**Escola de Engenharia**

Mestrado integrado em Engenharia Informática

Laboratórios de Informática III

**Relatório**

**Sistema de gestão de vendas – SGV**

**Projeto C**

Grupo 22

Beatriz Rocha a89571

Hugo Moreira a43148

Nuno Rodrigues a90439

**Índice**

[**Introdução** 2](#_Toc38130010)

[**Módulos e estruturas de dados** 3](#_Toc38130011)

[**Arquitetura Final** 5](#_Toc38130012)

[**Complexidade das estruturas e otimizações** 6](#_Toc38130013)

[**Resultado dos testes** 9](#_Toc38130014)

[**Observações** 10](#_Toc38130015)

[**Conclusão** 11](#_Toc38130016)

# **Introdução**

O projeto SGV é uma aplicação de software em grande escala, ou seja, é uma aplicação que trabalha com grandes volumes de dados e com uma elevada complexidade algorítmica e estrutural.

Este foi criado com o intuito de disponibilizar uma aplicação capaz de tratar e juntar dados específicos, de uma distribuidora com filiais, requeridos pelo utilizador.

A aplicação deverá contemplar a leitura e armazenamento de dados relativos aos clientes da instituição assim como dos seus próprios produtos e registos de vendas.

Durante a realização deste projeto deparámo-nos com vários desafios, tais como, encontrar a forma mais eficiente em termos de execução e espaço de memória necessário, para a sua leitura, para armazenar os dados e questionar a própria informação. Assim como a designação dos diferentes módulos necessários para a realização do projecto, isto para desenvolver a modularidade e o encapsulamento.

# **Módulos e estruturas de dados**

Relativamente aos módulos e estruturas de dados as decisões que tomamos foram relativas não só em termos do tempo de execução no carregamento dos dados, como na alocação/espaço em memória, e o próprio tempo de execução das respostas pretendidas.

Identificamos a informação dividida em três distintas entidades, referentes aos dados processados, que possuem a sua própria informação. Sendo estes:

* Produtos
* Clientes
* Vendas

Relativamente aos produtos a estrutura de dados utilizada foi uma Hashtable, definida em “produtos.h”, contendo esta em cada posição a primeira letra de cada código de produto e um apontador para uma AVL, definido em “produtosaux.h”, com os códigos ordenados de todos os produtos começados por essa mesma letra. Contendo ainda a informação do total de produtos existentes começados por essa mesma letra. Tendo ainda adicionalmente cada nodo a informação se esse produto foi comprado e as filiais em que esse produto foi comprado. Este armazenamento é efetuado após a leitura e validação do código de um produto.

Decidimos utilizar a estrutura Hashtable devido ao seu rápido tempo de acesso e inserção. Optamos por uma AVL por ser a estrutura mais adequada para armazenar a informação de forma ordenada, apesar de não ser tão eficaz em tempo de inserção e acesso como a Hashtable. Mas tendo em atenção a necessidade de que várias das respostas serem dadas ordenadamente e por uma melhor eficiência de alocação de memória.

Para os clientes não se verificou essa mesma necessidade uma vez que o código referente a um cliente apenas é constituído por apenas uma letra. E como já referido para os produtos, após a validação das informações correspondentes aos clientes, cria-se uma AVL (vazia), definido em “clientes.h”, onde são guardados os dados validos dos mesmos, esta AVL tem como elementos o código do cliente, a informação se este efetuou alguma compra e as filiais onde efetuou compras.

As vendas estão guardadas em um array de filiais[36], sendo que estão armazenados os dados dos 12 meses para cada filial e um de faturação[12], estando estes definidos em “vendas.h” que corresponde aos 12 meses do ano. Estes têm uma AVL ordenada por código produto, que por sua vez aponta para uma AVL ordenada por código cliente onde guarda a informação da venda de cada cliente para esse produto nesse mês. Tendo também a AVL dos produtos uma estrutura com os totais das vendas de todos os clientes para que este aponta/referência.

De forma a ser possível apresentar os dados às respostas pretendidas houve a necessidade de criar uma estrutura auxiliar para armazenar a informação de algumas respostas pretendidas pelo sistema, assim como a estrutura da própria apresentação da informação.

* View
* Estrutura auxiliar

O view, definido em “view.h” é o responsável por criar listas ligadas ou um array com os resultados das pesquisas, que irão ser utilizadas nas estruturas dos Produtos, Clientes, Vendas e pela estrutura auxiliar, definido em ”structaux.h”.

Havendo a necessidade do utilizador navegar sobre a aplicação e escolher o carregamento e as respostas/informações pretendidas foi por último definido:

* Navegador

Este módulo encontra-se definido em “menu.h” com os seus métodos definidos num módulo auxiliar, definido em “funcoes\_menu.h”.

De maneira a obtermos uma maior eficiência e segurança de dados no projeto, implementamos módulos de dados. Isto permite o encapsulamento dos dados para os seus respetivos módulos e de uma forma mais eficiente, caso encontrada a necessidade de efetuar alguma alteração ou mesmo correção, identificar o módulo em questão.

# **Arquitetura Final**

Estruturas de dados:

Clientes:

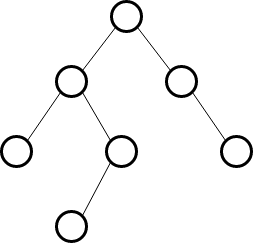


Figura: 1 AVL ordenada por código de cliente

Produtos:

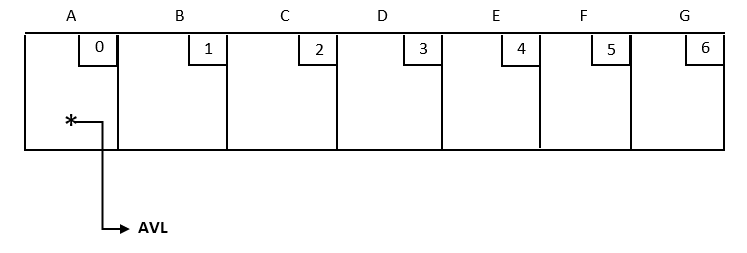


Figura: 2 HashTable elementos uma letra e um apontador para AVL ordenada por código de produtos

Vendas:

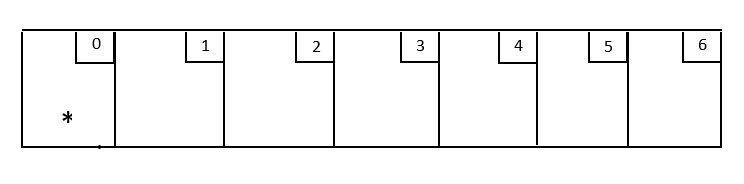


Figura: 3 Hashtable com as posições de cada mês ou dos mese vezes as filiais

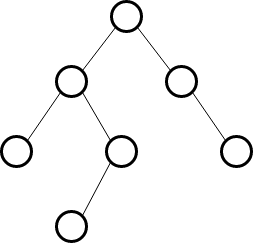
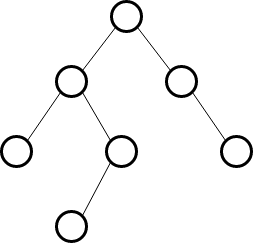
 

Figura: 4 AVL ordenada por código de produtos e com os totais da AVL que referência

Figura: 5 AVL ordenada por clientes que compraram o produto

# **Complexidade das estruturas e otimizações**

Após o levantamento de todos os requisitos necessários pretendidos pelo sistema tomamos as decisões da definição das estruturas de dados, estas serão as que mais impacto irão ter relativamente ao desempenho de todas as funcionalidades pretendidas. De referir que tomamos a decisão da ocupação do menor espaço de memória possível e tendo mais em consideração as respostas do sistema após a leitura, validação e carregamento da informação.

Clientes:

Como mencionado no ponto anterior a estrutura de dados escolhida foi uma AVL. Esta em termos de execução para pesquisas, ou inserção tem como pior caso um tempo de execução O(logn). E para apresentação dos seus resultados de forma ordenado um tempo de execução de O(n).

Sendo que o espaço alocado será determinado não só pelo número de elementos, mas também no pior caso por mais dois últimos elementos vezes a altura da árvore.

Produtos:

Os produtos, como mencionado anteriormente, foram definidos por uma hashtable que contém a primeira letra do código do produto, os número total dos produtos começados pela respetiva letra e um apontador para todos os produtos começados por essa letra.

Esta decisão foi tomada para uma optimização a uma das respostas requeridas pelo sistema. E pelo fato de o alfabeto não ser alterado.

Sendo que a hashtable tem como tempo de execução na inserção e pesquisa para o melhor O(1), e como referido não haverá a necessidade de haver realocação de memória.

Relativamente à AVL que é referenciada por cada posição da hashtable já foi referido acima o seu tempo de execução e espaçamento em memória.

Vendas:

A faturação global e as filiais estão definidas pelas estruturas de dados Vendas. Em que para aceder a um mês em específico terá o tempo de execução de O(1) e para cada uma das suas sub-árvores terá a complexidade mencionada nas AVLs.

Iremos analisar a complexidade de cada uma das funcionalidades:

Carregamento dos dados:

Relativamente ao carregamento de dados dos clientes o nosso programa tem como tempo de execução no pior caso O(logn), assim como para os produtos. No caso das vendas uma vez que existe a necessidade da validação da existência do produto e cliente que pertence ao registo de venda. Esta terá no pior caso o tempo de execução de uma pesquisa nos clientes e uma nos produtos tendo cada uma destas, tempo de execução no pior caso O(logn) mais a inserção da venda que tem também no pior caso O(1) + O(logn) + O(logn), sendo que em termos assimtópticos terá tempo O(logn), apesar de nos registos de vendas estarem partidos pelo número de meses ou filiais, e mesmo pelo número de produtos comprados nesse mês e pelo número de clientes que o compraram, este não terá um grande impacto na performance.

Isto apenas para referir, que teria bastante melhor performance e que iremos abordar em detalhe no ponto das otimizações, de que seria mais eficiente em termos de tempo de execução uma hashtable que referi-se uma hashtable com os produtos e que esta por sua vez para uma hashtable com os clientes.

Relativamente às restantes funcionalidades do programa iremos efetuar uma síntese do seu tempo de execução.

Existe a necessidade de quardar a informação dos clientes e produtos válidos esta terá como tempo de execução O(n).

2.

Os produtos começados por uma letra e apresentar os seus resultados ordenadamente. Essa operação terá como tempo de execução O(1) + O(n) sendo n o número de produtos começados por essa mesma letra.

3.

O tempo de execução para resposta a esta questão será de O(1) + O(logn) em que n é o número de produtos existentes nesse mês.

Sendo que para cada filial será o mesmo tempo de execução apenas o n será o número de produtos comprados nessa filial nesse mês.

4.

O(n) sendo n o número de produtos.

5.

O(n) sendo o n o número de clientes.

6.

O(n) + O(n) sendo o n o número de clientes ou produtos respetivamente.

7.

O(n)+O(logn) sendo o n o número de produtos e de que O(logn) será o número de clientes que compraram esse mesmo produto e terá esse tempo de execução multiplicado pelo número de produtos.

8.

O(n) sendo n o número de produtos comprados nesse intervalo.

9.

O(logn) + O(n) sendo n o número de produtos comprados nesse mês e o n o número de clientes que o compraram nesse mês.

10.

O(n) + O(logn) semelhante à anterior apenas com os “n” trocados.

11.

O(n)

12.

O(n)

Relativamente a otimizações como mencionado anteriormente e de que iríamos abordar neste ponto. Para um carregamento de dados mais eficiente poderíamos ter decidido por substituir todas as AVLs por Hashtables, seria muito mais eficiente uma vez que a inserção e procura tem tempo constante. Mas a informação não estaria ordenada e haveria a necessidade de em múltiplas questões feitas ao sistema de ordenar essa informação que teria como tempo de execução no pior caso O(nlogn). Além de que iríamos ter também num pior caso de dezenas de milhares de posições das hashtables vazias com memória alocado sem ser utilizada. De Realçar que as decisões foram tomados tendo em conta a performance de espaçamento em memória e de eficiência após o carregamento da informação.

De referir que uma vez que a estrutura da AVL dos produtos e dos clientes, existentes na faturação e nas filiais, isto é no “vendas.h”, poderiam ter sido definidas como uma Hashtable uma vez que não há a necessidade de obter os resultados de forma ordenada. Teríamos assim uma performance melhorada na pesquisa e inserção dos elementos.

# **Resultado dos testes**

# **Observações**

Ao longo do projeto reparamos que em certas partes ficaria mais eficiente, tanto na inserção como na procura, termos utilizado Hashtable em vez de AVL, visto que o tempo de execução seria 1 em vez de log(n). Contudo, na parte da impressão o tempo de execução iria piorar, nlogn em vez de n.

Embora a mudança no tempo de execução da impressão não apresente uma diferença muito significativa, acabamos por decidir não alterar a estrutura de dados, pois o tempo de execução já é aceitável e, em contrapartida, iríamos perder na performance de memória visto que íamos ter muitas posições vazias na Hashtable.

E como referido co capítulo das otimizações colocar a estrutura dos produtos nas vendas em forma de hashtable em vez de AVL.

# **Conclusão**

Em suma, conseguimos desenvolver o SGV pretendido usando conhecimentos adquiridos em diversas áreas curriculares anteriores, tais como Algoritmos e Complexidade. Estamos contentes com o resultado do projeto e acreditamos que todos os objetivos foram alcançados.

Contudo, temos noção que alguns aspetos podem ser melhorados como o uso de Hastables em vez de AVL, como já mencionado nas observações, e a performance da memória que diminuiria com a alteração.