MÉTODOS DE CONSULTA AOS DEMONSTRATIVOS FINANCEIROS DAS EMPRESAS LISTADAS NA BM&FBOVESPA

Rafael Costa Braga, Cesar Torres Fernandes
Fatec Zona Sul
rafaelcb21@gmail.com
cesartfernandes@gmail.com

RESUMO: As empresas de capital aberto devem apresentar seus demonstrativos financeiros ao mercado. Esses demonstrativos estão expostos no site da BM&FBovespa. Contudo, longos períodos anuais e trimestrais dos demonstrativos não podem ser comparados de forma automática, devido a forma em que estão publicados. Visando solucionar o assunto, são apresentadas duas soluções para buscar os demonstrativos de forma rápida a fim de compará-los. A primeira insere os demonstrativos num banco de dados utilizando a normatização, e a segunda insere-os em estruturas de dados hash, listas e matrizes multidimensionais, e todos são gravados em arquivos, e depois, é analisado o desempenho de ambas as soluções. Verificou-se que a segunda solução possui melhor tempo de execução, melhor consumo de memória secundaria, mas apresentou maior consumo de memória principal.

Palavras-chave: Estrutura de Dados **–** Banco de Dados – Bolsa de Valores – Demonstrativos Financeiros

ABSTRACT: The publicly traded companies must submit their financial statements to the market. These statements are set forth in the BM&FBovespa site. However, long annual and quarterly statements of the periods can not be compared automatically because of the way in which they are published. In order to solve the issue, are presented two solutions to get the statements quickly in order to compare them. The first inserts the statements in a database using normalization, and the second inserts them in hash data structures, lists and multidimensional arrays, and all are written to files, and then, we analyze the performance of both solutions. It was found that the second solution has better run-time, best use of secondary memory, but showed increased consumption of main memory.

Keywords: Structure Data - Database - Stock Market - Financial Statements

1. INTRODUÇÃO

Para uma empresa abrir seu capital no Brasil, elas devem se registrar na CVM (Comissão de Valores Mobiliários). No momento em que elas abrem o capital se tornam Sociedade Anônima (S/A.), e consequentemente, ficam sujeitas a inúmeras leis, decretos, medidas provisórias, resoluções, convênios, atos da CVM e outras regulamentações.

No artigo 176 da Lei n° 6.385, de 7 de Dezembro de 1976, diz que as Sociedades por Ações devem disponibilizar ao mercado suas demonstrações financeiras de forma clara e que exprimem a sua real situação econômico-financeiras. Por isso, as S/As enviam a

BM&FBovespa inúmeros demonstrativos em períodos trimestrais e anuais, nos quais, são todos públicos.

A BM&FBovespa disponibiliza os demonstrativos financeiros tanto via Web, quanto via software gratuito de própria autoria para acessar essas informações, contudo, só é possível comparar longos períodos de demonstrativos financeiros, de alguma empresa, de forma manual.

Visando solucionar esse problema, será proposto duas soluções A primeira refere-se a inserção dos demonstrativos financeiros num banco de dados, já a segunda solução a inserção será em estruturas de dados. Para que seja verificada a viabilidade de cada solução, ambas passaram por procedimentos que verifiquem a qualidade e o desempenho de cada solução.

Os capítulos dessa pesquisa são:

2. Fundamentação Teórica: esse capítulo expõe a classificação de algoritmo, para melhorar, e identificar a qualidade do mesmo, por meio da notação Grande-O. Também apresenta as principais estruturas de dados como vetor, listas, filas, pilhas, deques, árvores e hash, e o funcionamento de um banco de dados, por meio de seu SGDB (Sistema de Gerenciamento de Banco de Dados), juntamente com a técnica de Normatização para banco de dados.

3. Metodologia: nesse capítulo são descritos os procedimentos da pesquisa a serem realizados com o intuito de testar duas soluções possíveis para o problema relatado.

4. Desenvolvimento: nesse capítulo ocorre a execução dos procedimentos deflagrados no capítulo anterior, como a viabilização das soluções, a criação do banco de dados e da estrutura de dados, a inserção dos dados, e o desempenho da consulta em ambas as soluções.

5. Resultado: após o termino do desenvolvimento da pesquisa, os resultados obtidos são apresentados de forma sucinta, a respeito do consumo de memória principal, da secundária, do tempo de execução da consulta, e a notação Grande-O de cada solução.

6.Conclusão: nesse capítulo é exposto as vantagens e desvantagens de cada uma das soluções propostas, ao problema de poder comparar de forma rápida longos períodos de demonstrativos financeiros de empresas listadas na BM&FBovespa.

2. FUNDAMENTAÇÃO TEÓRICA

Para os dados serem utilizados eficientemente em um computador, eles precisam ser armazenados e organizados de forma eficaz, na Ciência da Computação, a área de Estrutura de Dados é responsável por esse estudo. Os algoritmos que irão percorrer essas estruturas de dados precisam ser avaliados e classificados pelo seu desempenho, e uma das técnicas utilizadas para a classificação é a notação Grande-O. Com a estrutura de dados e os algoritmos projetados, avaliados e classificados, o Sistema Gerenciador de Banco de Dados (SGDB) utiliza tais conceitos, técnicas e ferramentas para armazenar e recuperar informações de forma eficiente.

2.1. NOTAÇÃO GRANDE-O

Segundo KOFFMAN & WOLFGANG (2008) compreender como o tempo de execução, e os requisitos de memória de um algoritmo crescem em função do aumento do tamanho da entrada possibilita a comparação da performance dos algoritmos. O estudo dessa relação entre crescimento do tempo de execução e o tamanho da entrada de dados no limite (aumento indefinido, sem limitação) em um algoritmo se chama eficiência assintótica. Na matemática, analise assintótica é um método para descrever o comportamento de limites que no caso dos algoritmos, esta na relação entre tempo de execução com o tamanho da entrada ilimitada.

De acordo com CORMEM (2002), a notação Grande-O é uma notação matemática utilizada para analisar o comportamento assintótico de funções ou de algoritmos. Dessa forma, essa notação possibilita verificar o quanto um algoritmo é eficiente, de acordo com a relação entre numero de itens de entrada e a velocidade de execução do algoritmo.

LAFORE (2004) descreveu sobre a eficiência de alguns algoritmos em diferentes estruturas de dados, como vetor, lista, pilhas, filas, deques, tabela hash e outros. Os principais algoritmos para realizar operação em cada estrutura de dados são: inserção, exclusão e a busca de elementos.

No vetor, a inserção de elementos não depende da quantidade de itens em sua estrutura, o tempo de execução para inserir um elemento em um vetor será sempre igual, por isso, pode-se dizer que o tempo é constante, e a notação Grande-O é O(1). Isso ocorre, pois, um novo item em um vetor será sempre colocado na próxima posição disponível (LAFORE, 2004).

A busca linear, também conhecida por pesquisa linear, é um algoritmo que faz comparações com cada item de uma estrutura de dados, por isso, a sua eficiência depende da quantidade de itens na estrutura. Logo, pode-se dizer que o tempo de execução é proporcional ao numero de elementos, dessa forma, a notação Grande-O é O(n). (LAFORE, 2004)

A pesquisa binária é um algoritmo mais eficiente do que a pesquisa linear, sua notação Grande-O é O(log n). Trata-se de um algoritmo de busca em vetores que segue o paradigma de divisão e conquista. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor (LAFORE, 2004).

Taxa de Crescimento Comuns	
Grande-O	**Nome**
O(1)	Constante
O(log n)	Logarítmica
O(n)	Linear
O(n log n)	Log-Linear
$O(n^2)$	Quadrática
$O(n^3)$	Cúbica
$O(2^n)$	Exponencial
O(n!)	Fatorial

Tabela 1 – Taxa de Crescimento Comuns (KOFFMAN & WOLFGANG, 2008)

2.2. ESTRUTURA DE DADOS

Estrutura de dados é uma forma de armazenar e organizar dados, para que eles possam ser utilizados com eficiência. O catalogo telefônico é uma forma de dados estruturados, os assinantes e seus números estão ordenados alfabéticamentes, o que torna a busca rápida e simples. A escolha de uma determinada estrutura de dados pode levar um problema complicado a se tornar simples (VELOSO & SANTO & FURTATO & AZEREDO, 1984)

2.2.1. VETOR

Matriz é um conjunto retangular de números, símbolos ou expressões, organizados em linhas (linhas horizontais) e colunas (linhas verticais). Cada um dos itens de uma matriz é chamado de elemento. Para localizar determinado elemento, precisa-se fornecer dois índices: linha e coluna. Por essa razão são chamadas de matrizes bidimensionais (linha e coluna). Quando apenas um índice é suficiente, surge a matriz unidimensional, que costuma-se chamar vetor. (VELOSO & SANTO & FURTATO & AZEREDO, 1984)

De acordo com KOFFMAN & WOLFGANG (2008), vetor é uma estrutura de dados indexada, o que significa que os elementos deste podem ser acessados de forma arbitraria, bastando informar o índice do elemento, sendo possível também inserir ou remover um ou vários elementos de posições especificas.

Um vetor é alocado sequencialmente na memória principal do computador, ou seja, os elementos armazenados na memória possuem endereço contíguos e todos do mesmo tamanho, logo, todos os elementos devem ser do mesmo tipo. Por causa dessa representação física dos vetores, eles são criados com tamanho fixo na memória, sendo possível aumentar ou diminuir sua dimensão, contudo, o esforço computacional para alterar as dimensões de um vetor já criado são maiores, podendo até gerar paralisação do programa (KOFFMAN & WOLFGANG, 2008).

Segundo LAFORE (2004), a eficiência de um vetor na notação Grande-O é O(1) para inserção de elemento no vetor, O(N) para remoção de um elemento no vetor, O(1) para busca de elemento se o valor do índice for conhecido, e O(N) para pesquisa linear, pois não se conhece o valor do índice.

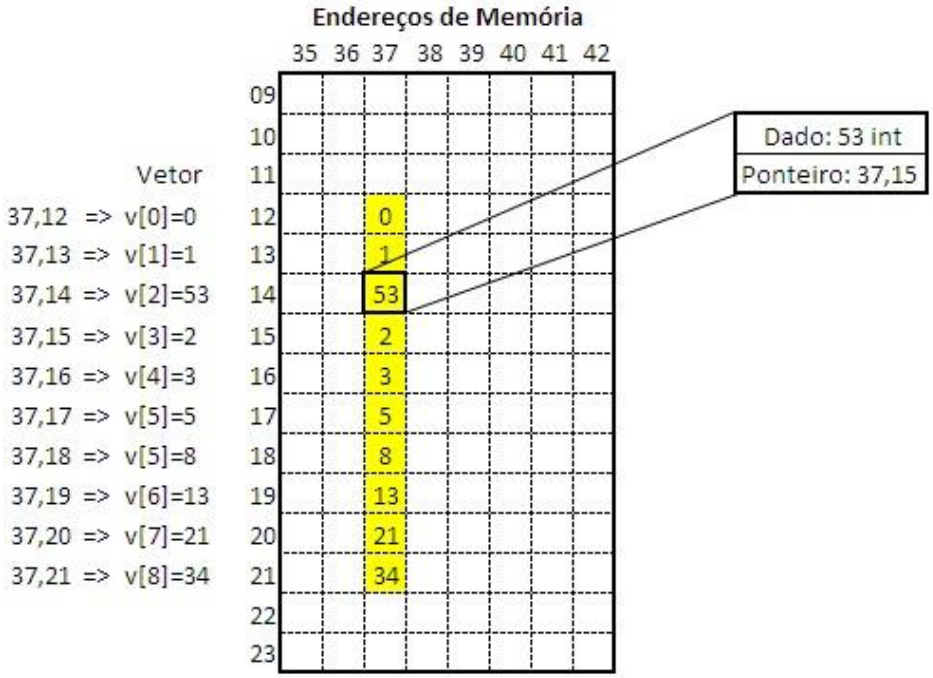

Figura 1 – Vetor e Endereçamento de Memória

2.2.2. LISTAS LINEARES ESTRUTURA

VELOSO & SANTO & FURTATO & AZEREDO (1984) descreveram que Lista Linear é uma estrutura que permite representar um conjunto de dados de forma a preservar a ordem linear entre eles. Uma lista linear é composta de nós que podem conter mais de um tipo dado primitivo (dados de um único tipo) ou composto (inteiros, floats, strings, e outros). Cada nó encontra-se num endereço de memória e aponta para o nó seguinte da Lista, dessa forma pode-se classificar uma lista como sendo dinâmica, ou seja, ela pode aumentar ou diminuir de tamanho com facilidade. Para encontrar um determinado nó numa lista, é necessário percorrê-la sequencialmente.

Segundo os mesmos autores, uma lista, por ter seus nós espalhados na memória, e cada nó apontando para o endereço de memória do nó seguinte e/ou antecessor, classifica-se a lista como sendo Encadeada. Quando o nó aponta somente ao nó seguinte, trata-se de uma Lista Encadeada Simples, mas se o nó aponta para o nó antecessor e o posterior, trata-se de uma Lista Duplamente Encadeada.

Pilhas, Filas e Deques são estruturas de dados bastante utilizadas e que podem ser implementadas em memória utilizando vetor ou listas, sendo esta ultima a mais utilizada, em

função de sua flexibilidade para aumentar ou diminuir a quantidade de elementos em sua estrutura de dados LAFORE (2004).

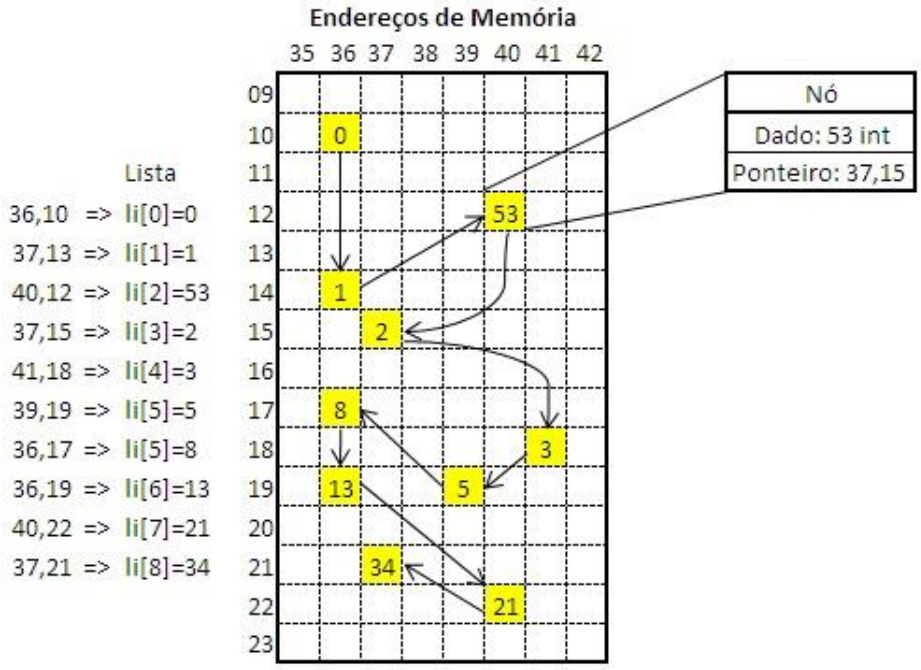

Figura 2 – Lista e Endereçamento de Memória

2.2.3. PILHAS

Segundo LAFORE (2004), Pilha é um tipo de estrutura de dados bastante utilizada na computação, a maioria dos microprocessadores usam uma arquitetura baseada em pilhas, e sua vida útil é tipicamente mais curta do que outras estruturas de dados. Elas são criadas e usadas para executarem certa tarefa durante a operação de um programa, e quando a tarefa for completada, elas são descartadas.

Ainda com o autor, uma pilha permite o acesso somente ao ultimo item inserido, ao remover esse item, pode-se acessar o item anterior ao ultimo inserido, e assim por diante, até chegar ao fim da pilha. Colocar um item de dado no topo da pilha é chamado de empilhar, e removê-lo do topo da pilha é chamado de desempilhar.

A pilha é dita ser um mecanismo de armazenamento LIFO (Last-In-First-Out : Último a entrar Primeiro ao Sair), pois, o ultimo item inserido é o primeiro a ser removido. A eficiência de uma pilha na inserção e remoção de dados é de ordem O(1) na notação Grande-

O, pois, não é necessário comparações e movimentos entre os elementos da pilha (LAFORE, 2004).

2.2.4. FILAS

A fila é uma estrutura de dados parecida com a Pilha, a única diferença é que seu mecanismo de armazenamento é o FIFO (Firdt-In-First-Out : Primeiro a Entrar Primeiro a Sair). A inserção de um novo dado na fila é inserida sempre no final desta, caracterizando assim, uma fila. Da mesma forma que as Pilhas, a eficiência da Fila na inserção e na remoção de seus elementos é de ordem O(1) (LAFORE, 2004).

2.2.5. DEQUES

De acordo com LAFORE (2004), um deque é uma fila com duas extremidades, é possível inserir e remover elementos em qualquer extremidade. Se a utilização do deque for somente para inserir e remover elementos da esquerda ele estará se comportando como uma pilha, mas se a utilização for inserir elementos na esquerda e removê-los na direita, ele estará se comportando como uma fila. Da mesma forma que as pilhas e filas, a eficiência do deque na inserção e na remoção de seus elementos é de tempo O(1). Contudo, embora a estrutura de um deque seja mais flexível do que uma pilha ou fila, ela é usada com menos frequência.

2.2.6. HASH

Segundo LAFORE (2004), Hash é uma estrutura de dados utilizada em tabela, na qual é composta por duas colunas, a primeira coluna são índices de um vetor, e a segunda coluna é o valor. Por se tratar de um vetor, a tabela é difícil de expandir depois de ter sido criada, contudo, a inserção e a busca são rápidas, estando na ordem O(1) na notação Grande-O.

O autor continua que a construção de uma tabela hash se dá por meio da função hash, que transforma uma chave-valor em um índice-valor. Após a geração dos índices, eles são inseridos numa tabela juntamente com o seu valor, e o conjunto de ambos, é chamado de tabela hash. Porém, existe tabela hash em que a chave não precisa ser transformada em um índice de um vetor, podendo assim, utilizar diretamente as chaves como índice, logo, não haverá necessidade de se utilizar a função hash.

De acordo com o mesmo autor, a função hash é um algoritmo utilizado para transformar um valor num índice, geralmente o índice criado é menor que o tamanho do valor. Com o índice criado representando o valor, para realizar uma busca, o valor desejado, será transformado num valor hash, é o algoritmo irá buscar o vetor cujo índice é igual ao valor hash.

Uma função hash eficiente gera um índice rapidamente, e é considerada perfeita se cada valor após ter sido transformado pela função hash for um índice único na tabela. Pois, existem funções hash que diferentes valores possuem o mesmo índice, gerando assim colisões entre os dados, ocorrendo perda de informação e de desempenho na velocidade de busca numa tabela hash. (LAFORE, 2004)

É possível utilizar tabela hash na memória secundaria, ou seja, no armazenamento externo, como Hard Disk e Memory Flash. O índice da tabela hash será o endereço de armazenamento do valor desejado, como é o caso de um arquivo. O índice será composto pelos números de blocos que se referem ao valor na memória secundaria. (CLAYBROOK, 1985)

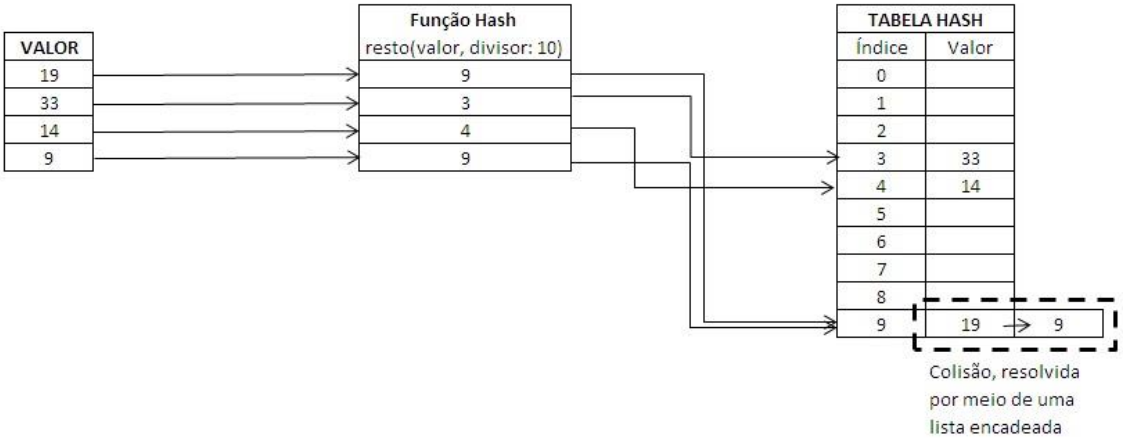

Figura 3 – Tabela Hash

2.2.7. ÁRVORE

As estruturas de dados vistas até o momento são lineares, ou seja, cada elemento possui apenas um predecessor ou sucessor. Para acessar todos os elementos em sequência é um processo da ordem de O(n). A estrutura de dados não-linear ou hierárquica se chama

Árvore. Em vez de ter apenas um sucessor, um elemento (nó) em uma árvore, pode ter vários sucessores, mas possui somente um predecessor. Na ciência da computação, o diagrama que representa uma árvore inicia-se da raiz da árvore, de cima para baixo (KOFFMAN & WOLFGANG, 2008).

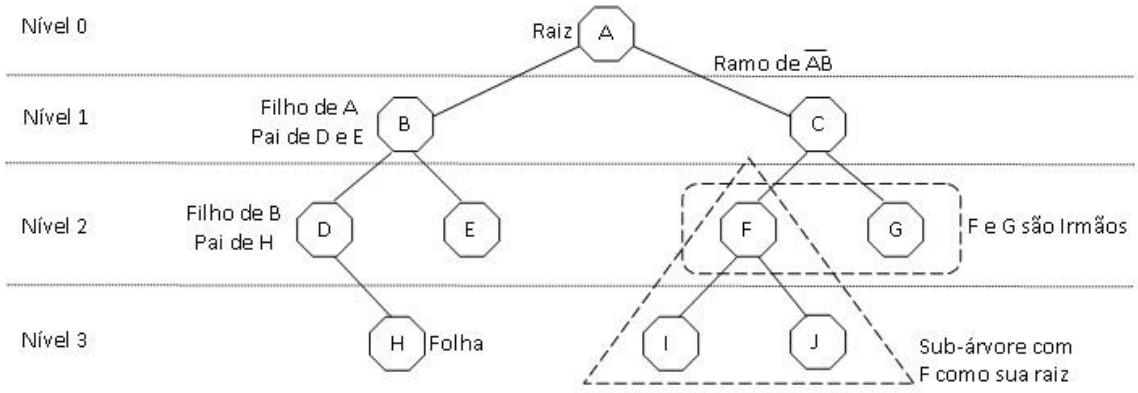

Figura 4 – Árvore Binária

Segundo KOFFMAN & WOLFGANG (2008) visto que árvores possuem estruturas hierárquicas, pode-se utilizá-las para representar informações que possuem organização hierárquica, como, hierárquica de classes, de diretórios de um disco e seus subdiretórios, uma árvore genealógica e outras.

O autor prossegue em o nó do topo de uma árvore é denominado Raiz, os vínculos de um nó com seus sucessores são denominados de Ramos. Com exceção da Raiz, cada nó possui somente um nó predecessor chamado de Pai, e os sucessores de um nó são chamado de Filho. Nós que possuem o mesmo pai são denominados de Irmãos. Nó que não possui filho é um nó Folha, conhecido também como Nó Externo, e nó não-Folha, ou seja, que possui filho é conhecido como Nó Interno.

Sub-árvore é quando denomina um determinado nó de uma árvore como sendo raiz, dessa forma, todos os seus descendentes são considerados filhos desse nó raiz, qualquer nó pode ser considerado como sendo raiz de uma sub-árvore. Nível, também chamado de profundidade de um determinado nó refere-se a quantas gerações o nó esta da raiz. Pode-se dizer que a raiz da árvore possui nível 0 (zero) e seus filhos terão nível 1 (hum), os filhos de seus filhos possuirão nível 2 (dois) e assim por diante. Com isso, pode-se encontrar a altura da árvore, que é a quantidade de níveis que possui. (KOFFMAN & WOLFGANG, 2008)

Árvores são estruturas de dados que admitem uma gama grande de algoritmos, como, de Busca, Inserção, Remoção, Mínimo, Máximo, Sucessor e Predecessor. A notação Grande-O irá depender do tipo de árvore, do tipo de algoritmo executado e da altura da arvore. (KOFFMAN & WOLFGANG, 2008)

2.2.7.1. ÁRVORE BINÁRIA

De acordo com KOFFMAN & WOLFGANG (2008) Árvore Binária é o tipo de árvore mais utilizado na computação. Cada nó dessa árvore possui de zero a dois filhos, denominando filho a esquerda e filho a direita. Em uma árvore todo nó contém dados, como por exemplo, um registro de funcionário, especificações de peças de carro e outros. Além dos dados, todos os nós, exceto as folhas, contêm referencias para outros nós. Logo, uma estrutura de arquivos não é uma arvore binária, pois, um diretório pode ter muitos filhos, e os subdiretórios não contêm dados, eles contêm apenas referências para outros subdiretórios ou arquivos.

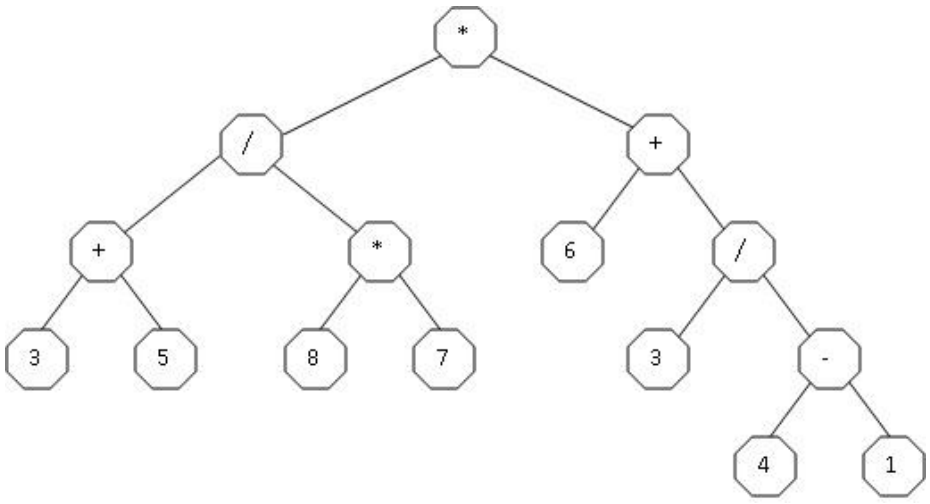

Figura 5 – Árvore Binária da expressão (3+5)/(8*7)*6+3/(4-1)

2.2.7.2. PERCURSOS EM ÁRVORE

KOFFMAN & WOLFGANG (2008) descreve sobre os 3 percursos em uma árvore. Percorrer uma árvore significa visitar cada nó em uma determinada ordem, com isso, é

possível identificá-los e relacioná-los. Há três maneiras de percorrer uma árvore, pré-ordem, ordem, e pós-ordem.

Percurso em pré-ordem, cada nó é registrado (visitado) a medida que se encontra o nó pela primeira vez. Ou seja, o nó é registrado antes de percorrer as sub-árvores. O percurso inicia-se sempre pela esquerda, e depois se direciona para as sub-árvores a direita.

Percurso em ordem, cada nó é registrado (visitado) a medida que se retorna do percurso de sua sub-árvore esquerda e depois direciona-se para a sub-árvore a direita.

Percurso em pós-ordem é quando o nó é registrado (visitado) somente depois de ter percorrido suas sub-árvores. Esse percurso numa expressão algébrica gera uma notação matemática chamada de pós-fixada, conhecida também por Notação Polonesa Reversa – NPR, que é utilizada nas calculadoras financeiras HP-12c.

- Percurso em pré-ordem da arvore da figura 2.0: */+35*87+6/3-41
- Percurso em ordem da arvore da figura 2.0: 3+5/8*7*6+34-1
- Percurso em pós-ordem da arvore da figura 2.0: 35+87*/6341-/+*

2.2.7.3. ÁRVORE DE BUSCA BINÁRIA

Árvore de Busca Binária é um tipo de árvore binária, contudo, os seus nós devem estar posicionados numa determinada ordem. Os nós das sub-árvores a esquerda devem ser menores que o valor da raiz, e os nós das sub-árvores a direita devem ser maiores que a raiz. Essa ordenação deve se repetir em cada nó raiz das sub-árvores (KOFFMAN & WOLFGANG, 2008).

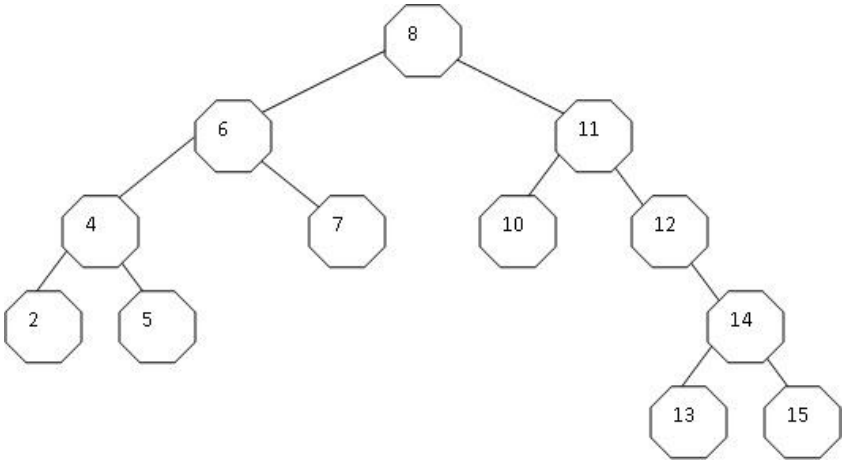

Figura 6 – Árvore de Busca Binária

2.2.7.4. LOCALIZAÇÃO DE UM NÓ EM UMA ÁRVORE DE BUSCA BINÁRIA

Segundo LAFORE (2004) para se encontrar um determinado nó em uma árvore de busca binária, o algoritmo de busca é simples, e seu desempenho na notação Grande O está na ordem de O(log n).

Como cada nó possui um valor, e seu filho a esquerda é menor que o pai, e o filho da direita é maior que o pai, bastaria comparar o valor desejado com o nó pai, se for menor percorre-se pela esquerda, se for maior percorre-se a direita, e assim sucessivamente em cada sub-árvore até encontrar ou não o valor desejado. (LAFORE, 2004)

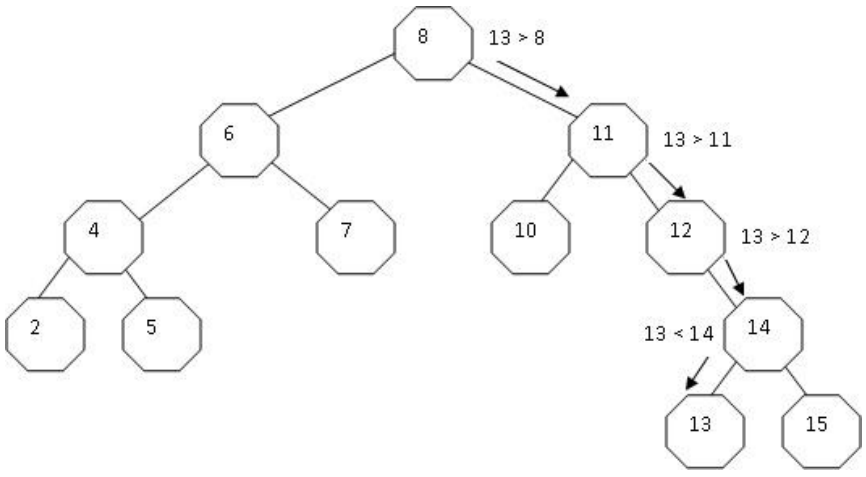

Figura 7 – Localização em Árvore de Busca Binária

2.2.7.5. INSERIR E REMOVER UM NÓ EM UMA ÁRVORE DE BUSCA BINÁRIA

De acordo com LAFORE (2004), para inserir um determinado nó em uma Árvore de Busca Binária, é necessário percorrer a árvore até o local em que deveria estar o nó, se o nó desejado não existir, o algoritmo deve inseri-lo. A eficiência da inserção esta na ordem O(logN).

Na remoção de um nó em uma Árvore de Busca Binária, o processo se inicia com a localização do nó desejado, contudo, no momento da remoção, deve-se levar em conta se o nó possui zero, um, ou dois filhos. Dependendo do cenário, o algoritmo deve funcionar de forma diferente. (LAFORE, 2004)

O autor continua na explicação de cada caso:

a) Caso 1: o nó a ser removido não tem filho.
    - Essa remoção é a mais simples, basta remover o nó e o nó pai não apontar mais para o nó removido
b) Caso 2: o nó a ser removido possui 1 filho.
    - Nessa remoção, substitui o nó removido pelo seu filho.
c) Caso 3: o nó a ser removido possui 2 filhos.
    - Para remover um nó que possui dois filhos, é necessário encontrar o nó sucessor do nó a ser removido, e inserir esse nó sucessor no local do nó que será removido. Para encontrar o nó sucessor, é necessário percorrer a sub-árvore, iniciando se do nó a ser removido, depois, caminha-se ao filho a direita desse nó, e depois percorre sempre os filhos à esquerda até encontrar um nó que não possui filho a esquerda, esse nó será o sucessor.

2.2.7.6. LOCALIZANDO VALORES MÁXIMOS E MÍNIMOS

Para se encontrar o valor mínimo de uma Arvore de Busca Binária, é necessário percorrer a árvore sempre em direção aos filhos da esquerda. O último filho a esquerda é o valor mínimo. Para encontrar o valor máximo é necessário percorrer a árvore sempre em direção aos filhos a direita. O último filho a direita é o valor máximo (LAFORE, 2004).

2.3. SISTEMA GERENCIADOR DE BANCO DE DADOS (SGDB).

Segundo SILBERSCHATZ & KORTH & SUDARSHAN (2006) o Sistema de Gerenciamento de Banco de Dados (SGDB) é uma coleção de dados inter-relacionados (banco de dados), e um conjunto de programas para trabalhar com os dados. O principal objetivo de um SGDB é fornecer uma maneira de armazenar e recuperar informações de um banco de dados, de forma conveniente e eficiente.

Os autores continuam, os SGDBs são projetado para gerenciar grandes blocos de informação. Esse gerenciamento consiste em definir estruturas de armazenamento, mecanismos para a manipulação da informação e garantir a segurança dessa informação armazenada. Para que o SGDB seja funcional, ele precisa recuperar os dados de maneira eficiente, e para isso, ele utiliza estruturas de dados complexas para representar os dados no banco de dados. Essa representação de como os dados são armazenados no banco de dados é descrita no primeiro nível de abstração, chamado de nível físico.

Um SGDB fornece uma única linguagem que irá definir a estrutura dos dados no banco de dados (DDL – Linguagem de Definição de Dados), e a forma de manipular os dados (DML – Linguagem de Manipulação de Dados). O DML permite a manipulação dos dados, como recuperar, inserir, excluir e atualizar informações armazenadas no banco de dados. Já o DDL é utilizado para implementar os esquemas de um banco de dados, como, por exemplo, as tabelas que serão criadas, os campos das tabelas, os campos responsáveis por gerar o relacionamento entre as tabelas, os índices das tabelas, em fim, o DDL implementa a estrutura de um banco de dados (SILBERSCHATZ & KORTH & SUDARSHAN, 2006).

O gerenciador de armazenamento do banco de dados traduz varias instruções DML em comandos de sistemas de arquivos de baixo nível, logo, ele é responsável por armazenar, recuperar e atualizar dados. Os dados de um banco de dados são armazenados em memória secundaria, pois, o custo do armazenamento é menor, e a expansão dessa memória além de ser mais fácil possui menor custo, do que armazenar toda a informação em memória primaria, que, embora seja mais rápida, não possui persistência, o espaço disponível é menor, e o custo da expansão e do armazenamento são maiores do que a memória secundaria (SILBERSCHATZ & KORTH & SUDARSHAN, 2006).

Segundo os mesmos, o gerenciador de armazenamento implementa vrias estruturas de dados como parte da implementação do sistema físico, como, os Índices, que podem fornecer

acesso rápido aos itens de dados, pois, os índices fornecem ponteiros (endereço de memória) para itens de dados que contem um valor especifico.

2.3.1. NORMATIZAÇÃO

Normalização é um conjunto de relações que permite armazenar informações sem redundância desnecessária, o que gera a recuperação de informação facilmente. A primeira forma normal (1FN) refere-se quando os atributos de uma entidade possuem somente um valor. A segunda forma normal (2FN) ocorre, somente se já existir a 1FN e se cada atributo não-chave for dependente da chave primaria por completo, e não dependente somente de parte da chave primaria (SILBERSCHATZ & KORTH & SUDARSHAN, 2006).

3. METODOLOGIA

De acordo com o Artigo 176 da Lei n° 6.385, de 7 de Dezembro de 1976, é dever das Sociedades por Ações ao fim de cada exercício social elaborar e disponibilizar ao mercado as demonstrações financeiras que exprimem com clareza a situação do patrimônio da companhia e as mutações ocorridas no exercício.

Os demonstrativos financeiros que deverão ser elaborados são:

1) Balanço Patrimonial (Ativo e Passivo)
2) Demonstração de Resultados (DRE)
3) Demonstração de Fluxo de Caixa (DFC)
4) Demonstração de Valor Adicionado (DVA)
5) Demonstração de Resultados Abrangentes (DRA)
6) Demonstração das Mutações do Patrimônio Liquido (DMPL)

Esses demonstrativos encontram-se disponíveis para download no site http://www.bmfbovespa.com.br/cias-listadas/empresas-listadas/BuscaEmpresaListada.aspx. Por esses demonstrativos estarem individualizados nesse site, por empresa e por ano, só é possível comparar longos períodos entre os demonstrativos financeiros, de uma empresa específica, de forma manual. Visando buscar soluções para essa limitação, abaixo se encontram os procedimentos para a realização da pesquisa:

1) Realizar o download dos demonstrativos financeiros.

2) Analisar e encontrar os atributos que tornem únicos cada demonstrativo financeiro.
3) Inserir os demonstrativos financeiros num banco de dados e em estruturas de dados, ambos de forma organizada.
4) Realizar consultas e verificar o desempenho da busca pelos demonstrativos financeiros no banco de dados e na estrutura de dados.
5) Apresentar os dados obtidos com os testes realizados.

4. DESENVOLVIMENTO

4.1. MATERIAL

Todos os experimentos foram realizados num computador pessoal que possui as seguintes características:

- Sistema Operacional: Windows 8 Enterprise
- Processador: Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz 2.99GHz
- Memória instalada (RAM): 4,00GB
- Tipo de Sistema: Sistema Operacional 64 bits, processador com base em x64
- Hard Drive (HD): 297GB
- Banco de Dados PostgreSQL 9.2
- Linguagem de programação Python 2.7.8

4.2. PRIMEIRO, SEGUNDO E TERCEIRO PROCEDIMENTOS

O primeiro procedimento é realizar o download dos demonstrativos financeiros, para isso foi construído um script em Python para realizar de forma automática o download. O período escolhido para a captura dos dados foi de 1997 à 2012 das 570 empresas listadas na bolsa de valores. Após essa captura obteve-se um universo de 317.457 demonstrativos financeiros, que possuíam juntos 16.590.074 de contas, pois, cada demonstrativo é composto por contas.

O segundo procedimento é analisar e encontrar os atributos que tornem únicos cada demonstrativo financeiro e consequentemente suas contas. Uma conta de um demonstrativo possui 11 atributos: ccvm, numero_da_conta, nome_da_conta, valor, aba_superior consolidado, origem, balanço, período, ano_documento, ano. Pode se observar que o conjunto

de 7 atributos é capaz de individualizar um demonstrativo financeiro como único. Os atributos são:

1) ccvm: código da empresa na CVM
2) consolidado: indica se o demonstrativo é o consolidado ou não.
3) origem: indica se o demonstrativo é anual (DFP) ou trimestral (ITR)
4) balanço: indica o tipo de demonstrativo que foi elaborado (Ativo, Passivo, DRE, DVA, DFC, DRA ou DMPL)
5) período: indica o período mensal ou anual em que o demonstrativo está se referindo (1-3, 1-6, 1-9, 1-12, 3-6, 6-9, 9-12)
6) ano_documento: indica o ano em que foi publicado o demonstrativo
7) ano: indica o ano em que o demonstrativo se refere

Com a individualização de cada conta no demonstrativo, é possível gerar uma chave para cada uma, dessa forma, todas as contas poderiam estar juntas, mas para encontrar um especifico demonstrativo, basta iniciar uma consulta utilizando como parâmetro a chave.

O terceiro procedimento que é inserir os demonstrativos financeiros num banco de dados e depois numa estruturas de dados para poder compará-los, se torna viável, por causa dessa chave, que foi construída como “string” com o seguinte layout:

ccvm	consolidado	origem	balanço	período	ano_documento	ano

Campo	**Formato**	**Observação**
ccvm	9(06)	Variável
consolidado	X(01)	Fixo
origem	X(03)	Fixo
balanco	9(01)	Fixo
período	X(04)	Variável
ano_documento	9(04)	Fixo
ano	9(04)	Fixo

Tabela 2 – Layout da chave que representa a consulta

Com base nessa chave, criou-se no banco BVMF no SGDB PostgreSQL 9.2, a tabela tb_consulta, que após a inserção dos dados ficou com 317.457 registros, no qual, cada registro representa uma chave. Após a criação dessa tabela, restou analisar os atributos: *numero_da_conta*, *nome_da_conta*, *valor*, *aba_superior*.

O atributo aba_superior é um tipo especial de nome_da_conta, que aparece somente no demonstrativo Mutações do Patrimônio Liquido, logo, pode ser unificado com o nome_da_conta, sobrando assim, 3 atributos: numero_da_conta, nome_da_conta, valor.

Para que as relações entre os atributos estejam de acordo com a primeira forma normal (1FN), numero_da_conta e nome_da_conta, por sererm multivalorados, necessitam possuir suas próprias tabelas. Após a criação e a inserção dos dados na tabela tb_nome_conta e tb_numero, elas ficaram respectivamente com 145.833 e 2.144 registros. E por fim, o atributo *valor* deverá se relacionar com todos os outros atributos, por isso, foi criado a tabela tb_conta que após a inserção dos dados resultou em 16.590.074 registros. Após a inserção dos dados o banco BVMF ficou com o tamanho em memória secundária de 1,419 GB.

Abaixo segue o diagrama Entidade-Relacionamento após o término do terceiro procedimento referente ao banco de dados:

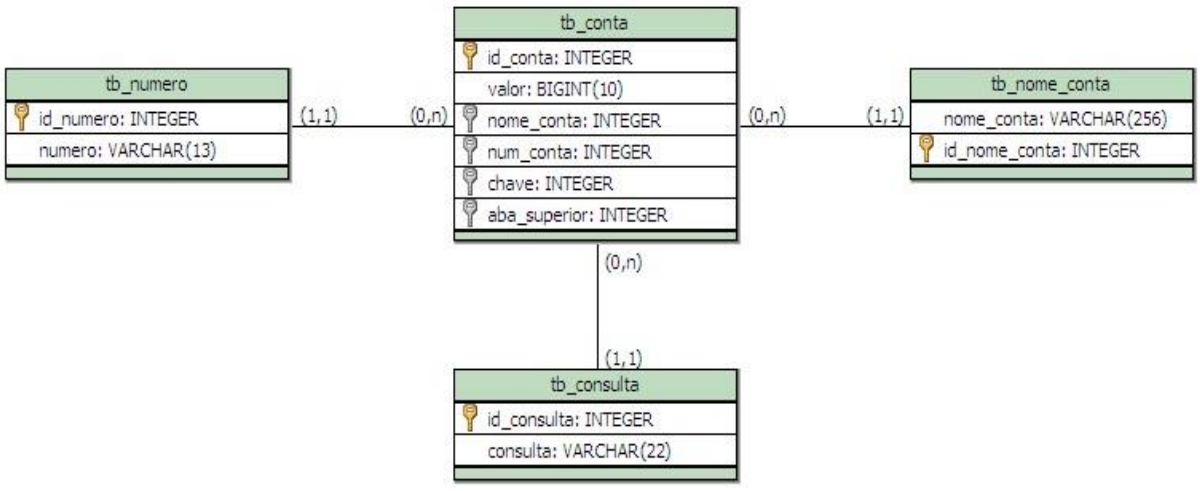

Figura 8 – Diagrama E-R do banco de dados BVMF

O terceiro procedimento também se refere em inserir os demonstrativos financeiros em estruturas de dados. Um demonstrativo financeiro é conjunto de contas, e uma conta é um conjunto de atributos, logo, a estrutura de dados é eficiente neste tipo de situação por se tratar de uma coleção de dados.

Foi capturado cada atributo de uma conta e inserido numa Lista: [num_conta, nome_conta, valor_conta, aba_superior].

Essa lista representa uma conta de uma consulta especifica, logo, reuniu-se todas as listas dessa consulta e inseriu-as numa outra lista, formando assim, uma matriz multidimensional. O nome da matriz é a consulta.

consulta = [[num_conta, nome_conta, valor_conta, aba_superior] ;
[num_conta, nome_conta, valor_conta, aba_superior] ;
[num_conta, nome_conta, valor_conta, aba_superior] ;
[num_conta, nome_conta, valor_conta, aba_superior]]

Para que essa matriz fosse mais fácil e rápido de manusear e menos custoso para memória principal e secundária, o nome da matriz e cada atributo são números que para serem interpretados é necessário outras estruturas de dados.

A primeira estrutura representa a relação entre a *consulta* e seu *numero*. Essa relação foi inserida numa tabela hash, na qual a *consulta* é o índice e o *numero* é o valor. Dessa forma, o sistema operacional, sabe a localização de cada índice na memória, não necessitando percorrer toda a tabela, logo, esta estrutura permite uma busca em ordem O(1) na notação Grande-O.

A segunda estrutura também é uma tabela hash, no qual o índice é o atributo *num_conta* da matriz, e o valor é o numero da conta no demonstrativo financeiro.

A terceira estrutura também é uma tabela hash, no qual o índice é o atributo *nome_conta* e a *aba_superior* da matriz, e o valor é o nome da conta no demonstrativo financeiro.

As três tabelas hash foram inseridas em arquivos para ficarem salvas permanentemente na memória secundaria e para que não houvesse a necessidade de estar sempre recriando-as.

A tabela hash { *consulta* key: *numero* value} foi salva no arquivo *chave_0.pck*, a segunda tabela hash { *num_conta* key: *numero_demonstrativo* value} foi salva no arquivo *chave_3.pck*, e a terceira tabela hash { *num_conta* key: *nome_demonstrativo* value} foi salva no arquivo *chave_4.pck*.

Em memória secundaria, o arquivo *chave_0.pck* ocupa 12,8 MB e possui 317.457 índices, o arquivo *chave_3.pck* ocupa 61 KB e possui 2.144 índices, e o arquivo *chave_4.pck* ocupa 9,3 MB e possui 145.833 índices. Percebe-se que a quantidade de índices é o mesmo da

quantidade de registros nas tabelas tb_consulta, tb_numero e tb_nome_conta do banco de dados BVMF.

A matriz multidimensional representa somente uma consulta do universo de 317.457. Cada matriz foi salva num arquivo individual, gerando ao final desse processo, 317.457 arquivos, no qual, o nome do arquivo é o numero da consulta. O espaço ocupado por todas os arquivos matrizes é de 527 MB.

Para a geração de todos os arquivos foi utilizado a linguagem de programação Python versão 2.7.8. Essa linguagem possui uma biblioteca chamada pickle, que permite empacotar estruturas de dados e salva-las diretamente em arquivos, dessa forma, basta carregar o arquivo por meio dessa biblioteca e utilizar a estrutura de dados.

Após a geração de 317.460 arquivos, todos foram compactados no formato ZIP, por meio da biblioteca *zipfile* da linguagem Python, gerando ao final do processo um arquivo *consultas.zip* com 317.460 arquivos, ocupando um espaço em memória secundária de 187,4MB.

4.3. QUARTO PROCEDIMENTO

4.3.1. BANCO DE DADOS

Com o banco de dados e as tabelas criadas, e seus registros inseridos, iniciou a verificação do desempenho da consulta. A consulta utilizada possui os seguintes valores aos atributos: ano = 2010, ccvm = 9512, ano_documento = 2011, consolidado = 't', periodo = '1-12', origem = 'DFP', balanco = '3'. O que gerou uma consulta com a seguinte chave '9512tDFP31-1220112010'.

O resultado dessa consulta foi de aproximadamente 10s para conclui-lá. O algoritmo que o banco de dados utilizou para realizar a busca é de ordem O(n) na notação Grande-O, pois, esse algoritmo percorre todos os 16 milhões de registros para retornar as contas que estão de acordo com a condição (chave) da consulta.

4.3.2. ESTRUTURA DE DADOS

Após a construção do arquivo *consultas.zip* iniciou-se a verificação do desempenho da busca nessa estrutura de dados. Para isso, foi criado um algoritmo escrito na linguagem Python, utilizando como condição de busca a chave '9512tDFP31-1220112010'.

O resultado dessa busca retornou o demonstrativo financeiro em 0,6s. Para realizar essa consulta, o tempo para abrir o arquivo *consultas.zip* foi de 4,36s, o tempo para carregar

todos os arquivos de *consultas.zip* foi de 4,09s totalizando um tempo de 8,45s para iniciar a busca. Esse tempo é despendido somente uma vez, não havendo necessidade de percorrê-lo novamente.

5. RESULTADO

A primeira solução foi utilizar as melhores práticas para um banco de dados, como as normalizações, e os relacionamentos entre as tabelas. Essas implementações possibilitaram a utilização de 1% da memória principal, 1,419 GB da memória secundária, e um tempo de execução da consulta em aproximadamente 10s.

A segunda solução utiliza as estrutura de dados Tabela Hash e Matrizes Multidimensionais em arquivos salvos na memória secundária. Esta solução permitiu a utilização de 5% da memória principal, 187,4 MB da memória secundária, e um tempo de execução de 0,6s.

Abaixo segue uma tabela resumindo o desempenho das soluções propostas:

	Memória Secundária utilizada (HD)	Tempo de Execução da Busca	Memória Primaria utilizada (RAM)	Notação Grande-O
Banco de Dados BVMF	1,419 GB	9849 ms	1%	O(n)
Arquivo *consulta.zip*	187,4 MB	620 ms	5%	O(1)

Tabela 3 – Comparação de desempenho entre as soluções propostas

6. CONCLUSÃO

Com a viabilização da inserção das contas dos demonstrativos financeiros num banco de dados e numa estrutura de dados, se tornou possível acessar de forma rápida longos períodos de demonstrativos financeiros de determinada empresa, podendo assim, compará-los de forma automática.

As duas soluções propostas possuem vantagens e desvantagens. A primeira solução, que é a utilização de um banco de dados, possui as vantagens da segurança, confiabilidade e facilidade de inserção de dados e de consultas que um SGDB proporciona. A desvantagem para essa solução é a utilização considerada de memória secundária.

Já na segunda solução, a vantagem dessa solução esta no tempo de resposta da consulta, que é dez vezes mais rápida que a primeira solução, e do espaço necessário para armazenamento da informação em memória secundária, que é 87% menos do que a primeira solução. A desvantagem da segunda solução é que ela requerer um domínio mais avançado de estrutura de dados e de relacionamento de informação, para que ocorra com sucesso inserções e consultas de dados, e necessita de uma quantidade maior de memória principal para funcionar, visto que essa memória possui um preço monetário maior para o computador.

7. REFERÊNCIA

BRASIL. Lei n. 6.385, de 7 de dezembro de 1976. Dispõe sobre o mercado de valores mobiliários e cria a Comissão de Valores Mobiliários.

CLAYBROOK, B. G. Técnicas de Gerenciamento de Arquivos. 1 ed. Rio de Janeiro: Campus, 248p, 1985.

CORMEM, T. H. Algoritmos: teoria e pratica. 2. ed. Rio de Janeiro: Elsevier, 916 p, 2002.

CUNHA, I. L. L. Estrutura de dados Mate Face e aplicações em geração e movimento de malhas. Tese de Mestrado. Universidade de São Paulo. São Carlos. Instituto de Ciências Matemáticas e de Computação. 112 p, 2009.

KOFFMAN, E. B.; WOLFGANG, P. A. T. Objetos, Abstração, Estruturas de Dados e Projeto Usando C++. Rio de Janeiro: LTC, 689 p, 2008.

LAFORE, R. Estrutura de Dados & Algoritmos em Java. 2.ed. Rio de Janeiro: Editora Ciência Moderna Ltda, 702 p, 2004.

SILBERSCHATZ, A; KORTH, H. F.; SUDARSHAN, S. Sistema de Banco de Dados. 5 ed. Rio de Janeiro: Elsevier, 781 p, 2006.

SOUZA, C. A. C. Implementação de uma estrutura de dados para visualização científica. Tese de Mestrado. Universidade de São Paulo. São Carlos. Instituto de Ciências Matemáticas e de Computação. 88 p, 2003.

VELOSO, P. A. S.; SANTOS, C. S.; FURTATO, A. L.; AZEREDO, P. A. Estrutura de Dados. 2.ed. Rio de Janeiro: Campus, 228 p, 1984.

WAZLAWICK, R.S. Metodologia de Pesquisa para Ciência da Computação. Rio de Janeiro: Editora Elsevier, 159 p, 2009.