

ProjetoC-LI 3-Grupo 24-MIEI

João Bernardo Freitas A74814
Alexandre Martins A77523

Abril 2020



1 Abstract

Este projeto tem como principal objetivo a criação de um sistema de gestão de vendas que permita, através de uma lista de produtos, clientes e vendas, extrair informação útil.

Assim, este relatório pretende sumarizar todos os esforços efetuados para alcançar o objetivo proposto.

Contents

1	Abstract	1
2	Introdução	3
3	Descrição do Problema	3
4	Estruturas de Dados	4
4.1	Clientes	4
4.2	Produtos	4
4.3	Vendas	5
4.3.1	Faturas	5
4.3.2	Filiais	5
4.4	SGV	6
5	Arquitetura	7
6	Otimizações	9
7	Resultados	9
8	Conclusão	15

2 Introdução

Este trabalho foi-nos proposto no âmbito da unidade curricular de Laboratórios de Informática III, com o objetivo de, através da consulta rápida de uma quantidade enorme de dados conseguirmos responder às queries fornecidas pelos docentes.

Assim, neste relatório, vamos descrever todos os passos dados para alcançar esse objetivo.

3 Descrição do Problema

O objetivo geral desta fase do projeto é o desenvolvimento de um sistema de gestão de vendas que nos permita responder a uma série de queries através de 3 ficheiros fornecidos pelos docentes, como essas respostas têm de ser o mais rápido possíveis teremos também de usar estruturas de dados para facilitar a escrita e leitura dos dados.

4 Estruturas de Dados

As *estruturas de dados* são um modo de armazenamento e organização de dados que podem ser usadas de forma eficiente, facilitando a procura e modificação delas mesmas.

Nesse sentido, e tendo em conta os requisitos deste trabalho, decidimos utilizar a biblioteca do sistema **GLib**, que inclui uma série de estruturas já implementadas. De entre todas as estruturas oferecidas decidimos utilizar as **GTree(Balanced Binary Tree)**, **GArray** e **GHashTable**, sendo que estas serão utilizadas em módulos diferentes dependendo do tipo de operações que esperamos que sejam efetuados neles.

Estas estruturas de dados irão ser populadas através dos ficheiros **Vendas_XM.txt**, **Clientes.txt** e **Produtos.txt**, fornecidos pelos docentes.

4.1 Clientes

Os clientes são definidos por uma letra maiúscula seguido de quatro números, por isso, para o catálogo de clientes decidimos utilizar uma lista de **GTrees**, uma para cada letra, onde essa letra irá servir para indicar em que árvore procurar um cliente.

Esta solução permite-nos focar só na **GTree** que poderá incluir um certo cliente, sendo que a procura, adição e remoção dos nós é $O(\log n)$.

```
struct clientes
{
    GTree *clis[26];
};
typedef struct clientes *Cat_Clientes;
```

4.2 Produtos

Os produtos são semelhantes aos clientes, sendo que a única diferença está no facto de ter duas letras maiúsculas em vez de apenas uma.

Como tal decidimos aplicar uma solução semelhante aos clientes, tendo assim um vetor de **GTrees**, uma para cada letra.

```
struct produtos
{
    GTree *prods[26];
};
typedef struct produtos *Cat_Produtos;
```

4.3 Vendas

Os ficheiros de input referentes às vendas incluem um número elevado de informação, sendo que nem toda essa informação poderá ser relevante para todas as queries, como tal decidimos criar duas estruturas distintas, **Filiais** e **Faturas**, que nos irão permitir otimizar a procura de dados dependendo da query.

4.3.1 Faturas

A estrutura fatura é uma estrutura que está mais focada na venda de um certo produto, como tal, inclui toda a informação relevante a essa venda e um identificador. Essa estrutura é depois guardada numa **GTree**, onde mais uma vez todas as operações são em **O(log n)**.

```
struct fat{
    gint id;
    Produto prod;
    Cliente cli;
    int mes;
    char tipo;
    int quant;
    double preco;
    int filial;
};
typedef struct fat *Fatura;

struct faturas{
    GTree* fat;
};
typedef struct faturas *Cat_Faturas;
```

4.3.2 Filiais

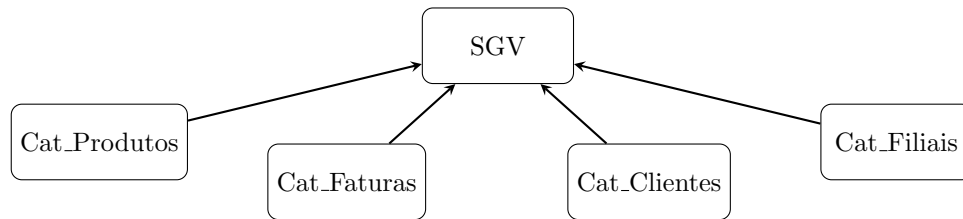
Esta estrutura foi criada de forma a responder rapidamente a todas as queries que focam nas filiais, por isso decidimos usar duas **Hashtables** por filial onde uma irá conter todas as informações relativas às compras de um certo produto numa filial e outra irá conter as informações das compras efetuadas por um cliente numa filial.

```
struct filial
{
    GHashTable *cliente[3];
    GHashTable *produto[3];
};
typedef struct filial *Cat_Filial;
```

4.4 SGV

A estrutura SGV é a estrutura que contém todas as anteriores estruturas, sendo que é também a estrutura que é utilizada em todas as queries.

```
struct sistemagestao
{
    Cat_Produtos produtos;
    Cat_Clientes clientes;
    Cat_Faturas faturas;
    Cat_Filial filiais;
};
typedef struct sistemagestao SGV;
```

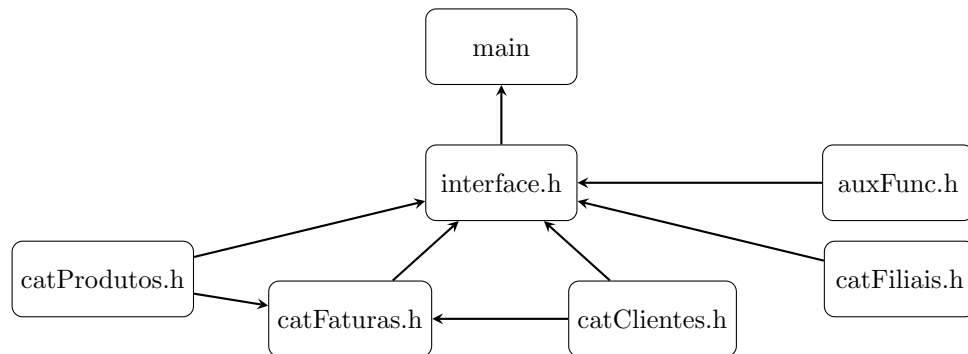


5 Arquitetura

Tendo em conta o tamanho do projeto era necessário implementar modularidade, ou seja, criar uma série de módulos que trabalhem em conjunto para atingir os objetivos propostos.

Como tal decidimos criar 7 módulos diferentes.

- **main**- Módulo principal do nosso programa, cria o menu e chama as funções das queries
- **interface.h**- Módulo central, neste módulo estão definidas todas as queries, é a partir deste módulo que se obtém acesso aos restantes módulos.
- **catClientes.h**- Módulo onde estão definidas todas as funções auxiliares relacionadas com o catálogo de clientes.
- **catProdutos.h**- Módulo onde estão definidas todas as funções auxiliares relacionadas com o catálogo de produtos.
- **catFaturas.h**- Módulo onde estão definidas todas as funções auxiliares relacionadas com o catálogo de faturas.
- **catFiliais.h**- Módulo onde estão definidas todas as funções auxiliares relacionadas com o catálogo de filiais.
- **auxFunc.h**- Módulo que contém diversas funções auxiliares necessárias para o funcionamento da interface



Esta modulação para além de facilitar a leitura permite-nos alterar um módulo sem ser necessário alterar os outros.

Para compilar o programa criamos a seguinte makefile.

```
CC = gcc
LIBS = 'pkg-config --libs --cflags glib-2.0'
FLAGS= -w -Wall -Wextra -ansi -std=c11 -O2 -D_GNU_SOURCE
INCLUDES=include
SRC=src
OBJ=obj
program:
    mkdir obj
    make produtos
    make clientes
    make faturas
    make filiais
    make auxiliar
    make interface
    make prog
    make allprog
    rm obj/*.o
    rmdir obj
    ./program
produtos: $(SRC)/catProdutos.c $(INCLUDES)/catProdutos.h
    $(CC) $(FLAGS) -o $(OBJ)/catProdutos.o $(LIBS) -c $(SRC)/catProdutos.c

clientes: $(SRC)/catClientes.c $(INCLUDES)/catClientes.h
    $(CC) $(FLAGS) -o $(OBJ)/catClientes.o $(LIBS) -c $(SRC)/catClientes.c

faturas: $(SRC)/catFaturas.c $(INCLUDES)/catFaturas.h
    $(CC) $(FLAGS) -o $(OBJ)/catFaturas.o $(LIBS) -c $(SRC)/catFaturas.c

filiais: $(SRC)/catFiliais.c $(INCLUDES)/catFiliais.h
    $(CC) $(FLAGS) -o $(OBJ)/catFiliais.o $(LIBS) -c $(SRC)/catFiliais.c

auxiliar: $(SRC)/auxFunc.c $(INCLUDES)/auxFunc.h
    $(CC) $(FLAGS) -o $(OBJ)/auxFunc.o $(LIBS) -c $(SRC)/auxFunc.c

interface: $(SRC)/interface.c $(INCLUDES)/interface.h
    $(CC) $(FLAGS) -o $(OBJ)/interface.o $(LIBS) -c $(SRC)/interface.c

prog: $(SRC)/main.c $(INCLUDES)/catClientes.h $(INCLUDES)/catProdutos.h
    $(INCLUDES)/catFaturas.h $(INCLUDES)/catFiliais.h
    $(INCLUDES)/auxFunc.h $(INCLUDES)/interface.h
    $(CC) $(FLAGS) -o $(OBJ)/main.o $(LIBS) -c $(SRC)/main.c
```



```

allprog: $(OBJ)/main.o $(OBJ)/catProdutos.o $(OBJ)/catClientes.o
         $(CC) $(FLAGS) -o program $(OBJ)/main.o $(OBJ)/catProdutos.o
         $(OBJ)/catClientes.o $(OBJ)/catFaturas.o $(OBJ)/catFiliais.o
         $(OBJ)/auxFunc.o $(OBJ)/interface.o $(LIBS)
memcheck:
         valgrind --leak-check=yes ./programa
clean:
         rm -rf obj
         rm program

```

6 Otimizações

No início do desenvolvimento do nosso sistema tínhamos duas estruturas semelhantes ao que apresentamos no final, tanto para os produtos como para os clientes. No entanto, em vez de termos uma árvore binária para cada letra do alfabeto tínhamos um array de arrays, onde, tal como nas árvores, a letra inicial representava o array que continha todos os produtos/clientes começados por essa letra. Após alguns testes percebemos que era pouco eficaz, e ao trocar esta estrutura por uma semelhante com árvores passamos de aproximadamente 50 segundos de leitura para 6.

7 Resultados

Ao executar a *makefile* o programa, e seus módulos, irão ser compilados sendo que no fim a *makefile* irá executar o binário automaticamente. O grupo decidiu que os nomes dos ficheiros input iriam ser fornecidos por omissão, ou seja estão hardcoded.

```

char clientes[13] = "Clientes.txt";
char produtos[13] = "Produtos.txt";
char vendas[14] = "Vendas_1M.txt";
sistema = loadSGVFromFiles(sistema, clientes, produtos, vendas);

```

Figure 1: Leitura HardCoded

Após leitura dos ficheiros irá ser apresentado um menu onde podemos escolher a query que queremos executar através do número que a precede.

```

Numero de produtos válidos 171008
Numero de clientes válidos 16384
Total de vendas 1000000
Vendas válidas 891108
Vendas inválidas 108892
Tempo de Leitura: 7.393602
-----
Escolha a Querie
1:Determinar a lista e o nº total de produtos cujo código se inicia por uma dada letra (maiúscula)
2:Dado um mês e um código de produto, ambos válidos, determinar e apresentar o número total de vendas e o total facturado com esse produto em tal mês
3:Determinar a lista ordenada dos códigos dos produtos que ninguém comprou
4:Determinar a lista ordenada de códigos de clientes que realizaram compras em todas as filiais
5:Determinar o número de clientes registados que não realizaram compras bem como o número de produtos que ninguém comprou.
6:Dado um código de cliente, criar uma tabela com o número total de produtos comprados, mês a mês.
7:Dado um intervalo fechado de meses determinar o total de vendas registadas nesse intervalo e o total faturado
8:Dado um código de produto e uma filial, determinar os códigos dos clientes que o compraram, distinguindo entre compra N e compra P
9:Dado um código de cliente e um mês, determinar a lista de códigos de produtos que mais comprou por quantidade, por ordem decrescente
10:Criar uma lista dos N produtos mais vendidos em todo o ano, indicando o número total de clientes e o número de unidades vendidas, filial a filial
11:Dado um código de cliente determinar quais os códigos dos N produtos em que mais gastou dinheiro durante o ano
12:Estatísticas de leitura
13:Saír

```

Figure 2: Menu Principal

Algumas queries poderão precisar de mais inputs, sendo que, nesse caso, o programa irá pedir ao utilizador para os fornecer.
No fim da execução de cada query também é indicado o tempo que essa mesma demorou a executar.
Seguem exemplos da execução de todas as queries.

```

AZ1932
AZ1933
AZ1934
AZ1937
AZ1942
AZ1943
AZ1946
AZ1948
AZ1960
AZ1962
AZ1965
AZ1971
AZ1974
AZ1975
AZ1977
AZ1983
AZ1985
AZ1986
AZ1987
AZ1994
Total de produtos começados por A: 6578
Execução Query 1: 0.017216
-----

```

Figure 3: Execução da query 1

```

2
-----
Inserir código de produto:
KR1583
Inserir mes:
2
Indique a filial - 0 para geral
0
ID: 0
Produto: KR1583
Cliente: 14891
Mes: 2
Tipo: P
Quantidade: 128
Preço: 77.72
Filial: 1
Compra em P
Total Faturado em cada Filial em P
Filial 1: 77.720000
Filial 2: 0.000000
Filial 2: 0.000000
Total Faturado em cada Filial em N
Filial 1: 0.000000
Filial 2: 0.000000
Filial 2: 0.000000
Total Vendido em cada Filial em P
Filial 1: 1.000000
Filial 2: 0.000000
Filial 2: 0.000000
Total Vendido em cada Filial em N
Filial 1: 0.000000
Filial 2: 0.000000
Filial 2: 0.000000
Total Vendido em N
0.000000
Total Vendido em P
1.000000
Total Faturado em N
0.000000
Total Faturado em P
77.720000
Execução Query 2: 0.046050
-----

```

Figure 4: Execução da query 2

```

ZZ1626
ZZ1631
ZZ1638
ZZ1688
ZZ1713
ZZ1717
ZZ1751
ZZ1758
ZZ1768
ZZ1805
ZZ1810
ZZ1821
ZZ1854
ZZ1875
ZZ1945
ZZ1948
ZZ1975
ZZ1985
ZZ1986
ZZ1990
Numero total = 30205
Execução Query 3: 0.269226

```

Figure 5: Execução da query 3

```

P3398
L4416
B1053
N1881
Q2749
X4169
T4425
U4177
D3648
K3550
H1039
T3527
Y2644
Q3535
S4320
Y2645
Z3296
J2901
F4435
C4442
Q3536
B1056
T1868
I2766
D1989
T3528
V4313
Execução Query 4: 0.041585
-----

```

Figure 6: Execução da query 4

```

5
-----
Clientes sem compras: 0
Produtos que nunca foram comprados: 928
Execução Query 5: 0.062448
-----

```

Figure 7: Execução da query 5

```

6
-----
Introduza o código de cliente:
L4891
+++++
++M  --  F1  --  F2  --  F3  ++
++1  --  159 --    0 --  132++
++2  --  172 --   77 --  163++
++3  --  180 --  202 --  196++
++4  --  112 --  267 --  182++
++5  --  183 --    0 --   93++
++6  --  180 --  127 --  150++
++7  --   19 --   90 --  190++
++8  --   33 --  161 --  275++
++9  --    0 --  175 --  325++
++10 --    0 --  195 --  128++
++11 --   47 --    0 --   94++
++12 --    0 --  268 --  372++
+++++
Execução Query 6: 0.000068
-----

```

Figure 8: Execução da query 6

```

7
-----
Introduzir primeiro mes:
1
Introduzir segundo mes:
4
Número de vendas: 297013
Total de vendas: 445562999.900196
Execução Query 7: 0.062505
-----

```

Figure 9: Execução da query 7

```

8
-----
Inserir código de produto:
KR1583
Inserir id da Filial:
1
Clientes que compraram em P:
L4891
R4193
Clientes que compraram em N:
Execução Query 8: 0.000104
-----

```

Figure 10: Execução da query 8

```

9
-----
Introduza o cliente
L4891
Introduza o mês
2
Produto->Quantidade
OD1288->77
NR1347->94
ZA1174->69
XC1676->44
KR1583->128
Execução Query 9: 0.000097
-----

```

Figure 11: Execução da query 9

```

-----
Inserir código de cliente:
L4891
Inserir limite:
10
GC1158
DY1402
AP1641
SP1467
NR1347
KR1583
QN1585
GR1361
LW1118
UP1448
Execução Query 11: 0.000093
-----

```

Figure 12: Execução da query 11

```

12
-----
Clientes.txt -> Linhas Validas: 16384
Produtos.txt -> Linhas Validas: 171008
Vendas_1M.txt -> Linhas Validas: 891108
Execução Query 12: 0.000036
-----

```

Figure 13: Execução da query 12

Por fim verificamos os tempos de execução do programa com os diversos ficheiros de **vendas**.

```

Numero de produtos válidos 171008
Numero de clientes válidos 16384
Total de vendas 3000000
Vendas válidas 3000000
Vendas inválidas 0
Tempo de Leitura: 23.008905
-----

```

Figure 14: Ficheiro Vendas_3M.txt

```

./program
Numero de produtos válidos 171008
Numero de clientes válidos 16384
Total de vendas 5000000
Vendas válidas 5000000
Vendas inválidas 0
Tempo de Leitura: 38.189670
-----

```

Figure 15: Ficheiro Vendas_5M.txt

Como seria de esperar os tempos de leitura aumentaram consideravelmente.

8 Conclusão

De acordo com o objetivo do projeto, sendo este, o acesso rápido a um conjunto de dados elevados, podemos concluir que este foi concluído com sucesso. Tivemos no entanto um problema em específico com a função **g-array-sort**, que acabamos por não conseguir implementar corretamente, e devido a isso, 3 queries não ficaram solucionadas em pleno, faltando no fim ordenar a resposta. Acabamos no fim também por não conseguir resolver completamente a query número 11 devido ao a problemas semelhantes. No entanto, no que diz respeito à primeira fase foi satisfatoriamente resolvida pelo que o código necessário para o funcionamento das queries foi desenvolvido. O presente trabalho serviu para um aprimoramento das nossas aptidões relativamente à organização de dados para obter uma maior eficiência tanto a lê-los como a interpreta-los.