
Sistema de Gestão de Vendas

TRABALHO REALIZADO POR:

JOÃO FIGUEIREDO MARTINS PEIXE DOS SANTOS

LUÍS FILIPE CRUZ SOBRAL

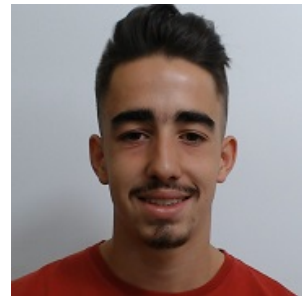
PAULO SILVA SOUSA



A89520
João Santos



A89474
Luís Sobral



A89465
Paulo Sousa

GRUPO 70
PROJETO LI3
2019/2020
UNIVERSIDADE DO MINHO

Índice

1	Introdução	1
2	Escolha da Solução Adotada	1
3	Estruturas de Dados	2
3.1	Catalogo	2
3.2	THashFact	2
3.3	GFiliais	3
3.4	SGV	3
4	Interação com o Utilizador	4
5	Estrutura do Projeto e Grafo de Dependências	5
6	Testes de Performance	6
6.1	Tempos de Execução	6
6.2	Libertação de Memória	8
7	Makefile	9
8	Conclusão e Reflexão Crítica	10

1 Introdução

Nesta unidade curricular foi-nos proposta a implementação de um sistema de resposta a queries sobre um Sistema de Gestão de Vendas - **SGV**.

Inicialmente, o projeto consistiu no desafio de implementar este sistema na linguagem C. Apesar da intenção de tornar rápida a execução das funcionalidades do programa a ser desenvolvido, 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 a implementação das estruturas onde seria organizada a informação, de forma a que o seu acesso fosse rápido e eficiente. Além disso, também considerámos um desafio a libertação da memória total do programa.

2 Escolha da Solução Adotada

Inicialmente, a estrutura que optamos por utilizar foi arrays ordenados com procura binária para todas as estruturas. Com a necessidade de melhorar a performance do programa, tivemos de começar a pensar em implementar estruturas de dados mais avançadas.

A opção de implementar AVL's foi logo posta de parte, porque tem o mesmo tempo de procura que a procura binária em arrays, tendo um pior tempo de inserção.

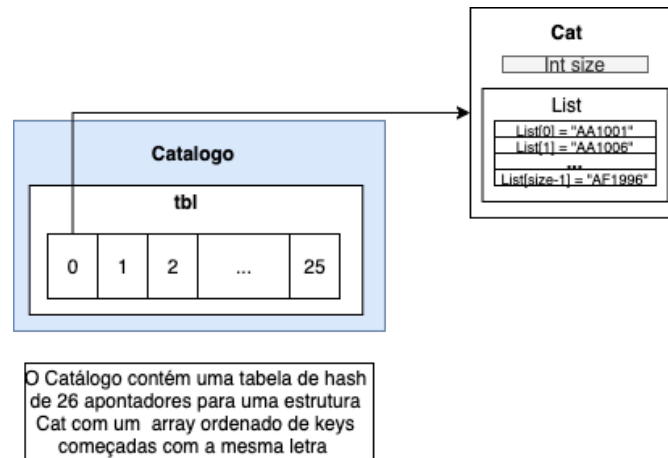
A opção que nos pareceu mais viável, foi a implementação de uma **Tabela de Hash** de **26** posições que em cada posição tinha um array ordenado.

Esta implementação permite-nos aceder a um produto/cliente em tempo logarítmico em função do número de elementos começados por um char, em vez do número total de elementos da estrutura.

No início, sempre que adicionávamos um produto/cliente ao array, executávamos o algoritmo BubbleSort para ordenar o array, sendo a complexidade temporal $\mathcal{O}(n^2)$. De forma a melhorar o tempo de ordenação do array, decidimos utilizar o algoritmo Quick-Sort após preencher o array todo, reduzindo a complexidade temporal para $\mathcal{O}(n \log n)$.

3 Estruturas de Dados

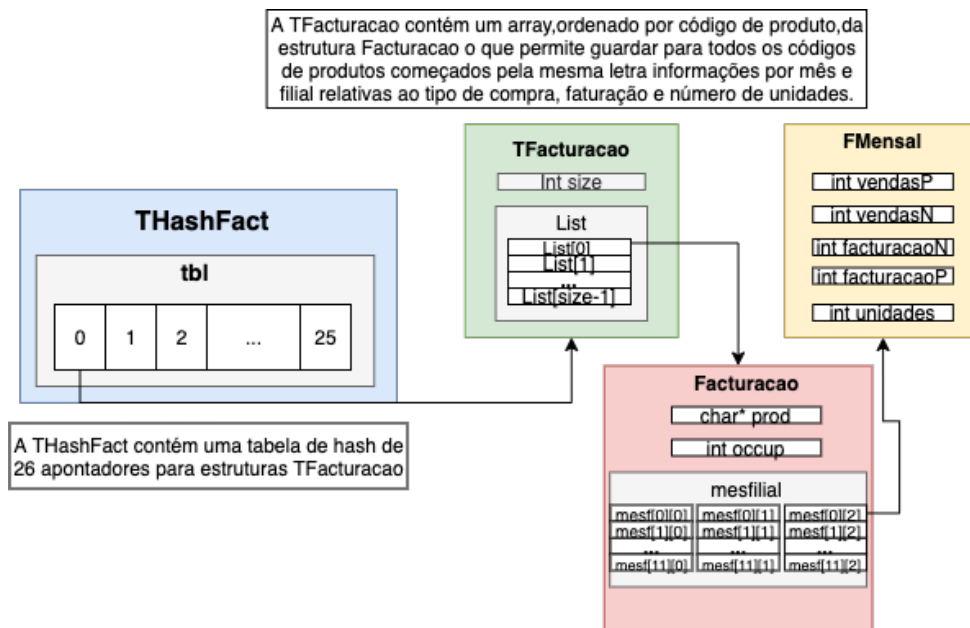
3.1 Catalogo



Esta é uma estrutura genérica utilizada para guardar os códigos de clientes ou produtos (aos quais chamámos Key) dos dados fornecidos pelo utilizador.

A função de Hash utilizada, recebe o primeiro carater da Key e devolve a correspondente posição na tabela.

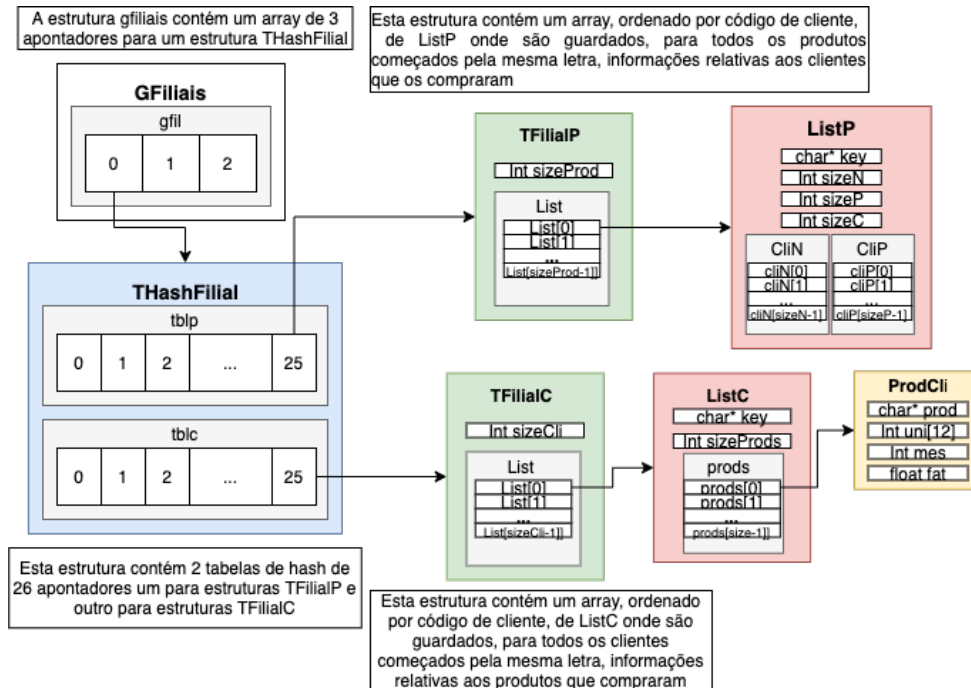
3.2 THashFact



Esta estrutura é utilizada para armazenar as informações relativas à faturação de todas as vendas.

Nessa estrutura, temos o produto e uma matriz de 12 por 3 da estrutura MesFilial, onde guardamos para um determinado mês e uma determinada Filial, o número de Vendas (N e P), o total faturado (N e P) e o número de unidades vendidas.

3.3 GFiliais

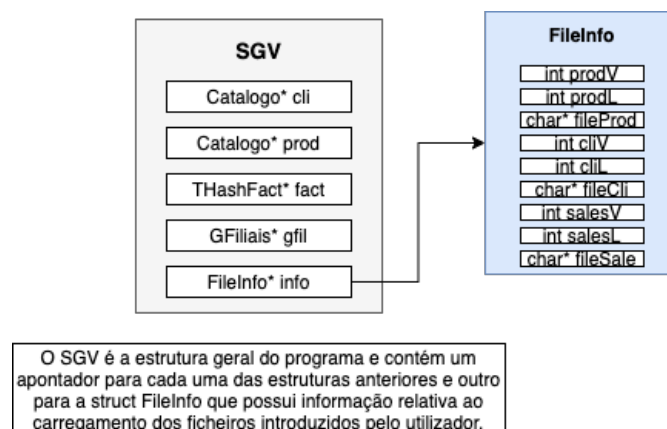


Esta é a estrutura utilizada para armazenar informação das vendas para cada filial.

Para cada produto temos uma estrutura ListP que contém, além de um apontador para o produto, dois arrays de clientes que compraram o produto, um com os clientes que compraram em modo normal e outro com os que compraram em promoção.

Já para os clientes temos um estrutura ListC onde podemos encontrar um apontador para o cliente e um array de ProdCli com informação de todas as compras deste cliente. Em cada posição deste array temos armazenado, para um produto, o número de unidades que o cliente comprou por mês e o total que gastou com esse produto.

3.4 SGV



A estrutura FileInfo, contém, para cada um dos ficheiros carregados, um inteiro com o número de linhas lidas, um inteiro com o número de linhas validadas e um char* com o caminho para o ficheiro lido.

4 Interação com o Utilizador

```
-----MENU-----
[Q] | Sair do programa
[1] | Iniciar o SGV
[2] | Produtos começados por uma letra
[3] | Informação de um produto por mês
[4] | Códigos de produto que ninguém comprou
[5] | Códigos de cliente que realizaram compras em todas as filiais
[6] | Número de Clientes e número de Produtos nunca utilizados
[7] | Informação das compras de um cliente
[8] | Número de Vendas para um intervalo de meses
[9] | Códigos de cliente que compraram um produto
[10] | Códigos de produto que um cliente comprou num mês
[11] | Códigos de produto mais vendidos
[12] | Códigos de produto em que um cliente gastou mais dinheiro
-----
Introduza o seu comando: █
```

Quando o utilizador corre o programa, é-lhe apresentado o menu com os comandos possíveis de utilizar. Os comandos 2 a 12 apenas podem ser executados após a leitura dos ficheiros, feita através do comando 1.

Durante a execução do programa o utilizador pode sempre utilizar o comando menu para voltar a visualizar o menu de comandos.

Após iniciar um comando, será pedido ao utilizador para introduzir os argumentos necessários para a execução da querie.

```
-----
Página 1/92
-----
AA1001 - 1    AA1006 - 2    AA1011 - 3    AA1017 - 4    AA1022 - 5    AA1032 - 6
AA1038 - 7    AA1041 - 8    AA1045 - 9    AA1055 - 10   AA1063 - 11   AA1064 - 12
AA1071 - 13   AA1073 - 14   AA1075 - 15   AA1078 - 16   AA1079 - 17   AA1081 - 18
AA1082 - 19   AA1083 - 20   AA1084 - 21   AA1085 - 22   AA1092 - 23   AA1099 - 24
AA1106 - 25   AA1112 - 26   AA1113 - 27   AA1118 - 28   AA1121 - 29   AA1124 - 30
AA1125 - 31   AA1134 - 32   AA1143 - 33   AA1144 - 34   AA1145 - 35   AA1150 - 36
AA1152 - 37   AA1155 - 38   AA1161 - 39   AA1165 - 40   AA1169 - 41   AA1170 - 42
AA1173 - 43   AA1175 - 44   AA1178 - 45   AA1179 - 46   AA1199 - 47   AA1201 - 48
AA1203 - 49   AA1206 - 50   AA1209 - 51   AA1210 - 52   AA1219 - 53   AA1226 - 54
AA1229 - 55   AA1231 - 56   AA1232 - 57   AA1233 - 58   AA1236 - 59   AA1240 - 60
AA1247 - 61   AA1252 - 62   AA1259 - 63   AA1263 - 64   AA1270 - 65   AA1274 - 66
AA1280 - 67   AA1282 - 68   AA1288 - 69   AA1292 - 70   AA1293 - 71   AA1295 - 72
-----
[N] Next Page | [P] Previous Page | [F] First Page | [L] Last Page | [Q] Quit
-----
█
```

Se o utilizador executar uma opção que devolve uma lista de produtos ou clientes, deparar-se-á com o menu de páginas, onde pode utilizar os comandos da barra para navegar nestas.

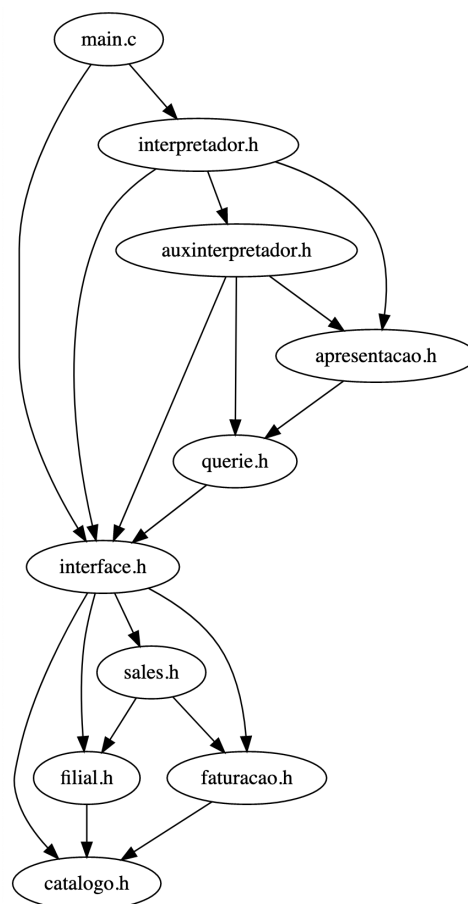
5 Estrutura do Projeto e Grafo de Dependências

O projeto está organizado segundo o padrão de design de Software MVC (Modelo Vista Controlador).

O Modelo do programa é gerido pelo módulo Interface. Este módulo contém a estrutura SGV e chama os módulos Filial, Sales, Faturacao e Catalogo para fazer qualquer alteração necessária aos dados do programa (carregar estrutura a partir de ficheiros, libertar memória, devolver dados, etc.).

A Vista do programa é controlada pelo módulo Apresentacao. Este módulo contém todas as funções que devolvem resultados visuais ao utilizador (resultados de queries, menus, etc.).

O Controlador do programa é gerido pelo módulo Interpretador. Este interage com o módulo Apresentação para devolver resultados visuais ao utilizador e com o módulo Interface para ter acesso ao SGV. Além disso, o Interpretador também chama funções do módulo Auxinterpretador que recebem os dados para execução das queries. Por sua vez, estas chamam as funções das queries do módulo Querie, dão o resultado ao módulo Apresentação para imprimir para o utilizador e libertam o espaço.



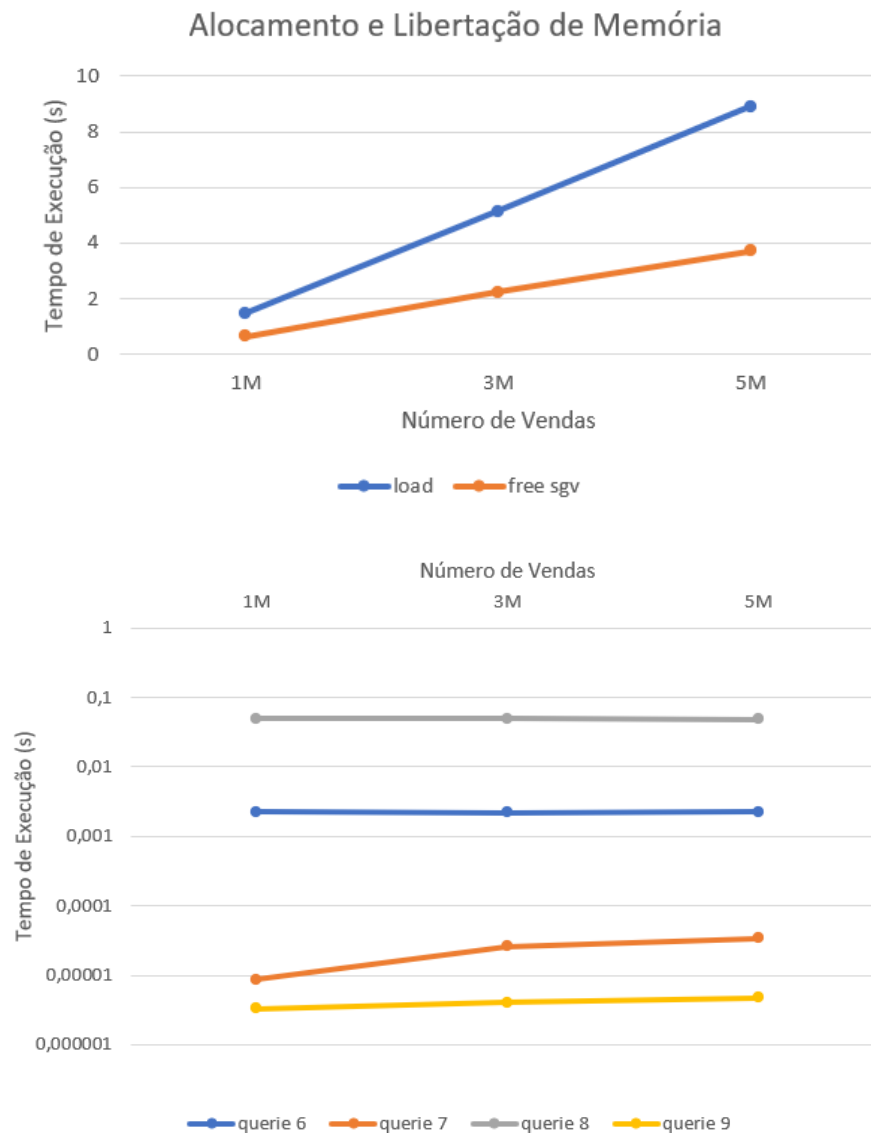
Grafo de dependências

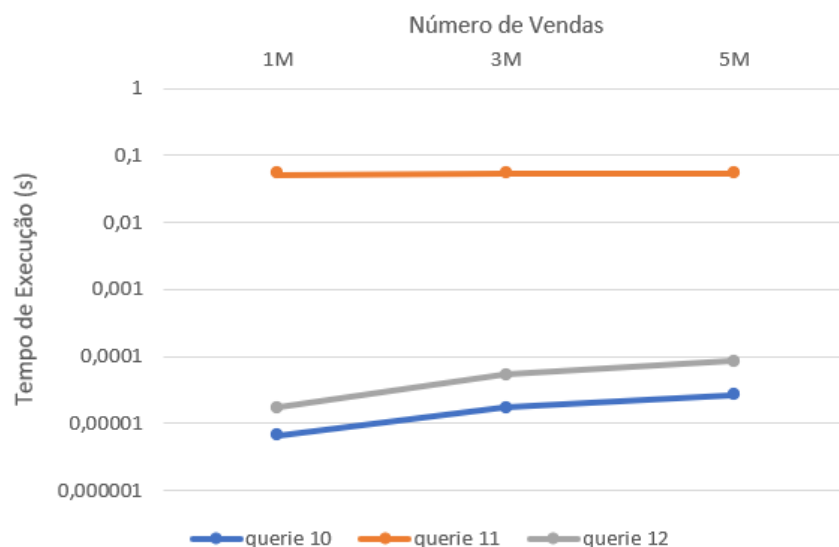
6 Testes de Performance

6.1 Tempos de Execução

Posteriormente ao desenvolvimento e codificação de todo o projeto foi-nos proposto realizar testes de performance que consistem na obtenção do tempo de carregamento dos ficheiros e dos tempos de execução das queries 6 a 12 usando os ficheiros de 1 Milhão, 3 Milhões e 5 Milhões de Vendas, para o qual usamos a biblioteca standard de C time.h.

Em seguida são demonstrados os resultados dos mesmos:





Como podemos ver a partir do Gráfico 1 (6.1), o tempo de carregamento de ficheiros foi de 1.4868s, 5.1501s e 8,9120s, para o ficheiro de 1 Milhão, 3 Milhões e 5 Milhões de vendas, respetivamente. Já em relação ao tempo de libertação de memória, obtivemos 0.6406s, 2.2396 e 3.7123s, para os mesmos ficheiros. O aumento do tempo de execução é aceitável, uma vez que é proporcional à variação do número de vendas.

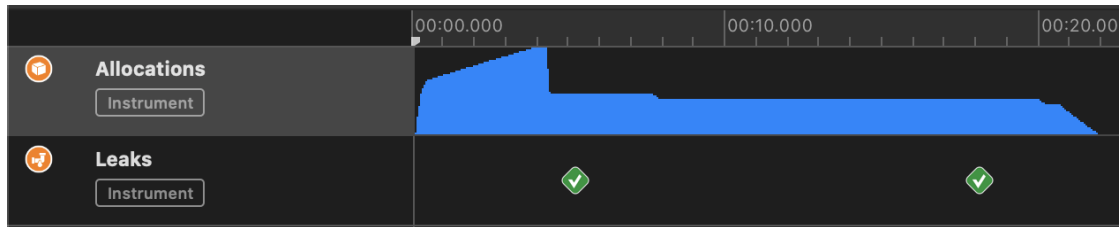
Em relação ao resultados das queries, notamos que algumas destas têm um tempo de execução constante em relação ao número de vendas. Isto deve-se ao facto de a querie fazer um varrimento dos produtos ou clientes (que mantêm o mesmo tamanho) ou ao facto de aceder diretamente a uma posição de memória.

As queries que mostraram resultados diferentes em relação ao número de vendas foram as queries 7, 10 e 12. Todas estas queries percorrem um array com as compras que um cliente fez, logo, com mais vendas, é perfeitamente aceitável o aumento do tempo de execução da querie.

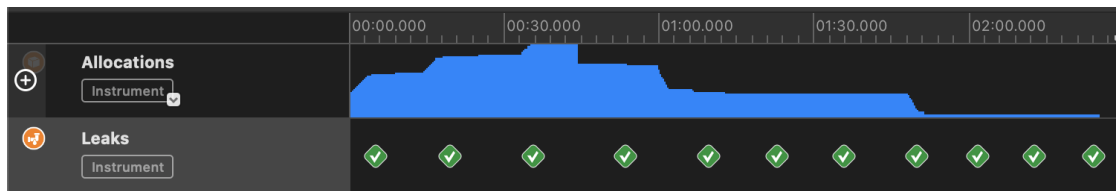
Estes testes foram realizados numa máquina com um processador i7 de 2,2 GHz, com uma memória Ram de 8GB. O produto utilizado para executar as Queries foi o "AF1184" e o cliente utilizado foi "Z5000". Em queries com um limite de resultados a apresentar usámos o valor 30.

6.2 Liberação de Memória

Além do tempo de execução do programa, fomos avisados que seríamos avaliados pela capacidade do nosso programa libertar toda a memória alocada. Para testar a funcionalidade, utilizamos a ferramenta Instruments do programa XCode.



Teste de Leaks para o ficheiro de 1 Milhão de Vendas



Teste de Leaks para o ficheiro de 3 Milhões de Vendas



Teste de Leaks para o ficheiro de 5 Milhões de Vendas

Como podemos observar a partir dos gráficos gerados pelo XCode, toda a memória foi libertada com sucesso para os três ficheiros. Além disso, o programa permitiu nos observar que para 1 Milhão de vendas o programa alocou cerca de 250MiB de memória, para 3 Milhões alocou cerca de 480 MiB e para 5 Milhões 760 MiB.

7 Makefile

```
CC = gcc      Compilador a utilizar

INCLUDE = -I include      Flag para incluir os header files

CFLAGS = -O3 -Wall -g -ansi      Flags de Compilação

SRC := src      Diretoria dos source files

OBJ := obj      Diretoria dos object files

SOURCES := $(wildcard $(SRC)/*.c)      Source files

NAME = program      Nome do executável

OBJECTS := $(patsubst $(SRC)/%.c, $(OBJ)/%.o, $(SOURCES))      Object Files

program: $(OBJECTS)
    $(CC) $(CFLAGS) $(INCLUDE) -o $(NAME) $(OBJECTS)      Compila o programa
    @echo Compilado

all: program
    doxygen Doxyfile      Compila o programa e
    @echo Documentação gerada      gera a documentação

.PHONY: clean
clean:
    rm -r $(OBJ)      Remove o executável e
    rm -f program      os object files
    @echo Objetos e Executável Apagados

.PHONY: cleanall
cleanall: clean      Remove o executável, os object
    rm -r docs/html      files e a documentação
    @echo Documentação Apagada

.PHONY: help
help:
    @echo "src: $(SOURCES)"      Imprime os source files e object files
    @echo "obj: $(OBJECTS)"

$(OBJ)/%.o: $(SRC)/%.c
    $(CC) $(CFLAGS) $(INCLUDE) -c $< -o $@      Gera os object files

$(shell mkdir -p $(OBJ))      Cria uma diretoria para os objetos, caso ela não exista
```

8 Conclusão e Reflexão Crítica

Desde logo, obtivemos resultados positivos no que toca à resposta das queries, cujos resultados foram os pedidos e o desempenho notável.

De um modo geral, estamos satisfeitos com a nossa solução, tanto a nível de desempenho do programa, como a nível de interação com o utilizador.

Para concluir, os objetivos deste projeto foram atingidos e durante a sua realização foram de reter determinados pontos. Primeiro, a necessidade de abdicar de performance para ter um código seguro, protegido do utilizador. Outro foi a importância da organização das estruturas de dados, quais os algoritmos a implementar e como diferentes relações entre estes dois conceitos tiveram um efeito considerável nos tempos de execução das funcionalidades do programa.