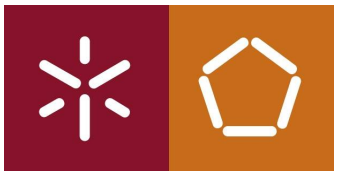

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática III

Relatório do Projeto em C

Grupo 60:

Ana Rita Rosendo
A84475

Gonçalo Esteves
A85731

Rui Oliveira
A83610

1 Introdução

No âmbito da Unidade Curricular de Laboratórios de Informática 3, foi-nos proposta a elaboração de um projeto em linguagem de programação C, cuja finalidade seria a leitura e análise de diversos ficheiros de dados, fornecidos pelos docentes. Estes ficheiros estão divididos em três tipos: ficheiros com códigos de produtos, ficheiros com códigos de clientes, e ficheiros com códigos de vendas. O objetivo do trabalho passa pela leitura e validação dos códigos dados nos ficheiros, e a sua inserção em estruturas de dados diversas que permitirão a análise da informação através das 12 *queries* sugeridas pelos docentes, cuja implementação também está ao nosso encargo.

2 Estrutura de Dados

De modo a elaborar o nosso projeto, debruçamo-nos sobre a matéria lecionada nas Unidades Curriculares de Programação Imperativa e Algoritmos e Complexidade, aproveitando os conhecimentos adquiridos para elaborar estruturas de dados eficientes, capazes de suportar a informação pretendida. Deste modo, a estrutura de dados principal é definida da seguinte forma:

```
typedef struct sgv{
   CatProd catp;
   CatCli catc;
   FatGlobal fat;
   VendasTotal vendas;
} *SGV;
```

Como pode ser observado, a estrutura de dados **SGV** é constituída pelo catálogo de produtos **CatProd**, pelo catálogo de clientes **CatCli**, por uma estrutura **FatGlobal** que corresponde à faturação e por uma estrutura **VendasTotal** que corresponde a todas as vendas efetuadas.

Comecemos por analisar as estruturas definidas no catálogo de produtos.

```
typedef char* Produto;

typedef struct nodoProd{
   Produto produto;
   struct nodoProd *dir, *esq;
} *ArvProd;

typedef ArvProd* CatProd;
```

A **ArvProd** é uma estrutura de dados do tipo árvore binária onde os produtos são guardados por ordem alfabética.

Portanto, o catálogo de produtos é um array dinâmico de diversas **ArvProd**, em que cada árvore suporta os produtos começados por uma mesma letra.

De seguida, analisemos as estruturas definidas catálogo de clientes.

```
typedef char* Cliente;

typedef struct nodoCli{
   Cliente cliente;
   struct nodoCli *dir, *esq;
} *ArvCli;

typedef ArvCli* CatCli;
```

Tal como a **ArvProd**, a **ArvCli** é uma árvore binária, onde os clientes são inseridos ordenadamente.

Logo, o catálogo de clientes será também um array dinâmico de árvores do tipo **ArvCli**, em que cada árvore suporta os clientes começados pela mesma letra.

Observemos agora estruturas definidas no ficheiro de faturação.

```
typedef struct fatura{
   Produto produto;
   double preco;
   int quantidade;
   char* tipo;
} *Fatura;

typedef struct nodoFat{
   Fatura fatura;
   struct nodoFat *dir, *esq;
} *ArvFat;

typedef ArvFat*** FatGlobal;
```

A **Fatura** é uma estrutura de dados que contém os dados de uma fatura sendo esses o produto, o preço, a quantidade de produto comprado e o seu tipo de compra que pode ser normal ou em promoção.

A **ArvFat** é uma estrutura de dados do tipo árvore binária que guarda as faturas ordenadas pelo produto.

Assim, a estrutura **FatGlobal** é constituída por diversas árvores **ArvFat** (mais precisamente, trata-se de um array tridimensional de árvores de faturas),

estando as suas dimensões associadas ao *Nº de Filiais*, *Nº de Meses* e *Nº de Letras*. Ou seja, cada árvore está associada a uma filial, um mês e uma letra (letra inicial do código do produto) específicos.

Por fim, consideremos as estruturas definidas no ficheiro que gere as filiais.

```
typedef struct venda{
   Produto produto;
   Cliente cliente;
   int quantidade;
   char* tipo;
   double preco;
} *Venda;

typedef struct nodoVenda{
   Venda venda;
   struct nodoVenda *dir, *esq;
} *ArvVenda;

typedef ArvVenda**** VendasTotal;
```

A **Venda** é uma estrutura de dados que contém os dados de uma venda sendo esses o produto vendido, o cliente que efetuou a compra, a quantidade de produto vendido, o tipo de compra e o preço do produto.

A **ArvVenda** é uma estrutura de dados do tipo árvore que guarda as vendas, ordenadas pelo produto.

Deste modo, podemos afirmar que a estrutura **VendasTotal** se trata de um array tetradimensional de árvores do tipo *ArvVenda*, onde as suas dimensões são determinadas pelo *Nº de Filiais*, o *Nº de Meses* e *Nº de Letras* (sendo este último considerado duas vezes). Assim, podemos afirmar que cada árvore esta associada a uma filial, um mês e a duas letras (iguais ou não) - a primeira será relacionada com o cliente, e a segunda com produto.

Para além das estruturas já analisadas, foi necessário criar outras estruturas essenciais para o desenvolvimento do projeto.

No ficheiro do catálogo dos produtos, foi criada uma estrutura de dados, a **ListaProd**, que suporta um array dinâmico de produtos e o número de produtos no array.

```
typedef struct listaProd{
   Produto* produtos;
   int quantos;
} *ListaProd;
```

À semelhança do caso anterior também no ficheiro do catálogo dos clientes foi criada uma estrutura de dados, a **ListaCli**, que guarda códigos de clientes num array dinâmico, e o número de clientes inseridos nesse array.

```
typedef struct listaCli{
   Cliente* clientes;
   int quantos;
} *ListaCli;
```

Também no ficheiro da faturação foi necessário definir novas estruturas, tais como:

- estrutura **VendaInicial**

```
typedef struct vendaInicial{
   Produto produto;
   Cliente cliente;
   int quantidade;
   double preco;
   char* tipo;
   int mes;
   int filial;
} *VendaInicial;
```

Esta estrutura contém os dados de uma venda completa, sendo esses o produto vendido, o cliente que efetuou a compra, a quantidade e o preço do produto vendido, e o mês e a filial em que o produto foi vendido.

- estrutura **Par**

```
typedef struct par{
   int elemI;
   double elemD;
} *Par;
```

Esta estrutura define um par de inteiros/double.

Também foram criadas novas estruturas no ficheiro da gestão de filiais, tais como:

- estrutura **ParProdInt**

```
typedef struct parProdInt{
   Produto produto;
   int quantidade;
} *ParProdInt;
```

Esta estrutura representa um par de produtos/int em que o int corresponde à quantidade vendida do produto em questão.

- estrutura **ListaParesInt**

```
typedef struct listaProdIntQuant{
   ParProdInt* pares;
   int quantos;
} *ListaParesInt;
```

Esta estrutura representa uma lista de pares produtos/int e a quantidade de pares.

- estrutura **ParProdDouble**

```
typedef struct parProdDouble{
   Produto produto;
   double quantidade;
} *ParProdDouble;
```

Esta estrutura representa um par de produtos/double em que a quantidade representa o total de dinheiro ganho com o produto.

- estrutura **ListaParesDouble**

```
typedef struct listaProdDoubleQuant{
   ParProdDouble* pares;
   int quantos;
} *ListaParesDouble;
```

Esta estrutura representa uma lista de pares produtos/double e a quantidade de pares.

- estrutura **TriploPCQ**

```
typedef struct triploPCQ{
   Produto produto;
   int nCli;
   int quantidade;
} *TriploPCQ;
```

Esta estrutura representa um triplo de produto/número de clientes/quantidade vendida.

- estrutura **ListaTriplos**

```
typedef struct listaTriplos{
   TriploPCQ* triplos;
   int quantos;
} *ListaTriplos;
```

Esta estrutura representa uma lista de triplos e a quantidade de triplos existentes na lista.

3 Queries

3.1 Query 1

A query 1 permite ao utilizador ler os ficheiros de Produtos, Clientes e Vendas e apresenta ao utilizador o nome do ficheiro lido, bem como o número de linhas lidas e o de linhas validadas.

Para responder a esta *query*, foram criados 3 algoritmos que lessem os diferentes ficheiros. Para os de Produtos e de Clientes usamos a mesma metadologia. Primeiro, lemos o ficheiro, percorremos código a código e, caso seja válido, inserimos num catálogo de produtos/clientes, definido pela estrutura **CatProd/CatCli**, mais precisamente, na árvore correspondente à primeira letra do código de produto ou cliente. À medida que lemos cada código, incrementamos o número de linhas lidas e se for válido, incrementamos também o número de linhas validadas.

No caso do ficheiro de Vendas, também começamos por lê-lo e analisamos código a código. Caso este seja válido, restringimos a venda em termos da sua informação e inserimo-la em duas estruturas de dados distintas, a de faturação, definida por **FatGlobal** (na qual se insere a venda apenas com a informação necessária a uma **Fatura**); e a de gestão de filiais, definida por **VendasTotal** (na qual se insere a venda apenas com a informação necessária a uma **Venda**). Em semelhança aos outros dois ficheiros, à medida que lemos cada código de venda, incrementamos o número de linhas lidas e caso válido, incrementamos o número de linhas validadas.

3.2 Query 2

A query 2 determina a lista e o nº total de produtos cujo código se inicia por uma dada letra. A lista é apresentada ao utilizador, por ordem alfabética, permitindo que o mesmo navegue nela.

Nesta *query*, foi criada uma lista, definida pela estrutura **ListaProd**, que guarda um array com códigos de produtos e o número de produtos contidos no mesmo.

Após sabermos a letra que o utilizador pretende analisar, acedemos ao nosso catálogo de produtos, definido pela estrutura **CatProd** e como este está ordenado de ordem alfabética, sabemos logo o índice onde se encontra a árvore binária que contém os códigos de produtos iniciados pela letra dada. Como tal, acedemos a essa árvore e inserimos cada código de produto no array da nossa lista, ordenado alfabeticamente, e incrementamos o número de produtos contidos. Tendo a lista, usamos o navegador que definimos, de forma a permitir ao

utilizador percorrer página a página o conteúdo da lista.

3.3 Query 3

A query 3 retorna o número total de vendas e o total facturado com um dado produto em determinado mês, ambos válidos, distinguindo os totais relativamente ao tipo de compra. O utilizador pode aceder aos resultados filial a filial ou para toda as 3.

De forma a responder a esta *query*, após sabermos o mês e o produto que o utilizador pretende analisar, precisamos de percorrer 3 árvores binárias da nossa estrutura **FatGlobal**, ou seja, uma árvore de cada filial. O algoritmo usado para cada filial é o mesmo: acedemos à estrututa, particularizando não só a filial que pretendemos (iremos analisar uma de cada vez), mas também o mês (escolhido pelo utilizador) e o índice onde se encontra a árvore que guarda o código das faturas iniciados pela letra dada. Após isto, inicialmente, vamos analisar a situação de compra Normal e percorremos a árvore código a código, e caso a fatura tenha código de produto igual ao que estamos a analisar e o tipo de compra do mesmo seja Normal, multiplicamos a quantidade de unidades vendidas do produto dessa fatura pelo preço unitário do mesmo e somamos esse valor aos outros das faturas que se encontram na mesma situação(ou seja, com código de produto igual ao que estamos analisar e compra do tipo N). Desta forma, chegando ao fim da árvore, sabemos quanto é que faturamos com dado produto num determinado mês numa filial. Fazemos agora o mesmo para as outras duas filiais. Agora fazemos novamente o mesmo, mas para o tipo de compra em Promoção.

Depois, voltamos a percorrer as mesmas árvores, mas agora para guardarmos apenas a quantidade de unidades vendidas pelo produto dado. Basicamente, usamos o mesmo processo que anteriormente, mas desta vez apenas guardamos o total de compras.

Por fim, caso o utilizador pretenda saber os resultados totais (ou seja, para todas as filais) somamos os valores encontrados para cada filial e retornamos o mesmo para cada tipo de compra. No entanto, se pretender os resultados por filial, retornarmos os valores já guardados, especificando a filial e o tipo de compra.

3.4 Query 4

Esta query determina a lista ordenada dos códigos dos produtos (e o seu número tota) que ninguém comprou, podendo o utilizador decidir igualmente se pretende valores totais ou divididos pelas filiais.

Nesta *query*, foi novamente necessário o uso da estrutura **ListaProd**. Primeiramente, acedemos ao nosso Catálogo de Produtos **CatProd** e, percorrendo todas as árvores do mesmo, convertemos cada uma delas numa lista do tipo que criamos inicialmente, uma de cada vez. À medida que convertemos uma árvore para uma lista (chamemos-lhe "listaPorLetra"), criamos também outra lista (chamemos-lhes "produtosNaoVendidos") e, caso o utilizador pretenda valores totais, percorremos todas as árvores da nossa **FatGlobal**, e caso haja produtos da "listaPorLetra" que não se encontram nos códigos das faturas das árvores, iniciadas por letra igual ao produto que estamos a analisar, da **FatGlobal**, inserimos esse produto na "produtosNaoVendidos". Continuamos a fazer isto recursivamente para todas as árvores do nosso cátologo de produtos e conforme a evolução do processo, vamos fundindo as listas "produtosNaoVendidos", somamondo também o número de elementos de cada lista para, no final, termos o número de produtos que nunca foram comprados. No entanto, caso o utilizador queira valores divididos por filiais, usamos o mesmo algoritmo de cima, mas quando percorrermos as àrvores da **FatGlobal**, percorremos apenas as árvores da filial que o utilizador pretende.

3.5 Query 5

A quinta query retorna a lista ordenada de códigos de clientes que realizaram compras em todas as filiais.

Para esta *query*, criamos uma lista do tipo **ListaCli**, composta por um array com códigos de clientes e o número de clientes contidos no mesmo. Em primeiro lugar, acedemos ao nosso Catálogo de Clientes **CatCli** e, percorrendo todas as árvores do mesmo, convertemos cada uma delas numa lista do tipo que criamos inicialmente, uma de cada vez. À medida que convertemos uma árvore para uma lista (vamos chamar-lhe "listaPorletra"), criamos também outra lista ligada(chamada, por exemplo, "clientesEmTodasAsFiliais") e percorremos todas as árvores da nossa **VendasTotal**, e, caso haja clientes da "listaPorLetra" que se encontram nos códigos das vendas das árvores, iniciadas por letra igual ao produto que estamos a analisar, de todas as filiais da **VendasTotal**, inserimos esse cliente na "clientesEmTodasAsFiliais". Continuamos a fazer isto recursivamente para todas as árvores do nosso cátologo de clientes e conforme a evolução do processo, vamos fundindo as listas "clienteEmTodasAsFiliais" e somamos o número de elementos de cada lista para, no final, termos o número de clientes efetuaram compras nas 3 filiais.

3.6 Query 6

A query 6 determina o número de clientes registados que não realizaram compras bem como o número de produtos que ninguém comprou.

Para a resolução desta *query*, criamos duas lista, uma delas guardava um array com códigos de clientes e o número de clientes contidos no mesmo (era do tipo **ListaCli**) e outra guardava um array com códigos de produtos e o número de produtos contidos no mesmo (do tipo **ListaProd**). Inicialmente, usamos o algoritmo usado na *query 4* para concluir o número produtos que não foram comprados por ninguém em todas as filiais. Seguidamente, de modo a determinar o número de clientes registados que não realizaram compras, começamos por aceder ao nosso Catálogo de Clientes **CatCli** e, percorrendo todas as àrvores do mesmo, convertemos cada uma delas numa lista de clientes do tipo que criamos inicialmente, uma de cada vez. À medida que convertemos uma árvore para uma lista (vamos chamar-lhe "listaPorletra"), criamos também outra lista (chamada, por exemplo, "clientesQueNaoCompraram") e percorremos todas as árvores da nossa **VendasTotal**, e caso haja clientes da "listaPorLetra" que não se encontram nos códigos das vendas das árvores, iniciadas por letra igual ao produto que estamos a analisar, da **VendasTotal**, inserimos esse cliente na "clientesQueNaoCompraram". Continuamos a fazer isto recursivamente para todas as árvores do nosso cátologo de clientes e conforme a evolução do processo, vamos fundindo as listas "clientesQueNaoCompraram". Por fim, percorremos as duas listas que temos agora, definidas no início da *query*, de modo a saber quantos elementos tem cada uma e, consequentemente, o número de clientes que nunca realizaram compras, bem como o de produtos que nunca foram comprados.

3.7 Query 7

Dado um código de cliente, criar uma tabela com o **número total de produtos comprados** (ou seja a soma das quantidades de todas as vendas do produto), mês a mês (para meses em que não comprou a entrada deverá ficar a 0). A tabela deverá ser apresentada em ecrã organizada por filial.

Para a resolução desta *query*, foi criada uma matriz dinâmica de inteiros, de dimensão *Nº de Filiais* * *Nº de Meses*, de modo a guardar em cada posição da matriz o total de produtos comprados pelo cliente numa determinada filial, num determinado mês.

De modo a preencher uma dada posição (x, y) da matriz, foi criada uma função que percorre a estrutura **VendasTotal**, onde estão armazenadas todas as vendas, e para todas as árvores de vendas associadas à filial "x", ao mês "y" e ao caractér "c" (primeiro caractér do código de cliente dado), contabiliza todas as vendas efetuadas pelo cliente, devolvendo o seu número total.

3.8 Query 8

Dado um intervalo fechado de meses, por exemplo [1..3], determinar o total de vendas (nº de registos de venda) registadas nesse intervalo e o total faturado.

Tendo em vista a elaboração desta *query*, criou-se uma função que, dada uma estrutura **FatGlobal** e dois meses, percorre todas as árvores de faturas associadas a esse intervalo de meses (ou seja, para todas as filiais e todas as letras pelas quais os códigos de produtos são iniciados, percorre as árvores cujo mês associado está compreendido no intervalo fechado desses dois meses). Ao percorrer cada uma destas árvores, a função guarda os resultados que obtém (nº de vendas e total faturado) numa estrutura do tipo **Par**, sendo esta retornada no fim, após a travessia de todas as árvores, estando nela guardada o nº de vendas total e o total faturado, para o intervalo de meses pretendido.

3.9 Query 9

Dado um código de produto e uma filial, determinar os códigos (e número total) dos clientes que o compraram, distinguindo entre compra N e compra P.

Com o objetivo de resolução desta *query*, foi construída uma função tal que, dada uma estrutura **CatCli**, uma **VendasTotal**, o produto e a filial que se pretende analisar, e um tipo de compra ('N' ou 'P'), devolve a lista de todos os clientes que efetivamente compraram esse produto nessa filial, para o dado tipo de compra. Esta função executa da seguinte forma: para o catálogo de clientes fornecido, vai criando, letra a letra, uma lista com os clientes (**ListaCli**) iniciados por uma dada letra; depois, através de uma função auxiliar, percorre a lista de clientes, indo verificar, para cada um dos clientes, se ele comprou o produto em questão, tendo em conta o tipo de venda que está a ser analisado. Ou seja, a função percorre todos os clientes, verificando quais os que compraram o produto para um dado tipo de venda, guardando o código daqueles cujo teste deu positivo numa lista de clientes, que será devolvida ao utilizador.

Para a correta execução da *query*, a função acima referida é evocada duas vezes, uma para o tipo de vendas 'N' e outra para o tipo de vendas 'P', sendo que ambas as listas são exibidas ao utilizador, distinguindo as compras normais das promocionais.

3.10 Query 10

Dado um código de cliente e um mês, determinar a lista de códigos de produtos que mais comprou (por quantidade e não por faturação), por ordem descendente.

Com o intuito de desenvolver esta *query*, foi criada uma estrutura **ParProdInt**, que é um par composto por um código de produto e um inteiro, e uma lista destes pares, designada de **ListaParesInt**, que possui um array dinâmico de pares e a quantidade de pares guardada.

Posto isto, foi feita uma função que, dada uma estrutura **VendasTotal**, um mês e um código de um cliente, retorna uma lista de pares, com todos os produtos comprados por esse cliente nesse mês e a respetiva quantidade. De modo a fazer isto, a função percorre, para todas as filiais, todas as árvores associadas ao mês em questão, particularizando também apenas as que estão associadas à letra pela qual é iniciado o código do cliente (no entanto, claramente tem de percorrer todas as letras dos produtos). Durante este processo, e com a ajuda de uma função auxiliar, são inseridos numa lista de pares (que virá a ser devolvida como sendo a lista final) todos os produtos, e a respetiva quantidade comprada, adquiridos pelo cliente em questão.

3.11 Query 11

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.

De modo a elaborar esta *query*, foi definida uma função que, dada uma estrutura **VendasTotal**, uma **CatProd**, um inteiro N e uma filial, devolve uma lista de triplos com os N produtos mais vendidos. Um triplo está definido na estrutura **TriploPCQ**, sendo que este possui um código de produto e dois inteiros. Uma lista de triplos é dada pela estrutura **ListaTriplos**, que possui um array dinâmico de **TriploPCQ** e um inteiro, que indica a quantidade de triplos.

Para a resolução do problema, a função cria, para cada um dos produtos presentes no catálogo de produtos, um triplo com o número de clientes que o compraram e o total de produtos comprados, percorrendo todas as árvores de vendas associadas à filial que se pretende analisar e à primeira letra do produto em questão. Posto isto, insere cada triplo numa lista de tamanho N, caso a quantidade de produtos vendidos seja superior à de pelo menos um dos produtos previamente inseridos nessa lista.

Por fim, retorna a lista com os N produtos mais vendidos.

3.12 Query 12

Dado um código de cliente determinar quais os códigos dos 3 produtos em que mais gastou dinheiro durante o ano.

Com a intenção do desenvolvimento desta *query*, foi criada uma função que, dada uma estrutura **VendasTotal** e um cliente, retorna uma lista de pares **ParProdDouble** com os 3 produtos em que esse cliente mais gastou dinheiro. Um **ParProdDouble** é um par com um código de produto e um double, sendo a sua lista definida na estrutura **ListaParesDouble**, constituída por um array de pares e o inteiro com a quantidade de pares inseridos.

A referidada função percorre a estrutura de vendas, passando por todas as árvores associadas à primeira letra do cliente em questão, inserindo numa lista de pares todos os produtos (e o total gasto) cujo cliente comprou. Posto isto, essa lista é ordenada de forma decrescente, tendo em conta o total de dinheiro gasto, sendo devolvida ao cliente uma lista com os três primeiros elementos da lista ordenada (que são, consequentemente, os produtos nos quais foi gasto mais dinheiro).

4 Modularidade

O código foi criado de forma dividida e organizada de forma a que o utilizador não tivesse hipótese de alterar as estruturas de dados, de modo a tornar os módulos do programa o mais isolados possíveis, permitindo um encapsulamento dos dados. Tendo isto em vista, o código foi dividido em seis módulos: "catClientes.h", "catProd.h", "faturacao.h", "gestaoDeFiliais.h", "queries.h" e "navegador.h".

O primeiro módulo trata das funções que implementam a estrutura de dados que guarda os códigos dos clientes.

O segundo módulo trata das funções que implementam a estrutura de dados que guarda os códigos dos produtos.

Relativamente ao módulo "faturacao.h", este permite a implementação da estrutura de dados que guarda os códigos das faturas, bem como funções auxiliares para a resolucão de queries envolvendo a Faturação.

O módulo "gestaoDeFiliais.h" trata dos algoritmos que implementam a estrutura de dados responsável por guardar os códigos das vendas relativamente à Gestão de Filiais, bem como funções auxiliares para uso nas queries envolvendo a Gestão de Filiais.

No módulo "queries.h" é onde se situam as funções responsáveis pela resolução das queries.

Por fim, o módulo "navegador.h" é o que permite a navegação do conteúdo de listas de produtos ou clientes, pedidas nas queries.

Estes módulos servem como uma API que permite ao utilizador apenas aceder determinados dados, escolhidos conforme o critério de quem está a programar, neste caso optando por um maior isolamento dos dados, e funcionam também como um auxílio ao programador, visto que permite uma partilha de dados entre módulos. Uma vez que o código está divido em vários módulos, permite que o programa em si esteja melhor organizado e de fácil leitura.

5 Otimização do Desempenho

Em relação a estratégias de desempenho, pode-se referir o facto de que, inicialmente guardavamos os códigos dos produtos, dos clientes e das vendas em diferentes arrays de strings. Porém, desta forma demoravam imenso tempo a ser lidos, principalmente o ficheiro de vendas (demorava cerca de 10 minutos).

Passamos, então, a guardar os códigos em árvores binárias, organizadas em arrays das mais variadas dimensões, de modo a estruturar melhor a distribuição dos códigos, e a agilizar as inserções dos mesmos.

Para além disto, ao longo da resolução das *queries* foram criadas também estruturas auxiliares, como por exemplo estruturas que guardavam pares de informação, de forma a otimizar as mesmas, permitindo um melhor desempenho.

6 Conclusão

Após a elaboração deste projeto, podemos afirmar que aumentamos o nosso nível de conhecimento, não só no que toca ao domínio da linguagem de programação C, mas também à compreensão de determinados conceitos, nomeadamente a modularidade, o encapsulamento de dados e a otimização de código. Com o desenvolvimento deste trabalho, fomo-nos deparando com diversos obstáculos que se colocaram à nossa frente, sendo necessário não só fazer uso de conhecimentos previamente adquiridos, mas também de outros novos, conseguidos através de uma procura nas fontes disponíveis.