

**Laboratórios de Informática III (LI3)**

**Projeto de Java 2014/2015**

***GEREVENDAS***

Gestão de Produtos, Clientes e Compras de um Hipermercado

MIEI – 2º ano – 2º semestre

**GRUPO 84**

Beatriz Loureiro Márcia Costa José Santos Ruben Santos

A68876 A67672 A67687 A70644

Índice

Introdução…………………………………………………………………………………………. 2

Descrição dos módulos....................................................................................................... 3

Catálogo de Clientes........................................................................................................... 4

Catálogo de Produtos.......................................................................................................... 5

Compras.............................................................................................................................. 6

Contabilidade……………………………………………………………………………………… 8

Main.c………………............................................................................................................10

Interface do utilizador..........................................................................................................10

Resultados e comentários sobre os testes de performance……....................................... 11

Makefile.............................................................................................................................. 13

Grafo de dependências...................................................................................................... 13

Conclusão………………………………………………………………………………………… 14

Introdução

No âmbito da unidade curricular de Laboratórios de Informática III do 2ºano da licenciatura de Engenharia Informática foi proposto o desenvolvimento de um projecto em linguagem C que tem por objectivo fundamental 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.

Este projeto considera-se um grande desafio para nós pelo facto de passarmos a realizar programação em grande escala, uma vez que se trata 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 protecção e acessos controlados aos dados).

# Descrição dos módulos

A arquitetura do software é definida por quatro módulos principais: Catálogo de produtos, Catálogo de clientes, Contabilidade e Compras, cujas fontes de dados são os três ficheiros de texto FichProdutos.txt, FichClientes.txt e Compras.txt, e uma interface que permita a comunicação com o cliente.

Gesthiper

FichProdutos.txt

FichClientes.txt

Compras.txt

**CatálogoClientes**

**CatálogoProdutos**

**Contabilidade**

**Compras**

O ficheiro FichProdutos.txt contem 200.000 códigos de produtos. Cada linha representa o código de um produto do hipermercado onde cada código é formado por duas letras maiúsculas seguidas de quatro dígitos.

O ficheiro FichClientes.txt contem 20.000 códigos de clientes. Em cada linha de código existe um cliente identificado do hipermercado. Cada código é formado por duas letras maiúsculas e três dígitos.

O ficheiro Compras.txt contem 500.000 registos de compras e é a nossa maior fonte de dados. Cada linha representa o registo de uma compra efetuada no hipermercado. Cada código é formado por: código de produto, preço do produto unitário decimal, o número de unidades compradas, a letra N ou a letra P, o código do cliente que lhe corresponde e o mês em que foi efetuada a compra.

# Catálogo de Clientes

Módulo de dados onde são guardados os códigos de todos os clientes do ficheiro FichClientes.txt. O array de árvores (**struct avl\_table \*\*array\_arv**) é um array de 26 posições cujos índices se encontram organizados alfabeticamente. Cada índice contem um apontador para uma árvore correspondente á letra respetiva desse índice.

0

1

2

3

…

\*

* **Clientes.h**

**Tipos Opacos**

**typedef struct clientes \*Clientes**

**/\*API\*/**

* **Clientes create\_clientes():** Função que cria uma estrutura de clientes vazia.
* **Clientes insere\_clientes(Clientes c, char \*s):** Função que insere um cliente à estrutura de clientes.
* **int cliente\_cmp(const void\*a,const void \*b, void\*p):** Função de comparação para o uso da biblioteca AVL.
* **int cliente\_valida(Clientes c, char \*s):** Função que verifica se um dado cliente está na estrutura de clientes.
* **int total\_clientes(Clientes c):** Função que retorna o total de clientes inseridos num array de árvores.
* **int conta\_clientes(Clientes c, char ch):** Função que determina o número de clientes iniciados por uma dada letra.
* **struct avl\_table \*avl\_cli(Clientes c):** Função que retorna o elemento struct avl\_table \*lcli pertencente à estrutura de clientes.
* **char \*\*letra\_cli(Clientes cl, char c):** Função pertencente à querie 6, que recebe uma dada letra e retorna um array ordenado de clientes dessa mesma letra.
* **int letra\_cli\_total(Clientes cl, char c):** Função auxiliar à querie 6 que devolve o número de clientes que começam pela letra dada.

# Catálogo de Produtos

Módulo de dados onde são guardados os códigos de todos os produtos do ficheiro FichProdutos.txt. Os códigos encontram-se organizados alfabeticamente, permitindo desta forma, saber quais são os produtos que se iniciam por uma dada letra, ou então saber quantos são. As informações acima descritas encontram-se respectivamente na **struct avl\_table \*\*array\_arv** e no array **contador**. Relativamente à **struct avl\_table \*listprod,** esta é uma árvore que irá conter todos os produtos do ficheiro FichProdutos.txt. Isto servirá mais tarde para uma mais eficiente realização de algumas queries.

0

1

2

3

…

\*

* **Produtos.h**

**Tipos Opacos**

**typedef struct produtos \*Produtos**

**/\*API\*/**

* Produtos create\_produtos(): Função que cria uma estrutura de produtos vazia.
* Produtos insere\_produtos(Produtos p, char \*s): Função que insere um produto à estrutura de produtos.
* int prod\_valida(Produtos p, char \*s): Função que verifica se um dado produto está na estrutura de produtos.
* int produtos\_cmp(const void\*a,const void \*b, void\*p): Função de comparação para o uso da biblioteca AVL.
* int total\_produtos(Produtos p): Função que retorna o total de produtos inseridos num array de árvores.
* int conta\_produtos(Produtos p, char ch): Função que determina o número de produtos iniciados por uma dada letra.
* char \*\*letra\_prod(Produtos p, char c): Função pertencente à querie 2, que recebe uma dada letra e devolve um array ordenado de produtos dessa mesma letra.
* int letra\_prod\_total(Produtos p, char c): Função auxiliar à querie 2 que devolve o número de produtos que começam pela dada letra.
* char\* mystrdup(const char\* s): Função que substitui o strdup.
* struct avl\_table \*listaprodutos(Produtos p): Função que devolve elemento listprod da estrutura de produtos.

# Compras

Estrutura de compras que contem cada um dos elementos de uma linha do ficheiro compras.txt.

Array de apontadores para estruturas, que guarda todas as compras do ficheiro Compras.txt.

Array de 12 posições correspondente ao número de meses do ano, em que cada índice corresponde a um apontador para uma árvore que contem todos os clientes que fizeram compras nesse mês. Este array foi criado para uma melhor eficiência da querie 10(Cli\_Todos\_Meses).

Árvore que contém informação de cada produto do ficheiro Compras.txt onde essa informação é a quantidade dos produtos comprados e uma árvore com todos os clientes que compraram esse mesmo produto. Esta árvore foi-nos útil para a execução da querie 12( produtos\_por\_venda).

Árvore ListaProdNaoComprados e ListaClientesSemCompras contêm respetivamente os produtos que nunca foram comprados e os clientes que nunca efetuaram uma compra. Estas árvores foram criadas para a execução da querie 14(prod\_n\_comprados e cliente\_n\_compraram).

LClientes é uma árvore que contém todos os clientes. É usada tambem na querie10(Cli\_Todos\_Meses).

* **Compras.h**

**Tipos Opacos**

**typedef struct compras \*compra;**

**typedef struct data\_compras \*Datac;**

**Estruturas Públicas**

Estrutura definida para a querie 12 que contém um produto, uma árvore de clientes que compraram esse produto e o número de unidades vendidas.

Estrutura definida para a querie 13 que contém um produto e o número de unidades vendidas desse produto.

Estrutura definida para a queri8, com um array de clientes, um array com os tipos de compra(N e P), o numero total de compras em N, numero total de compras em P e total de compras

Estrutura definida para a querie 10, que contém um array dos clientes que compraram todos os meses e o tamanho desse array.

Estrutura definida para a querie 9 que contém um array de produtos, um array de inteiros que corresponde à quantidade comprada para cada produto e o total de elementos desses arrays.

Estrutura definida para a querie 11, que contém um array de arvores que correspondente a todos os clientes que aparecem nas compras, o número de compras de cada mês e o número de clientes que compram cada mês.

**/\*API\*/**

Datac insere\_compras(Datac d, char \*s, Clientes cl, Produtos p): insere uma compra num array de compras na estrutura Datac.

Datac create\_compras(): Função que inicializa a estrutura Datac e todos os seus elementos.

int \*Total\_Prod\_Comp\_Mes(char \*s, Datac d): Função pertencente à querie 5, que retorna um array de inteiros com o total de produtos comprados em cada mês por esse cliente.

int prod\_n\_comprados(Datac d): Função correspondente à querie 14, que retorna o número de produtos que não foram comprados.

int cliente\_n\_compraram(Datac d): Função correspondente à querie 14, que retorna o número de clientes que nunca efetuaram uma compra.

cli\_Prod \*Cli\_Prod(char \*s, Datac d): Função pertencente à querie 8, que dado um produto, retorna a estrutura chamada cli\_Prod que contém um array de clientes que compraram esse produto, um array com o tipo de compra que cada cliente efetuou desse produto, e três inteiros: número de compras em N, número de compras em P e total de compras.

prod\_mais\_Comp \*Prod\_Mais\_Comp(char \*s, int m, Datac d): Função pertencente à querie 9, que dado um mês e um cliente, retorna uma estrutura prod\_mais\_Comp que contém um array de produtos e de inteiros ordenados pela quantidade de produtos que o cliente comprou.

cli\_todos\_Meses \*Cli\_Todos\_Meses(Datac d): Função correspomdente à querie 10, que devolve uma estrutura que contém um array com todos os clientes que efetuaram compras todos os meses do ano.

total\_comp\_cli\_Mes \*Total\_Comp\_Cli\_Mes(Datac d): Função referente à querie 11, que devolve uma estrutura um array de inteiros com o número de compras efetuadas a cada mês e um array com o número de cliente que fizeram compras em cada mês.

ProdPorVenda \*\*produtos\_por\_venda(Datac d, int n): Função pertencente à querie 12, que dado um inteiro n, retorna um array de estruturas com tamanho n, do tipo ProdPorVenda, que contém determinado produto e suas informações como uma árvore dos clientes que o compraram e um inteiro que corresponde ao número de unidades vendidas desse produto.

tres\_prod\_mais\_Comp \*\*Tres\_Prod\_Mais\_Comp(char \*s, Datac d): Função correspondente à querie 13, que recebe um cliente, e devolve um array de estruturas do tipo tres\_prod\_mais que contém um produto e um inteiro corresponde ao número de unidades comprados desse produto pelo cliente. Os três primeiros elementos deste array encontram-se ordenados.

int linhascompras(Datac d): Função que devolve a informação do número de elementos lidos do ficheiro de compras.

int comprasvalidas(Datac d): Função que devolve o número de compras validadas.

int invalprod(Datac d): Função que devolve o número de produtos inválidos do ficheiro de compras.

int invalcli(Datac d): Função que devolve o número de clientes inválidos do ficheiro de compras.

Contabilidade

Árvore que contem todos os produtos do ficheiro FichProdutos.txt, que mais tarde conterá apenas os produtos não comprados tal como pedido na querie 4(Nao\_Comprados).

Total faturado de todas as compras válidas.

Array de apontadores para estruturas, que guarda todas as compras do ficheiro Compras.txt.

* **Contabilidade.h**

**Tipos Opacos**

**typedef struct contabilidade \*Contabilidade;**

**Estruturas Públicas**

Estrutura definida para a querie 3, em que são guardadas todas as informações relativas ao tipo de compra e o seu respectivo preço num dado mês.

Estrutura definida para a querie 4, em que o array produtos\_ncomprados contém os elementos remanescentes que não foram validados aquando da colocação em memória do ficheiro de compras, bem como o seu total.

Estrutura definida para a querie 7, em que são guardadas informações quanto ao total de compras efetuadas e a respectiva faturação total.

**/\*API\*/**

Contabilidade create\_cat\_contabil(Produtos p): Função que inicializa a estrutura Contabilidade e todos os seus elementos. Recebe como parâmetro a estrutura Produtos p pois é passado como argumento uma árvore com todos os produtos, útil para a realização eficiente da querie 4.

Contabilidade insere\_Compras(Contabilidade c, char \*s, Clientes cl, Produtos p): insere uma compra num array de compras na estrutura Contabilidade à semelhança do efetuado no módulo de compras.

modo\_Mes \*Modo\_Mes(Contabilidade c, int m, char \*ch): Função pertencente à querie 3, que retorna o total de compras, bem como total faturado (com distinção do tipo da compra) de um dado produto introduzido pelo utilizador num mês também definido pelo utilizador.

nao\_Comprados \*Nao\_Comprados(Contabilidade c): Função pertencente à querie 4, que retorna todos os produtos não comprados fazendo uso do elemento da estrutura Contabilidade que contém todos os produtos numa árvore à qual são removidos os produtos que foram validados.

intervalo\_Mes \*Intervalo\_Mes(Contabilidade c, int m1, int m2): Função pertencente à querie 7, que fornece informações acerca do total de compras efetuadas e a sua faturação total, num intervalo de meses introduzido pelo utilizador.

double facturacao\_anual(Contabilidade c): Função que retorna a faturação total de todas as compras no ficheiro de compras em uso, usada para apresentar as informações sobre o ficheiro na opção 15 do menu.

# Main.c

O programa é controlado pelo ficheiro main.c. A função main() invoca funções auxiliares como a função *carregarficheiros* que carrega os ficheiros de compras, produtos e clientes em variáveis do tipo FILE. Depois, são usadas funções derivadas dos vários módulos para carregar os ficheiros em memória nas devidas estruturas. Após a colocação em memória, é chamada a função auxiliar *interface*, que servirá para selecionar a querie que o utilizador pretende executar. Escolhida a queire, com o auxílio de um switch, temos acesso a funções também contidas no main.c chamadas *printQn* (sendo n o número da queire escolhida) que invocam a função que faz exatamente o que a querie pede e mostra ao utilizador os resultados pretendidos.

# Interface do utilizador

11198759_853994978021493_370523283_n.jpg

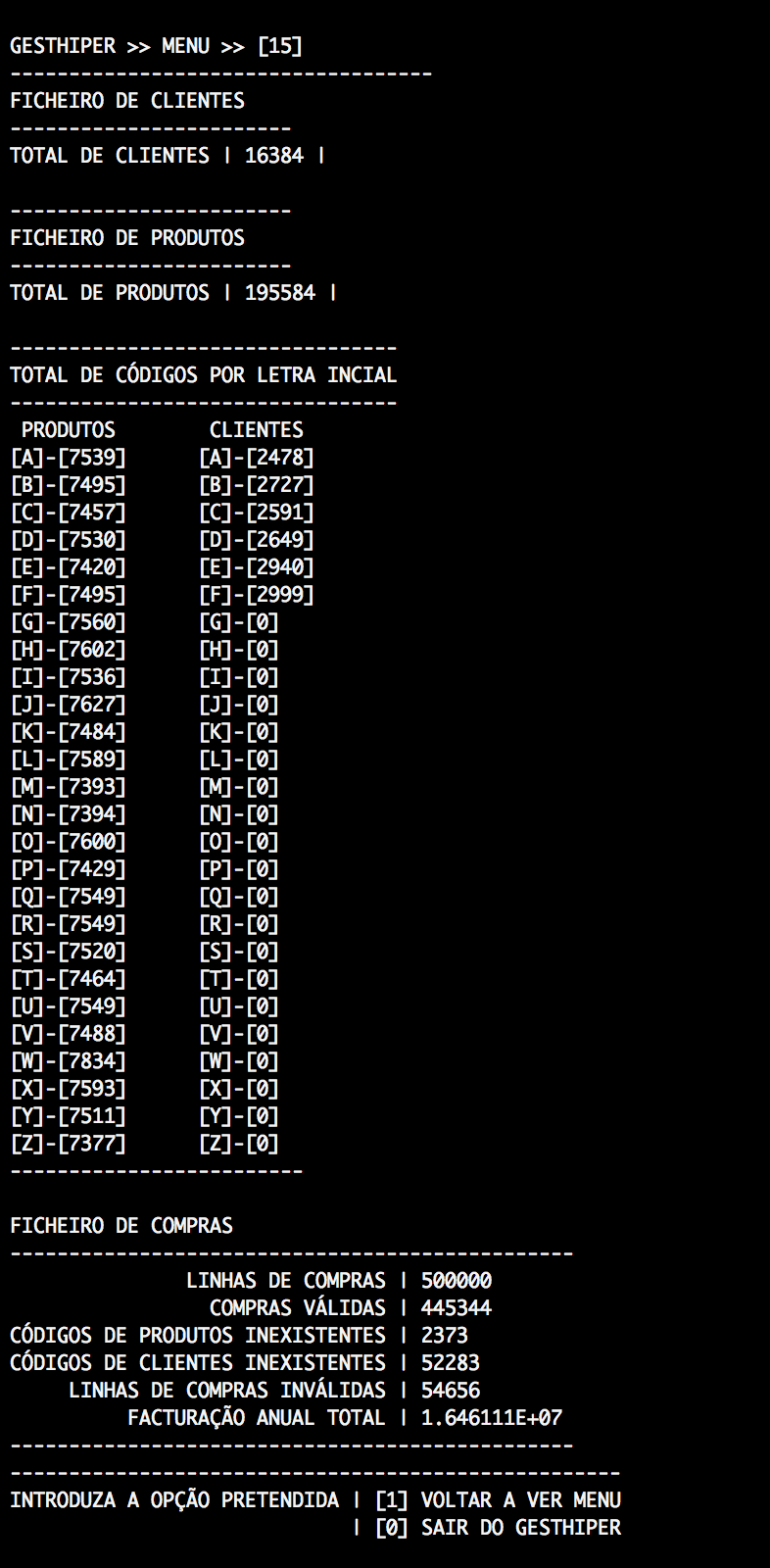
Quando o utilizador executa o programa é-lhe pedido que escolha qual o documento de texto que pretende analisar, como podemos observar na figura à direita.

O ficheiro é carregado e de seguida aparece um menú com 15 opções. As 14 primeiras referentes às 14 queries do projeto, sendo que decidimos usar o [0] para sair do GESTHIPER e o [1] para voltar a ver o menu inicial. O objetivo é que o utilizador prima a tecla correspondente à opção do menu pretendida.



A última opção, a [15], contém informações sobre os ficheiros (compras, clientes e produtos) indicando o nome do ficheiro lido, o total de clientes, total de produtos e total de códigos por letra inicial, assim como o total de códigos de produtos e clientes inexistentes. Relativamente às compras temos ainda as linhas de compras válidas e inválidas e a faturação anual total.

Quando o resultado da querie é demasiado extenso (como por exemplo na querie 6 em que são devolvidos milhares de códigos de clientes) decidimos não apresentar ao utilizador os resultados de uma vez só. Tomamos a opção de apresentar os primeiros 20 resultados e permitir ao utilizador navegar com a opção [+] e [-] podendo visualizar assim a próxima página ou a anterior.

11166001_853991801355144_569899418_n.jpg

# 

# 

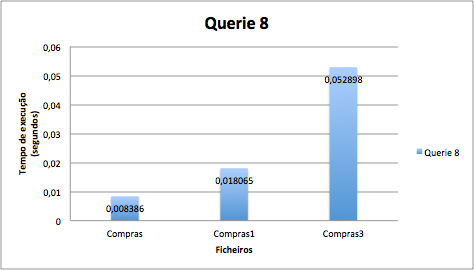
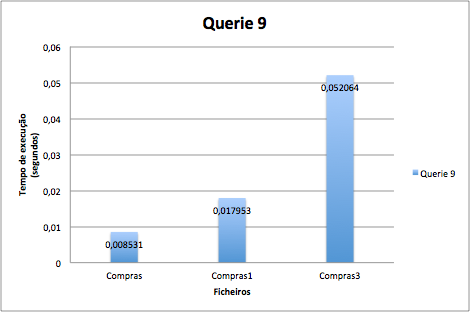
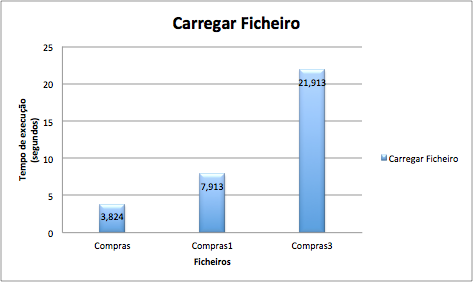
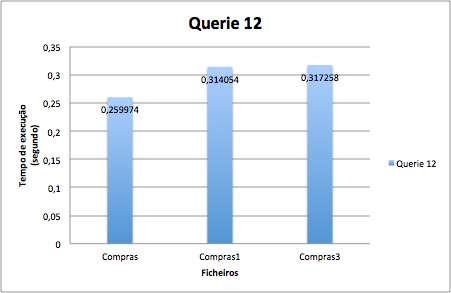
# 

# Resultados e comentários sobre os testes de performance

Depois de desenvolver e codificar todo o projecto foi-nos proposto realizar alguns testes de performance que consistem em comparar os tempos de execução das querias 8, 9 e 12 usando os ficheiros Compras.txt (500 000 compras), Compras1.txt (1 milhão de compras) e Compras3.txt (3 milhões de compras).

Uma vez que a quantidade de compras vai aumentando de ficheiro para ficheiro é aceitável que os tempos de execução para os carregar, ou seja, a fase de leitura e inserção também aumente significativamente.

Comparando os valores de execução das queries pretendidas, como podemos observar nos respetivos gráficos apresentados, o tempo de execução não varia significativamente, uma vez que são variações na ordem das centésimas de segundo.

*Nota: Na querie 8 usamos o seguinte código produto: ZE6026 e na querie 9 usamos o código de cliente: AF202.*

# 

# 

# 

# Makefile

A makefile, ficheiro contido no “.zip”, 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.

11198923_854001314687526_1442570_n.jpg

# Grafo de dependências

11198542_10203991618220703_1237801650_n.jpg

# Conclusão

O desenvolvimento deste projeto contribuiu para que pudéssemos aplicar os conhecimentos que tínhamos apreendido nas diversas unidades curriculares e também a colmatar algumas dificuldades que sentíamos.

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.

Inicialmente, tivemos dificuldades nas AVLs, visto que não tínhamos muita prática quanto à sua implementação. Depois de alguns problemas com o seu balanceamento, acabamos por apostar na utilização da biblioteca standard AVL da GNU, que nos facilitou não só o carregamento dos ficheiros em memória, mas também na realização de algumas queries, devido ao vasto conjunto de úteis funções que a biblioteca contém.

A querie que para nós foi mais complicado realizar foi a 12 em que inicialmente optamos por guardar a informação de cada produto num array. Esta implementação não se mostrou eficiente, visto que se trata de muita informação a ser executada, penalizando o tempo de execução em +/- 15 minutos. A solução para este problema foi a criação de uma árvore de estruturas que contém o produto e quantidade vendida e uma árvore com os clientes que compraram esse produto. O tempo de execução tornou-se então razoável: 0,25 segundos para o Compras.txt. Contudo conseguimos responder de forma eficiente a todas as queries.

Assim sendo, tornou­-se realmente um grande desafio e por isso bastante mais gratificante pois deparamo-­nos com alguns obstáculos que conseguimos ultrapassar adquirindo assim uma maior experiência no que toca ao uso desta linguagem.