Escola de Engenharia
Universidade do Minho

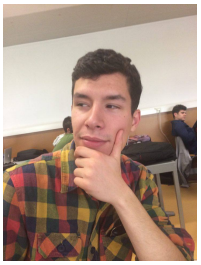
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado Integrado em Engenharia Informática
Laboratórios de Informática III

Gestão de Vendas de uma cadeia de Distribuição com 3 filiais
GEREVENDAS

Grupo 84


Célia Figueiredo
a67637


Gil Gonçalves
a67738


Humberto Vaz
a73236


Ricardo Lopes
a72062

Braga, 27 de Abril de 2016

1. Introdução

No âmbito da unidade curricular de Laboratórios de Informática III do 2ºano do curso de MIEI, foi proposto o desenvolvimento de um projeto em linguagem C que tem por objetivo ajudar à consolidação dos conteúdos teóricos e práticos e enriquecer os conhecimentos adquiridos nas UCs de Programação Imperativa, de Algoritmos e Complexidade, e da disciplina de Arquitetura de Computadores.

O Projeto, denominado GereVendas, baseia-se num programa de gestão de hipermercados com 3 filiais que dependem de uma lista de clientes, uma lista de produtos e uma lista de vendas efetuadas. Cada uma destas listas estará num ficheiro .txt e para cada um dos ficheiros o programa percorre o ficheiro, executando operações que permitam guardar estes dados em memória. Para ajudar nesta tarefa repartir-se-á as tarefas em quatro módulos módulos. Estes módulos são: um catálogo de clientes; um catálogo de produtos; um módulo de faturação global; um módulo de gestão de filial.

De forma a preservar o encapsulamento de dados será disponibilizada uma API de forma a que o utilizador apenas possa aceder através destas funções públicas. Depois dos ficheiros serem carregados o utilizador será capz de executar uma lista de queries previamente fornecida pela equipa docente. Para responder às diferentes queries utilizam-se as funções definidas nas API dos diferentes módulos.

Este projeto considera-se um grande desafio, pelo facto de passarmos a realizar programação em grande escala, uma vez que se tratam de grandes volumes de dados e por isso uma maior complexidade. Nesse sentido, o desenvolvimento deste programa será realizado à luz dos princípios da modularidade (divisão do código fonte em unidades separadas coerentes) e do encapsulamento (garantia de proteção e acessos controlados aos dados).

Conteúdo

2. Descrição dos Módulos

A arquitetura da aplicação a desenvolver é definida por quatro módulos principais: Catálogo de clientes, Catálogo de produtos, Faturação Global e Vendas por Filial, cujas fontes de dados são três ficheiros de texto detalhados abaixo.

No ficheiro **Produtos.txt** cada linha representa o código de um produto vendável no hipermercado, sendo cada código formado por duas letras maiúsculas e 4 dígitos (que representam um inteiro entre 1000 e 1999), como no exemplo:

```
AB9012
XY1185
BC9190
```

O ficheiro de produtos contém cerca de 200.000 códigos de produto.

No ficheiro **Clientes.txt** cada linha representa o código de um cliente identificado no hipermercado, sendo cada código de cliente formado por uma letra maiúscula e 4 dígitos que representam um inteiro entre 1000 e 5000, segue um exemplo:

```
F2916
W1219
F2915
```

O ficheiro de clientes contém cerca de 20.000 códigos de cliente.

O ficheiro **Vendas_1M.txt**, no qual cada linha representa o registo de uma venda efectuada numa qualquer das 3 filiais da Cadeia de Distribuição. Cada linha (a que chamaremos compra ou venda, o que apenas depende do ponto de vista) será formada por um código de produto, um preço unitário decimal (entre 0.0 e 999.99), o número inteiro de unidades compradas (entre 1 e 200), a letra **N** ou **P** conforme tenha sido uma compra **Normal** ou uma compra em **Promoção**, o código do cliente, o mês da compra (1 ... 12) e a filial (de 1 a 3) onde a venda foi realizada, como se pode verificar nos exemplos seguintes:

```
KR1583 77.72 128 P L4891 2 1
QQ1041 536.53 194 P X4054 12 3
OP1244 481.43 67 P Q3869 9 1
JP1982 343.2 168 N T1805 10 2
IZ1636 923.72 193 P T2220 4 2
```

O ficheiro de vendas inicial, **Vendas_1M.txt** , conterá 1.000.000 (1 milhão) de registos de vendas realizadas nas 3 filiais da cadeia de distribuição. Existirão também os ficheiros **Vendas_3M.txt** e **Vendas_5M.txt** utilizados para as questões de performance da aplicação.

A aplicação possuiu uma arquitectura tal como apresentado na figura seguinte, em que se identificam as fontes de dados, a sua leitura e os módulos de dados a construir:

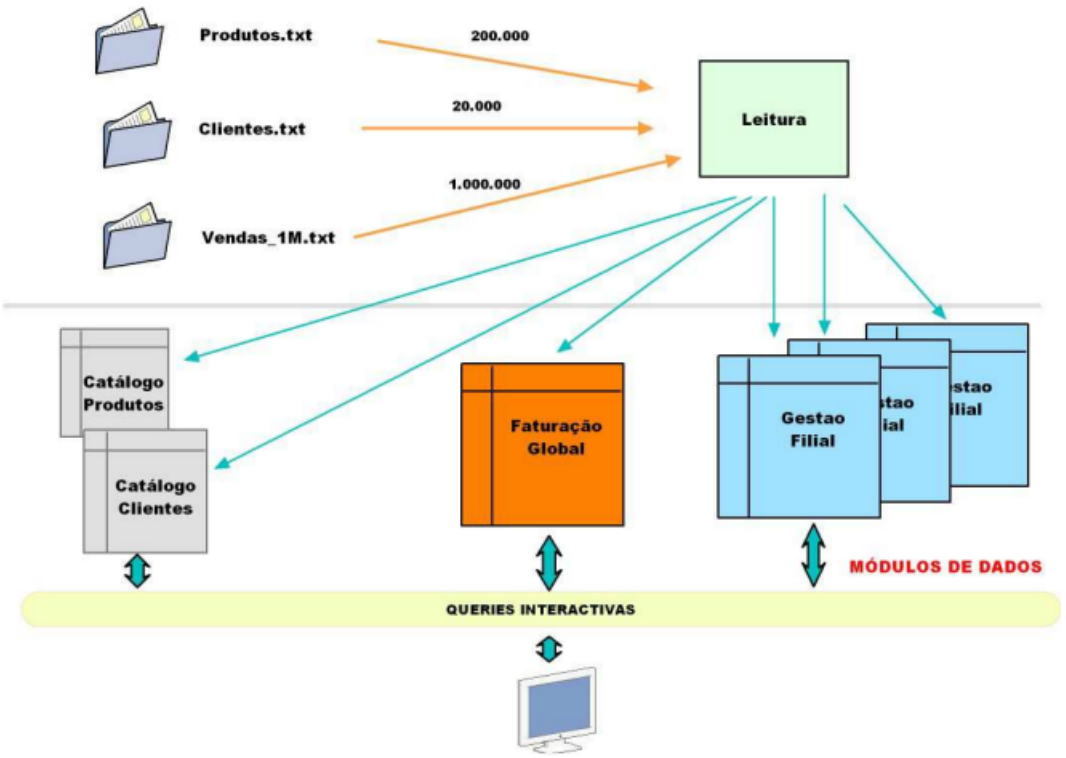


Figura 2.1: Arquitetura da aplicação

2.1 Catálogo de Clientes

É o módulo de dados onde são guardados os códigos de todos os clientes do ficheiro **Clientes.txt**, organizados por índice alfabético;

2.1.1 Clientes.h

Módulo de dados onde são guardados os códigos de todos os clientes do ficheiro **Clientes.txt**. O array de árvores este que é um array de 26 posições cujos índices se encontram organizados alfabeticamente. Cada índice contém um apontador para uma árvore correspondente à letra respetiva desse índice.

Tipos Opacos

```
typedef struct catalogo_clientes *CatClientes;
```

O typedef no ficheiro Clientes.h é a única informação que o utilizador têm relativamente à implementação de dados, não tendo acesso ao ficheiro .c dos clientes, não conseguindo conhecer a verdadeira implementação da estrutura AVL. Deste modo garantimos o encapsulamento de dados e a única forma de o utilizador interagir com o catálogo de clientes será através da API;

clientes.c

```
struct catalogo_clientes{
ARVORE indices[27];
};
```

Para guardar os clientes lidos a partir do ficheiro resolvemos usar AVLs, pois executa as operações de inserção, busca e remoção em tempo O(log n), sendo rápida para aplicações que fazem uma quantidade excessiva de procuras, porém esta estrutura é um pouco mais lenta para inserção e remoção. Isso deve-se ao facto de as árvores AVL serem rigidamente balanceadas.

/*API*/

- CatClientes inicializa_catalogo_clientes() - Função que cria uma estrutura de clientes vazia;
- void insertC(CatClientes c, char * valor) - insere um dado cliente na estrutura CatClientes, de maneira ordenada alfabeticamente;
- void cat_remove_cliente(CatClientes cat, char *str) - remove um cliente da estrutura CatCliente
- void free_catalogo_Clientes(CatClientes cat) - função que liberta todo o espaço ocupado pela estrutura CatClientes
- int existeCliente (char *cliente,CatClientes cat) - função que verifica se um cliente existe
- int numeroClientes(CatClientes cat) - função que conta todos os nodos
- int numeroClientesLetra(CatClientes cat, char letra) - função que conta os nodos por que começam por determinada letra

2.2 Catálogo de Produtos

Módulo de dados onde são guardados os códigos de todos os produtos do ficheiro **Produtos.txt**, organizados por índice alfabético, o que irá permitir, de forma eficaz, saber quais são os produtos cujos códigos começam por uma dada letra do alfabeto e saber quantos produtos são contabilizados.

2.2.1 Produtos.h

Tipos Opacos

```
typedef struct catalogo_produtos *CatProdutos;
```

Este typedef é a única informação que o utilizador tem relativamente à implementação de dados, não tendo acesso ao ficheiro .c dos produtos não consegue conhecer a verdadeira implementação da estrutura AVL. Assim, garantimos o encapsulamento de dados e a única forma de o utilizador interagir com o catálogo de produtos será através da API.

produtos.c

```
struct catalogo_produtos{
ARVORE indices[27];
};
```

Para guardar os produtos lidos a partir do ficheiro utilizámos uma avl,

/*API*/

- CatProdutos inicializa_catalogo_produtos();
- void insertP(CatProdutos c, char * valor);
- void cat_remove_produto(CatProdutos cat, char *str);
- void free_catalogo_produtos(CatProdutos cat);
- int existeProduto (char *produto,CatProdutos cat);
- int numeroProdutos(CatProdutos cat);
- int numeroProdutosLetra(CatProdutos cat, char letra);
- ARRAY listaProdutosLetra(CatProdutos cat, char l);

2.3 Faturação Global

Módulo de dados que contém as estruturas de dados responsáveis pela resposta a questões quantitativas que relacionam os produtos às suas vendas mensais, em modo Normal (N) ou em Promoção (P), para cada um dos casos guardando o número de vendas e o valor total de faturação de cada um destes tipos. Este módulo refecencia todos os produtos, mesmo os que nunca foram vendidos, não contém qualquer referência a clientes, mas é capaz de distinguir os valores obtidos em cada filial.

2.3.1 faturacao.h

Tipos Opacos

```
typedef struct faturacao *Faturacao;
typedef struct info *Info;
```

faturacao.c

```
struct faturacao{
int totalvendas[12];
float totalfaturado[12];
ARVORE produtos;
};

struct info{
char *code;
```

```
int vendasP[12][3];
int vendasN[12][3];
float faturadoN[12][3];
float faturadoP[12][3];
int quantidadeP[12][3];
int quantidadeN[12][3];
};
```

Para guardar a informação relativa à faturação utilizámos um array de 12 AVLs, pois esta estrutura permite ordenar estruturas complexas de acordo com um método de comparação, separando pelos 12 meses. Como existe ordem alfabética, a AVL permite fazer procura rápida, preservando assim memória.

/*API*/

- Faturacao inicializa_faturacao() - A funcão inicializa_faturacao aloca uma estrutura Faturacao e cria uma avl para os produtos, bem como inicializa os 12 nodos totalvendas e totalfaurado a zero;
- void cont_regista_produto(Faturacao fat, char *prod) - A funcão cont_regista_produto, dada a estrutura faturacão e um id de produto, insere-o através da funcão avl_insert;
- void cont_insere_venda(Faturacao fat, char *produto, int q, float preco, char M,int mes, int filial) - A funcão cont_insere_venda, dada a estrutura faturacão, atualiza as vendas, quantidades e faturado conforme o produto em questão ser Promocão (P) ou Normal (N);
- void cont_remove_produto(Faturacao fat, char *produto) - A funcão cont_remove_produto remove um produto da Faturacao passados ambos por parametro. Inicialmente calcula o id do produto através da funcão fat_procura_info para depois eliminar o produto em questão através da funcão avl_delete e logo a seguir faz free ao nodo eliminado através da funcão free_info;
- void free_faturacao(Faturacao fat) - A funcão free_faturacao elimina todos os nodos da faturacao através da funcão avl_destroy e faz free à estrutura faturacao passada por parâmetro;
- float getTotalFatPFilialX (char* prod,int mes,Faturacao fat, int filial) - A funcão getTotalFatPFilialX vai calcular o total faturado no modo Promocão (P) de um dado produto, num determinado mês e em determinada filial;
- float getTotalFatNFilialX (char* prod,int mes,Faturacao fat, int filial) - A funcão getTotalFatNFilialX vai calcular o total faturado no modo Normal (N) de um dado produto, num determinado mes e em determinada filial;
- int getVendasNFilialX (char* prod,int mes,Faturacao fat, int filial) - A funcão getVendasNFilialX calcula o nr de vendas em modo Normal (N) de um produto num determinado mês e filial. Inicialmente vai buscar o produto através do seu id, se o encontrar devolve o nr de vendas nas condicões anteriores;
- int getVendasPFilialX (char* prod,int mes,Faturacao fat, int filial) - A funcão getVendasPFilialX calcula o nr de vendas em modo Promocao (P) de um produto num determinado mês e filial. Inicialmente vai buscar o produto através do seu id, se o encontrar devolve o nr de vendas nas condicões anteriores;

- int getQuantidadeNFilialX (char* prod,int mes,Faturacao fat, int filial) - A funcão getQuantidadeNFilialX calcula a quantidade vendida em modo Normal (N) de um determinado produto num dado mês e filial;
- int getQuantidadePFilialX (char* prod,int mes,Faturacao fat, int filial) - A funcão getQuantidadePFilialX calcula a quantidade vendida em modo Promocão (P) de um determinado produto num dado mês e filial. Inicialmente vai buscar o nodo do produto através do id do produto e retorna finalmente a quantidade vendida;
- ARRAY naoCompradosFilial(Faturacao fat, int filial) - A funcão naoCompradosFilial percorre a estrutura da faturacao numa dada filial em todos os meses através do Traverser t, e verificia se este foi ou não comprado,se foi nada faz, se não insere-o num array a. No final faz free ao Traversser t e retorna o novo array;
- ARRAY naoComprados(Faturacao fat) - A funcão naoComprados percorre a estrutura da faturacao em todas as filiais e em todos os meses através do Traverser t, e verificia se este foi ou não comprado,se foi nada faz, se não insere-o num array a. No final faz free ao Traversser t e retorna o novo array;
- int totalVendasMeses(Faturacao fat, int a, int b) - A funcão totalVendasMeses calcula o total de vendas num determinado intervalo de meses;
- float totalFatMeses(Faturacao fat, int a, int b) - A funcão totalFatMeses calcula o total faturado num dado intervalo de meses;
- ARRAY nMaisVendidos(Faturacao fat, int n) - A funcão nMaisVendidos calcula os n produtos mais vendidos de todas as filiais através de um traversser t. Inicializa dois arrays a e b através da funcão inicializa_array . Copia os elementos do tipo Info para o array a, ordena-o através da funcão ordena e depois devolve o array b com os elementos ids dos produtos do array já ordenados;
- int getQuantidadeFilial(Faturacao fat, char*prod, int filial) - A funcão getQuantidadeFilial calcula a quantidade vendida de um determinado produto através do seu id de produto numa determinada filial em todos os meses.

2.4 Gestão da Filial

Módulo de dados que, a partir dos ficheiros lidos, contém as estruturas de dados adequadas à representação dos relacionamentos, fundamentais para a aplicação, entre produtos e clientes, ou seja, para cada produto, saber quais os clientes que o compraram, quantas unidades cada um comprou, em que mês e em que filial.

Para a estruturação optimizada dos dados deste módulo de dados tivemos em atenção que pretendemos ter o histórico de vendas organizado por filiais para uma melhor análise, nunca esquecendo que existem 3 filiais nesta cadeia.

2.4.1 Filial.h

Tipos Opacos

```
typedef struct filial *Filial;
typedef struct icliente *Icliente;
```

```
typedef struct iprodutos *Iprodutos;
```

Filial.c

```
struct filial{
ARVORE infoCliente;
};

struct icliente{
char *cliente;
int quantidade[12];
ARVORE infoprodutos[2];
};

struct iprodutos{
char *prod;
int quantidadeT;
float gastouT;
int quantidade[12];
float gastou[12];
};
```

- Filial inicializa_filial() - Inicializa a estrutura da filial;
- void fil_regista_cliente(Filial fil, char *cliente) - Insere um cliente na filial;
- void fil_insere_prod(Filial fil, char *cliente, char *produto,int q, int mes, float preco, char p) - InserE um produto comprado por um dado cliente num dado mês, se o produto já tiver sido inserido a quantidade=quantidadeAntiga+ quantidadeComprada, aumenta também o dinheiro gasto nesse produto. Senão tiver sido inserido a quantidade=quantidadeComprada e o preco=precoComprado;
- int getQuantidadeMesCliente(Filial fil, char *cliente, int mes) - Retorna a quantidade dos produtos comprados por um dado cliente, num dado mês;
- ARRAY naoCompraram(Filial fil);
- ARRAY compraram(Filial fil);
- void clientesCompraram(Filial fil,ARRAY a) - Remove do array todos os clientes que não compraram em todas as filiais;
- void free_filial(Filial fil);
- ARRAY topMaisGastou(ARRAY a) - Vai retornar os 3 produtos que um cliente mais gastou;
- ARRAY clientesCompraramProduto(Filial fil, char* produto) - Retorna a lista de clientes que compraram um determinado produto;

- int comprouProdutoN(Filial fil, char* cliente, char* produto) - função diz se um dado cliente comprou um dado produto em normal;
- int comprouProdutoP(Filial fil, char* cliente, char* produto) - função que diz se um dado cliente comprou um dado produto em promoção;
- int getNumClientesFilial(Filial fil, char* produto) - diz quantos clientes compraram um determinado produto;
- void getIProdMes(Filial fil, char* cliente, int mes, ARRAY a) - Para um dado cliente e para um mês, inser a informacao, dos produtos que o cliente comprou nesse mês numa dada filial;
- ARRAY extraiPorQuantidade(ARRAY a,int mes) - Devolve uma lista de produtos mais comprados, por quantidade, de um dado mês;
- void removeNaoCompraram(Filial fil, ARRAY a);
- void removeCompraram(Filial fil, ARRAY a);

3. Main.c

4. Interface do utilizador

Quando o utilizador executa o programa é-lhe pedido que escolha qual o documento de texto que pretende analisar, como podemos observar na figura seguinte:

Figura 4.1: Escolha do ficheiro de vendas a analisar

O ficheiro é carregado e de seguida aparece um menu com 12 opções, referentes às 12 queries do projeto, sendo que decidimos usar o [0] para sair do GereVendas. O objetivo é que o utilizador prima a tecla correspondente à opção do menu pretendida.

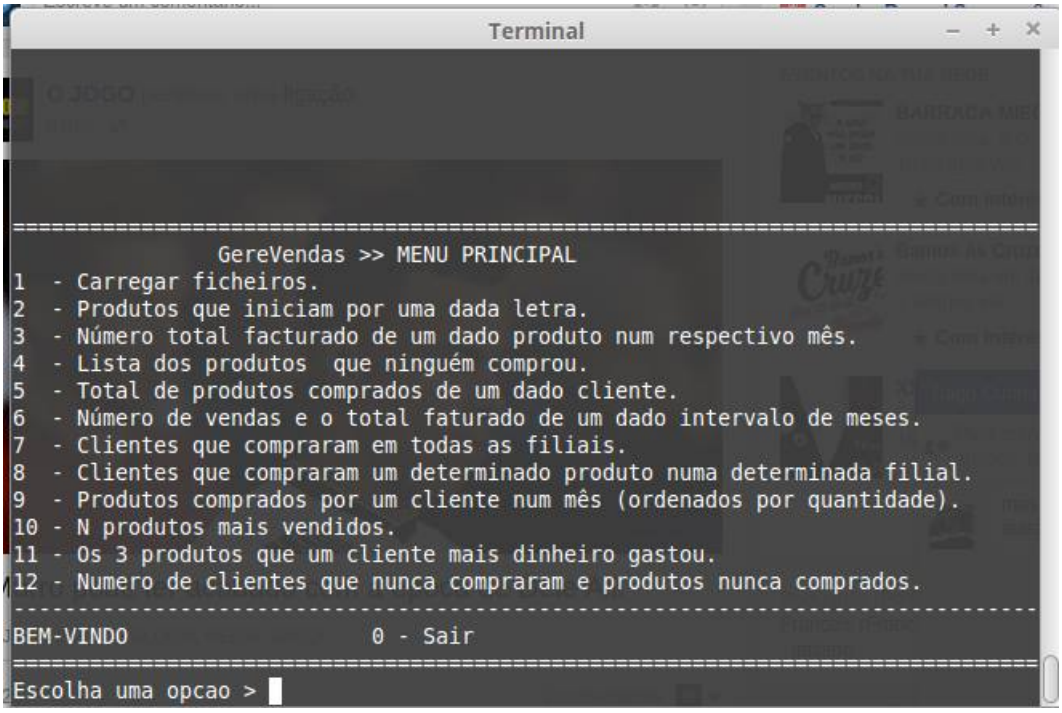


Figura 4.2: Menu principal da aplicação

5. Resultados e comentários sobre os testes de performance

Depois de desenvolver e codificar todo o projeto foi-nos proposto realizar alguns testes de performance que consistem em comparar os tempos de execução das queries 8, 9, 10, 11 e 12 usando os ficheiros Vendas_1M.txt (1000 000 vendas), Vendas_3M.txt (3 milhões de vendas) e Vendas_5M.txt (5 milhões de vendas). Uma vez que a quantidade de vendas vai aumentando de ficheiro para ficheiro é aceitável que os tempos de execução para os carregar aumente.

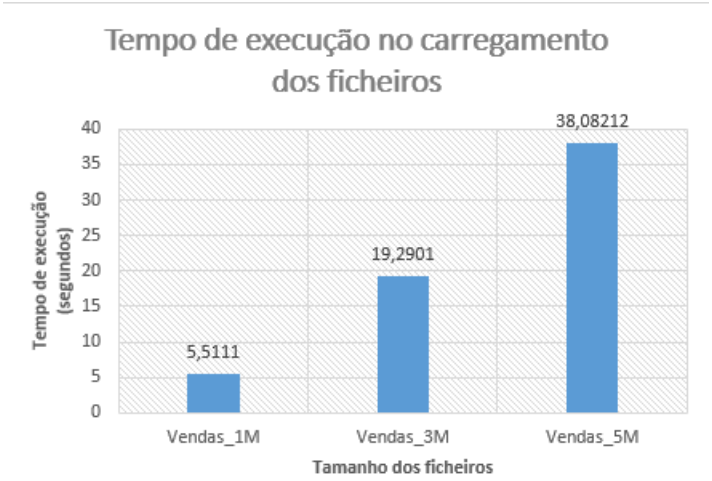


Figura 5.1: Gráfico do tempo de execução do carregamento dos 3 ficheiros de vendas

Comparando os valores de execução das queries pretendidas, como podemos observar nos respetivos gráficos apresentados,

6. Makefile e Grafo de dependências

A makefile permite correr todo o software escrevendo apenas *"make"* no terminal. Posto isto, apresenta-se a makefile utilizada cujas flags utilizadas como opção de compilação são –Wall –Wextra –ansi – pedantic –O2. Possui ainda a opção "make clean" que elimina todos os ".o" que foram criados quando se compilou o software.

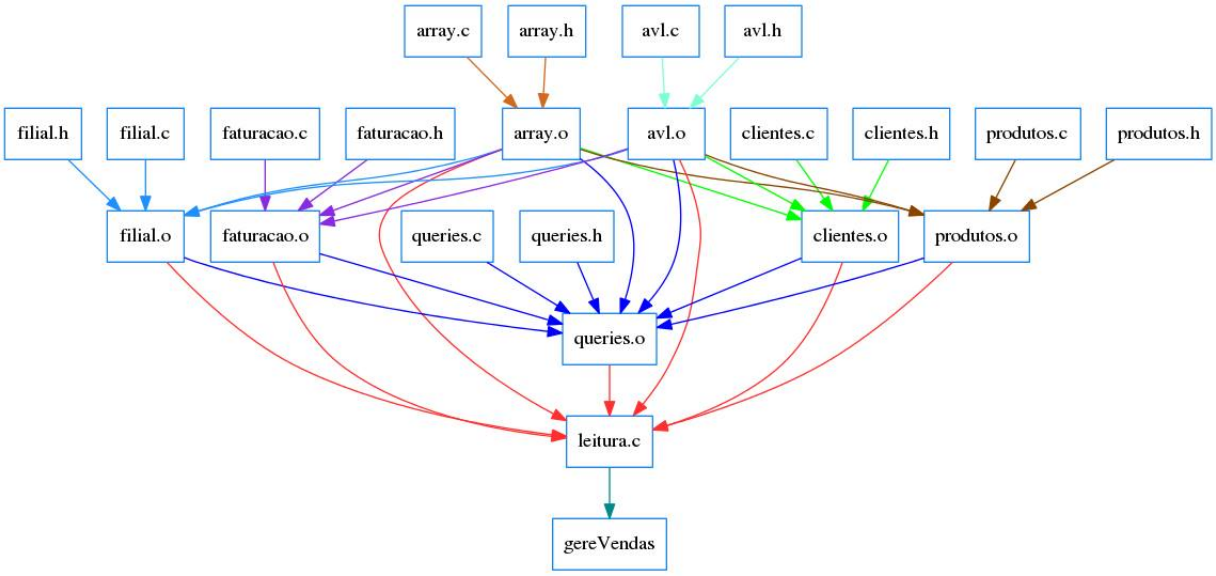


Figura 6.1: Grafo de depêndencias

```
objects = array.o avl.o clientes.o faturacao.o filial.o \
produtos.o queries.o

CFLAGS=-Wall -ansi -pedantic -O2

all:
make clean
make produtos
make array
make avl
make clientes
make faturacao
make filial
make queries
make leitura
```

```
leitura: src/leitura.c array.o avl.o clientes.o faturacao.o filial.o
 produtos.o queries.o
gcc src/leitura.c array.o avl.o clientes.o faturacao.o filial.o
 produtos.o queries.o $(CFLAGS) -o gereVendas -lm

queries: src/queries.c src/headers/queries.h
gcc src/queries.c -c $(CFLAGS)

clientes: src/clientes.c src/headers/clientes.h
gcc src/clientes.c -c $(CFLAGS)

produtos: src/produtos.c src/headers/produtos.h
gcc src/produtos.c -c $(CFLAGS)

array: src/array.c src/headers/array.h
gcc src/array.c -c $(CFLAGS)

faturacao: src/faturacao.c src/headers/faturacao.h
gcc src/faturacao.c -c $(CFLAGS)

filial: src/filial.c src/headers/filial.h
gcc src/filial.c -c $(CFLAGS)

avl: src/avl.c src/headers/avl.h
gcc src/avl.c -c $(CFLAGS)

.PHONY : clean
clean :
rm -f gereVendas
rm -f $(objects)
rm -f gesval
```

7. Conclusão

Uma vez que se tratou de um trabalho de uma dimensão já considerável comparando com o que estávamos habituados envolveu utilização de técnicas particulares e tivemos sempre como objetivo que este trabalho fosse concebido de modo a que seja facilmente modificável, e seja, apesar da complexidade, o mais optimizado possível a todos os níveis.