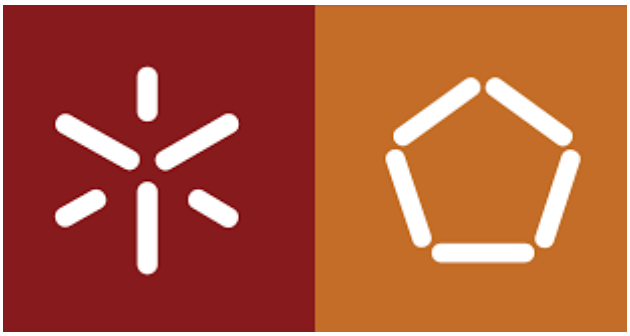


Relatório da 2ª Fase do Projeto

Laboratórios de Informática III

Grupo 23

Licenciatura em Engenharia Informática
Universidade do Minho
Janeiro 2024

Grupo de trabalho:
Fábio Rafael Oliveira Magalhães, A104365
Miguel Tomás Antunes Pinto, A100815
Pedro Miguel Costa Azevedo, A100557

Indíce

1 Evolução desde a primeira fase

Estruturas de dados

Na segunda fase do trabalho, mantivemos a utilização das seguintes estruturas de dados:

- **Struct**
- **Estrutura de dados principais – HashTable**
- **Estrutura de dados auxiliares - GList**

As vantagens da estrutura de dados HashTable continuaram a ser essenciais para o êxito das operações. Sua eficiência de pesquisa permaneceu notável, permitindo acesso direto aos elementos por meio de uma chave específica. Isso se revelou crucial para a busca e recuperação eficiente de dados, especialmente em conjuntos de dados extensos. Além disso, a implementação com uma função de hash bem distribuída assegurou um desempenho médio constante para operações de busca, inserção e remoção, mantendo-se como uma escolha ideal para lidar com grandes volumes de informações.

Quanto às estruturas de dados auxiliares, a GList continuou a desempenhar um papel fundamental na manipulação dinâmica dos dados. Sua flexibilidade permitiu a eficiente inserção e remoção em qualquer ponto da lista, o que se mostrou vantajoso em cenários que exigiam manipulação dinâmica dos elementos. A capacidade de crescimento dinâmico da GList, expandindo-se conforme os dados são adicionados, continuou a ser uma característica valiosa para otimizar o uso de memória quando o tamanho total da lista não era previsível. A versatilidade da GList em adaptar-se a diferentes volumes de dados e sua eficiência na iteração, facilitando a travessia sequencial de elementos, foram aspectos que contribuíram para simplificar a manipulação e acesso aos dados na ordem em que estavam organizados, mantendo-se como uma opção robusta na segunda fase do trabalho.

Modularidade e Encapsulamento

A adesão aos princípios de modularidade e encapsulamento continuou a ser uma prioridade na segunda fase deste trabalho de desenvolvimento de software.

Dividimos o código-fonte em unidades distintas, buscando garantir uma implementação controlada e reutilizável, o que facilita a detecção e correção de erros de maneira eficiente.

Um exemplo notável dessa abordagem é o módulo "statistics", que exemplifica a modularidade implementada no projeto (valores constantes,

apontadores para dados, utilização de getters (declarados no parsing) e a utilização de funções do Glib, com estruturas de dados pré-definidas e null-safety, são algumas das suas características). Este módulo realiza cálculos relacionados ao preço total em uma reserva, compara datas, entre outras funcionalidades. O módulo é concebido de maneira independente, permitindo sua reutilização ao longo do trabalho. Acesso aos dados é restrito apenas ao necessário, garantindo proteção e controle adequados. Os dados aos quais o módulo tem acesso são preservados, e os dados resultantes de suas operações são disponibilizados de forma controlada.

Da mesma forma, os módulos dedicados à realização de queries foram estruturados para manter a consistência dos princípios de modularidade e encapsulamento. A execução das queries ocorre de forma isolada em módulos específicos, assegurando que possíveis erros e correções nesses módulos não interfiram no funcionamento do restante do código. Esta abordagem não só facilita a identificação e resolução de problemas, mas também promove a reutilização eficaz de código, contribuindo para a robustez e manutenção do software ao longo do tempo.

2 Batch Mode e Queries

Na fase inicial de implementação deste projeto, estabelecemos o modo batch como parte integrante. Nesse modo, o programa é executado com três argumentos, sendo o primeiro o caminho para o diretório contendo os ficheiros de entrada, o segundo correspondente ao caminho para um arquivo de texto que contém uma lista de comandos a serem executados.

Na segunda fase do projeto, mantivemos e aprimoramos essa abordagem, expandindo a solução existente para incluir a resolução de todas as queries propostas (1 a 10). Dessa forma, no modo batch, cada comando é interpretado e executado de acordo com sua natureza, produzindo resultados que são salvos no diretório de saída especificado através das funções e buffers expostos no módulo `utils`.

Além disso, para atender aos requisitos específicos da segunda fase, implementamos o modo de operação interativo, incluindo um menu de interação com o programa e um módulo de paginação para apresentação de resultados extensos. Realizamos análises de desempenho, incluindo testes de tempo de CPU e memória, para garantir a eficiência da solução desenvolvida.

Aprimoramos ainda mais as estratégias de modularidade e encapsulamento, conceitos fundamentais nesta unidade curricular. A correta aplicação desses conceitos é vital para uma avaliação positiva em Laboratórios de Informática III, refletindo no desenvolvimento sustentável e eficiente do software ao longo das diferentes fases do projeto.

Leitura de Inputs e Parsing de Dados

O código mantém a sua estrutura de execução e interpretação de inputs, uma função chamada `batchMode` que lida com a leitura de inputs de um ficheiro. Essa função abre o arquivo especificado e de seguida lê linha por linha do ficheiro. A cada iteração do loop, é lida uma linha do ficheiro usando a função *fgets*. Essa linha é então passada para a função *execute_command*.

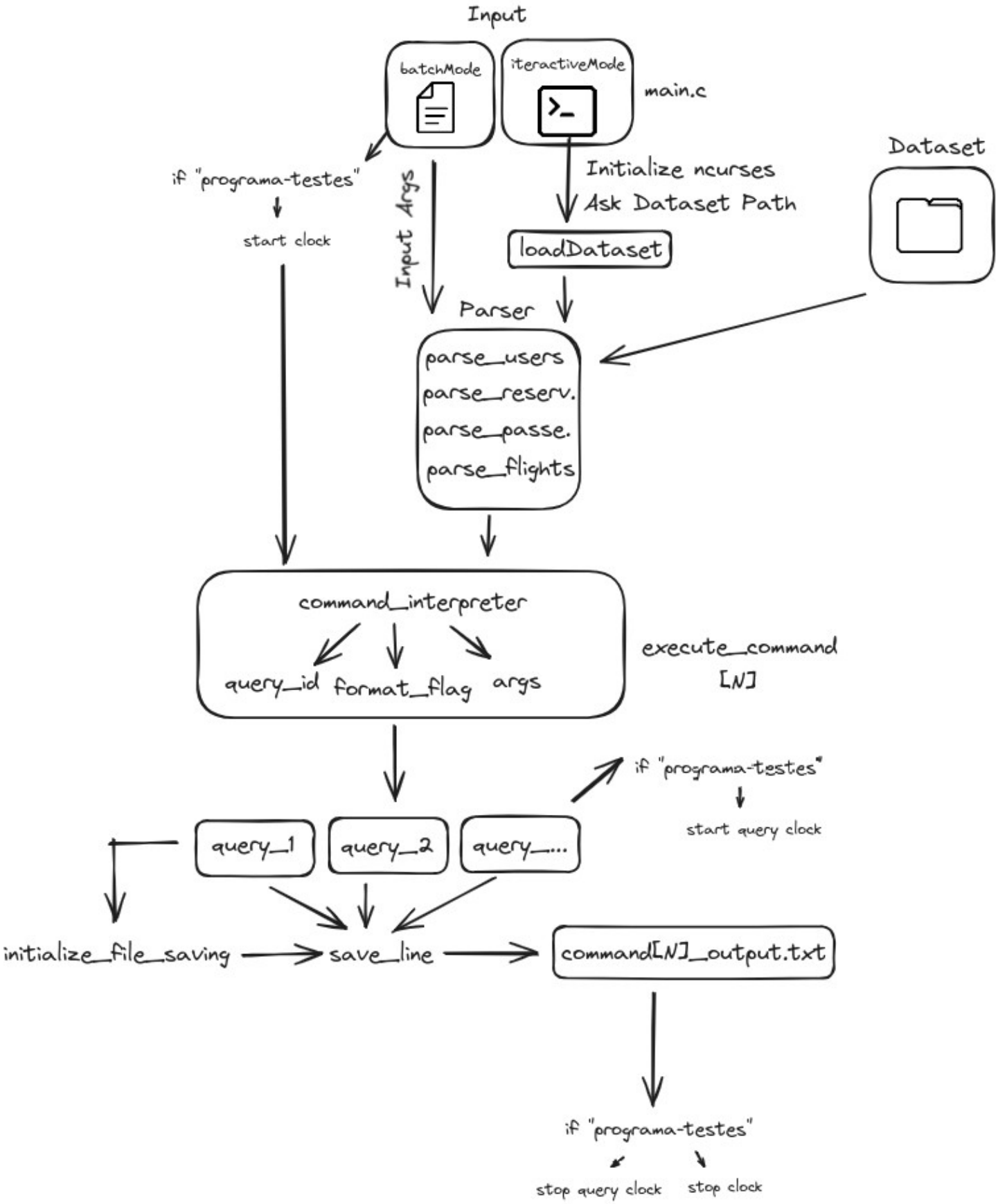
Dentro da função *execute_command*, o conteúdo da linha é processado. Em primeiro lugar, é feita uma interpretação do comando usando a função *command_interpreter*, que extrai informações específicas do comando, como o *ID* da consulta, a *flag* de formato e os argumentos, os quais são manipulados de acordo com o tipo de consulta a ser realizada.

Com base no *ID* da consulta identificado, a função *execute_command* determina qual função de consulta específica deve ser executada. Dependendo do caso, a função apropriada é chamada, executando a consulta e processando os resultados de acordo com a lógica de cada consulta.

A estrutura geral do código garante que cada linha do arquivo de entrada seja lida, interpretada e processada sequencialmente até que todas as linhas tenham sido processadas. Isso permite que os comandos de consulta sejam executados com base nas informações fornecidas no ficheiro de entrada, gerando os resultados correspondentes conforme cada consulta é processada.

A mesma afirmação pode ser dita para os nossos métodos de parsing e buffering (e escrita) de dados, dado que se manteve a mesma estrutura inicial dos mesmos, apenas com a alteração algumas adições e substituições de funções pré-definidas (*Standard C*) por algumas funcionalidades do *Glib* que, apesar de uma pior performance na execução *Valgrind*, apresentavam diversas vantagens como null-safety, prevenção na alocação de memória (no caso de “*g_malloc*”) e melhor compatibilidade com as linhas obtidas durante o parsing (por exemplo a “*g_strdup*” e “*g_strsplit*”), facilitando a definição de algumas funções.

Estrutura da Aplicação



3 Modo Iterativo

Execução de exemplo em modo iterativo da query 1 com os argumentos "LGarcia1208" com o dataset regular de exemplo proveniente da equipa docente.

Neste módulo “iteractiveMode.c”, foram declaradas algumas funções visuais como por exemplos as “*initialize_colors*”, “*centeredString*”, “*showLoadingScreen*” e “*showTitleScreen*”, que tiram partido da da biblioteca *ncruses* e desenham no ecrã do utilizador elementos mais chamativos e de fácil utilização, como por exemplo colorização, centralização de elementos, telas de carregamento ou inicio, etc. Melhorando assim a *user-experience (UX)*.

Não só isso mas também nesse módulo foram declaradas funções mais funcionais, como por exemplo as “*loadDataset*”, “*askDatasetPath*”, “*askForQArgs*”, “*print_result*”, “*showQResult*” e “*execute_query*”, que não só tiram partido desse mesma biblioteca como também carregam, interpretam ou executam funções importantes para o cálculo das queries e a visualização desse resultado de execução.

Para tais cálculos foi utilizada então primeiramente, e durante a execução da aplicação principal (que é executada aquando se passa a tela de inicio), a função “*loadDataset*”, que carrega/faz o parsing de todas a informação do dataset (durante esse carregamento é mostrada uma tela de carregamento, “*showLoadingScreen*”), em seguida é então executada a aplicação principal e mostrado o menu de queries disponiveis a executar. Quando uma queries é selecionada são perguntados alguns parâmetros para a sua execução (“*askForQArgs*”) e é adicionado o resultado (“*print_result*”) na demonstração da paginação do mesmo (“*showQResult*”). Caso a query não produza nenhum resultado, o utilizador ficará a saber através do texto/informação mostrado na paginação.

Testes Unitários

Criamos o módulo “unitTesting.c” para realizar a validação dos dados no conjunto de dados. Este módulo contém funções que são executadas durante a execução do programa de testes (execução do programa com o ficheiro “programa-testes”) e com uma pasta que corresponde aos outputs esperados. Caso essa pasta não seja passada nos argumentos a quando a execução, o programa apenas irá fazer testes de desempenho (*CPU time* e memória).

Para funcionamento são abertos ambos os diretórios (valores esperados e valores obtidos/resultados) e são comparados todos os comandos nos valores esperados com os ficheiros de resultados linha a linha. Caso uma diferença seja encontrada esta à alertada no terminal, indicando o ficheiro e linha onde a desigualdade foi encontrada.

4 Conclusão

Um **ponto importante a realçar e interiorizar** na execução desta segunda fase foi que, apesar de todas as otimizações de tempo de execução de código realizadas pelo grupo, seria ainda necessário a reestruturação (e a rescrição) de todo o projeto devido ao modo de estruturação de uma estrutura de dados durante o parsing e execução de código. Tal limitação levou a que o grupo tivesse a decisão de realizar a totalidade das queries restantes (queries 7 e 10 que não foram realizadas na primeira fase) **mas** com a impossibilidade de realizar os testes automáticos da plataforma de testes para um "dataset com uma ordem de grandeza superior", dado a outra opção seria a realização de tais testes mas sem a completude das queries pretendidas pelo grupo.

Apesar de tudo. Na realização da segunda fase do projeto foi possível ainda melhorar certos aspetos onde reconhecemos falhas, tais como a modularização e encapsulamento do código. Sendo ainda possível a realização dos restantes objetivos propostos no enunciado e pela equipa docente.

Deste modo, ambicionamos ainda a otimização das estrutura de dados utilizadas e tempo de execução de código. Mas não obstante, e como forma de aprendizagem desta unidade curricular, realizámos e explorámos diversos modos de otimização, gestão e modularização de código, e pretendemos utilizar essas aprendizagens num futuro.