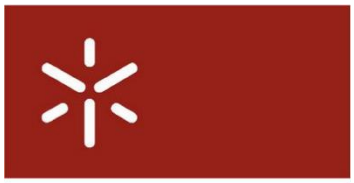


Universidade do Minho

Universidade do Minho

Relatório do Trabalho Prático LI3

Mestrado Integrado em Engenharia Informática

Grupo 31

André Morais	A83899
Luís Ribeiro	A85954
Pedro Rodrigues	A84783

Introdução

O trabalho realizado foi desenvolvido em linguagem C e teve como principal objetivo a gestão das vendas de uma distribuidora com 3 filiais. Para a concretização do objetivo, utilizamos ficheiros ".txt" com informação útil sobre clientes, vendas e produtos. Inicialmente, usamos arrays para extrair a informação dos ficheiros para a manipulação da mesma. Para facilitar a extração de informação dos ficheiros criamos algumas estruturas de dados. A exigência e a demora prolongada do tempo de execução levou ao uso da biblioteca GLib que disponibiliza acesso a estruturas já implementadas, melhorando drasticamente o tempo de execução.

Assim, diminuímos o número de estruturas criadas, tirando o máximo proveito das diferentes estruturas que esta biblioteca disponibiliza.

O objetivo deste relatório é justificar as escolhas dos tipos de dados, estruturas de armazenamento e também explicar as estratégias usadas para a resposta das questões propostas.

Módulos e Estrutura de Ficheiros

Foram criados vários módulos de forma a garantir o encapsulamento. Segue-se uma breve descrição de cada um desses módulos e estruturas de dados:

CatProds

```
typedef struct cat_prods{
  GHashTable* produtos [26][26];
  int nr_produtos;
}cat_prods;
```

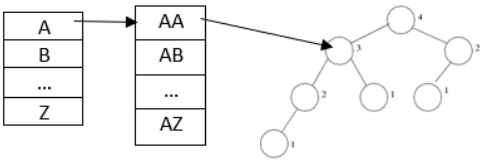
Estrutura de dados

Este módulo serve para guardar todos os produtos válidos lidos do ficheiro "Produtos.txt".

Neste módulo, inicialmente, foi adotada uma estrutura de arrays, em que foi facilmente detetado que não era a melhor opção devido aos tempos de procura elevados.

Sendo assim adotamos outro método:

- Um array com 26 posições correspondentes à primeira letra do produto.
- Cada posição desse array aponta para um novo array, novamente com 26 posições, associado à 2º letra do código do produto.
- Cada posição do segundo array aponta para uma GTree, que contém todos os produtos começados por essas letras.

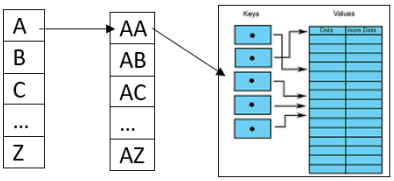



```
=> Vendas.txt :  Das 1000000 vendas lidas, 891108 sao validas

CPU Time: 8.642567
```

Tempo de execução com GTree

Mantivemos esta estrutura durante quase todo o trabalho, mas no final reparamos que poderíamos melhorar a eficiência da leitura de dados e portanto, decidimos utilizar GHashTable em vez de GTree, tendo esta um tempo de procura constante, o que nos fez ganhar 2 segundos relativamente aos 8 segundos anteriores, visto que a maior parte das funções utilizadas por este módulo são apenas de procura.



Estrutura de dados final

Na imagem seguinte, apresentamos a API do módulo:

```
Cat_Prods inicializa_CatProds();
Cat_Prods insereProd (Cat_Prods catp, Produto p);
int existeProd (Cat_Prods catp, Produto p);
int nrProds (Cat_Prods catp);
Lista_Prods produtosPorLetra (Cat_Prods catp, char letra);
GHashTable* getProdutosLetra (Lista_Prods lst, int i);
```

API

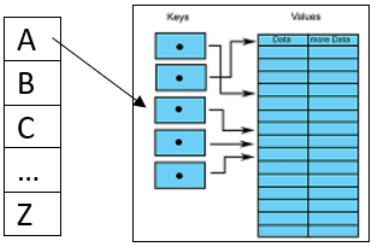
CatCli

```
typedef struct cat_cli{
  GHashTable* clientes [26];
  int nr_clientes;
}cat_cli;
```

Estrutura de dados

Este módulo é idêntico ao anterior e, portanto, utilizamos sempre a mesma estratégia e estruturas de dados. Serve para guardar todos os clientes válidos lidos do ficheiro "Clientes.txt".

Tal como o módulo acima, foi adotada inicialmente uma estrutura de arrays. Mais tarde um array em que cada posição correspondia à primeira letra do código do cliente e apontava para uma GTree que guardava todos os códigos começados por essa letra. No final, para melhorar o desempenho de procura, alteramos, tal como anteriormente, a GTree por uma GHashTable e as diferenças foram notórias.



Para este módulo definimos a seguinte API:

```
Cat_Cli inicializa_CatCli();
Cat_Cli insereCli(Cat_Cli catc, Cliente c);
int existeCli (Cat_Cli catc, Cliente c);
GHashTable* clientesLetra(Cat_Cli catc, char l);
int nrCli (Cat_Cli catc);
```

API

Clientes

Este módulo é utilizado para encapsular o cliente. Definimos o cliente como uma string (ou char*) e definimos a seguinte API para que os outros módulos pudessem utilizar a estrutura Clientes:

```
Cliente criaCliente (char* cliente);
char* getCliente (Cliente c);
char primeiraLetraCli (Cliente c);
int validaCliente (Cliente c);
```

API

Produto

Tal como o anterior, é utilizado para encapsular o produto, sendo este definido como uma string (char *) e foi definida a seguinte API:

```
Produto criaProduto (char * produto);
char* getProduto (Produto p);
char primeiraLetra (Produto p);
char segundaLetra (Produto p);
int validaProduto (Produto p);
```

API

Venda

Este módulo serve para encapsular o tipo Venda, que contém uma estrutura diferente das outras:

```
typedef struct vend{
  char* produto;
  float preco;
  int unidades;
  char compra;
  char *cliente;
  int mes;
  int filial;
}vend;
```

Estrutura de dados

Perante a requisição dos outros módulos, foi definida a seguinte API:

```
Venda criaVenda (char* venda,Cat_Prods catp,Cat_Cli catc);
int getUniVenda (Venda v);
float getPrecoVenda (Venda v);
char * getProdVenda (Venda v);
int getMes (Venda v);
char getTipoVenda (Venda v);
char * getCliVenda (Venda v);
int getFilial (Venda v);
```

API

Faturação

Este módulo guarda a informação correspondente às vendas, organizadas consoante o produto vendido.

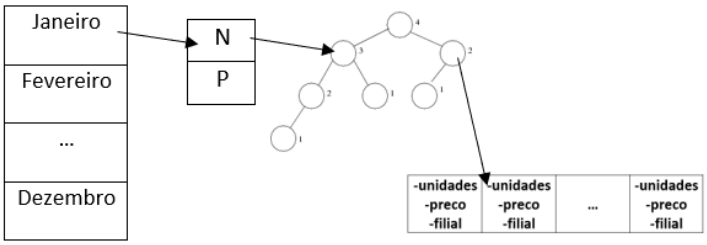
```
typedef struct faturacao{
  /*
     N 0
     P 1
  */
  GTree* fat [12][2];
  GArray* nc;
}faturacao;

typedef struct fatnode{
  GPtrArray* fatvendas;
}fatnode;

typedef struct fat_str{
  int unidades;
  float preco;
  int filial;
}fat_str;
```

Estrutura de dados

Neste módulo decidimos organizar a informação por meses e tipo de compra. Assim, a estrutura de dados da *Faturacao* é composta por um array com 12 posições corresponde ao mês em que a venda foi feita em que cada posição aponta para um array com 2 posições distinguido assim a venda do tipo normal e em promoção. Cada posição desse array aponta para uma *GTree* onde as vendas são organizadas por produtos, ou seja, a key da *GTree* corresponde ao código do produto vendido e os dados dentro do nodo são guardados num *GPtrArray* que contém uma estrutura com as unidades, o preço e a filial correspondente a essa venda.



Estrutura de dados Final

O *GArray nc* presente na estrutura da *Faturacao* corresponde ao código dos produtos que nunca foram vendidos, é atualizado quando é pedida a query correspondente e mantido durante o correr do programa, aumentando assim a execução dessa query caso esta seja corrida mais do que uma vez.

Para este módulo, achamos adequada a seguinte API:

```
Faturacao inicializaFaturacao();
Faturacao insereVenda(Faturacao fatp, Venda v);
int nrVendaProduto(Faturacao fatp, Produto p, int mes, char tipo);
float fatProduto(Faturacao fatp, Produto p, int mes, char tipo);
int atualizanc (Faturacao fatp, Cat_Prods catp);
int getNaoComprados_nr (Faturacao fatp);
GArray* getNaoComprados (Faturacao fatp);
int getVendasEntre(Faturacao fatp, int x, int y);
float getFatEntre(Faturacao fatp, int x, int y);
GArray* getNaoCompFil(Faturacao fatp, Cat_Prods catp, int filial);
```

API

Filial

```c
typedef struct filial{
  /*
    N 0
    A
    P 1
  */
  GTree* fil [3][12];
  GArray* compT;
  int nc;
}filial;

typedef struct fnode{
  GPtrArray* filvendas;
}fnode;

typedef struct fi_str{
  char* p;
  char tipo;
  int unidades;
  float preco;
}fi_str;
```

Estrutura de dados

Este módulo tem como objetivo guardar as informações da compra organizadas filial, contendo as informações de cliente e produto.

A sua estrutura de dados é muito semelhante à *Faturacao*, temos um array que aponta para outro array que aponta para uma *GTree* de *GPtrArray*.

Assim sendo, a estrutura é composta por um array de 3 posições que correspondem às filiais e divide as vendas pelas três filiais. Cada uma dessas posições aponta novamente para um array, em que cada posição representa o mês em que a venda foi registada, ou seja, será um array de 12 posições. Cada posição aponta para uma *GTree* onde estão organizadas por código de cliente, ou seja, a key de cada nodo da *GTree* é o código do cliente, dentro de cada nodo temos um *GPtrArray* que contém a informação do produto, do tipo da venda, das unidades e do preço unitário.

A estrutura contém ainda um *GArray* compT que guarda os clientes que comparam em todas as filiais e o inteiro nc que guarda os clientes que não compraram.

Definimos então a seguinte API para tornar possível a execução das queries:

```c
Filial inicializaFilial();
Filial insereFilV (Filial f, Venda v);
int nr_clientesCompraram(Filial f, int filial, int mes);
GArray* clientesCompraram(Filial f, int filial, int mes);
int nr_compraCli (Filial f, Cliente c, int mes, int filial);
int fat_compraCli (Filial f, Cliente c, int mes, int filial);
int cliente_nc(Filial f, Cat_Cli catc);
GArray* getCompT(Filial f, Cat_Cli catc);
int vendasProdutoFil(Filial f, Produto p, int mes, char tipo, int fil);
float fatProdutoFil(Filial f, Produto p, int mes, char tipo, int fil);
GArray* cliCompProd (Filial f, Produto p, int filial, char tipo);
GArray* tresProdutos (Filial f, Cliente c);
GArray* prodCompDec (Filial f, Cliente c, int mes);
GArray* prodNcomp (Filial f, int filial, int N);
```

API

Navegador

Neste módulo contemos todas as funções referentes a menus. Inicialmente é chamada no início do programa a função que apresenta o menu inicial, que está ilustrado na figura abaixo, e, consoante a escolha do utilizador, é chamada a função do menu da respetiva query onde se pede ao utilizador os argumentos necessários para a execução da mesma.

Menu Inicial

Queries

Este módulo contém todas as respostas a todas as queries e apresenta os resultados para o utilizador, definimos então a seguinte API para que fosse possível executar as queries no nosso *Navegador.*

```
int querieUmP (Cat_Prods catp);
int querieUmC (Cat_Cli catc);
int querieUmV (Faturacao fatp, Filial f, Cat_Prods catp, Cat_Cli catc);
int querieDois (Cat_Prods catp, char l, int x, int y);
int querieTres (Faturacao fatp, Filial f, Produto p, int mes);
int querieCinco (Filial f, Cat_Cli catc, int x, int y);
int querieQuatro (Faturacao fatp, Cat_Prods catp, int x, int y, int fil);
int querieSeis (Filial f, Faturacao fatp, Cat_Cli catc);
int querieSete (Filial f, Cliente c);
int querieOito (Faturacao fatp,int x,int y);
int querieNove (Filial f, Produto p, int filial, char tipo, int x, int y);
int querieDez (Filial f, Cliente c, int mes);
int querieOnze (Filial f, int filial, int n);
int querieDoze (Filial f, Cliente c);
```

→ API

→ Query 1:

- Implementamos funções de criação e validação, onde a função de criação aloca memória para a estrutura da informação extraída do ficheiro e a função de validação valida essa informação.

→ Query 2:

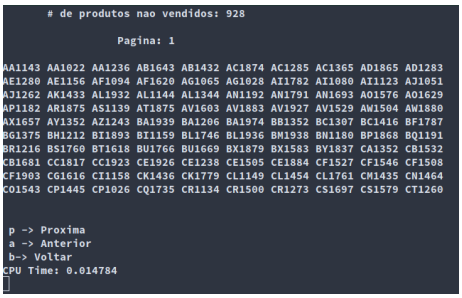
- Consoante a primeira, localizamos facilmente a GHashTable corresponde aquelas iniciais. Em seguida apenas precisamos de percorrer o array que dá informação da segunda letra e imprimir os elementos da GHashTable presente em cada uma dessas posições.

→ Query 3:

- Conforme o mês e o tipo de venda recebido como argumento, localiza a GTree que contém toda a informação das vendas daquele produto naquele mês e daquele tipo. A partir daqui bastou utilizar funções da glib que procuram pela chave que, neste caso, é o código do produto e imprime o tamanho do array com as vendas bem como a faturação total com as vendas desse produto, nesse mês.

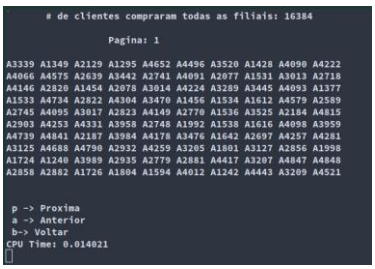
→ Query 4:

- Antes de qualquer execução, é verificado se o array de não comprados presente na estrutura da Faturacao está vazio, se não estiver, imprime os resultados já calculados anteriormente. Se estiver vazio, vai procurar pelos produtos não comprados, sendo assim, com funções de travessia da glib, percorremos o catálogo de produtos e verificamos o número de vendas para esse produto.



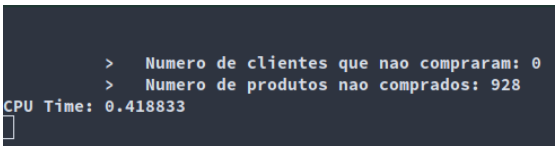
→ Query 5:

- Recebendo o catálogo de clientes, percorre todas as árvores com funções da glib e verifica se cada cliente vendeu em todas as filiais. Se vendeu então escreve-o no array presente na estrutura da Filial. O que facilita caso esta query seja usada várias vezes durante a execução do programa.



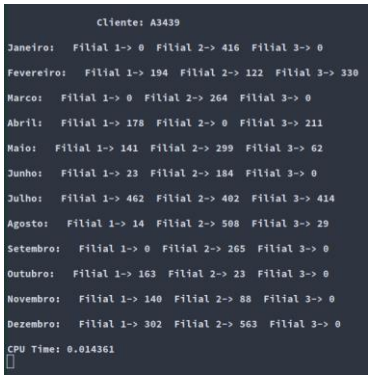
→ Query 6:

- Esta query foi das mais simples de implementar porque apenas usamos duas funções, uma já usada na query 4, e outra que conta o número de clientes que não compraram através de uma travessia ao catálogo de clientes e Filial através de funções da glib.



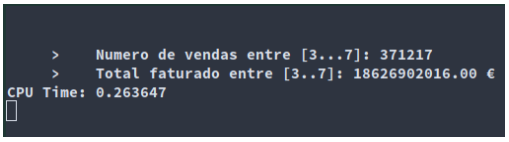
→ Query 7:

- Aqui é simples, basta procurar o cliente recebido em cada mês e em cada filial, percorrer o GPtrArray e ir sempre somando o número de unidades vendidas. Imprimindo depois todas as informações de forma organizada.



→ Query 8:

- Percorre as arvores dos dois tipos naqueles meses recebidos como argumento com funções da glib e, para o número de vendas, calcula apenas o tamanho do array, para a faturação, percorre o array com as vendas e multiplica o preço unitário com as unidades vendidas.

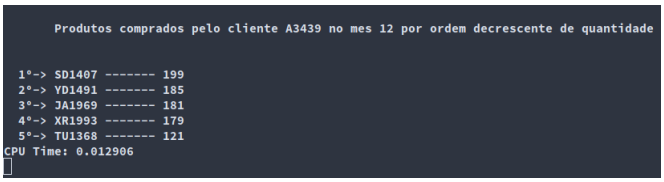


→ Query 9:

- Usamos uma estrutura auxiliar, que tem um GArray e os argumentos da função de modo a ser possível fazer a travessia das GTree com as funções implementadas na glib. Assim, à medida que percorremos as GTree verificamos se o produto e o tipo de compra correspondem ao pedido pelo utilizador, se sim, guardamos no array.

→ Query 10:

- Foi criado um GPtrArray com uma estrutura que continha o código de produto e as unidades vendidas por este. Assim, para cada filial, é procurado o array correspondente às vendas do cliente passado como argumento. Para cada elemento do array, verificamos se o produto já foi adicionado ao GPtrArray auxiliar, se sim, apenas soma-se as unidades, se não, adicionamos o produto. No final, foi necessário criar uma função de comparação para que a glib ordenasse o array por ordem decrescente consoante as unidades vendidas. No final, simplificamos o GPtrArray para um GArray contendo só o código de produto e assim, poder ser imediatamente utilizado para imprimir no ecrã.

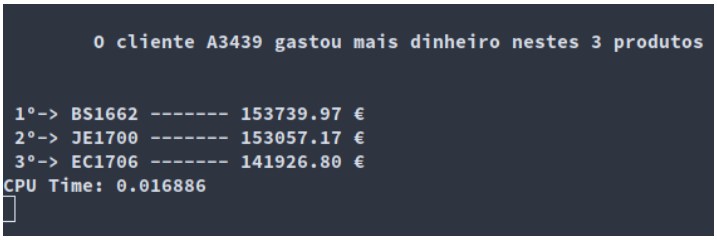


→ Query 11:

- Esta query é muito complexa. Inicialmente é iniciada uma GTree que contém em cada nodo uma struct auxiliar com código de produto, unidades vendidas e um GArray com todos os clientes que compraram aquele produto. Durante a travessia da Filial, o cliente só é inserido no array se não pertencer ao mesmo. Assim, no final, tal como na query anterior, é dado sort com funções de glib segundo as unidades vendidas e mais tarde passado para um GArray pronto a ser utilizado para imprimir no ecrã os resultados.

→ Query 12:

- Esta query foi pensada e estruturada estrategicamente da mesma maneira que a query 10, é criada uma estrutura auxiliar e é percorrida a informação. A maior diferença é que neste caso, guardamos o valor total gasto e não as unidades vendidas. No final, o GPtrArray é ordenado consoante o total gasto e é passado para um GArray pronto, novamente, a ser imediatamente utilizado para imprimir os resultados.



Profiling

Nos últimos dias do tempo limite dado pela equipa docente, conseguimos fazer um pouco de Profiling, no sentido de melhorar a eficiência e os resultados do nosso código, visto que reparamos estar a demorar demasiado tempo para carregar os ficheiros. Sendo assim conseguimos verificar alguns problemas no nosso código e corrigido alguns, apesar de termos noção que pelos resultados, muita coisa poderia ser melhorada. Alteramos os nossos catálogos de clientes e produtos para GHashTable devido ao tempo que demorava a validar as vendas e tivemos resultados significativos.

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
25.02	0.04	0.04	1000000	0.00	0.00	validaVenda
18.76	0.07	0.03	1000000	0.00	0.00	criaVenda
12.51	0.09	0.02	1171008	0.00	0.00	criaProduto
12.51	0.11	0.02	891108	0.00	0.00	insereFilV
6.25	0.12	0.01	2583165	0.00	0.00	getProdVenda
6.25	0.13	0.01	1782216	0.00	0.00	getMes
6.25	0.14	0.01	1782216	0.00	0.00	getTipoVenda
6.25	0.15	0.01	891108	0.00	0.00	insereVenda
3.13	0.16	0.01	901565	0.00	0.00	existeProd
3.13	0.16	0.01	171008	0.00	0.00	insereProd
0.00	0.16	0.00	1782216	0.00	0.00	getFilial
0.00	0.16	0.00	1782216	0.00	0.00	getPrecoVenda
0.00	0.16	0.00	1782216	0.00	0.00	getUniVenda
0.00	0.16	0.00	1350792	0.00	0.00	getCliVenda
0.00	0.16	0.00	1243581	0.00	0.00	getProduto
0.00	0.16	0.00	1072573	0.00	0.00	primeiraLetra
0.00	0.16	0.00	1072573	0.00	0.00	segundaLetra
0.00	0.16	0.00	1032768	0.00	0.00	getCliente
0.00	0.16	0.00	1016384	0.00	0.00	criaCliente
0.00	0.16	0.00	1016384	0.00	0.00	primeiraLetraCli
0.00	0.16	0.00	1000000	0.00	0.00	existeCli
0.00	0.16	0.00	171008	0.00	0.00	validaProduto
0.00	0.16	0.00	16384	0.00	0.00	insereCli
0.00	0.16	0.00	16384	0.00	0.00	validaCliente
0.00	0.16	0.00	1	0.00	0.00	querieUmC
0.00	0.16	0.00	1	0.00	160.10	querieUmMenu
0.00	0.16	0.00	1	0.00	7.93	querieUmP
0.00	0.16	0.00	1	0.00	152.18	querieUmV

Quando o CatProds e o CatCli eram compostos por GTree os resultados eram estes

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
50.03	0.27	0.27				prod_n
12.97	0.34	0.07	721028	0.00	0.00	elemArr
9.27	0.39	0.05				cmp_n_h
5.56	0.42	0.03	891108	0.00	0.00	insereVenda
3.71	0.44	0.02	1000000	0.00	0.00	criaVenda
3.71	0.46	0.02	1000000	0.00	0.00	validaVenda
3.71	0.48	0.02	891108	0.00	0.00	insereFilV
2.78	0.50	0.02	1016384	0.00	0.00	criaCliente
2.78	0.51	0.02	171008	0.00	0.00	validaProduto
1.85	0.52	0.01	1782216	0.00	0.00	getTipoVenda
1.85	0.53	0.01	1171008	0.00	0.00	criaProduto
1.85	0.54	0.01	901565	0.00	0.00	existeProd
0.00	0.54	0.00	2583165	0.00	0.00	getProdVenda
0.00	0.54	0.00	1782216	0.00	0.00	getFilial
0.00	0.54	0.00	1782216	0.00	0.00	getMes
0.00	0.54	0.00	1782216	0.00	0.00	getPrecoVenda
0.00	0.54	0.00	1782216	0.00	0.00	getUniVenda
0.00	0.54	0.00	1350792	0.00	0.00	getCliVenda
0.00	0.54	0.00	1243581	0.00	0.00	getProduto
0.00	0.54	0.00	1072573	0.00	0.00	primeiraLetra
0.00	0.54	0.00	1072573	0.00	0.00	segundaLetra
0.00	0.54	0.00	1032768	0.00	0.00	getCliente
0.00	0.54	0.00	1016384	0.00	0.00	primeiraLetraCli
0.00	0.54	0.00	1000000	0.00	0.00	existeCli
0.00	0.54	0.00	171008	0.00	0.00	insereProd
0.00	0.54	0.00	16384	0.00	0.00	insereCli
0.00	0.54	0.00	16384	0.00	0.00	validaCliente
0.00	0.54	0.00	26	0.00	0.00	getProdutosLetra
0.00	0.54	0.00	1	0.00	0.00	prodNcomp
0.00	0.54	0.00	1	0.00	0.00	produtosPorLetra
0.00	0.54	0.00	1	0.00	0.00	querieDois
0.00	0.54	0.00	1	0.00	0.00	querieDoisMenu
0.00	0.54	0.00	1	0.00	0.00	querieOnze
0.00	0.54	0.00	1	0.00	0.00	querieOnzeMenu
0.00	0.54	0.00	1	0.00	0.24	querieUmC
0.00	0.54	0.00	1	0.00	150.09	querieUmMenu
0.00	0.54	0.00	1	0.00	16.47	querieUmP
0.00	0.54	0.00	1	0.00	133.38	querieUmV

Demasiado tempo passado numa função responsável pela query 11, seria um caso a estudar com tempo

Conclusão

Concluído o trabalho, entendemos que a solução apresentada é eficiente e bem estruturada. Foram efetuadas várias medições de tempo, para além do *profiling*, que comprovam a boa performance das abordagens seguidas nas várias queries. A nível de memória, houve o cuidado de dar *free* das estruturas usadas para evitar redundâncias e alocações de memória desnecessárias. Houve certamente uma aprendizagem notória relativamente às estruturas implementadas da biblioteca glib, e tentou fazer-se o uso máximo dessas estruturas, tirando assim partido da estabilidade e a fiabilidade das mesmas. Apesar de muito código reescrito e melhorado, haveria certamente espaço para melhorar, como o caso da query 11.

No geral faz-se uma avaliação muito positiva das diversas métricas consideradas e entende-se que a abordagem ao problema foi concisa e direta.