Universidade do Minho - Escola de Engenharia
Licenciatura em Engenharia Informática - 2º ano

Laboratórios de Informática III

Fase 2

Grupo 29

Ano Letivo 2022/2023

Diogo Gomes Matos - a100741
José Paulino Ribeiro Freitas - a96140
Paula Frederika Janovska Marques - a90088

Índice

Introdução

O presente relatório serve como suporte à Fase 2 do projeto da unidade curricular Laboratórios de Informática III (LI3) do ano letivo 2022/2023, que visa alcançar os seguintes objetivos:

- Consolidação de conhecimentos essenciais da linguagem C e de Engenharia de Software, nomeadamente, modularidade e encapsulamento, estruturas dinâmicas de dados, validação funcional e medição de desempenho (computacional, consumo de memória, etc);
- Consolidação do uso de ferramentas essenciais ao desenvolvimento de projetos em C, nomeadamente, compilação, linkagem, definição de objetivos de projeto com base nas suas dependências e depuração de erros, e de gestão de repositórios colaborativos.

Neste, abordaremos em detalhe a arquitetura e estruturação do nosso projeto, clarificando todas as melhorias e ajustes que se observam em relação à 1ª fase, incluindo também uma análise estimativa do desempenho da nossa aplicação.

Foi criado um programa capaz de ler e interpretar ficheiros de dados de grande extensão, com dois diferentes modos de execução, recorrendo a uma arquitetura capaz e eficiente que permitisse a fácil manipulação dos mesmos. Foi desenvolvido também um programa de testes capaz de avaliar o desempenho e a corretude da execução dos comandos.

Arquitetura do Projeto

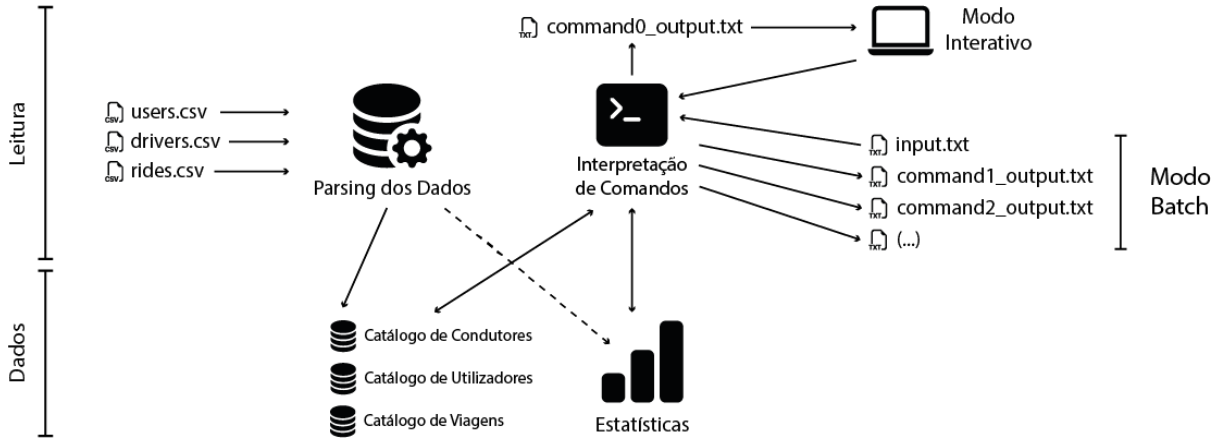

Figura 1 - Arquitetura da aplicação

Parsing e Estruturação dos Dados

O *parsing* é, naturalmente, a primeira tarefa executada pelo programa. É responsável por ler cada linha dos três ficheiros de dados e guardar as suas informações em estruturas adequadas que facilitem a interpretação dos mesmos mais tarde.

A nossa estratégia de estruturação consistiu na criação de 4 distintos módulos de dados: três catálogos e uma estrutura de estatísticas.

❖ Catálogos

Para o armazenamento dos dados relativos aos condutores, utilizadores e viagens criamos três catálogos de dados, cada um deles constituído por uma tabela de hash. Cada uma destas tabelas de hash recorre a uma diferente estrutura de dados que inclui os parâmetros de cada perfil (*DRIVER*, *USER* ou *RIDE*).

Por outras palavras, estas tabelas são um conjunto de estruturas, onde cada uma delas é identificada pelo seu identificador (*id*, *username*, *etc*) e inclui os dados relativos a um dos condutores/utilizadores/viagens.

❖ Estatísticas

De modo a melhorar significativamente o tempo de execução de cada comando, durante o *parsing* é também criada uma estrutura de estatísticas (*STATS*) que inclui diversos dados úteis à execução das diferentes queries.

Esta estrutura é constituída por 7 tabelas de hash, que são construídas de forma simultânea à leitura dos dados das viagens (*rides.csv*), e 4 listas ligadas que constituem ordenações de estatísticas já existentes.

As tabelas permitem o acesso praticamente instantâneo a estatísticas e as listas permitem poupar trabalho de ordenação, que acontece em queries como a 2 e 3.

Leitura e Interpretação de Comandos

Para a leitura de comandos a aplicação conta com dois modos de execução:

❖ Modo Batch

No modo Batch o programa recebe como argumento o caminho para a pasta onde se encontram os ficheiros de dados e o caminho para um ficheiro de texto com a lista de comandos, executando imediatamente e sem ambiente gráfico.

Depois de efetuado o *parsing* dos ficheiros, o ficheiro de comandos é lido linha a linha, e o conteúdo de cada uma das linhas é interpretado e direcionado para a execução da query em questão. O resultado de cada comando é colocado numa pasta *Resultados*, criada na compilação, sob a forma de um ficheiro de texto. Cada ficheiro de texto é identificado por *command<i>_output.txt* onde ***<i>*** corresponde ao número do resultado.

❖ Modo Interativo

No modo Interativo o programa é executado sem argumentos e é apresentado ao utilizador um ambiente gráfico com base no terminal e implementado utilizando a biblioteca *ncurses*.

O utilizador depara-se imediatamente com um ecrã de seleção de *dataset* onde o mesmo deve introduzir o caminho para os ficheiros de dados, depois de confirmada a validade do caminho, o *parsing* dos ficheiros é efetuado, permitindo o acesso ao menu principal onde pode escolher executar uma query, mudar o *dataset* que se encontra carregado ou sair do programa.

Na execução de uma query é apresentado um menu de seleção para escolha de uma das 9 disponíveis e para cada uma é apresentado um menu que recebe os argumentos para a sua execução e que inclui também informações úteis relativas ao seu funcionamento.

O resultado da query executada é escrito num ficheiro *command0_output.txt* que é apresentado ao utilizador a partir de um ecrã de resultados que inclui um módulo de paginação, permitindo a navegação pelas linhas do ficheiro diretamente, sem sair do programa.

Funcionamento das Queries

Cada query conta com uma estrutura de dados diferente que engloba especificamente parâmetros para os dados estatísticos que lhe são necessários e que são armazenados em módulos de dados únicos, constituídos por tabelas de hash e listas ligadas.

Na execução de uma query as estatísticas necessárias são acedidas e, em alguns casos, devolvidas diretamente. Noutros casos são filtradas de acordo com os argumentos dados pelo comando. Por exemplo, de todos os condutores com maior avaliação média, já ordenados, são apenas selecionados os primeiros N, onde N é o argumento passado para a query.

Grande parte destas estatísticas são criadas no *parsing* das viagens, o que melhora significativamente o tempo de execução do programa. A única exceção a esta regra será a query 9, que não usa qualquer das estatísticas calculadas no *parsing*, sendo que as nossas tentativas de tirar partido do *parsing* nesta query resultaram em piores resultados. Apesar disso, continua a ser a query mais lenta na execução.

Encapsulamento e Modulação

O processo de decisão e alteração da arquitetura do projeto passou não só por encontrar a resposta mais rápida, eficiente e com menos consumo de memória como também por ter especial atenção à forma como os diferentes processos interagem entre si, garantindo o encapsulamento de cada um deles e efetuando a modulação dos dados.

Observemos a ***Figura 1***, se prestarmos especial atenção na secção *Dados* podemos concluir que cada um dos módulos de dados existentes é independente entre si. Certamente que criar uma estrutura de dados acessível globalmente e que englobasse todos os módulos de dados num só permitiria um acesso mais fácil e rápido, sendo apenas necessário partilhar um apontador pelos diversos processos. No entanto, esta solução prova-se frágil, sendo vulnerável a erros à medida que a aplicação é desenvolvida e à medida que a sua complexidade aumenta, pelo que a modulação é essencial.

O não encapsulamento dos diferentes processos permitiria também a partilha dos parâmetros das estruturas e a reciclagem de dados ao longo da execução, sendo desnecessária a clonagem de certos tipos de dados, como *strings*, quando fossem necessários fora do ambiente onde foram criados. Pelos mesmos motivos mencionados anteriormente, esta abordagem prova-se incapaz no desenvolvimento de um programa estável e coerente, pelo que todos os diferentes processos são encapsulados e protegidos.

Melhorias, Ajustes e Desafios

Ao longo das duas fases foi evidente o crescimento do nosso projeto e dos nossos conhecimentos, isto conjugado com as sugestões abordadas na apresentação da 1ª fase permitiu-nos realizar melhorias e ajustes significativos. Nesta secção abordamos em detalhe as alterações mais significativas e quais foram as mais desafiantes.

Alterações Gerais

Num âmbito geral, as primeiras melhorias que fizemos tiveram em conta sugestões dadas pelos professores na apresentação, nomeadamente no que toca à reutilização de código ao longo dos processos e à organização do mesmo.

Assim, transformamos conjuntos repetitivos de pequenas instruções em funções auxiliares que introduzimos em *utils.c*, o nosso módulo de utilidades. Por exemplo, a função *get_dataset_path()*, responsável por transformar o caminho para a pasta dos ficheiros num caminho concreto para cada um deles.

Outras das alterações consistiu em mover a interpretação dos argumentos das queries para a função *handle_input()*, o que permitiu que as funções de execução de cada uma das queries recebessem unicamente as variáveis que necessitavam, em vez de uma string não interpretada.

Cálculo de Estatísticas

No que toca ao cálculo de estatísticas, introduzimos no programa o cálculo de estatísticas de forma simultânea ao *parsing* dos dados, tal como tínhamos planeado fazer no relatório da fase 1.

Inicialmente implementamos este método apenas nas estatísticas de cada condutor, cada utilizador e cada cidade (no cálculo do *avg_score*), das quais apenas tiravam partido as queries 1, 2, 3 e 4, com todas as restantes implementadas sobre uma abordagem repetitiva, onde para cada uma delas a totalidade da tabela hash das viagens era percorrida. Esta abordagem resultou numa enorme ineficiência, com tempos de execução a ultrapassar os 20 minutos no grande *dataset.*

Rapidamente percebemos que tínhamos grandes melhorias a fazer, mais tarde acabamos por criar diversas tabelas hash e listas ligadas auxiliares durante o *parsing*, que permitiram um aumento significativo de desempenho que é discutido em *Avaliação de Desempenho*. Esta foi sem dúvida a alteração mais desafiante, que obrigou a um repensar do funcionamento de todas as queries e uma procura pelos processos mais eficientes de o fazer, para não mencionar a mudança de toda a estrutura do projeto de modo a suportar e organizar um conjunto de dados adicional.

Arquitetura do Projeto

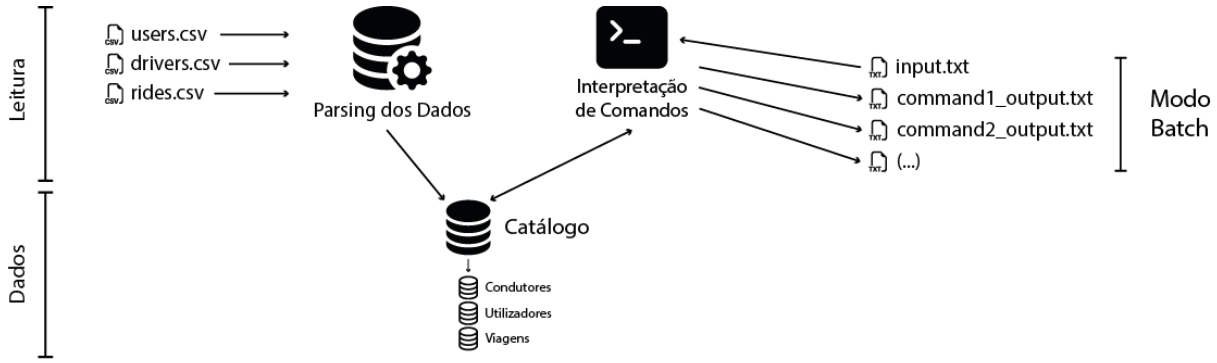

Figura 2 - Arquitetura da aplicação na fase 1

Também a arquitetura da nossa aplicação sofreu grandes alterações desde a 1º fase deste projeto. Desde já na forma como organizamos as nossas estatísticas:

Observemos a ***Figura 2***, uma breve comparação com a ***Figura 1*** permite identificar a falta de um módulo de dados de estatísticas, isto porque, como referido acima, o nosso programa não tirava proveito do *parsing* para o cálculo de estatísticas. Em vez disso, era utilizada uma estrutura de dados *STATS* - a não confundir com o atual catálogo de estatísticas que utiliza uma estrutura pelo mesmo nome - que era constituída por **todos** os parâmetros de estatística necessários a **todas** as queries. Cada função de cálculo de estatística tirava proveito desta mesma estrutura para criar o seu módulo de dados, que era usado apenas por uma query.

O primeiro momento desta alteração consistiu em separar esta estrutura de dados em várias, cada uma direcionada para um objetivo específico e por, depois, colocar cada uma das funções de cálculo de estatística num ficheiro diferente, juntamente com a nova estrutura dedicada à mesma. O que, naturalmente, foi uma grande melhoria para o encapsulamento do código.

Atualmente, e com o cálculo das estatísticas no *parsing*, contamos com o módulo de estatísticas que centra os diferentes módulos de dados estatísticos numa estrutura de dados *STATS*, encapsulada.

Finalmente, a organização dos catálogos de dados mudou significativamente. Inicialmente contávamos com uma estrutura *CATALOG* que unificava as três tabelas de hash com os dados dos condutores, utilizadores e viagens. Por motivos já referidos em *Encapsulamento e Modulação*, descontinuamos esta estrutura, individualizando cada um dos dados.

Avaliação de Desempenho

Para obter uma imagem geral da *performance* do nosso programa, foi efetuada uma análise metódica recorrendo a testes em modo batch do programa.

Metodologia dos Testes

Para cada um dos 4 *datasets* fornecidos (*regular*, *regular-errors*, *large*, *large-errors*) foram efetuados 5 testes (5 amostras) em 3 máquinas distintas para inputs iguais.

O tempo de execução e uso de memória foram calculados com recurso à biblioteca **<sys/resources>** da **GNU C Library**.

O tempo de execução representa o tempo **total** de execução (*user time* + *system time*) e o uso de memória corresponde ao uso **máximo** de memória durante a execução do programa.

Detalhes sobre os cálculos efetuados e os valores lidos em cada amostra estão disponíveis no ficheiro *testes-desempenho-grupo29.xlsx* nos anexos do relatório.

❖ Inputs

Dataset	***regular***	***regular-erros***	***large***	***large-errors***
Nº comandos	50	54	500	500
Fonte	exemplos de queries	BlackBoard	BlackBoard	BlackBoard

Tabela 1 *- Inputs utilizados para cada dataset*

❖ Máquinas

Máquina	**1**	**2**	**3**
Processador	AMD Ryzen 5 2500U	AMD Ryzen 5 5600U	Intel Core i5 7400
Memória RAM (GB)	8	8	16
Sistema Operativo	Linux (Ubuntu)	Linux (Ubuntu)	Windows (WSL - Ubuntu)

Tabela 2 *- Máquinas utilizadas*

Tempo de Execução (seg)

Dataset	*regular*	*regular-errors*	*large*	*large-errors*
Máquina 1	14,115	9,793	390,139	380,388
Máquina 2	8,176	5,601	201,549	194,272
Máquina 3	11,744	7,969	294,719	284,103
Média	**11,703**	**7,969**	**294,672**	**284,188**

***Tabela 3** - Tempo médio de execução em segundos*

Uso Máximo de Memória e Fugas (MB)

Dataset	*regular*	*regular-errors*	*large*	*large-errors*
Máquina 1	335	310	3338	3243,6
Máquina 2	335	311	3338	3244
Máquina 3	334	309	3337	3243
Média	**335**	**310**	**3338**	**3244**

***Tabela 4** - Uso máximo de memória em MegaBytes*

A existência de fugas de memória (*memory leaks*) foi testada utilizando a ferramenta ***valgrind*** que para todos os testes devolveu os seguintes valores:

- *definitely lost*: 0 bytes
- *indirectly lost*: 0 bytes
- *possibly lost*: 0 bytes
- *still reachable*: 18,804 bytes in 9 blocks
- *suppressed*: 0 bytes

Dos quais as únicas fugas de memórias detectadas advêm do uso da biblioteca **GLib** para a construção dos módulos de dados.

Programa de Testes

Foi ainda desenvolvido um programa de testes capaz de testar o desempenho do programa e validar a corretude dos seus resultados, comparando os resultados obtidos com os esperados. Este programa utiliza processos do programa principal na sua execução.

Conclusão

Durante a realização do projeto defrontamos-nos com vários obstáculos e a cada aparição tomávamos proveito do que nos foi lecionado para os ultrapassar, pouco a pouco os obstáculos tornaram-se mais pequenos e notamos um crescimento das nossas competências.

Após concretizarmos todos os desafios propostos, decidimos tentar identificar eventuais falhas e lacunas no projeto realizado e otimizá-lo o máximo possível.

Em testes com o novo *dataset*, que possui uma quantidade de dados bastante maior, todos os processos continuam a dar um resultado correto com um tempo de execução muito satisfatório.

Tendo em conta todos os desafios propostos no decorrer do projeto consideramos que a resposta dada esteve sempre à altura dos mesmos. Apesar de haver sempre coisas que podem ser melhoradas, acreditamos que produzimos um bom trabalho.

Para concluir, consideramos que com os novos conhecimentos adquiridos temos segurança de que estaremos muito melhor preparados para enfrentar outros desafios e para a nossa futura carreira profissional.