Universidade do Minho

Departamento de Informática

Laboratórios de Informática III

Grupo 4

Gonçalo Costa	José Oliveira	Pedro Lopes
a100824	a100547	a100759

Índice

1. Introdução..... 2
   1.1. Descrição do problema..... 2
   1.2. Análise do problema..... 2
   1.3. Análise das Soluções..... 3
   1.4. Soluções abandonadas..... 3
2. Código do projeto..... 4
   2.1. Parser..... 4
      2.1.1 Tratamento dos Dados..... 4
         2.1.1.1 Transformação em Structs..... 4
         2.1.1.2 Criação das DataBases..... 5
   2.2. Estatísticas..... 5
   2.3. Dates..... 5
   2.4. Queries..... 6
      2.4.1 Queries 4 e 7..... 6
      2.4.2 Queries 5,6 e 9..... 6
      2.4.3 Query 7..... 7
   2.5. Main..... 7
   2.6. Modo Interativo..... 8
      2.6.1. Paginação..... 8
   2.7 Tabela de Performance..... 9
   2.8 Mudanças que poderíamos ter feito..... 10
3. Conclusão..... 11

Chapter 1

1. Introdução

O presente relatório tem como objetivo apresentar o projeto realizado no âmbito da unidade curricular de Laboratórios de Informática III, ao longo do primeiro semestre, do segundo ano da Licenciatura em Engenharia Informática da Universidade do Minho. Este consiste na criação de um programa capaz de ler, armazenar e gerir toda a informação válida contida em ficheiros CSV e devolver os dados pretendidos pelo utilizador da forma mais eficiente e rápida possível, tentando manter os dados encapsulados.

1.1. Descrição do Problema

Os 3 ficheiros CSV deverão ser processados: users.csv, drivers.csv e rides.csv, cada linha nesses ficheiros deverá conter um registo com a informação de um utilizador, condutor e viagem, respetivamente. Um dos objetivos deste trabalho é armazenar toda a informação contida nesses ficheiros nas estruturas de dados mais adequadas para cada fim e responder aos pedidos do utilizador sobre as mesmas.

1.2 Análise do Problema

Este projeto foi dividido em 3 problemas chave que foram precisos resolver para completar o trabalho. O primeiro problema foi a escolha das estruturas de dados adequadas para responder às queries dadas. O segundo grande problema foi a resposta às mesmas, que exigia a implementação de diferentes estratégias para podermos responder. O terceiro grande problema foi a gestão de memória, visto que a quantidade de dados que era preciso armazenar era muito grande, e tínhamos de ter cuidado com memory leaks.

1.3 Análise das Soluções

Para responder a cada um destes problemas, nós desenvolvemos as seguintes soluções.

1. Para estruturas de dados, nós escolhemos utilizar arrays e hashtables, visto que o array é uma boa estrutura para a ordenação, e também para a travessia ordenada. A hashtable, a segunda estrutura escolhida, permitia o acesso em tempo constante aos dados de qualquer utilizador.
2. Para as respostas às queries, nós separamo-las em 2 estilos diferentes de perguntas: as perguntas de acesso, que implicariam uma resposta cujo valor poderia ser facilmente pré-calculado e armazenado, e as perguntas de ordenação, que pediam a ordenação e a escolha de certos indivíduos que satisfazem certa condição.
3. Para gestão de memória, decidimos utilizar o mínimo de memória para guardar o máximo de informação, utilizando, por exemplo, shorts quando não precisávamos de 4 bytes para inteiros, guardar um único char para distinguir entre verdadeiro e falso, e a tentativa de evitar memory leaks quando se chamava um getter que alocava memória.

1.4 Soluções abandonadas

A ideia de utilizar uma árvore para armazenar alguns dados foi considerada, porém foi rapidamente abandonada visto que não era eficiente, visto que tínhamos de criar a árvore de raiz, enquanto que o array era criado de modo muito mais rápido, e a inserção ordenada não era algo muito necessário para completar o projeto, logo decidimos continuar a utilizar um array.

Chapter 2

2. Código do Projeto

A finalidade deste capítulo é expor detalhadamente, e com a devida fundamentação, a nossa abordagem para o armazenamento, gestão de dados e resolução das Queries do utilizador. Esta componente constitui a maior parte da lógica do nosso programa.

2.1. Parser

Para a conversão dos ficheiros CSV, criamos um parser genérico que converte os 3 ficheiros csv em strings através do uso das funções fgets() e strsep(), pois tivemos em conta os campos vazios nos comentários dos utilizadores. Feita a conversão de cada linha dos ficheiros num array de tokens, estes vão ser transformados numa struct de cada tipo e os valores dos tokens vão ser atribuídos aos campos das structs através do uso das funções de processamento, e uma função para organizar cada catálogo numa hashtable e num array.

2.1.1 Tratamento dos dados

2.1.1.1 Transformação em Structs

O parser.c irá ler os ficheiros e, separar cada linha e armazenar todas as informações num array de string tokens, que depois será passado à parse_func, que redireciona para o módulo correto para ser processado, identificando se o utilizador/condutor/viagem é válida para depois ser armazenado numa estrutura dedicada a receber os dados importantes para a realização das estatísticas e da resposta às queries.

2.1.1.2 Criação das DataBases

Estando as estruturas criadas com os valores dos tokens nos respectivos campos, vamos agora inseri-las numa base de dados criada pela organize_func(), e depois retorna um pointer para ela.

2.2. Estatísticas

No módulo de estatísticas o nosso grupo decidiu pegar nas informações de cada viagem e realizar alguns cálculos para depois poder armazenar em estruturas de dados apropriadas para responder às queries. Foram armazenadas 3 diferentes informações, cada um com as suas estruturas auxiliares e com algumas informações para resposta de queries:

1. As cidades existentes, que foram guardadas numa hashtable, e cada cidade tinha o preço total e o número de viagens, bem como um array de todos os condutores que passaram nessa cidade, juntamente com a avaliação deles.
2. As datas das viagens, armazenadas numa hashtable e que contém as viagens feitas em cada dia.
3. Dois arrays com as viagens feitas por condutores e utilizadores do mesmo sexo, e com a respetiva idade dos seus perfis.

2.3. Dates

Inicialmente tivemos a ideia de usar strings para guardar os valores pretendidos, no entanto chegamos à conclusão que ocuparia demasiada memória e não seria eficiente visto que teríamos de comparar strings sempre que quiséssemos comparar datas, por isso criamos um módulo próprio onde utilizamos shorts para tal efeito, pois assumimos também como data "base" o dia 1 de Janeiro 1950 que se encontra a 26579 dias da data presente (valor menor do que o máximo que um short aguenta). Criamos também uma estrutura para facilitar a utilização e o cálculo entre as datas .

2.4. Queries

Para a resolução das queries, depois da criação das bases de dados e armazenamento dos dados provenientes do parser, criamos funções para responder a cada query, dependendo dos dados a que pretendemos aceder. Por exemplo, nas primeiras 3 queries acedemos às estruturas dos utilizadores e dos condutores através de getters e fomos buscar as informações da pessoa a que pretendíamos aceder na hashtable ou ordenamos o array pedido e fomos buscar os melhores condutores/utilizadores, dependendo da query.Nas queries 4 e 7 acedemos à hashtable das cidades e ordenamos as cidades, enquanto que na 5, 6 e 9 acedemos à hashtable das datas. A query 8 foi a única que utilizou os arrays com os géneros.

2.4.1 Queries 4 e 7

Devido aos pedidos semelhantes destas 2 queries (ambas envolvem valores de viagens numa cidade), estas queries foram resolvidas de forma semelhante.

Para a query 4, para o cálculo do preço médio em uma cidade, acedemos à nossa estrutura auxiliar que armazena todas as cidades, fazemos a divisão do dinheiro total gasto em viagens nessa cidade (sem considerar gorjetas) na cidade pedida pelo número total de viagens e, finalmente, damos print a esse resultado, convertido para double com 3 casas decimais.

Para a query 7, semelhante à query 4, temos que utilizar a avaliação média dos condutores na cidade pedida para ordenar os top N condutores numa determinada cidade. Para isso, ordenamos com um shell sort todos os condutores, através do uso das estruturas auxiliares Info, que vão conter o id, avaliação e número de viagens de cada condutor, valores necessários para o cálculo da avaliação média de cada condutor. Com a ordenação feita, demos print das informações dos primeiros N condutores do array de condutores ordenados, passando à frente aqueles com o estado de conta inativo e informações erradas (verificação de erros).

2.4.2 Queries 5,6 e 9

Nestas queries, nós necessitamos de verificar e calcular estatísticas entre 2 datas. Para isto, criamos uma estrutura auxiliar para as datas para armazenar as informações necessárias para a resolução destas queries, como por exemplo, o dinheiro total de viagens nesse dia, o número de viagens e mais.

Para a query 5, transformamos as datas recebidas como argumentos em um valor inteiro (o número de dias desde a 01/01/1950 até à data usada na função) com a função calc_Date(). Com estes valores, pegamos e somamos todas as quantidades de dinheiro total e números de viagens de cada em dia entre e incluindo as 2 datas em 2 variáveis distintas. Com os valores calculados, finalmente fazemos a divisão dos 2 e damos print ao valor.

Para a query 6, tratamos as datas como na query 4 e transformamo-las em inteiros. Depois disto, somamos todas as distâncias de todas as viagens que são da cidade pedida em cada dia entre e inclusive as 2 datas e, sempre que somamos alguma distância, incrementamos o número de viagens. Com isto feito, fazemos a divisão do total das distâncias com o número total de viagens e calculamos assim o valor da distância média, numa determinada cidade, num intervalo de tempo.

Para a query 9, primeiro guardamos todos os ids de todas as viagens com gorjeta diferente de zero entre e inclusive as 2 datas num array de ids de viagens e registamos o último índice do array após a última inserção para depois usarmos este valor para listar todas as viagens. Com isto feito, ordenamos com outro shell sort semelhante todos os ids do array de acordo com os parâmetros selecionados da query e, finalmente, listamos todas as viagens de forma ordenada.

2.4.3 Query 8

Para a query 8, sendo a única query que envolve o género de utilizadores e condutores, criamos uma estrutura auxiliar **Sexo**, que vai armazenar o id da viagem, a idade das contas do condutor e utilizador em que ambos o condutor e utilizador são do mesmo género. Com isto feito, vamos guardar dois arrays, um para cada género, com todas estas informações previamente mencionadas e os tamanhos destes arrays na estrutura auxiliar **Statistics.** Sabendo isto, ordenamos os 2 arrays de viagens do mesmo género mencionados atrás e verificamos as idades das 2 pessoas de cada viagem e, caso sejam maiores que a idade pedida pela query e o estado de ambas as contas seja ativo, registamos essa viagem.

2.5. Main

Aqui, para a execução do código, primeiro verificamos se o número de argumentos da main é menor que 2 e, caso seja, vai para o modo interativo e, caso não seja, é aberto o modo batch.

No modo batch, procedemos à abertura, envio dos 3 ficheiros csv para o parser e armazenamento dos dados provenientes do parser em arrays de void *. Uma dificuldade que tivemos aqui foi em como usar o primeiro argumento da main (argv[1]) para a abertura dos ficheiros csv, mas depois concatenamos o primeiro argumento da main com a localização dos ficheiros csv e já resolvemos isso. Depois disto feito, abrimos e damos parse do ficheiro de input também e armazenamos o resultado da mesma forma que com os CSV e, finalmente, com estes dados, respondemos as queries através da função answer_queries(), que vai usar os arrays provenientes na abertura e conversão dos ficheiros para responder as queries. Finalmente, depois de respondidas as queries, libertamos toda a memória alocada na execução do projeto.

No modo interativo, depois de muitas dúvidas quanto ao que íamos usar para este modo, decidimos finalmente utilizar a biblioteca ncurses para este fim, pois parecia a forma mais simples e eficiente de criar a paginação. Procedemos então à criação de uma janela, onde

vai estar o pedido da path da localização dos ficheiros csv. Antes disto, tal e qual como no modo batch, procedemos à abertura e envio dos 3 ficheiros csv para o parser. Com isto, chama-mos então a função queries_menu (), que vai abrir o menu de queries.

.

2.6. Modo interativo

Para o modo interativo, começamos por entrar na função queries_menu() com a informação retirada dos CSV. Esta função vai ser usada sempre que for necessário voltar para o menu de queries, ou seja, depois do input da path dos ficheiros csv e depois da conclusão e obtenção do resultado de uma query qualquer. Na função queries_menu(), criamos a janela do menu e damos highlight à opção de query sobre a qual estamos através de um ciclo while que vai decidir a opção highlighted dependendo da seta clicada. Clicando 'Enter' em alguma query, entramos na função query_menu8(), que vai receber o valor da query selecionada e as informações dos CSV e, com isto, entramos em uma das 9 funções auxiliares de resposta às queries no modo interativo queryx_UI(), que algumas usam uma lógica semelhante e não idêntica às funções de resposta do modo batch devido à implementação da paginação em algumas das queries que podem dar muitos inputs que podem não caber no terminal. Nestas funções, tal e qual como mencionado anteriormente, se a resposta da query for só um valor específico, então usamos a mesma lógica das outras funções do modo batch e apenas damos print na janela nova desse valor.

2.6.1 Paginação

Para as queries que necessitam de paginação, decidimos utilizar um ciclo while, que acaba apenas quando a tecla 'Q' é pressionada, ou seja, quando saímos da janela de resultado da query. Até lá, antes de entrar neste ciclo, calculamos ou utilizamos o número de resultados que vamos imprimir. Com isto, já dentro do ciclo while, vamos calcular o número de páginas necessárias, dividindo este valor por 30 (número de linhas por página). Com isto calculado, vamos depois calcular o valor inicial e final da página, utilizando o número de página * 30 para o valor inicial e este valor calculado + 30 para o fim. Com isto feito, vamos dar print aos resultados e, caso o número de resultados seja maior que 30 e seja possível ir para a página seguinte ou anterior, então incrementamos ou decrementamos o número de página e o índice do array onde vamos buscar os ids ordenados ou não, dependendo da query e a forma como navegamos os ids, para a obtenção do que queremos e, com isto, voltamos a dar print dos valores da página nova com os índices do array calculados e os novo início e fim da página.

Com isto feito, como mencionado no início deste sub-capítulo, ao pressionarmos o botão 'Q', o ciclo while vai acabar e vai ser chamada de novo a função query_menu() e vamos voltar ao menu de queries, onde vamos poder selecionar outra query outra vez.

2.7 Tabela de Performance

Utilizadores/ Datasets	Large - Errors	Regular - Errors
Pedro Lopes	69.693 s	3.073 s
Zé Oliveira	76.475 s	3.452 s
Gonçalo Costa	90.569 s	4.236 s

Especificações das máquinas:

Pedro Lopes – **CPU**: AMD Ryzen 7 5800H with Radeon Graphics (16) @ 3.200GHz
GPU: NVIDIA GeForce RTX 3060 Mobile / Max-Q
GPU: AMD ATI Radeon Vega Series / Radeon Vega Mobile Series
Memory: 10018MiB / 13856MiB

Zé Oliveira - **CPU**: 11th Gen Intel i7-11370H (4) @ 3.302GHz
GPU: 00:02.0 VMware SVGA II Adapter Memory: 2317MiB / 3922MiB

Gonçalo Costa - **CPU**: Intel i5-7200U (4) @ 3.100GHz
GPU: NVIDIA GeForce 920MX
GPU: Intel HD Graphics 620 Memory: 4331MiB / 7839MiB

2.8 Mudanças que poderíamos ter feito

Algumas mudanças que chegamos à conclusão que deveríamos ter feito durante e antes da entrega do projeto, devido a falta de tempo para alterar ou o raciocínio já estar demasiado fundido com o código, foi o uso de ncurses para o modo interativo, pois chegamos a ter demasiado problemas com a implementação e adaptação das funções de resposta das queries do modo batch para o modo interativo, juntamente com as funções da biblioteca ncurses, o que nos levou a ter mais dificuldades e a perder mais tempo nesta parte do projeto.

Outra mudança que pensámos em fazer foi, em vez de armazenarmos apenas os valores que necessitamos para depois, nós acabamos por armazenar os apontadores para estes valores, o que nos diminui a performance um bocado.

Mais uma mudança que pensámos fazer também é que nós necessitamos de ler 2 vezes as informações das viagens, o que nos ocupa mais memória, pois é necessária a utilização de getters. Em vez disso, poderíamos ter organizado as estatísticas das viagens em tempo de parse, o que nos ia ajudar bastante na performance.

Chapter 3

Conclusão

Mediante os exercícios propostos enfrentamos bastantes desafios, como a gestão de memória, devido a algumas alocações de memória que, apesar de terem sido resolvidas, levaram o nosso projeto a falhar várias vezes, e o encapsulamento, que nem sempre foi respeitado durante a duração do projeto, mas acreditamos que fomos bem sucedidos a tentar ultrapassar estas dificuldades. Ao longo do projeto tivemos de aprender a utilizar novas ferramentas, como o valgrind e o Gdb para debugging e verificação de memory leaks, bem como aprendemos a utilizar hashtables e novas estruturas de dados.