PROJETO LI3

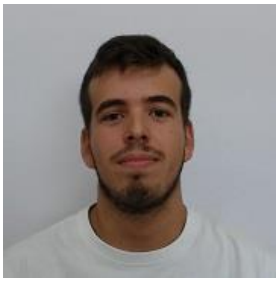
2019

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Filipa Santos, Hugo Cardoso, João Cunha





ÍNDICE

1. INTRODUÇÃO

O projeto de Laboratórios de Informática III encontra-se dividido em duas etapas, uma em C e outra em Java. Neste relatório, vamos abordar a primeira. O tema deste ano consiste em fazer uma aplicação que simula a gestão de vendas de uma empresa, isto é, determinar numa certa venda o código do cliente que efetuou essa tal venda, o código do produto que comprou, a quantidade de unidades, o preço total, etc.

A aplicação que criamos começa por ler os ficheiros de produtos, clientes e vendas de uma certa empresa. Esses dados são guardados em estruturas escolhidas por nós (explicadas em detalhe na secção dos módulos), de modo a tornar esta leitura mais rápida. De seguida, é possível consultar estes dados e, através das queries, adquirir resultados específicos, como, por exemplo, os códigos de clientes que realizaram compras em todas as filiais.

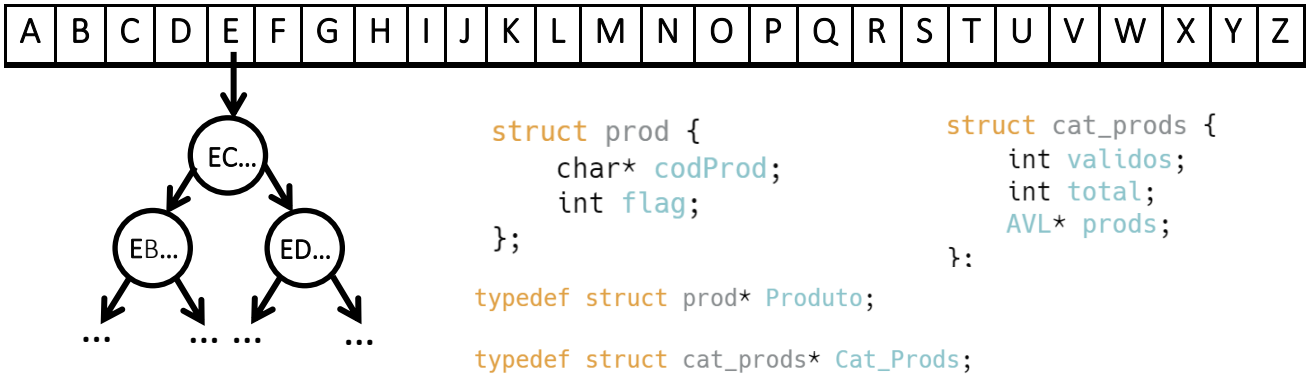
Neste relatório será abordado o motivo de optarmos por certas estruturas para os módulos criados e alterações feitas para que a consulta das queries seja mais eficiente.

2. MÓDULOS

2.1 CATÁLOGO DE PRODUTOS (CAT_PRODS)

Inicialmente, a estrutura optada para guardar os produtos era um array que, quando ficava sem memória, era realocado para outro com o dobro do tamanho. Rapidamente se constatou que esta não era a solução mais eficaz, nem para inserção dos elementos nem para a sua consulta.

Então, balanceando a memória e a rapidez da nossa aplicação, concluimos num array de 26 posições (correspondentes às 26 letras do alfabeto), em que cada célula aponta para uma AVL de produtos começados por uma certa letra e ordenada alfabeticamente.



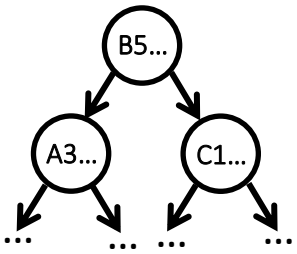

```
struct prod {
    char* codProd;
    int flag;
};

typedef struct prod* Produto;
```

```
struct cat_prods {
    int validos;
    int total;
    AVL* prods;
};

typedef struct cat_prods* Cat_Prods;
```

A estrutura do produto possuía inicialmente apenas o código deste, de maneira a torná-lo privado. Os restantes constituintes das estruturas (como a flag, o total, etc) serão explicados na parte das queries, sendo que foram adicionados mais tarde de maneira a facilitar a execução das mesmas.

2.2 CATÁLOGO DE CLIENTES (CAT_CLIS)

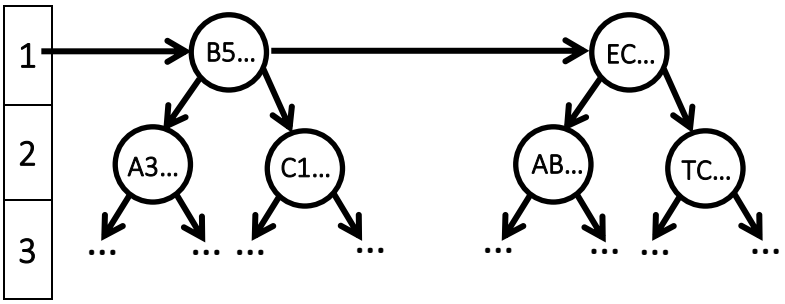
Para os clientes, tivemos uma abordagem semelhante. Como neste caso o código de cliente tem apenas uma letra, não achamos necessário criar um array, sendo a que a estrutura final acabou por ser apenas uma AVL, organizada alfabeticamente, com os conteúdos dentro que achamos necessários.




```c
struct cli {
    char* codCli;
    int flag;
};

struct cat_clis {
    int validos;
    int total;
    AVL clis;
};

typedef struct cli* Cliente;

typedef struct cat_clis* Cat_Clis;
```

2.3 FILIAIS (CAT_VFILS)

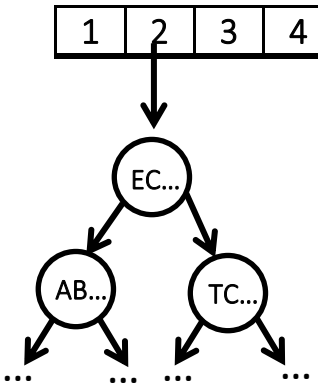
Relativamente às vendas, foi necessário dividir a sua informação por dois módulos distintos, um relativamente à facturação e outro às filiais. Neste último, decidimos criar um array de tamanho 3 (correspondente às 3 filiais), em que cada entrada aponta para uma AVL de clientes (organizada alfabeticamente). Por sua vez, cada nodo desta aponta para uma AVL com os códigos dos produtos (organizada alfabeticamente) que esse mesmo cliente comprou, tal como outras informações dessa compra (unidades, mês, etc).




```c
struct pcfil {
    Produto  produto;
    double   preco;
    int      unidades;
    char     tipo;
    int      mes;
};

typedef struct pcfil* PCFil;

typedef struct vfil* VFil;

typedef struct cat_vfils* Cat_VFils;

struct vfil {
    Cliente cliente;
    AVL prodsc;
};

struct cat_vfils {
    AVL* fils;
};
```

2.4 FACTURAÇÃO (CAT_VFACTS)

Na facturação, novamente, optamos por uma estrutura que consiste num array de tamanho 12 (correspondente aos 12 meses). Cada entrada aponta para uma AVL de produtos (organizada alfabeticamente) que contém também outros dados necessários para a resolução das queries que consultam este módulo.




```c
struct cat_vfacts {
    int totalVendas;
    int* vendasValidas;
    AVL* mes;
};
```

```c
struct vfact {
    Produto produto;
    double  preco;
    int     unidades;
    char    tipo;
    int     filial;
};
```

```c
typedef struct vfact* VFact;

typedef struct cat_vfacts* Cat_VFacts;
```

2.5 NAVEGADOR

O navegador é necessário para as queries que devolvem uma ou várias listas de dados, nomeadamente códigos de produtos, de maneira a navegar organizadamente pelos mesmos, neste caso visualizando-os em grupos de 20 de cada vez. Assim, evita-se que o utilizador receba logo a lista inteira de uma só vez, o que não seria prático nem organizado.

A estrutura do navegador possui a lista de strings (códigos de produtos/clientes) que vai ser exibida e a posição atual na mesma, de maneira a ser possível avançar ou retroceder uma página, bem como outro parâmetro relevante para as queries.

```c
struct lst_str {
    char** strings;
    int pos;
    int num;
};

typedef struct lst_str* Lista_Strings;
```

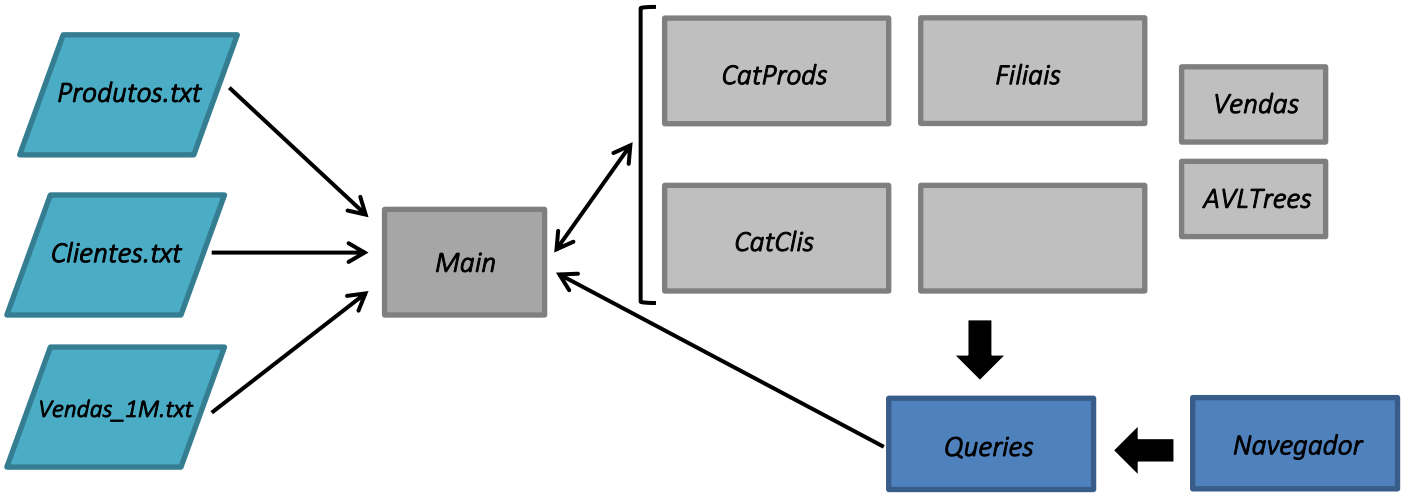
2.6 OUTROS

Para auxiliar o funcionamento da aplicação, houve a necessidade de criar duas estruturas de dados meramente auxiliares. Uma delas, chamada de "Venda", tem o mero propósito de guardar momentaneamente todos os parâmetros da venda e aferir a sua validade, antes de serem fatiados pelas filiais e facturação. A outra, "SGV", foi criada com o objetivo de facilitar acesso às diversas estruturas de dados, sendo representativa de todo o Sistema de Gestão de Vendas.

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

```
typedef struct sgv {
    Cat_Prods catp;
    Cat_Clis catc;
    Cat_VFacts catvfact;
    Cat_VFils catvfil;
} *SGV;
```

3. ARQUITETURA FINAL DA APLICAÇÃO



4. OPTIMIZAÇÃO

4.1 LEITURA DOS CATÁLOGOS

No final, consideramos AVL's a escolha mais apropriada para usar em diversas estruturas. Com esta estrutura, conseguimos aceder aos dados pretendidos num tempo reduzido - O(log N) - devido à organização e balanceamento da mesma. Contudo, consideramos os nossos algoritmos relativamente às AVL's, especificamente o responsável por balancear a árvore, um pouco ineficiente. Isto causou um aumento da duração da inserção na árvore, o que tornou o tempo de execução do nosso programa indesejavelmente alto, apesar das tentativas infrutíferas de otimizar tais algortimos.

4.2 QUERIES

Na realização das queries, tivemos de efetuar algumas mudanças às estruturas de dados para obter uma consulta mais eficaz. Assim, eis alguns exemplos das decisões tomadas baseadas em certas queries:

QUERY 1: foram criadas 2 variáveis nas estruturas Cat_Prods, Cat_Clis e Cat_VFacts chamadas "válidos" e "total" para que esta query seja logo determinada durante a inserção. Assim, quando esta query é pedida pelo utilizador, não é necessário percorrer as estruturas inteiras, contando nodo a nodo, para determinar os valores pedidos (número de linhas lidas e validadas de cada módulo), é apenas necessário aceder a estas variáveis. Além disso, dado que cada estrutura da facturação e das filiais é criada a partir da mesma venda, seria contraprodutivo ter os contadores em ambos estes catálogos, visto que os seus valores seriam iguais.

QUERY 2: considerando que é necessário apenas uma das árvores (árvore da respetiva letra) do catálogo de produtos, aqui a decisão do array de 26 entradas, cada uma correspondente a cada letra do alfabeto, beneficia muito esta query.

QUERY 4: para esta query foi adicionada uma "flag" à estrutura de cada produto de modo a que, para verificar se um produto foi ou não comprado, seja apenas necessário consultar esta variável, ao revés de consultar a árvore da facturação para verificar se o produto consta na mesma.

QUERIES 5/6: de modo semelhante ao 4, colocamos uma "flag" na estrutura dos clientes idêntica à dos produtos a fim de saber se realizaram compras em todas as filiais. Note-se que a flag pode assumir vários valores, indicando se o cliente/produto fez compras/foi comprado, e em que filial(ais). Por exemplo, um cliente com "flag" de valor 7 realizou compras em todas as filiais.

QUERIES 7/10/12: inspiraram-nos a colocar uma AVL dos produtos (e os outros promenores da compra) em cada cliente no módulo de dados correspondente às filiais, dado que precisamos de percorrer todas as compras de um certo cliente.

QUERIES 2/4/9: para estas queries adicionamos um parâmetro "num" à estrutura do navegador com o número total de elementos da lista que este vai imprimir no ecrã, de maneira a não ser necessário percorrer de novo a lista para fazer a contagem.

5. CONCLUSÃO

Em suma, consideramos a nossa prestação nesta trabalho boa, dado que fizemos um bom trabalho em alcançar algoritmos imediatos para as queries pedidas. Foi um projeto interessante para desenvolver o racicíonio sobre estruturas de dados mais e menos eficientes e sobre a organização de uma aplicação comum.

Por outro lado, sentimos que ficamos aquém em termos de tempo de leitura dos ficheiros dados. Fica assim como objetivo para a 2ª parte deste projeto melhorar este aspeto.

Do grupo 23, do turno PL5, constituído por:

João da Cunha e Costa, nº 84775

Hugo André Coelho Cardoso, nº 85006

Filipa Alves dos Santos, nº 83631