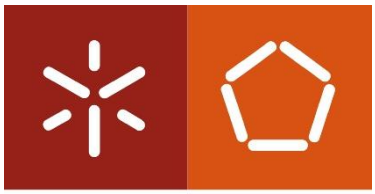


Universidade do Minho

Departamento de Informática

Sistema de Gestão de Vendas

Grupo Nº 18

Luís Martins (A89600)

Rui Gonçalves (A89590)

Tiago Alves (A89554)

12 de Abril de 2020

Conteúdo

1 Introdução

Nesta unidade curricular fomos desafiados a implementar um sistema de resposta (a *queries*) sobre um Sistema de Gestão de Vendas - **SGV**.

A primeira fase deste projeto consistiu no desafio de implementar este sistema na linguagem C. Foi-nos permitido o uso de recursos tais como bibliotecas de código, por exemplo *glib*, mas no entanto decidimos criar as nossas estruturas de raiz. Apesar de uma das intenções deste projeto ser tornar rápida a execução das funcionalidades do programa, alguns dos focos prioritários deste projeto foram a modularidade e o encapsulamento das estruturas de dados por nós utilizadas.

Ao longo do desenvolvimento deste projeto, consideramos que os maiores desafios foram implementar as estruturas onde organizar a informação.

2 Módulos e Estruturas de dados

De modo a conseguirmos responder às *queries* propostas da forma mais eficiente possível, o mais importante é a forma como organizámos a grande quantidade de informação que nos é fornecida.

Antes de começar a criar as estruturas, analisámos tanto os ficheiros fornecidos como as *queries* que nos eram propostas, de modo a perceber que dados eram importantes para cada uma delas.

2.1 Listas

O módulo de Listas é um módulo genérico de um array dinâmico de *strings* (em C dado por um array de apontadores) e contém funções de adição de elementos ao array, realocação de memória (caso o array se encontre preenchido) e funções de prócura binária. Cada estrutra LISTAS contém o tamanho total do array, o número de elementos ocupados e o array.

```
typedef struct listas{
        int size;
        int ocup;
        char **listas;
}LISTAS;
```

2.2 Venda

O módulo de Venda contém as funções necessárias à verificação da validade de uma venda usando as listas e a função de prócura binária mencionadas anteriormente (fazendo a procura de um cliente e de um produto nas suas respetivas listas). Este módulo contém uma estrutura Venda que contém todas a informações relativas à mesma, tem também os

getters dos parâmetros desta estruta retornando sempre que possível uma cópia (de modo a permitir o encapsulamento dos dados).

```
struct venda{
        char *prod;
        double preco;
        int unidades;
        char tipo;
        char *cliente;
        int mes;
        int filial;
};
```

2.3 AVL Tree

Este módulo contém tanto as funções de implementação de uma AVL balanceada como os getters para os parâmetros da mesma. Cada estrutura AVL é constituída por um nodo, um apontador para o nodo da direita e outro para o nodo da esquerda e,o mais importante, um apontador void que permite colocar no nodo da árvore um apontador para uma qualquer outra estrutura que desejemos, permitindo deste modo que este módulo seja a base da implementação dos restante módulos.

```
struct AVBin {
    char* key;
    void *str;
    struct AVBin *left;
    struct AVBin *right;
    int altura;
};
```

2.4 Fatura

Neste módulo, a API está definida de forma a ser possível, a partir de uma Venda, criar ou atualizar uma fatura para um certo produto. Em cada fatura temos os seguintes arrays:

- dois de inteiros, um para a quantidade vendida por filial e outro, ordenado por tipo de venda, mês e filial, onde está o numero de vendas de cada produto.
- um de double, que contém o valor faturado por produto, separado, também por tipo de venda, mês e filial.

Temos também ,neste módulo, os getter's de cada um dos parâmetros da estrutura tal como uma função que cria um clone de uma fatura que será necessário para manter o encapsulamento dos dados na resposta às *queries*.

```
struct fatura {
        char *produto;
        int numVendas[2][12][3];
    double fatTotal[2][12][3];
        int quantPorFilial[3];
};
```

2.5 Faturas

Este módulo contém a estrutura Faturas. Esta mesma estrutura é constituída por um array de inteiros que contém o número total de vendas divididas por mês , um array de double que contém o total faturado dividído por mês , contém por fim, um apontador para uma AVL que tem no seu nodo a Estrutura Fatura, sendo assim uma árvore de faturas.

Este módulo contém funções de inserção de vendas e de faturas na estrutura Faturas, tal como getter's e ,por fim, funções de resposta a diversas *queries*.

```
struct faturas {
    double totalFatMes[12];
    int totalVendasMes[12];
    AVL *avlF;
};
```

2.6 Filial

Este módulo contém a estrutura Filiais. Esta mesma estrutura é constituída por dois arrays de AVL's.

- O primeiro array, é um array de AVL's com apontadores para a estrutura CliVendas
  - estrutura composta por um inteiro (que indica o número de produtos diferentes comprados pelo cliente), um array que contém a quantidade de produtos comprados pelo cliente separado por mês e, por fim, um array de estruturas ProdCliente.
    - estrutura composta por uma string (que indica o nome do produto), a quantidade de vendas do produto desse mês e, por fim, a faturação total desse mesmo produto.

  dividido por filial e pela primeira letra pelo qual o cliente começa.

- O segundo array, é um array de AVL's com apontadores para a estrutura ListaProds
  - estrutura composta por duas listas, em que a primeira lista é composta por todos os clientes que compraram o produto sem promoção, e a segunda lista composta por todos os clientes que compraram o produto em promoção.

dividido por filial e pela primeira letra pelo qual o produto começa.

```
struct filiais {
    AVL* cliVendas[3][26];
    AVL* prodVendas[3][26];

};
```

```
typedef struct cliVendas {
    int ocup;
    int quantidades[12];
    ProdCliente **produtos;
} CliVendas;
```

```
typedef struct prodCliente {
    char *prod;
    int quantidades[12];
    double fatTotal;
} ProdCliente;
```

```
typedef struct listaProds {
    LISTAS *clienteN;
    LISTAS *clienteP;

} ListaProds;
```

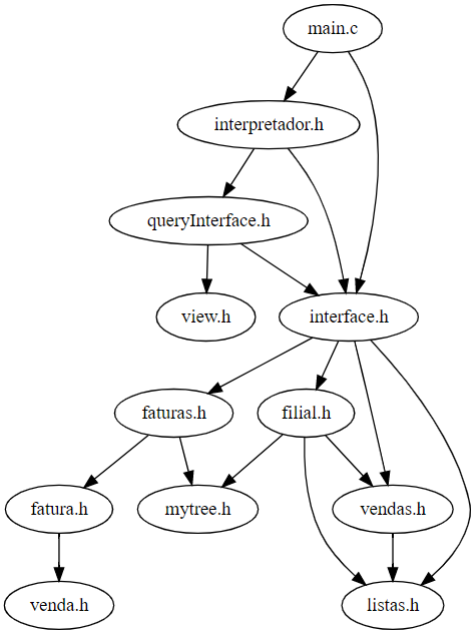
2.7 SGV

Este módulo contem a estrutura SGV, estrutura esta que contém um apontador para uma lista de clientes,um apontador para uma lista de produtos, um apontador para a estrutura Faturas com todas as faturas, um apontador para a estrutura Filais com todas as filiais e, por fim, um apontador para uma nova estrutura chamada Infos, estrutura esta que contém o caminho para o ficheiro, o número de linhas lidas do ficheiro e o número de linha válidas do ficheiro para cada um dos 3 ficheiros.

```
struct infos{
        char *fileC;
        int linhasC;
        int validasC;
        char *fileP;
        int linhasP;
        int validasP;
        char *fileV;
        int linhasV;
        int validasV;
};
```

Este módulo contém também as várias funções de getter's dos resultados que permitem responder às *queries*, onde sempre que possivel retornamos cópias dos valores/estruturas necessárias de modo a manter o encapsulamento dos dados das outras estruturas.

3 Grafo de dependências e estrutura do Projeto



3.1 Modelo

O modelo tem como base dois grandes módulos, o módulo Filial e o módulo Faturas e tem como módulos auxiliares, o módulo Listas e Árvores.

Estes módulos estão unidos no módulo mais geral, o interface, que contém a estrutura SGV e este recebe os pedidos do Controlador e trata a informação obtida nos módulos necessários de forma a devolver a informação pedida de forma mais eficiente.

É nesta camada onde se encontram toda a parte dos algorítmos e dados que resolvem as *queries* propostas.

3.2 Controlador

O Controlador tem como base dois grandes módulos, o módulo queryInterface e o Interpretador, o módulo Interpretador contém os menus de escolha das *queries* e o queryInterface contém os menus para cada *query* individualmente.

3.3 Apresentação

A apresentação contém apenas o módulo View, este contém funções de apresentação de arrays de strings genéricas e para apresentar tabelas de inteiros.

4 Tempos de execução

	1 Milhão	3 Milhões	5 Milhões
LoadTime(Q1)	4.7	16.5	26.68
LoadTime(Q2)	0.00038	0.00038	0.00038
LoadTime(Q3)	0.000006 (ZZ1999)	0.000005	0.000009
LoadTime(Q4)	0.11 (Todas Filiais)	0.125	0.188
LoadTime(Q5)	0.015	0.0156	0.0156
LoadTime(Q6)	0.11	0.125	0.125
LoadTime(Q7)	0.000018 (Z5000)	0.00001	0.000032
LoadTime(Q8)	0.0000113	0.000014	0.000018
LoadTime(Q9)	0.000006 (ZZ1999,1)	0.000005	0.000005
LoadTime(Q10)	0.005 (Z5000,1)	0.005	0.00722
LoadTime(Q11)	0.2372 (5)	0.2572	0.308
LoadTime(Q12)	0.000086 (Z5000,5)	0.000155	0.000218

Depois do desenvolvimento e codificação de todo o projeto foi-nos proposto realizar testes de performance das queries 6 a 12 usando os ficheiros de 1, 3 e 5 milhões de vendas e a biblioteca standard de C, time.h.

Uma vez que o número de vendas aumenta de ficheiro para ficheiro, seria expectável que o tempo de execução das queries aumentasse, no entanto apenas observamos um aumento significativo no tempo de carregamento dos dados para as estruturas,mantendo-se os tempos das *queries* aproximadamente iguais. Isto deve-se ao facto de a base da estrutura ser uma AVL cuja chave é ou o código de cliente ou o código do produto, como tanto o número de clientes como o de produtos não se altera, o número de nodos da AVL irá manter-se o mesmo, fazendo assim com que só o tempo de inicialização dessa estrutura (carregamento das vendas) tenha um aumento significativo mantendo-se o tempo de execução das *queries* aproximadamente igual.

Estes testes foram realizados numa máquina com processador de 3.5 GHZ com memória RAM de 8Gb.

5 Uso e libertação de mémoria

Utilizando a ferramenta *Valgrind*, inicialmente, e de modo a verificar se as queries e as estruturas libertavam todas a memória, executamos todas as *queries* apenas uma vez e obtivemos os seguintes resultados:

- Total heap usage: 11,750,616 allocs, 11,750,616 frees.
- 577,601,801 bytes alocados o que equivale aproximadamente a 550.84 Mb.

Para o caso do ficheiro de 3 milhões de vendas, sem executar as queries, obtivemos os seguintes resultados:

- Total heap usage: 30,162,994 allocs, 30,162,994 frees.
- 1,443,712,905 bytes alocados o que equivale aproximadamente a 1,34Gb.

Por fim, para o caso do ficheiro de 5 milhões de vendas, sem executar as queries, obtivemos os seguintes resultados:

- Total heap usage: 47,088,672 allocs, 47,088,672 frees.
- 3,080,231,299 bytes alocados o que equivale aproximadamente a 2.87Gb.

Como era expectável o número de alocações para os ficheiros de 3 e 5 milhões, é 3 e 5 vezes maior, respetivamente.

6 Makefile

```
IDIR =../include      Diretoria das includes

CC=gcc      Compilador a ser usado

CFLAGS=-O3 -ansi -D_GNU_SOURCE -Wall -g -I$(IDIR) Flags de compilação

ODIR=obj    Diretoria onde ficam os .o

_DEPS = vendas.h listas.h mytree.h fatura.h faturas.h filial.h view.h interface.h queryInterface.h macros.h interpretador.h    Todos os headers do projeto
DEPS = $(patsubst %,$(IDIR)/%,$(_DEPS)) Todos os heards decritos estão na dirétoria include


_OBJ = main.o listas.o vendas.o mytree.o fatura.o faturas.o filial.o view.o interface.o queryInterface.o interpretador.o Todos os obj que os .c geram
OBJ = $(patsubst %,$(ODIR)/%,$(_OBJ))   Todos os obj descritos estão na dirétoria obj


$(ODIR)/%.o: %.c $(DEPS)
    [ -d $(ODIR) ] || mkdir -p $(ODIR) Cria a diretoria obj caso não exista
    $(CC) -c -o $@ $< $(CFLAGS)  Cria os objetos mecionados anteriormente na pasta Obj



program: $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS)  Gera o executável com todos os .o na pasta obj e com as flags que foram mencionadas
    doxygen Doxyfile  Gera a documentação


clean:
    rm program  Remove o executavel
    rm -rf ../docs/html
    rm -rf ../docs/latex    Remove a documentação
    rm -f $(ODIR)/*.o *~ core $(INCDIR)/*~ Remove os objetos na diretoria obj
```

7 Conclusão

Para concluir, conseguimos cumprir todos os requisitos propostos, conseguindo implementar todos os módulos e estruturá-los como pedido, mantendo o encapsulamento dos dados e a modularidade, sendo assim capaz de responder a todas as *queries* da forma que nos pareceu mais eficiente. Inicialmente tivemos como ideia utilizar como base para o projeto a estrutura de AVL balanceadas, pois a nosso ver era uma estrutura que permitiria um rápido acesso dos dados, o que foi o caso. No entanto, após analisármos os tempos de execução e a quantidade de memória utilizada, reparamos que o tempo de load dos ficheiros de venda era relativamente maior ao que estavamos à espera, especialmente nos ficheiros de 3 e 5 milhões, e também que a quantidade de memória usada é muito maior do que aquilo que estávamos à espera chegando a ultrapassar as 2Gb de memória no caso dos ficheiro de 5 milhões de vendas.

Apesar de tudo, os resultados que obtemos foram satisfatórios e com a realização deste projeto adquirimos novos conhecimentos sobre técnicas de modulação e encapsulamento de dados, e também sobre o quão importante é organizar uma grande quantidade de dados em estruturas adequadas.