Laboratórios de Informática III

MIEI - 2º ano - 2º semestre
Universidade do Minho

SGV

Grupo 61

Ana César
A86038

Angélica Cunha
A84398

Tiago Gomes
A69853

Braga, 12 de Abril de 2020

Índice

1. Introdução

O presente trabalho tem como propósito o desenvolvimento de um sistema de gestão de vendas, com capacidade para suportar grandes quantidades de dados. Para tal, foi desenvolvida uma aplicação modular e facilmente adaptável a futuras necessidades de escalabilidade. Partindo dos princípios de encapsulamento e criação de código reutilizável, apresentamos de seguida a estrutura da mesma, bem como as decisões tomadas a nível de estruturas de dados e algoritmos, para responder às necessidades propostas.

2. Interpretação do enunciado

Numa fase inicial do projeto, e por boas práticas de programação, começamos por aferir atentamente toda a informação presente no enunciado do projeto, tentado chegar a uma estrutura compacta, tanto na arquitetura do projeto, como nos vários módulos que tivemos que desenvolver.

Começamos por tratar dos módulos que serviam de base ao nosso trabalho: Clientes e Produtos. Estes dois módulos foram os que encontramos menos problemas relativamente ao "desenho"das estruturas e a implementação do módulo em si. É de realçar ainda que conseguimos chegar à solução mais eficiente relativamente aos restantes módulos: faturação e filial tirando partido da implementação base dos módulos anteriormente referidos.
Ainda assim, antes de chegar, ao que achamos ser a mais correta implementação destes dois últimos módulos, fomos deparados com alguns problemas até mesmo em relação à interpretação do enunciado do trabalho prático. Um dos maiores desafios foi mesmo perceber, qual era a intenção de cada um dos módulos, até porque, de certa forma, podíamos ter a informação destes dois módulos conjugadas num único módulo, mas posteriormente percebemos o porquê da necessidade de fazermos esta distinção importante. A faturacao, que referencia exclusivamente o sistema de vendas de uma forma interna, isto é, ao nível dos produtos e valores a estes associados. As filiais, que têm o propósito de relacionar o próprio sistema com o fator externo, os seus clientes.

A nível da estruturação do projeto, o nosso maior entrave foi encontrar uma solução que fosse facilmente modificável e escalável a grandes volumes de dados. Contudo, após várias tentativas e diferentes implementações, chegamos a uma solução, cuja complexidade se adapta relativamente bem ao escalonamento dos dados.

3. Modularidade e Conceção

Inicialmente na concepção da estrutura da aplicação, estávamos a ponderar implementar uma *HashTable* para gerir Clientes, em que tínhamos como *chave* a letra correspondente ao primeiro dígito do código, e se o código fosse referente a um produto, teríamos também associada a cada posição outra HashTable correspondente à segunda letra do código. No entanto, dado à redundância de código, optámos por criar um módulo genérico catálogo. Após uma melhor análise das necessidades impostas pelas *queries*, utilizamos a facturação para relacionar os produtos com o número de registos de vendas e os totais facturados, enquanto que as filiais fazem a relação fundamental dos clientes com os produtos, e ainda com as questões quantitativas das vendas (tanto a nível das quantidades vendidas como a nível dos valores lucrativos das vendas).

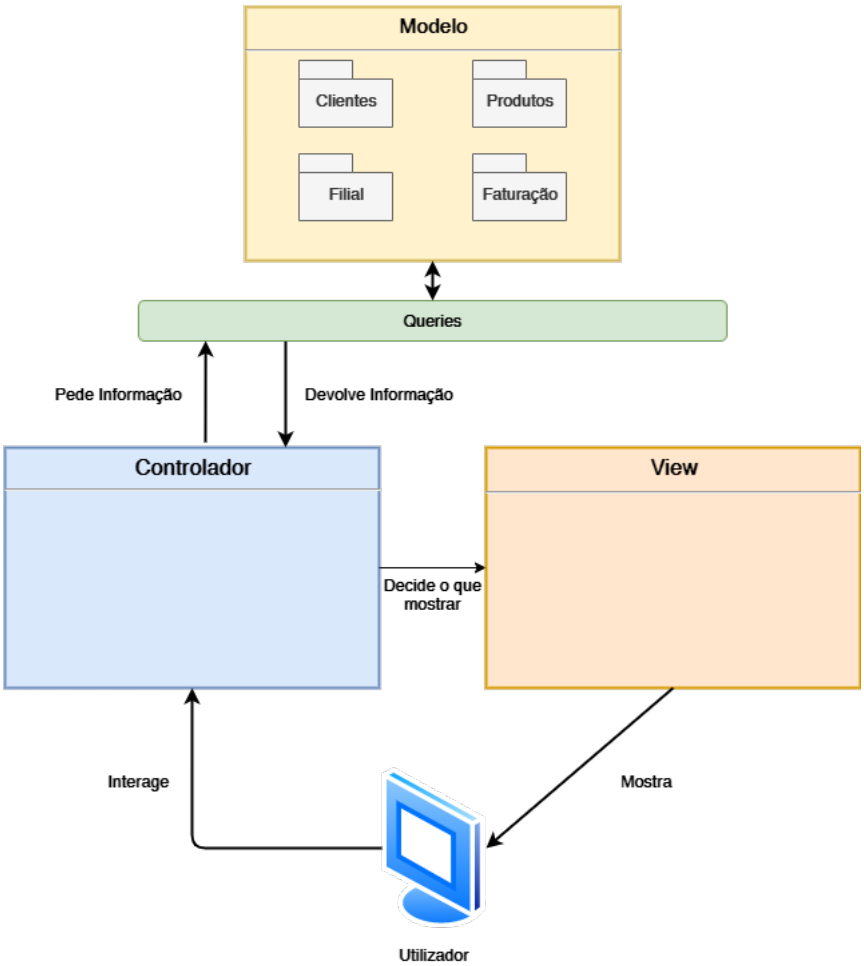

Figura 3.1: Arquitetura do projeto

Como podemos aferir, a arquitetura do nosso programa tem por base o padrão de desenvolvimento de software MVC (Modell View Controller), onde a apresentação dos dados e a interação dos utilizadores são separados dos métodos que interagem com a camada lógica.

3.1 Modelos

3.1.1 Catalogo

O módulo catalogo consiste numa estrutura de dados auxiliar que constitui a base dos principais módulos do modelo, de forma a optimizar a reutilização de código. Para tal, optamos por implementar uma *struct* onde está presente um HashTable estático, assim como um campo inteiro profundidade. Este campo profundidade define o tamanho do nosso HashTable ($BASE^{profundidade}$), que pode tomar os valores 1 e 2. No nosso caso, a BASE foi definida como 26 porque devido ao problema implementado, usamos o número de letras do alfabeto português. A cada posição deste HashTable corresponde um apontador para uma árvore balanceada da biblioteca Glib, que contêm os códigos (produto ou cliente), ordenados alfabeticamente. É ainda de realçar, que o índice associado a um código é devidamente calculado através de uma função *hashkey*, consoante a profundidade do catálogo em causa. Associadas a esta estrutura, foram implementadas várias funções de manutenção da mesma tanto a nível de recursos, como a nível de funcionalidade. Sendo esta estrutura de dados a base dos restantes módulos, que definem o modelo de dados, foram implementadas funções que garantem, não só a correta alocação e libertação de memória da estrutura, como funções que nos permitem gerir os restantes módulos baseados nesta estrutura.

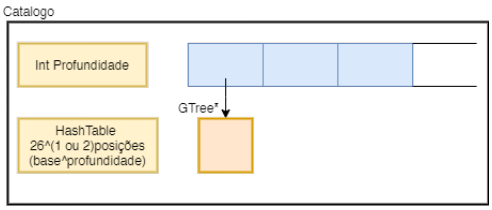

Figura 3.2: Estrutura do Catálogo

3.1.2 Clientes

O módulo clientes tem o propósito de guardar todos os códigos dos Clientes registados no SGV. Para tal, a *struct*, denominada **Clientes**, não é nada mais do que um catálogo com profundidade 1, uma vez que os códigos de clientes são inicializados por uma letra maiúscula do alfabeto português.

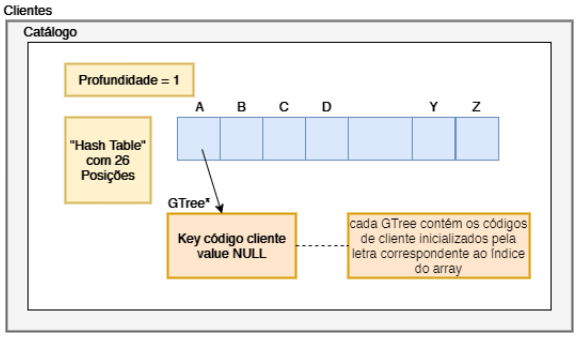

Figura 3.3: Estrutura de Clientes

3.1.3 Produtos

Este módulo tem a função de modelar os Produtos na SGV, sendo que a estrutura Produtos, à semelhança da estrutura Clientes contêm um Catalogo, onde são guardados todos os códigos dos produtos por ordem alfabética. Contudo, o campo profundidade deste catálogo de produtos é 2, implicando que o HashTable presente na estrutura tenha 676 posições (26*26 combinações de letras), uma vez que os códigos de produtos são inicializados por duas letras maiúsculas.

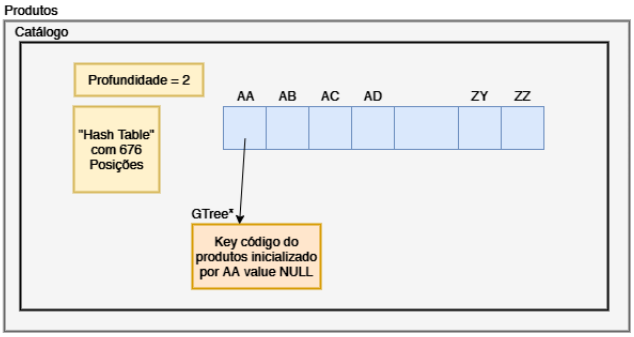

Figura 3.4: Estrutura de Produtos

3.1.4 Facturação

O módulo facturacao tem como finalidade modelar a informação relativa às vendas de cada um dos produtos. Para isso, a estrutura Faturacao consiste num array, com tamanho igual ao número de filiais do sistema (neste caso 3) em que a cada posição está associado um catálogo de profundidade 2, onde nas várias GTrees do HashTable são guardados todos os códigos de produtos vendidos nessa mesma filial. Adicionalmente, a cada um desses códigos está associada uma estrutura auxiliar Valores, que contêm informação relativa ao número de registos de venda desse produto, como o total faturado desse produto para essa filial. Estes valores estão guardados em quatro arrays de inteiros com 12 posições, relativas aos 12 meses do ano, em que dois desses arrays armazenam inteiros com informação relativa ao número de registos de vendas desse produto, em modo Normal e modo Promoção, e outros dois arrays que armazenam doubles que correspondem ao total faturado desse produto, também distinguindo entre modo Normal e modo Promoção.

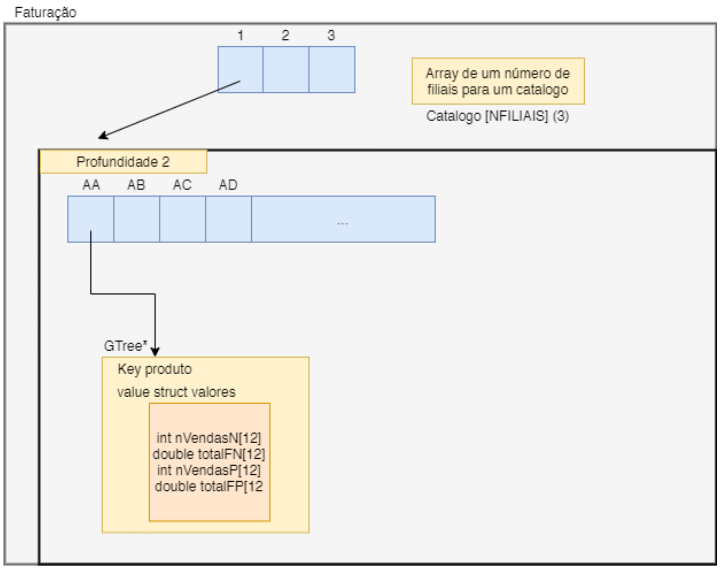

Figura 3.5: Estrutura da Faturação

3.1.5 Filial

O módulo filial tem como propósito relacionar todos os clientes que efetuaram compras numa determinada filial com os produtos que compraram. Para isto, a estrutura filial é composta por um catalogo de profundidade 1 (onde são guardados os códigos dos clientes), e em que a cada cliente está associada uma GTree (árvore balanceada da biblioteca Glib) cujas chaves são os códigos dos produtos comprados por esse cliente. Ademais, a cada um desses produtos está associado (no parâmetro value da gtree) uma **struct** Info. Esta estrutura auxiliar é responsável por guardar o total gasto por esse cliente nesse produto, assim como a quantidade comprada desse produto em cada mês do ano. Adicionalmente está presente também um *array* utilizado como *flag* para saber se esse produto foi comprado por esse cliente em modo Normal e/ou em Promoção.

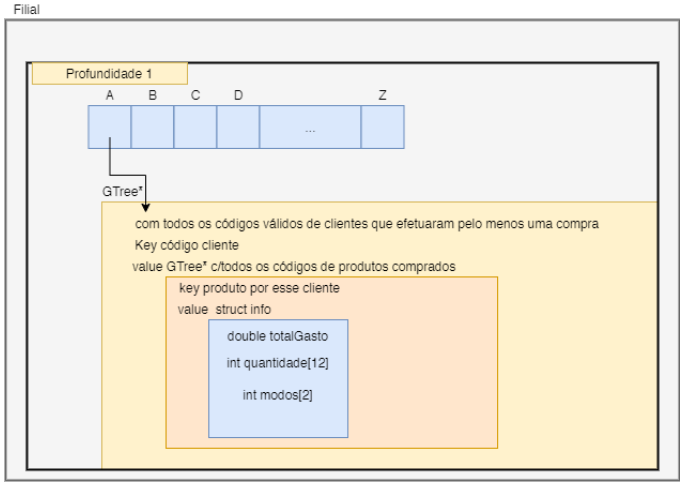

Figura 3.6: Estrutura da Filial

Neste módulo foram ainda criadas três estruturas auxiliares, com a intenção de possibilitar a resposta de certas queries impostas pelo projeto, cuja informação necessária às suas respostas reside no módulo filial.

3.1.6 Venda

O módulo auxiliar Venda foi implementado apenas para armazenar a informação relativa a uma venda numa *struct* venda (código de produto, preço unitário, quantidade, tipo, código de cliente, mês e filial). Associadas a esta estrutura, estão presentes os respectivos *getters* e *setters* necessários ao encapsulamento do módulo.

3.1.7 Parser

Este módulo tem o propósito de lidar com a importação de dados, comportando as funções necessárias para validação dos códigos de produtos, de clientes e das linhas de vendas, presentes nos diferentes ficheiros e posterior adequação às estruturas de dados.

3.1.8 Queries

No módulo Queries, foi implementada a estrutura principal do SGV, que contém todas as estruturas e informação necessárias para o correto funcionamento da aplicação (criação, manutenção e consulta de dados). Esta estrutura SGV, que é a estrutura principal do programa é constituída por:

- **Estrutura Clientes** que guarda todos os códigos dos clientes válidos e registados no sistema de vendas.
- **Estrutura Produtos** que guarda, por sua vez, todos os códigos dos produtos válidos e que podem ser vendidos no sistema de vendas.
- **Estrutura *ProdutosNãoVendidos*** que contém todos os códigos dos produtos que não foram vendidos em nenhuma das filiais.
- **Três Estruturas Infofile** que guardam toda a informação relativa aos 3 ficheiros que são lidos para o carregamento dos dados).
- **Estrutura Faturacao**
- **Array Filiais** com tamanho igual ao número de filiais do sistema. Cada posição contém uma estrutura Filial correspondente (0-Filial 1, 1-Filial 2, 2-Filial 3).

Neste módulo foram implementadas, à base de funções que nos permitiram manter o encapsulamento da própria estrutura SGV, todas as funções necessárias para responder às queries pretendidas, assim como todas as funções necessárias à criação e libertação da própria estrutura SGV. Estas funções de encapsulamento são essenciais para mantermos a estrutura interna SGV desconhecida às funções que dão resposta às queries e impedindo estas mesmas de alterar qualquer tipo de conteúdo presente no SGV.

3.1.9 Resultados

Este módulo foi criado com a intenção de conter as estruturas auxiliares criadas para conseguir dar resposta às diferentes queries, de modo a evitar a criação dessas mesmas estruturas nos módulos principais tais como, Clientes, Produtos, Faturacao e Filial. É um módulo inteiramente auxiliar, cujas estruturas nos permitiram manter o encapsulamento da informação nelas contida.

3.2 View

A view é o módulo responsável por notificar o utilizador daquilo que é suposto este conhecer, sendo assim uma espécie de "canal"de apresentação não só de dados, como de menus do programa ou até mesmo de informar eventuais erros que possam acontecer, como a introdução de opções inválidas pelo utilizador.

3.3 Controlador

O Controlador (**Main**) encarrega-se do correto fluxo de execução do programa. É responsável por recolher as informações da interação do utilizador e tratar de recolher as respostas e encaminhá-las para o módulo view que por sua vez as mostra.

3.4 Outras Abordagens

Como foi anteriormente referido, a solução apresentada não foi a única solução proposta e testada para os diferentes módulos do modelo. Onde foram verificadas maiores diferenças relativamente à quantidade necessária de memória alocada e até mesmo aos tempos de execução do programa, nomeadamente nos tempos de carregamento dos dados, foi ao nível dos módulos da Faturacao e da Filial...

Relativamente ao módulo **Faturacao** uma primeira abordagem ao problema foi implementar dois arrays aninhados, em que o primeiro visava a divisão dos meses, e um segundo associado a cada uma das posições do primeiro que diferenciava as filiais. Neste segundo array, a cada posição estava associado um HashTable dinâmico da biblioteca Glib cujas chaves referenciavam os códigos de produtos correspondentes, e os values uma estrutura auxiliar que continha informação relevante.

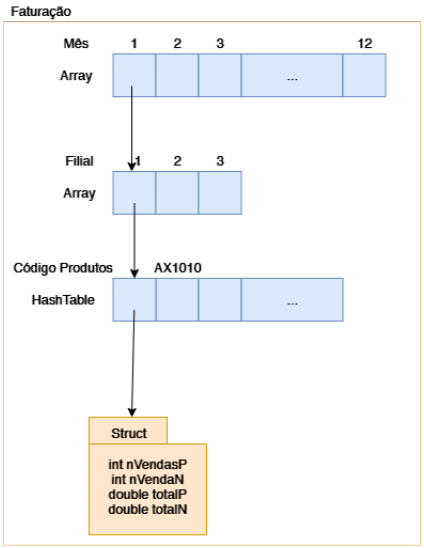

Figura 3.7: Primeira tentativa de implementação do módulo faturacao

Contudo, esta abordagem foi rapidamente posta de parte uma vez que implicava um grande défice de rapidez no processamento dos dados sempre que uma venda era adicionada à estrutura faturacao. Não só havia o problema do elevado número de acessos à memória, como o principal entrave foi no desperdício de alocação de memória que o HashTable dinâmico estava a necessitar, tornando todo o desempenho do programa muito menos eficiente.

Uma segunda abordagem de implementação deste módulo, não difere em muito da estrutura final do mesmo. Contudo, o carregamento dos dados estava a ser efetuado de forma significativamente mais demorada, tendo como causa um maior número de acessos à memória a ser realizados.

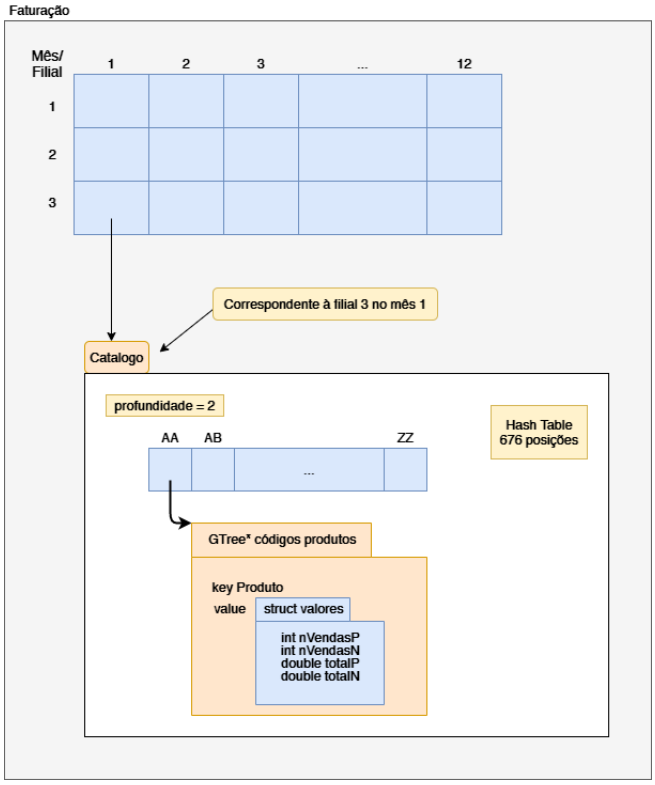

Figura 3.8: Segunda tentativa de implementação do módulo faturacao

Relativamente ao módulo **Filial** a primeira abordagem seguida, não fugiu muito à implementação final do módulo.

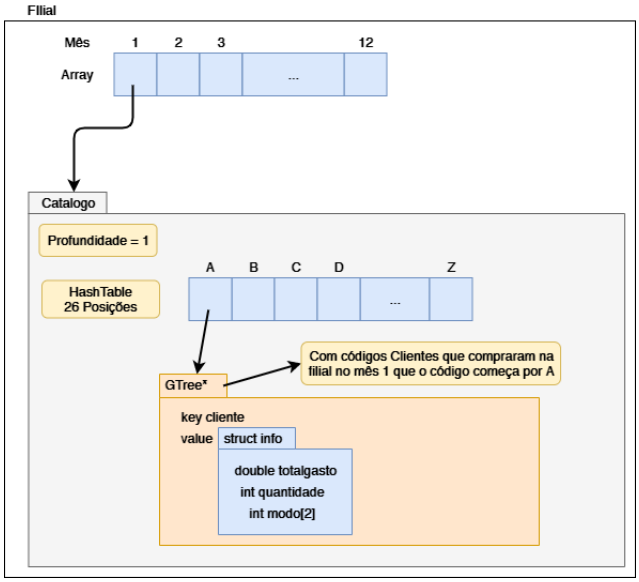

Figura 3.9: Primeira tentativa de implementação do módulo filial

Como podemos verificar, o módulo filial passava a ser constituido não só por um catálogo de clientes, mas por 12 catálogos de clientes, um para cada mês do ano. Esta abordagem é claramente mais dispendiosa a nível de memória, contudo tornava a inserção de uma venda no módulo filial ligeiramente mais rápida... Optamos então pela eficiência a nível de alocação de memória.

4. Dependências e Funcionamento

Nesta secção iremos proceder à explicação do funcionamento da aplicação, nomeadamente as suas dependências, com o objectivo de esclarecer como o programa executa cada interação do utilizador à finalidade do programa.

4.1 Funcionamento

Inicialmente, é feita a leitura dos ficheiros através das directorias inseridas pelo utilizador para carregamento de dados, ou se escolhidos os caminhos default, são carregados os ficheiros inicialmente fornecidos. A função "main" chama continuamente a função "*runController*" até que o utilizador decida sair do programa. É esta função que se compromete a manter o correto funcionamento do programa relativamente às decisões do utilizador. Para isso, é apresentado um menu ao utilizador, com o leque de opções disponíveis e que após recolher a informação introduzida pelo utilizador, "manda"executar a querie correspondente que comunica com a camada lógica, e posteriormente recebe os dados resultado e encaminhando-os para a função correspondente da view, de forma adequada à mostragem do resultado.

4.2 Dependências

A makefile tem como objetivo gerar o executável do programa. Para isso, usamos vários "Switches"que ajudam na identificação de erros e optimização do programa.

```
FLAGS = -Wall -O2 -g -ansi -Qunused-arguments
```

Figura 4.1: Flags usadas na Makefile

A flag -Wall é usada para warnings, -O2 para otimização na compilação, -g para debugging, -ansi e a flag -Qunused-arguments foi usada devido a warnings gerados pelo uso da biblioteca Glib.

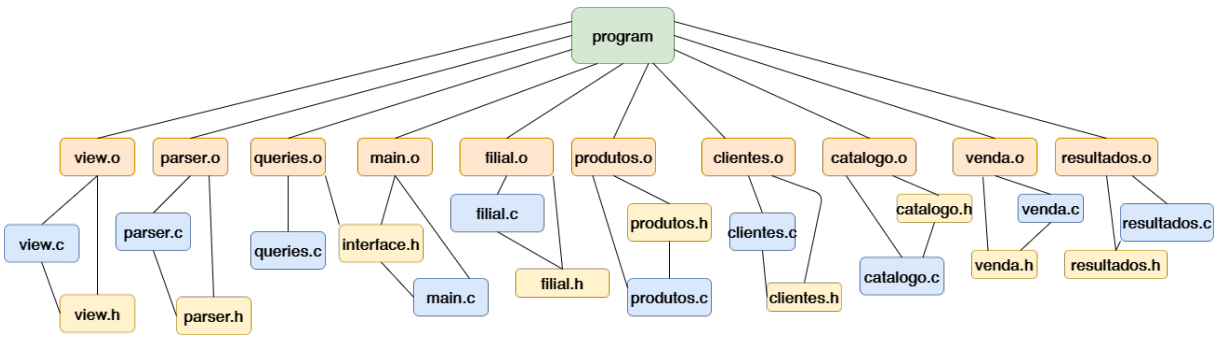

Figura 4.2: Grafo de Dependências

5. Desempenho e Considerações

Neste capítulo vamos ter em consideração o desempenho do nosso programa, tendo em conta principalmente os tempos de execução de todas as queries. Estes tempos vão ser ainda comparados usando 3 diferentes ficheiros de vendas: Vendas_1M.txt com um milhão de linhas de vendas, "Vendas_1M.txt","Vendas_3M.txt"e "Vendas_5M.txt", com três e cinco milhões, respetivamente. Para o carregamento dos Clientes e Produtos usamos os ficheiros "Clientes.txt"e "Produtos.txt", respetivamente.

Nas figuras seguintes, mostramos os tempos de execução das queries quando o carregamento de dados é feito com os ficheiros base de clientes, de produtos e os diferentes ficheiros de vendas.

Como podemos averiguar, os tempos de carregamento dos dados para o primeiro caso (Vendas_1M.txt) são bastante mais reduzidos do que quando o problema é escalado a um tamanho de dados significativamente maior. Era de esperar até, que o tempo de carregamento fosse de alguma forma proporcional ao tamanho do ficheiro, contudo isso não foi verificado. Houve uma acentuação muito mais significativa no tempo de execução da querie 1, quando passamos do ficheiro com um milhão de linhas de vendas para o de três milhões, do que quando passamos do ficheiro com "Vendas_3M"para o ficheiro "Vendas_5M.txt".
Um outro aumento significativo verificado foi a nível da querie 11, cujo tempo de resposta, à semelhança do tempo de carregamento, sofreu um grande aumento na primeira mudança de ficheiro e um aumento muito menos significativo na segunda mudança.
Fizemos o teste de comparar os tempos de carregamento dos dados, sem fazer a inserção das vendas nas estruturas correspondentes às filiais e percebemos que o grande aumento do tempo está relacionado com esta. Isto acontece principalmente, devido ao maior número de vendas, onde consequentemente vai haver um maior número de associações dos produtos aos clientes, pelo que a inserção e a consulta das vendas na estrutura filial vai ser cada vez mais custosa.

Outro aspeto a salientar, foi observar precisamente o contrário relativamente ao tempo de resposta das restantes queries, em que as oscilações foram praticamente nulas. Isto, porque na maioria dos casos a resposta às queries advêm de travessias feitas às estruturas em que o custo é praticamente o mesmo, e sendo que os ficheiros de carregamento de clientes e produtos são os mesmos nos três casos, não há nenhuma alteração no número de códigos nos catálogos.

```
./program
Query1: 5.596951 seconds
Query2: 0.102148 seconds
Query3: 0.000078 seconds
Query4: 0.000720 seconds
Query5: 0.017592 seconds
Query6: 0.014160 seconds
Query7: 0.000040 seconds
Query8: 0.136849 seconds
Query9: 0.011836 seconds
Query10: 0.000084 seconds
Query11: 1.361284 seconds
Query12: 0.000105 seconds
MBP-de-Ana:projC anacesar$
```

(a) Tempos de Execução Vendas_1M.txt

```
./program
Query1: 20.403670 seconds
Query2: 0.114115 seconds
Query3: 0.000268 seconds
Query4: 0.000057 seconds
Query5: 0.017930 seconds
Query6: 0.011955 seconds
Query7: 0.000046 seconds
Query8: 0.157427 seconds
Query9: 0.016355 seconds
Query10: 0.000058 seconds
Query11: 4.817215 seconds
Query12: 0.000370 seconds
MBP-de-Ana:projC anacesar$
```

(b) Tempos de Execução Vendas_3M.txt

```
./program
Query1: 31.547867 seconds
Query2: 0.101261 seconds
Query3: 0.000050 seconds
Query4: 0.000033 seconds
Query5: 0.016996 seconds
Query6: 0.010990 seconds
Query7: 0.000085 seconds
Query8: 0.158968 seconds
Query9: 0.025287 seconds
Query10: 0.000087 seconds
Query11: 7.493588 seconds
Query12: 0.000476 seconds
MBP-de-Ana:projC anacesar$
```

(c) Tempos de Execução Vendas_5M.txt

6. Conclusão

Este projecto permitiu permitiu-nos uma melhor compreensão dos desafios associados ao desenvolvimento de uma aplicação com grandes volumes de dados e complexidade elevada a nível algorítmico e estrutural, assim como da importância do encapsulamento de dados para a manutenção de uma aplicação. São de realçar três pontos principais que achamos que de certa forma falharam na nossa implementação:

- O módulo resultados foi criado com a intenção de conter todas as estruturas auxiliares criadas para dar respostas às queries de maior complexidade. Contudo, no caso do módulo filial, optamos por não colocar as estruturas auxiliares no módulo resultados, devido à complexidade das estruturas, pois isso comprometeria a sua rapidez e dificultaria o próprio acesso.

- Como forma de manter o encapsulamento do sgv, optamos por trabalhar com cópias das estruturas, pois dessa forma também garantíamos que o acesso feito às estruturas pelas queries não alteravam o conteúdo das mesmas. Para isso, implementamos funções, neste caso gets, a cada uma das estruturas que compõem o SGV. Estas funções têm o objetivo de criar e devolver cópias completas das estruturas originais, para que as queries trabalhem sobre estas. Contudo, os gets implementados à estrutura faturacao e filial apenas devolvem um novo apontador para a estrutura, uma vez que devido à maior complexidade deste módulos efetuar cópias completas destas estruturas era muito custoso, o que estava a provocar um maior défice no tempo de resposta às queries que envolvem estas estruturas. Assim, decidimos dar prioridade à eficiência no tempo de resposta.

- Um último ponto a referir vai de encontro com a estrutura do módulo filial, em que desde o momento que implementamos este módulo e fizemos a correta inserção das vendas, os tempos de carregamento dos dados aumentaram em cerca de dois segundos[1]. Ainda relativamente a uma possível melhoria de tempos de execução podemos voltar a referir a querie 11, que é a nossa querie mais demorada. Contudo, a solução encontrada para a sua resolução foi a mais eficiente e correta que encontramos.

Para finalizar, podemos referir que esta primeira fase da implementação do projeto na linguagem C é bastante desafiante, mas por outro lado bastante gratificante uma vez que pensamos ter cumprido os objetivos deste trabalho prático.

[1]Está presente na pasta docs do projeto, um ficheiro de texto "notas.txt"com as comparações dos tempos de execução em diferentes fases e implementações do projeto.