Relatório do projeto SGV - LI3

Grupo 66

José Magalhães, Luís Ramos e Luís Vila

9 de Abril de 2019



Conteúdo

1 Introdução

No âmbito da unidade curricular Laboratórios de Informática III, foi proposto desenvolver um projeto em *C*, mais concretamente um Sistema de Gestão de Vendas (SGV). Este passa por armazenar e tratar informação presente em ficheiros TXT. Esses ficheiros incluem informação sobre os Produtos, os Clientes e sobre as Vendas. Toda esta informação está organizada por códigos.

Utilizou-se no nosso trabalho estruturas de dados criadas por nós, apresentadas à frente, pelo que esta decisão permitiu moldar as estruturas conforme as nossas necessidades.

Assim, criaram-se estruturas que fossem de fácil acesso conforme as nossas necessidades de coletar informação, tentando assim potenciar a eficiência do nosso programa.

2 Estruturas de dados

2.1 Módulo *main*

De modo a auxiliar o cumprimento da *query 1*, foi necessário a criação das seguintes estruturas:

```
typedef struct sgv{
   int prodLidos;
   int cliLidos;
   int vendasLidas;
   int vendasValidas;
   Cat_Produtos prod;
   Cat_Clientes cli;
   Cat_FatGlobal fat;
   Filial fili;
} *SGV;
```

Estrutura que funciona como elo de ligação entre as várias estruturas criadas ao longo dos vários módulos. Inclui, ainda, quatro contadores necessários para que a resposta à *query* em causa seja a mais eficiente possível.

```
typedef struct files{
   char *fc;
   char *fp;
   char *fv;
} *FILES;
```

Com a função de colocar no respetivo campo da *struct* os nomes dos ficheiros associados, serão feitas alocações de memória de modo a que o nome do ficheiro de clientes seja associado a fc, o dos produtos a fp e o das vendas a fv.

```
typedef struct venda{
    char* codProd;
    char* codCli;
    double preco;
    int quantidade;
    int mes;
    int filial;
    float total;
}*Venda;
```

Este tipo de dados foi construído de modo a que, quando é lida uma linha de venda, os vários campos desta estrutura sejam preenchidos, conforme as divisões fornecidas pela mesma.

2.2 Módulo *Catálogo de produtos*

```
struct prod {
    char* produto;
    int filial1;
    int filial2;
    int filial3;
};
typedef struct prod *Produto;
```

Estrutura criada com o objetivo de guardar o código de um dado produto, assim como três *flags* de modo a guardar o número da filial em que este fora comprado, o que é útil para o desenvolvimento de outras *querys*.

```
struct cat_prod {
    AVL ArrayLetra[N];
    int TamAVL[N];
    int TotalProd;
};
typedef struct cat_prod *Cat_Produtos;
```

Este *typedef* é responsável por armazenar, num *array* de AVL'S, organizado por letra, os códigos de produto correspondentes a essa mesma letra. Note-se que, na sua totalidade, serão vinte e seis. Contém, ainda, um *array* de inteiros que indica o tamanho respetivo de cada AVL. Por último, guarda, também, o número total de produtos.

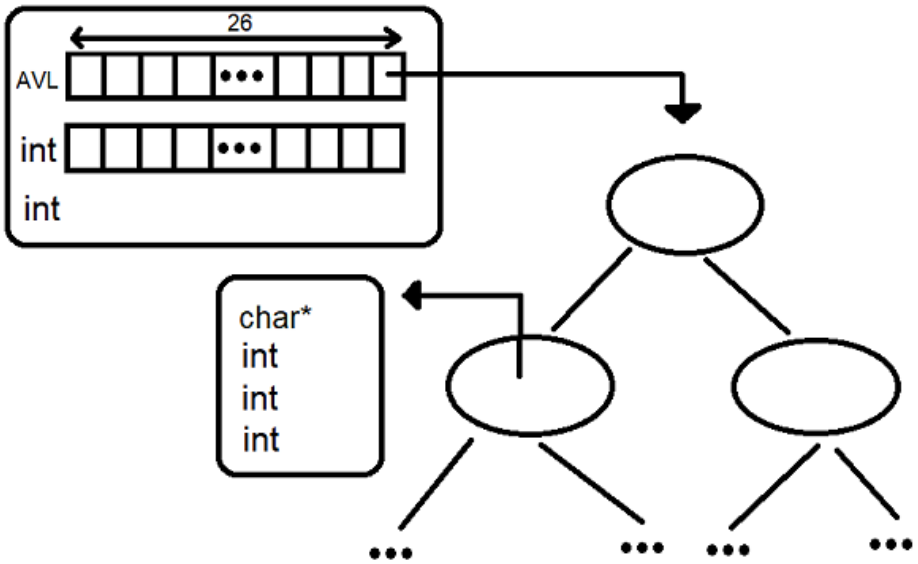

Figura 1: Estrutura do catálogo de produtos.

2.3 Módulo *Catálogo de clientes*

```
struct cli {
    char* cliente;
};
typedef struct cli *Cliente;
```

Estrutura criada com o objetivo de guardar o código de um dado cliente.

```
struct cat_cli {
    AVL ArrayLetra[N];
    int TamAVL[N];
    int TotalCli;
};
typedef struct cat_cli *Cat_Clientes;
```

Este *typedef* é responsável por armazenar, num *array* de AVL'S, organizado por letra, os códigos de cliente correspondentes a essa mesma letra. Note-se que, na sua totalidade, serão vinte e seis. Contém, ainda, um *array* de inteiros que indica o tamanho respetivo de cada AVL. Por último, guarda, também, o número total de clientes.

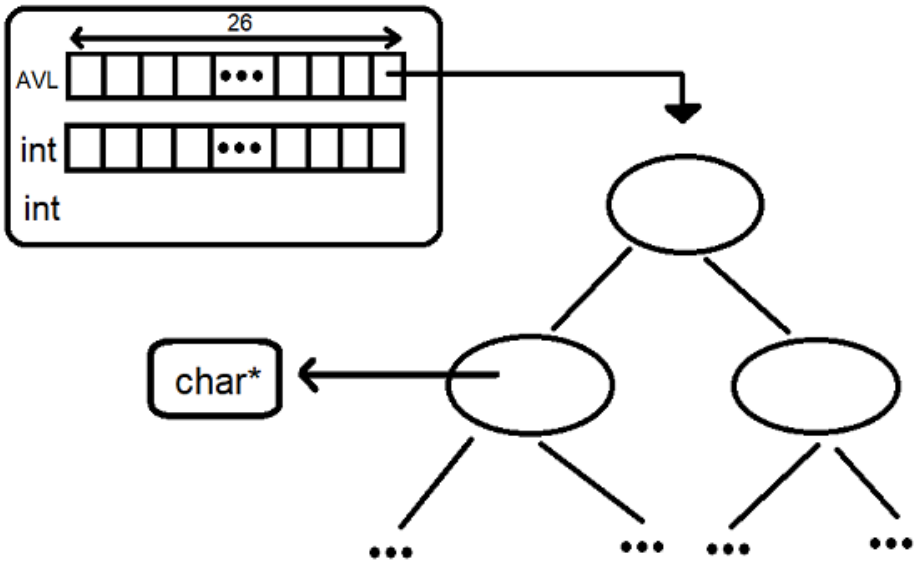

Figura 2: Estrutura do catálogo de clientes.

2.4 Módulo *Faturação*

```
 struct vendaP{
    int numVendas,quantidadeN,quantidadeP;
    double receitasN,receitasP;
};
typedef struct vendaP *VendaProd;
```

Esta *struct* é utilizada de modo a guardar os valores de uma determinada venda de um produto, sendo estes o seu número de vendas, a sua quantidade do tipo N, a sua quantidade do tipo P e, ainda, o valor das receitas do tipo N e do tipo P. De notar que esta estrutura será fundamental para a concretização de várias *querys*.

```
struct prodV{
    char* produto;
};
typedef struct prodV *ProdVenda;
```

Estrutura criada com o objetivo de guardar o código de um dado produto de uma dada venda.

```
struct fatura{
    ProdVenda prod;
    VendaProd filial[3];
typedef struct fatura *Fatura;
```

É responsável por guardar o código de um produto, assim como as suas vendas divididas pelas três filiais.

```
struct cat_fatglobal{
    AVL Mes[12];
};
typedef struct cat_fatglobal *Cat_FatGlobal;
```

Esta *struct* é constituída por um *array* de AVL's, distribuído ao longo dos doze meses sendo que, em cada um destes, é guardada a faturação dos produtos respetivos a essa mesmo mês.

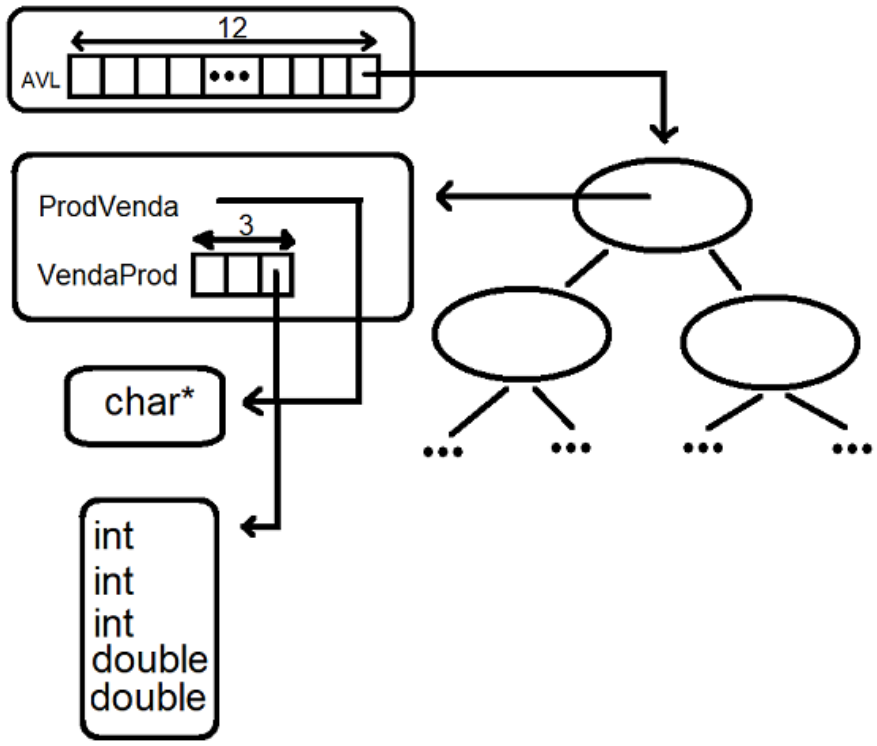

Figura 3: Estrutura da faturação.

2.5 Módulo *Filial*

```
struct Prods{
  ProdVenda produto;
  int qtdN;
  int qtdP;
  double receita;
  char* tipo;
};
typedef struct Prods *Prods;
```

Pertencente ao módulo *Filial*, este conjunto de dados, para um determinado produto em que este é guardado, possui a respetiva quantidade vendida do tipo *N*, bem como o do tipo *P*. Inclui também a faturação total do produto em causa e o seu tipo.

```
struct Clis{
  Cliente cliente;
  AVL produtosCompraram[3][12];
};
typedef struct Clis *Clis;
```

Para um determinado cliente, cujo código é guardado nesta estrutura, contém a AVL de produtos que este comprou,sendo esta distribuída não só por filiais, mas também pelos meses.

```
struct Filial{
    AVL Clis[26];
};
typedef struct Filial *Filial;
```

Estamos perante um *array* de AVL's, com vinte e seis posições,organizados pela letra de um cliente, o que se tornará eficiente para a resolução de uma grande parte das *querys*.

```
struct maisCaros{
  double gastos[3];
  ProdVenda pmaiores[3];
};
typedef struct maisCaros *MC;
```

Estrutura destinada à *query 12* que tem, por objetivo, guardar os três produtos em que mais dinheiro foi gasto sendo então criados dois *arrays*, ambos com três posições. O primeiro guarda os valores e o segundo guarda os códigos.

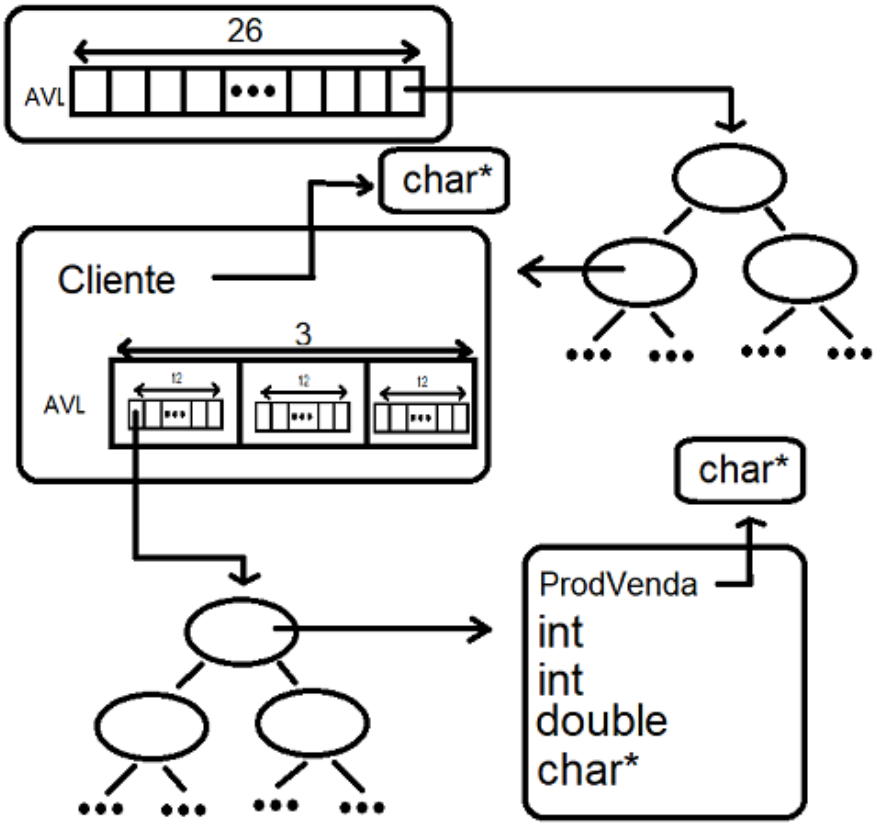

Figura 4: Estrutura da filial.

2.6 Módulo *Lista de strings*

```
typedef struct ListaProds{
    char** listP;
    int contador;
    int index;
}*ListP;
typedef struct ListaProds *ListP;
```

Struct necessária não só para a resolução de várias *querys*, mas também para a implementação do navegador, bem como todas as suas funções adjacentes. Possui, então um *array de strings*, um contador que nos indica o tamanho da mesma e um *index* necessário para a navegação.

```
struct duasList{
    ListP ln;
    ListP lp;
};
typedef struct duasList *DListP;
```

Partindo do definido em cima, e em resposta à *query 12*, esta estrutura contém duas *ListP*.

3 Análise de complexidade

3.1 Relativa à *query 8*

O tempo de execução desta *query* varia de acordo com as opções pretendidas pelo utilizador. No seu melhor caso,ou seja dados meses iguais, demora a executar, em média, cerca de 21,863 segundos, pois só é necessário percorre uma das doze AVL's. No seu pior caso, terá de as percorrrer a todas com um tempo aproximadamente doze vezes maior.

3.2 Relativa à *query 9*

Esta *query* é um pouco mais demorada que as restantes, pois, embora as AVL's estejam divididas por mês, é necessário percorrer a totalidade das mesmas. Em média, executa em $\theta(Nlog(N))$.

.

4 Discussão de resultados

4.1 *Query 1*

A *query 1* pretende que se leiam três ficheiros do formato txt. O utilizador tem a opção de escolher os ficheiros que quer ler, ou então assumir os ficheiros por omissão, correndo assim os ficheiros já pré-definidos. Para isto, optou-se por criar um menu inicial que fará a pergunta ao utilizador, agindo consoante a vontade do mesmo. Caso pretenda reinicializar o programa, então as estruturas deverão ser completamente removidas e novamente iniciadas e, consequentemente, preenchidas. Assim, desenvolveram-se funções que analisam linha a linha cada ficheiro validando e guardando cada linha na respetiva estrutura, contando ainda quantas foram lidas e quantas destas continham códigos válidos, apresentando de imediato este resultado ao utilizador. Utiliza-se para tal funções auxiliares que tratam da validação e inserção dos diferentes tipos de códigos.

4.2 *Query 2*

A *query 2* pretende que, dada uma letra, seja apresentado ao utilizador , através de um navegador, a lista de produtos e o número total da mesma. Para tal, visto que o catálogo de produtos está organizado por letra, apenas é preciso procurar pela AVL correspondente, sendo assim de resposta imediata, primando pela eficiência. Esta é guardada numa lista de strings, para ser possível, através do navegador, apresentar os códigos ao utilizador.

4.3 *Query 3*

A *query3* recebe dois *inputs*, um mês e um código produto de modo a apresentar ao utilizador o número total de vendas e o total faturado, distinguido entre N e P. Com a faturação organizada por mês, já só é necessário obter os dados relativos a esse mês, estando estes numa AVL, sendo assim mais eficiente. Dentro desta, é necessário recorrer a uma travessia à procura de um dado produto. Esta *query* também inclui a opção de o utilizador escolher entre valores totais, ou valor por filiais, sendo esta decisão tomada através de um interpretador de comandos. Estes valores são apresentados recorrendo a funções auxiliares.

4.4 *Query 4*

A *query 4* pretende que se apresente a lista de produtos não vendidos, assim como o seu valor total. Esta lista pode ser obtida na sua totalidade, ou dividida por filiais, consoante a vontade. Para tal, é percorrido o catálogo de produtos , verificando as *flags* de cada produto e, caso estas contivessem o valor zero, o código seria adiconado a uma lista . Por fim, apresenta-se a lista ao utilizador através do navegador.

4.5 *Query 5*

Para o desenvolvimento desta *query*, criou-se uma função que percorre a Filial na sua totalidade e verifica se respetivo código de cliente realizou compras em todas as filiais. Caso se verifique, o cliente é, então, colocado numa lista, que será depois impressa e, posteriormente, navegada.

4.6 *Query 6*

Para a sua resolução, percorre-se a Filial e procede-se à contagem do número de clientes que realizaram pelo menos uma compra. A resposta a esta *query* corresponde ao valor de clientes válidos, subtraindo o numero de clientes que realizaram qualquer compra. É necessário, ainda, percorrer o catálogo de produtos, contabilizando o número de produtos com as três flags a 0.

4.7 *Query 7*

Para esta *query*, criou-se uma função que, recebendo um código de cliente, procura esse mesmo cliente na estrutura Filial. Sendo esta organizada por letra de cliente, esta procura acaba por ser bastante rápida. Uma vez encontrado, definiu-se umas funções auxiliares para ir buscar os valores prosseguindo à impressão da respetiva tabela, em que os valores são apresentados por filiais, dando, ainda, ao utilizador a opção de escolher outro cliente.

4.8 *Query 8*

É pedido para determinar o número de vendas e o total faturado entre dois dados meses. Estando a faturação organizada por mês, esta *query* torna-se bastante eficiente. Assim sendo, percorre se a Faturação apenas entre os meses dados, obtendo assim os valores necessarios, apresentando os mesmos ao utilizador.

4.9 *Query 9*

Dado um código de produto e uma filial, é pretendido determinar os clientes que o compraram, distinguindo as compras N e P. Esta *query* é pouco eficiente pois terá de "varrer" a Filial por completo, uma vez que é necessário procurar por um determinado produto, em todos os clientes. Para a realização do pedido, cria-se duas listas, uma para compras N e outra para compras P. De seguida, faz-se a inserção de cada cliente (caso este tenha comprado o produto dado) na respetiva lista.

4.10 *Query 10*

Esta *query* pretende que, dado um código de cliente e um mês, seja apresentada a lista de produtos mais comprados (por quantidade), por ordem descendente. Para esta *query* a resposta é rápida pois a filial está organizada por cliente e, por sua vez, dentro de cada cliente, organizada por mês. Ou seja, percorre-se a AVL que corresponde à letra do cliente, até encontrar o cliente certo. Dentro desse cliente basta apenas percorrer a AVL correspondente ao mês fornecido, nas três filiais.

4.11 *Query 11*

O objetivo desta *query* é apresentar a lista dos N produtos mais vendidos, apresentando o número total de clientes e número de unidades vendidas, filial a filial. Para a resolução desta *query* tínhamos como ideia inicial criar uma *mini-heap*, com N elementos e ir substituindo o valor à cabeça, conforme este fosse menor (em unidades vendidas) ao novo elemento, reordenando de seguida a *mini-heap*. No final, ficaríamos apenas com os N produtos mais vendidos e respetiva faturação. Uma vez que não foi possível implementar este método, surgiu um plano B, menos eficiente. Consistia em percorrer a Faturação e ir colocando cada produto e respetiva faturação num *array*, ordenando esse *array* a cada iteração. Por último, surgiu ainda um plano C também pouco eficiente. O objetivo era a criação de uma nova AVL, ordenada por quantidades, sendo que esta iria ficar "gigante". No final, teríamos que colocar essa AVL numa lista, mas colocariamos apenas os N elementos pedido. O pouco tempo disponível impossibilitou a implementação deste método.

4.12 *Query 12*

Esta *query* pretende que, dado um código de cliente, seja apresentado ao utilizador os três produtos em que mais gastou dinheiro. Sendo a Filial organizada por letra de cliente, esta *query* também é bastante rápida. Utiliza-se uma nova estrutura que contém dois *arrays*, um para guardar os códigos de produtos e outro para guardar os respetivos valores. Depois disso, é ir comparando os valores um a um, e, caso seja maior, insere nessa posição, dando *shift* nas outras (sendo a menor quantidade apagada). Posteriormente atualiza o código sendo que, no final, apresenta-se ao utilizador os códigos obtidos.

Tabela 1: Testes de performance

Query	**Tempo (ms)**
6	14,401
7	0,400
8	21,863——239,071
9	123,672
10	0,029
11	-
12	0,057

5 Conclusão

Quanto à eficiência do projeto, no geral, consideramos que foi positiva, pois permite carregar os ficheiros numa média de 6,2 segundos pelo que ainda resolve as queries em tempo aceitável. Quanto à parte pedagógica, este projeto foi muito enriquecedor para o nosso coletivo, pois melhorou a nossa capacidade de trabalhar com estruturas de dados, assim como fez com que adquirissemos agilidade com a linguagem de programação *C*. Permitiu também que descobríssemos propriedades sobre esta que desconhecíamos até ao momento. Para além disso, ensinou-nos também a fazer o tratamento de grandes quantidades de informação, tendo sido bastante desafiante encontrar um equilíbrio entre a qualidade e eficiência. Em suma, o esforço coletivo foi grande, de modo a garantir boas soluções para o proposto, mas que, por fim, consideramos que os objetivos principais foram cumpridos.