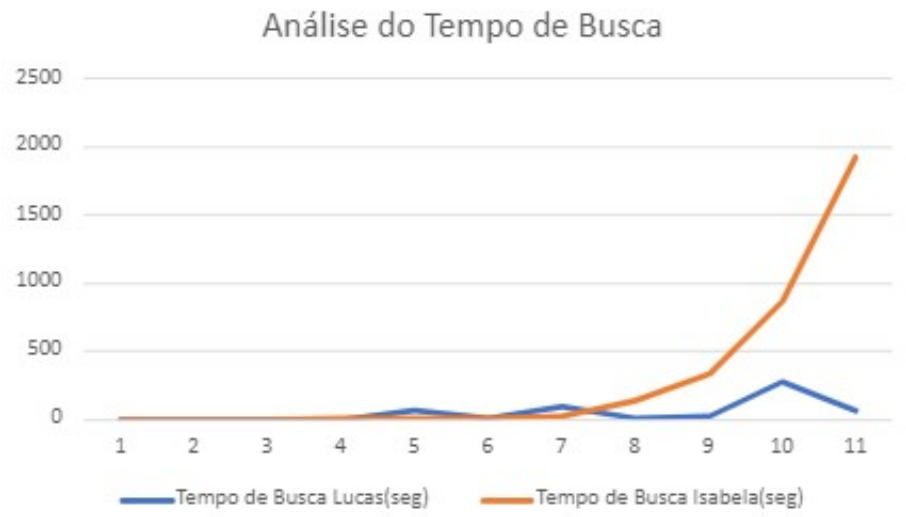

Figure 10. Comparação entre as duas máquinas - Busca

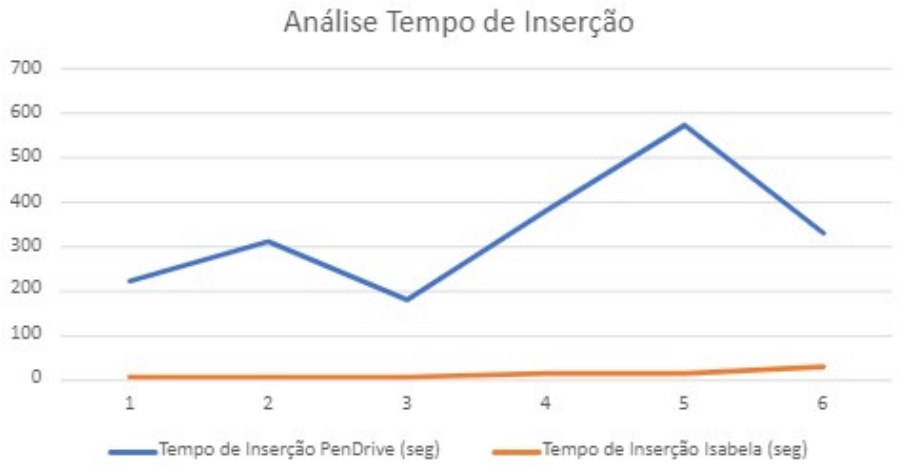

Figure 11. Análise pen-drive - Inserção

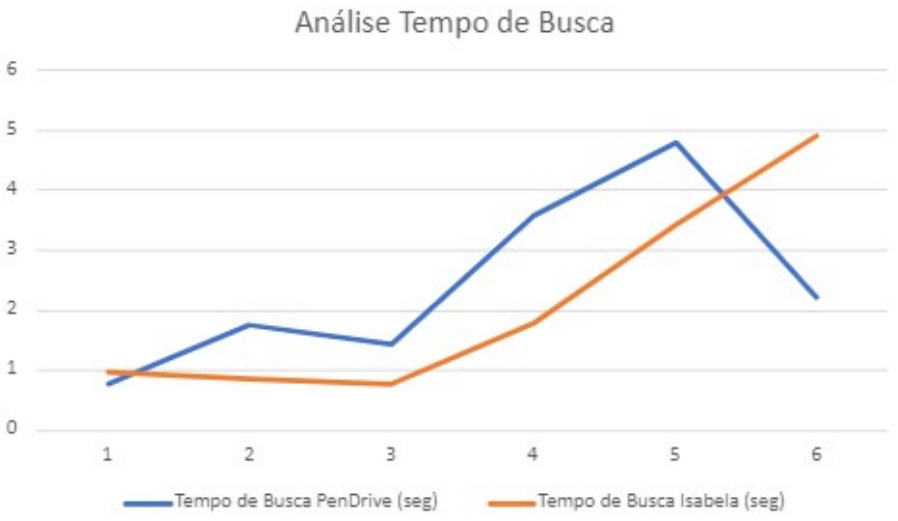

Figure 12. Análise pen-drive - Busca