Universidade do Minho
Departamento de Informática

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática 3

Primeiro projeto - C

A82238 João Gomes
A80376 Pedro Pereira

Braga
2020

Conteúdo

1 Introdução

Neste primeiro projeto, foi-nos pedido para fazer uso de conceitos como Modularidade, Encapsulamento e o Modelo de Controlador para fazer um projeto em C, cujo objetivo é ler e dar parse a ficheiros de grande tamanho, guarda-los em estruturas de dados de forma a poder responder a Queries referente à informação contida nestes ficheiros.

2 Interpretação do Enunciado

Após ler e interpretar o enunciado, chegamos à conclusão que no que toca aos ficheiros *clientes.txt* e *produtos.txt*, a única dificuldade seria agrupa-los numa estrutura que permitisse ordena-los facilmente, tivesse um tempo de inserção rápido e permitisse fazer um *lookup* rápido, já que para cada linha do ficheiro de Vendas, teriamos de aceder ao catalogo de produtos e clientes para verificar se um dado código existe.

No que toca ao ficheiro *vendas.txt*, alguns problemas surgiram logo à partida após a leitura das Queries interativas. Havia muitas possibilidades possiveis. Podiamos ter uma estrutura em que se guardava individualmente toda a informação de cada venda, mas descartamos logo esta possibilidade pois aparentava ser muito ineficiente. Restou-nos então decidir se iriamos organizar a informação de vendas por cliente, por produto ou então ter ambos e em troca ter alguma informação redundante em várias estruturas.

3 Descrição do Espaço de Desenho

3.1 Modularidade

Um dos conceitos mais importantes abordados neste Unidade Curricular foi o conceito de Modularidade, que consiste na separação do código fonte em vários ficheiros, normalmente agrupados por utilidade ou funcionalidade.

Apesar de ser um conceito fácil de captar e compreender, revelou ser muito mais dificil de o aplicar à partida, visto que nunca antes tivemos de estruturar o nosso código de uma maneira tão rigida e específica.

Estruturamos então o nosso projeto em módulos, como pedido no enunciado. São então os módulos:

- **Módulo de Leitura e load da Estrutura**: Módulo composto pelos ficheiros *reader.c* e *reader.h* responsáveis por fazer a leitura e parse dos códigos dos ficheiros fornecidos e de guardar a informação nas nossas estruturas de dados.
- **Catalogo de Produtos**: Módulo composto pelos ficheiros *catalogoProdutos.c* e *catalogoProdutos.h* que contêm as estruturas que guardam os códigos dos produtos por ordem alfabética e que contém as funções que permitem interagir com a estrutura.
- **Catalogo de Clientes**: Módulo composto pelos ficheiros *catalogoClientes.c* e *catalogoClientes.h*, que tal como o catalogo de Produtos, contêm as estruturas que guardam os códigos dos clientes por ordem alfabética que contém as funções que permitem interagir com a estrutura.
- **Faturação Global**: Módulo composto pelos ficheiros *faturacaoGlobal.c* e *faturacaoGlobal.h*, que contém as estruturas responsáveis por dar resposta a todas as queries relacionadas com faturação e que não mencionem os clientes.
- **Gestao de Filial**: Módulo composto pelos ficheiros *gestaoFilial.c* e *gestaoFilial.h* que contém as estruturas responsáveis por dar resposta às queries relacionadas com faturação que mencionem tanto produtos como clientes.
- **Sistema de Gestão de Vendas**: Módulo composto pelos ficheiros *sistemaGestaoVendas.c* e *sistemaGestaoVendas.h* que atua como **modelo** do projeto. Contém a estrutura SGV que incorpora todas as outras estruturas e este é o responsável por obter respostas às queries pedidas pelo user.
- **Controlador**: Módulo composto pelo ficheiro *controlador.c* que é o responsável por pedir informação sobre as respostas às queries ao Modelo.

3.2 Estruturas de Dados

No que toca às estruturas de dados a usar, a nossa primeira abordagem não foi conseguida com sucesso. Começamos por criar duas estrutura que continham um array de Strings onde guardariamos todos os códigos válidos de produtos e clientes, ordenados alfabéticamente. Não conseguimos desenvolver uma função de lookup eficiente o suficiente, o que levou a que os tempos de read e load do ficheiro de vendas fossem perto dos 5 minutos, o que não é aceitável. Após este tentativa falhada, decidimos usar as estruturas da biblioteca **Glib**, pois oferecem-nos a eficiência que procuramos.

- **Catalogo de Produtos e Clientes**: Para guardar a informação dos códigos de clientes e produtos, usamos a estrutura **GSequence**, que atua como uma árvore balanceada. Escolhemos esta estrutura pois fornece funções de tempos de lookup muito eficientes, que é o pretendido deste módulo. Após a leitura de uma linha do ficheiro *clientes.txt* e *produtos.txt*, procedemos à sua validação e depois adicionamos à estrutura correspondente usando a funçao g_sequence_append, que insere um elemento na GSequence de forma ordenada.

- **Faturação Global**: Para guardar a informação relacionada com cada produto e as suas vendas e faturações, decidimos usar uma árvore binária balanceada do Glib, **GTree**. Esta estrutura é uma árvore que apresenta funcionalidades de um **HashMap** pois cada node na árvore é um par de (key,value). Foi este pormenor que nos levou a usar esta estrutura pois permitiu-nos associar a cada produto, uma estrutura por nós feita, **struct faturação** no ficheiro faturacaoGlobal.c. Esta estrutura é constituida por 4 arrays, 2 de inteiros e 2 de floats, alocados dinamicamente. Os arrays de inteiros *vendasN* e *vendasP* representam respetivamente, a quantidade de vendas que o produto em questão tem em modo normal ou em modo Promoção. Os arrays de floats *totalN* e *totalP* representam respetivamente, a faturação feita que o produto em questão teve em modo normal e em modo Promoção. A introdução de dados nesta estrutura dá-se da seguinte maneira: quando uma linha de vendas válida é lida, é feito um lookup na **GTree** que nos dá acesso à estrutura *Faturacao* associada ao produto lido na linha de vendas, permitindo-nos depois apenas somar os valores lidos na linha, aos valores já guardados na estrutura.

- **Gestão de Filiais**: Este conjunto de estruturas foi o que nos deu mais trabalho e causou mais problemas. Começamos por ponderar guardar a informação em função de cada cliente ou em função de cada produto, mas rapidamente nos apercebemos que havia várias queries feitas em função ou de um ou de outro. Decidimos então, ter duas árvores binárias balanceadas diferentes. Uma em função dos produtos e outra em função dos clientes. Isto apesar de resultar na duplicação de alguma informação e na perda de algum tempo de load resultante da inserção da mesma informação em várias estruturas diferentes, acabou por valer a pena, pois resultava numa maior eficiência nas respostas às queries. Temos portanto, duas árvores **GTree**. Os nodes da primeira, têm como key o código de um produto e como value têm outra árvore. As nodes desta árvore, por sua vez, têm como key o código de um cliente e como value uma estrutura *Vendas*, parecida com a estrutura *Faturação* mencionada em cima, que tem 4 arrays com os valores das vendas. A segunda tree é o oposto desta primeito. Cada node tem como key um código de um cliente e como value também uma árvore. As nodes desta segunda árvore, por sua vez, têm como key o código de produto e como value a estrutura *Vendas* referida em cima. Estas estruturas dão-nos a flexibilidade de poder percorrer toda a informação de vendas a partir de um dado cliente ou produto, consoante a nossa necessidade. A inserção de informação nesta estrutura é feita da seguinte forma: quando lida uma linha de vendas válida, consoante o valor da failial, a informação será inserida na estrutura correspondente. Depois, a informação é inserida em ambas as trees ao mesmo tempo. Na primeira tree, usa-se o código do produto para fazer lookup e encontrar a tree referente a esta. Depois usa-se o código do cliente para fazer lookup e encontrar a tree referente a este cliente, podendo-se depois dar fetch à estrutura *Vendas* associada a este produto e cliente, e acrescentado os valores correspondentes a esta venda. A inserção na segunda tree é feita de igual modo, mas com os valores de cliente e produto trocados.

- **Sistema de Gestão de Vendas (SGV)**: Esta é apenas uma estrutura que dá wrap a todas as outras estruturas, que resulta numa estrutura que dentro terá, uma estrutura de catalogo de produtos, uma de catalogo de clientes, uma de faturação global e três estruturas de Gestão de Filial, uma para cada filial.

4 Escolha Justificada da Solução Adotada

Inicialmente foi planeado um método com estruturas criadas por nós. No entanto, à medida que foram postas em práticas as várias estratégias de resolução das *queries* foi constatado que a eficiência desta estruturas não era a melhor.

Desta maneira, foi decidido alterar parte das estruturas para as já existentes do **GLib**, uma vez que estas teriam uma eficiência e acessibilidade superior.

Quanto ao tipo de estruturas escolhidas, foi tido em conta que estruturas como a lista ligada tornariam complicado aceder a partes da estrutura sem percorrer a mesma do ínicio. Assim, foram usadas a **GTree** e o **HashMap** que são mais adequados ao que se pretende neste trabalho.

Os resultados obtidos nos testes que fizemos dão-nos também segurança que a solução apresentada vai de encontro com os nossos objetivos.

5 Descrição da Estrutura do Projeto

A estrutura deste projeto foi feita tendo em conta o modelo MVC (Model View Controller).

Neste modelo, o Controlador é o responsável por pedir ao Modelo as respostas das queries pretendidas e depois passa estas respostas à View que é a responsável por apresentar os resultados ao utilizador. Neste trabalho, não tivemos tempo para implementar a View, acabando por quebrar este Modelo. Ficamos então apenas com o Modelo e o Controlador, sendo que a forma como a informação é apresentada ao User é feita no Modelo, nas funções que respondem às Queries.

Sabemos que a forma correta seria, o Modelo devolver ao Controlador como resultado da Query uma estrutura com a informação. O controlador depois passaria à View essa estrutura e seria responsabilidade da View o print da informação para o utilizador ver.

No nosso projeto em especifico, o Controlador encontra-se implementado no ficheiro *controlador.c* e o modelo encontra-se implementado no ficheiro *sistemaGestaoVendas.c* onde se encontra a estrutura que contém todas as outras estruturas e as funções que respondem às queries.

6 Testes de Desempenho

Realizamos apenas algunst testes de desempenho, relativos ao tempo de leitura e load da informação dos ficheiros, e ao tempo de execução das Query 6 e 12.

Relativo ao tempo de dar read e load à informação nas estruturas, fizemos 3 testes para cada ficheiro.

- **1 Milhão de entradas**: Os valores obtidos foram: 9.495565 segundos, 9.650223 segundos e 9.479384 segundos, resultando numa média de 9.541724 segundos.
- **3 Milhões de entradas**: Os valores obtidos foram: 30.824088 segundos, 30.853567 segundos e 30.879946 segundos, resultando numa média de 30.852533 segundos.
- **5 Milhões de entradas**: Os valores obtidos foram: 55.82239 segundos, 52.628500 segundos e 53.203128 segundos, resultando numa média de 53.641289 segundos.

Relativamente à query4, fizemos 3 testes para cada ficheiro.

- **1 Milhão de entradas**: Os valores obtidos foram: 0.261834 segundos, 0.256985 segundos e 0.256157 segundos, resultando numa média de 0.258325 segundos.
- **3 Milhões de entradas**: Os valores obtidos foram: 0.265824 segundos, 0.264228 segundos e 0.267713 segundos, resultando numa média de 0.265921 segundos.

- **5 Milhões de entradas**: Os valores obtidos foram: 0.263212 segundos, 0.269368 segundos e 0.267809 segundos, resultando numa média de 0.266819 segundos.

Relativamente à query12, fizemos 3 testes para cada ficheiro.

- **1 Milhão de entradas**: Os valores obtidos foram: 0.005713 segundos, 0.017404 segundos e 0.007072 segundos, resultando numa média de 0.010063 segundos.
- **3 Milhões de entradas**: Os valores obtidos foram: 0.019289 segundos, 0.018706 segundos e 0.018201 segundos, resultando numa média de 0.018732 segundos.
- **5 Milhões de entradas**: Os valores obtidos foram: 0.230802 segundos, 0.224726 segundos e 0.221538 segundos, resultando numa média de 0.225689 segundos.

7 Interpretação

Podemos observar pelos resultados indicados acima, que o tempo de read e load da informação para os ficheiros é feito em tempo linear, multiplicando por 3 o tempo obtido quando usando o ficheiro de 3 milhões de vendas e multiplicando por 5 o tempo obtido quando usando o ficheiro de 5 milhões. Este resultado é o esperado e vai de encontro com as nossas espetativas.

O tempo de execução da query 4 é praticamente constante, mantendo-se igual qualquer que seja o número de vendas dadas.

O tempo de execução da query 12 vai aumentando de forma quase linear com o aumento do número de vendas, o que é expectavel, pois quantas mais vendas forem inseridas, mais pares cliente-produto vamos ter que não existiriam nos ficheiros mais pequenos, levando a um lookup time maior.

8 Conclusão

Este trabalho deu-nos uma perspetiva nova sobre como projetos a grande escala são realizados e planeados. Obrigou-nos a pensar e a estruturar o nosso projeto em torno de conceitos como encapsulamento e modularidade, o que se revelou uma dificuldade com a qual nós não contavamos. Revelou também uma certa inexperiência a trabalhar com a linguagem C, pois deparamo-nos muitas vezes com incapacidade de realizar algo que sabemos ser possível, além da quantidade enorme de erros com que nos deparamos constantemente.

Em suma, estamos satisfeitos com o resultado deste trabalho apesar do MVC não ter sido realizado totalmente.