

UNIVERSIDADE DO MINHO
LICENCIATURA EM ENGENHARIA INFORMÁTICA

Fase 2
PROJETO LI3

Airline Manager

PL6 - Grupo 27

Hugo Abelheira(a95151)
Luís França(a104259)
Mariana Rocha(a90817)

25 de janeiro de 2024

Conteúdo

1	Introdução	3
2	Nova Arquitetura e Estruturas	3
2.1	Alterações Iniciais	3
2.2	Arquitetura	3
2.3	Queries	4
3	Modo Interativo	5
3.1	Requisitos	5
3.1.1	Não Funcionais	5
3.1.2	Funcionais	6
3.2	Desenvolvimento	6
3.3	Interface Final	7
4	Testes	7
5	Análise de Desempenho	8
5.1	Valgrind	9
6	Dificuldades Sentidas	9
6.1	Modo Interativo	10
6.2	Avaliação do Sistema	10
7	Conclusão	10

Lista de Figuras

1	Representação da Arquitetura do Projeto	4
2	Home e Instructions Page	7
3	Exit Option e Settings Page	7
4	Query Solver e Result Pages	7
5	Tempo de execução por máquina e média de tempo de execução de cada query	9
6	Quantidade de <i>memory leaks</i> do programa-principal e do programa-testes	9

1 Introdução

A segunda fase deste projeto representa um avanço essencial na implementação do nosso *Query Solver*, concentrando-se em melhorias práticas para a interação do utilizador e na conclusão das funcionalidades pendentes. O foco principal desta fase é a criação de um modo interativo, oferecendo uma interface mais amigável e intuitiva para a execução das *queries* e para manipulação dos dados especificados.

Nesta etapa vamos também finalizar as quatro *queries* restantes, proporcionando aos utilizadores um leque mais abrangente de opções para análise de dados. A complexidade adicional traz consigo o desafio contínuo de manter a estrutura modular do projeto, garantindo escalabilidade e eficiência na gestão dos dados específicos fornecidos.

Além disso, a introdução de testes funcionais e de desempenho visa validar e aprimorar a viabilidade e eficiência do sistema. Esta abordagem sistemática tem como objetivo consolidar a solidez da solução, proporcionando uma base robusta para o processamento e análise eficaz dos dados específicos do projeto.

2 Nova Arquitetura e Estruturas

2.1 Alterações Iniciais

Durante a evolução do projeto, realizamos ajustes estruturais e refinamentos para otimizar a eficiência do sistema.

Simplificamos as estruturas de dados, reduzindo a quantidade de informações armazenadas não apenas para aprimorar o desempenho, mas também para promover uma maior escalabilidade.

Além disso, realizamos uma revisão crítica das *hash tables* existentes, eliminando aquelas consideradas desnecessárias. Essa redução não resultou apenas da simplificação, mas, principalmente, de uma análise cuidadosa que revelou estratégias anteriores mais complexas do que o necessário.

A decisão de eliminar o módulo *statistics* foi acompanhada pela redistribuição estratégica do seu conteúdo, integrando-o em áreas que se alinham melhor com a lógica de negócios e a modularidade do sistema.

Simultaneamente, introduzimos novas *hash tables* eficientes, projetadas para lidar especificamente com desafios de *queries* particulares.

Essas mudanças não só simplificaram a estrutura global e fortaleceram a coesão do código, mas também contribuíram para um sistema mais ágil e preparado para futuras expansões, com uma gestão de *hash tables* mais eficiente.

2.2 Arquitetura

Ao conceber a arquitetura deste projeto, mantivemos uma abordagem modular que se baseou na estrutura previamente estabelecida na fase anterior, com a incorporação essencial do modo interativo nesta segunda fase. O sistema permanece dividido em três componentes principais: **Processamento de Entrada**, **Lógica de Negócios** e **Geração de Resultados**. Essa organização modular permite uma implementação robusta e escalável, onde cada módulo desempenha funções específicas, contribuindo para a coesão global do sistema.

No âmbito da **modularidade**, cada componente opera independentemente, facilitando a manutenção, a expansão e a compreensão individual das suas funcionalidades. A estrutura modular oferece flexibilidade, permitindo ajustes e adições sem afetar outras partes do sistema. Esta abordagem foi crucial para a incorporação do modo interativo, uma vez que adicionamos funcionalidades sem comprometer a estabilidade global do programa.

Quanto ao **encapsulamento**, cada módulo encapsula as suas operações internas, expondo apenas as interfaces necessárias para interações externas. Essa prática não apenas promove a segurança e a integridade dos dados, mas também simplifica a compreensão e o uso do sistema. A estratégia de encapsulamento permaneceu consistente ao longo do trabalho, garantindo uma implementação coesa e de fácil manutenção.

A Figura 1 apresenta um diagrama simplificado que ilustra as principais classes e suas interações no sistema. Este diagrama proporciona uma visão visual clara da estrutura do projeto.

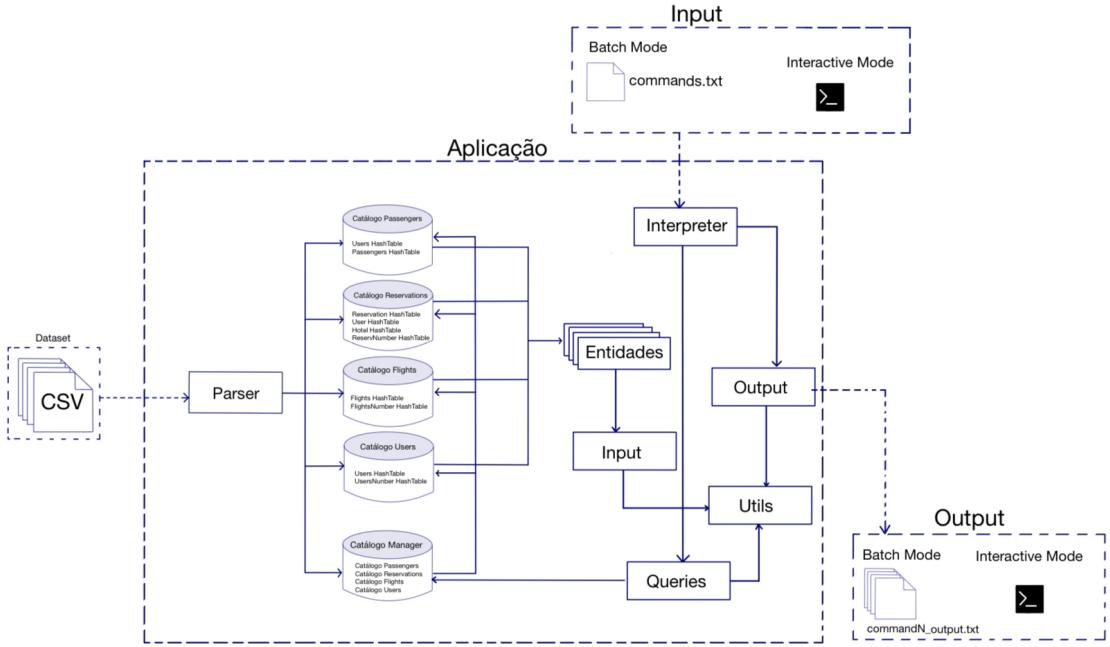


Figura 1: Representação da Arquitetura do Projeto

2.3 Queries

Query 7: "Listar o *top N* aeroportos com a maior mediana de atrasos. Atrasos num aeroporto são calculados a partir da diferença entre a data estimada e a data real de partida, para voos com origem nesse aeroporto. O valor do atraso deverá ser apresentado em segundos. Caso dois aeroportos tenham a mesma mediana, o nome do aeroporto deverá ser usado como critério de desempate (de forma crescente)."

A query 7 tem como objetivo listar o *top N* aeroportos com a maior mediana de atrasos. Utilizando uma estrutura de dados denominada *AirportInfo2*, o código itera sobre os voos armazenados no catálogo, calculando os atrasos com base na diferença entre as datas estimada e real de partida. Esses atrasos são acumulados num *array* dinâmico, associados a cada aeroporto. A mediana dos atrasos é então calculada para cada aeroporto, e o array de *AirportInfo2* é ordenada utilizando a função *qsort*. A função retorna os resultados formatados, limitados ao *top N* aeroportos com as maiores medianas de atrasos, após a devida libertação da memória alocada para evitar quaisquer *memory leaks*.

Query 8: "Apresentar a receita total de um hotel entre duas datas (inclusive), a partir do seu identificador. As receitas de um hotel devem considerar apenas o preço por noite (*price_per_night*) de todas as reservas com noites entre as duas datas. E.g., caso um hotel tenha apenas uma reserva de 100Eur/noite de 2023/10/01 a 2023/10/10, e quisermos saber as receitas entre 2023/10/01 a 2023/10/02, deverá ser retornado 200Eur (duas noites). Por outro lado, caso a reserva seja entre 2023/10/01 a 2023/10/02, deverá ser retornado 100Eur (uma noite)."

A query 8 é destinada a apresentar a receita total de um hotel entre duas datas, considerando o preço por noite das reservas. A função utiliza um identificador de hotel e intervalo de datas como argumentos. A lógica consiste em iterar sobre as reservas associadas ao hotel, verificando se as datas da reserva estão contidas no intervalo especificado. Caso haja sobreposição, a função calcula o número de noites dentro desse intervalo e soma o preço por noite multiplicado pelo número de noites à receita total do hotel. O resultado final é retornado como uma *string* representando o valor total da receita.

Query 9: "Listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome (de forma crescente). Caso dois utilizadores tenham o mesmo nome, deverá ser usado o seu identificador como critério de desempate (de forma crescente). Utilizadores inativos não deverão ser considerados pela pesquisa."

A query 9 foi projetada para listar todos os utilizadores cujo nome começa com o prefixo passado por argumento, ordenados por nome de forma crescente. Em caso de nomes iguais, o identificador é utilizado como critério de desempate também de forma crescente. A função exclui utilizadores inativos da pesquisa, percorrendo a tabela de utilizadores, identificando aqueles que atendem aos critérios e armazenando as informações em uma estrutura dinâmica chamada *User_list*. Posteriormente, os resultados são ordenados usando a função *qsort* e formatados num *array* de

strings representando o número total de utilizadores encontrados, juntamente com detalhes sobre cada utilizador, como identificador e nome. A memória, como sempre, é gerida para evitar *leaks*.

Query 10: "Apresentar várias métricas gerais da aplicação. As métricas consideradas são: número de novos utilizadores registados (de acordo com o campo *account_creation*); número de voos (de acordo com o campo *schedule_departure_date*); número de passageiros; número de passageiros únicos; e 12 número de reservas (de acordo com o campo *begin_date*). Caso a *query* seja executada sem argumentos, apresentar os dados agregados por ano, para todos os anos que a aplicação tem registo. Caso a *query* seja executada com um argumento, *year*, apresentar os dados desse ano agregados por mês. Finalmente, caso a *query* seja executada com dois argumentos, *year* e *month*, apresentar os dados desse ano e mês agregados por dia. O output deverá ser ordenado de forma crescente consoante o ano/mês/dia.

A *query 10*, destinada a apresentar diversas métricas gerais da aplicação, incluindo o número de novos utilizadores registados, voos, passageiros, passageiros únicos e reservas. A apresentação dessas métricas varia conforme os argumentos fornecidos à *query*: sem argumentos, os dados são agregados por ano para todos os anos registados; com um argumento (*year*), os dados desse ano são apresentados agregados por mês; com dois argumentos (*year* e *month*), os dados desse ano e mês são apresentados agregados por dia. A função utiliza estruturas de dados e funções auxiliares para processar as informações dos catálogos e gera um *array* de *strings* representando as métricas agregadas, ordenadas de forma crescente. A gestão de memória é realizada de forma apropriada para prevenir possíveis *memory leaks*.

3 Modo Interativo

Nesta secção, abordaremos detalhadamente o modo interativo do nosso projeto, implementado utilizando a biblioteca *NCurses*. Desenvolvemos uma série de menus intuitivos que oferecem aos utilizadores uma experiência mais amigável e acessível com o nosso *Query Solver*. A inclusão de uma aba de instruções fornece informações relevantes, permitindo que os usuários explorem as funcionalidades do sistema de forma eficiente e percepçável. Adicionalmente, incorporamos um menu de configurações, visando melhorar a qualidade de vida do usuário, proporcionando ajustes personalizáveis para uma experiência mais adaptada e eficaz. Essas inovações não apenas facilitam a interação com o *Query Solver*, mas também reforçam o nosso compromisso com uma interface mais intuitiva e centrada no utilizador.

3.1 Requisitos

3.1.1 Não Funcionais

- O caminho padrão sugerido deve ser conveniente para o usuário e, quando possível, relativo à pasta de execução do programa.
- O *layout* dos resultados deve ser claramente definido, proporcionando uma apresentação organizada e legível das informações.
- A personalização do formato de output deve ser eficiente e intuitiva para melhorar a experiência do utilizador.
- O sistema de paginação deve ser eficiente para lidar com grandes conjuntos de dados, garantindo uma resposta rápida e uma navegação suave.
- O sistema de navegação por setas deve ser intuitivo, proporcionando uma experiência de usuário agradável e sem complicações.
- Quando ocorrerem erros por parte do utilizador, uma caixa de texto colorida deve ser exibida para destacar e explicar claramente o argumento inserido incorretamente.
- A aba de instruções deve ser apresentada de forma clara e intuitiva, garantindo que o utilizador possa entender facilmente como interagir com o programa e fornecendo exemplos práticos.
- A aba de *settings* deve ser de fácil acesso e compreensão, proporcionando uma configuração conveniente e rápida antes do início do programa.
- Mesmo seguindo um formato específico, o design do input deve ser intuitivo, facilitando a inserção correta de informações por parte do utilizador e minimizando possíveis erros devido a ambiguidades.

3.1.2 Funcionais

- O programa deve ser capaz de iniciar sem a necessidade de argumentos e, quando necessário, sugerir um caminho padrão relativo à pasta de execução do programa.
- O modo interativo deve fornecer instruções claras para cada *query* disponível, orientando o utilizador sobre como formatar e inserir as consultas.
- O sistema deve incluir um módulo de paginação para lidar com resultados extensos, permitindo ao usuário navegar pelos dados de maneira eficiente.
- O programa deve adotar um formato de input padronizado para cada *query*, conforme detalhado nas instruções, proporcionando consistência e previsibilidade na interação do usuário com o sistema.
- O programa deve permitir ao utilizador a escolha do formato de *output*, incluindo as opções de visualização num ficheiro *txt*, visualização por número de páginas ou por número de *outputs* ou um por um.
- O sistema deve ser capaz de identificar e lidar graciosamente com erros comuns do usuário, como inserção inadequada de texto em *queries* de *top N* ou especificação de uma pasta inválida para o *dataset*.
- Deve existir um sistema de navegação por setas para facilitar a movimentação do usuário dentro do programa, especialmente ao explorar os resultados paginados.
- O modo interativo deve incluir uma aba de instruções que forneça informações detalhadas sobre o funcionamento do programa, exemplos do *input* para diferentes *queries* e exemplos de *outputs* esperados.

3.2 Desenvolvimento

Iniciamos o desenvolvimento do modo interativo implementando uma página inicial que apresenta as principais opções do programa, nomeadamente o *Airline Query Solver*, que encapsula toda a lógica de negócio, uma aba de instruções detalhada e a opção *Exit* para encerrar o programa. A aba de instruções foi estruturada em três páginas distintas: a primeira fornece uma introdução ao sistema e o seu propósito, a segunda explica em detalhes o funcionamento do programa, e a terceira contém informações específicas de cada *query*, incluindo um resumo, exemplos de *input* e exemplos de *output* corretos.

Ao entrar no *Airline Query Solver*, o utilizador é recebido com uma aba de *settings* que permite a personalização de aspectos como a *path* do *dataset* e os formatos de *output*, conforme especificado nos requisitos. Após a configuração, os catálogos são preenchidos com a informação da *path* fornecida, e o sistema avança para o *Query Solver*. Caso ocorram problemas, o programa está preparado para apresentar mensagens e códigos de erro, garantindo uma experiência informativa.

No *Query Solver*, o utilizador tem a oportunidade de especificar a *query* desejada e os respetivos argumentos, personalizados para cada *query* individual. Destaca-se a atenção especial dada aos menus e à qualidade de vida do utilizador, com a inclusão de notas explicativas que o acompanham ao longo do programa. Por exemplo, se o formato da data inserida for inválido, será exibida de mensagem de erro no menu flutuante em questão, evitando que o utilizador perca tempo inserindo os outros argumentos.

O menu final, denominado *Query Result*, permite a visualização dos *outputs* de acordo com o formato escolhido nas *settings*. As opções incluem *One by One*, *Txt Format* para exportar para ficheiros *txt*, escolher o número de páginas ou definir o número de *outputs* por página. Cada uma destas opções será detalhadamente explorada a seguir.

- **Txt Format:** Ao escolher esta opção, o utilizador solicita a geração de um ficheiro *txt* contendo o *output* da *query* selecionada. Esta abordagem permite uma fácil exportação e partilha dos resultados, tornando a gestão da informação obtida bastante prática.
- **One by One:** Optando por este formato, cada *output* é apresentado individualmente. O utilizador pressiona a tecla "c"(de *continue*) para avançar para o próximo resultado. Quando a janela atinge a capacidade máxima, é criada uma nova página, mantendo a apresentação dos *outputs* de forma sequencial.
- **Number of Pages:** Nesta opção, o utilizador especifica o número desejado de páginas para visualização dos *outputs*. Os resultados são distribuídos de maneira proporcional, sendo importante notar que, caso o número de total de *outputs* seja superior ao limite da página, será necessário fazer *scroll down* para aceder aos restantes resultados dessa página.
- **Number of Outputs per Page:** Ao indicar a quantidade de *outputs* desejada por página, o sistema calcula automaticamente o número total de páginas necessárias para apresentar todos os resultados. Importante salientar que, se o utilizador solicitar mais *outputs* por página do que a janela consegue mostrar, será necessário utilizar a opção de *scroll down* para visualizar os resultados adicionais na mesma página.

3.3 Interface Final

A apresentação visual do nosso sistema final é exemplificada pelas figuras abaixo que revelam a interface gráfica dos menus desenvolvidos. Estes *screenshots* fornecem uma visão prática dos diferentes menus, demonstrando a organização, opções e a interatividade intuitiva que os utilizadores podem esperar ao utilizar o nosso sistema. Cada imagem destaca um aspecto específico da interface final, proporcionando uma representação sucinta da experiência visual global.

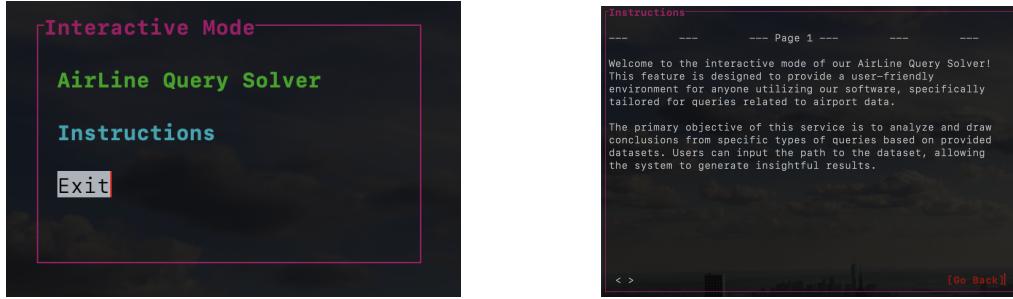


Figura 2: Home e Instructions Page



Figura 3: Exit Option e Settings Page

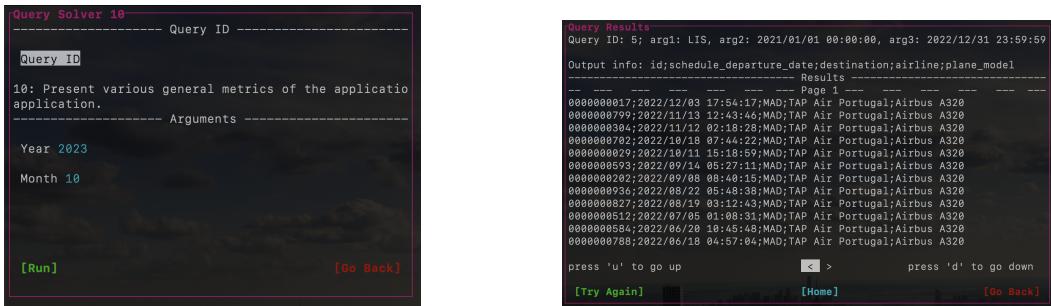


Figura 4: Query Solver e Result Pages

4 Testes

Os testes funcionais foram essenciais para verificar a correta implementação das funcionalidades do sistema, validando a lógica e a precisão das respostas. Em paralelo, os testes de desempenho abordaram métricas críticas, como tempo de execução das *queries* e uso eficiente da memória, proporcionando uma análise abrangente do desempenho do sistema em diferentes cenários.

Os **testes funcionais** foram concebidos seguindo os mesmos princípios de modularidade e encapsulamento. Cada teste avalia individualmente a funcionalidade específica de uma *query*, isolando-a de outras partes do sistema. Essa abordagem não apenas facilita a identificação e correção de problemas, mas também contribui para a confiabilidade geral do sistema, garantindo que cada módulo funcione conforme o esperado, mesmo em cenários de interação complexa.

Para os testes funcionais, adotamos uma abordagem específica, utilizando a técnica que os professores sugeriram para verificar a correta execução das *queries*. Desenvolvemos uma variedade de testes para cada *query*, variando os tipos de argumentos (válidos e inválidos) para garantir uma cobertura abrangente. Cada teste incluiu a medição do tempo de execução, proporcionando uma compreensão sobre a eficiência de cada *query* em diferentes condições.

No âmbito dos **testes de desempenho**, também seguimos as orientações fornecidas pelos docentes. Implementamos um "programa-testes" que recebe o caminho para o *dataset*, o ficheiro txt de comandos e a pasta com os resultados esperados. Este programa compara os resultados obtidos com os esperados, indicando se o teste foi bem-sucedido ou apontando a linha da primeira incongruência encontrada. Além disso, registamos o tempo de execução de cada *query* e o tempo geral de execução, bem como a quantidade de memória utilizada pelo programa.

Os valores dos testes funcionais podem ser consultados no arquivo `analysis.txt`, enquanto a análise de desempenho está disponível no arquivo `analysisTest.txt`.

5 Análise de Desempenho

A análise de desempenho desempenha um papel crucial na avaliação da eficácia e eficiência do sistema desenvolvido. Nesta secção são explorados os resultados dos testes realizados em diferentes máquinas, proporcionando uma visão aprofundada do comportamento do sistema em cenários diversos.

Mostraremos dados quantitativos sobre o tempo de execução de cada *query*, o tempo geral de execução do sistema e a quantidade de memória utilizada. Essas métricas fornecerão percepções significativas sobre a eficiência operacional do sistema, permitindo-nos entender melhor como ele responde a diferentes demandas de processamento.

Este capítulo destaca a importância de compreender não apenas a funcionalidade individual das *queries*, mas também o impacto coletivo delas no desempenho global do sistema.

Apresentamos a tabela seguinte que contém as médias dos tempos de execução obtidos ao realizar o programa de testes em cada máquina, 10 iterações de 100 *queries*. Este conjunto diversificado de execuções permite uma visão abrangente do desempenho do sistema sob várias condições. Além das médias dos tempos de execução, fornecemos informações específicas sobre cada computador utilizado, destacando particularidades relevantes para a análise dos resultados.

	Máquina 1	Máquina 2	Máquina 3
Modelo	Acer SF315-41-R34L	MackBook Pro 2020	HP 15s-fq1028np
CPU	amd ryzen 5	i5	i7
RAM	8Gb	16Gb	12Gb
Sistema Operacional	Zorin OS	MacOS	Ubuntu
Compilador	gcc 12.2.1	Apple clang 15.0.0	gcc 11.4.0
Memória máxima do programa	33Mb	Error	33Mb
Tempo de execução	0.32s	0.37s	0.49s

Tabela 1: Tabela Informativa

Embora os computadores com hardware mais avançado demonstrem tempos de execução superiores em relação ao computador com desempenho inferior, é importante notar que as diferenças observadas não são significativamente pronunciadas. A marginal discrepância nos tempos de execução sugere que, apesar das variações no hardware, o sistema mantém uma consistência notável em sua capacidade de resposta. Essa relativa uniformidade indica uma eficiência robusta do sistema, capaz de fornecer resultados satisfatórios mesmo em ambientes com configurações de hardware distintas. Essa constância no desempenho contribui para a versatilidade do sistema, adaptando-se efetivamente a diferentes contextos de utilização.

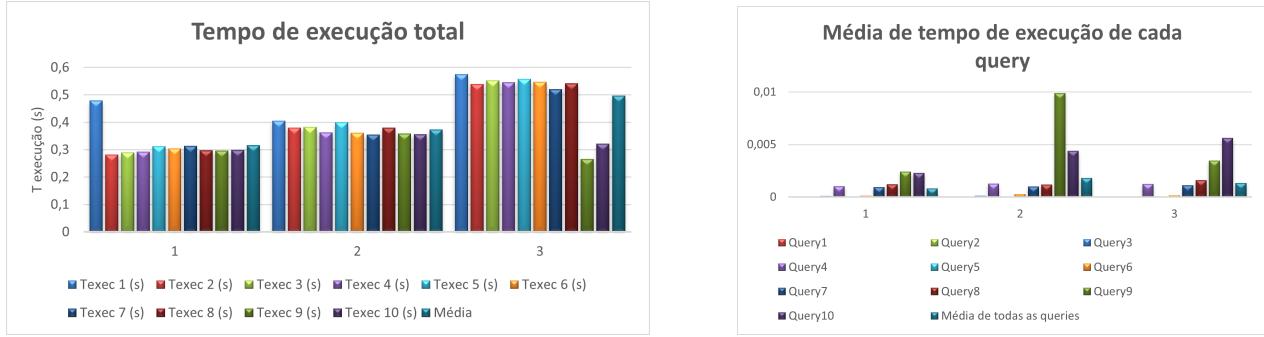


Figura 5: Tempo de execução por máquina e média de tempo de execução de cada query

Os valores baixos de execução de *queries* refletem a eficiência das escolhas de estruturas de dados adotadas no projeto. A utilização extensiva de *hashTables* contribui significativamente para tempos de acesso rápidos aos dados. Além disso, a implementação de *hashTables* específicas para determinadas *queries* otimiza ainda mais o desempenho, oferecendo um acesso mais eficiente aos dados relevantes.

A decisão de empregar a função *qsort* para a ordenação de dados nas *queries* necessárias também se alinha com a busca pela eficiência. A complexidade assintótica de $O(N^2 \log N)$ desta função torna-a numa escolha apropriada para ordenar conjuntos de dados, garantindo um processamento rápido e escalável, essencial para operações que envolvem a organização eficiente de informações em diferentes contextos.

Estas escolhas fundamentadas nas características das estruturas de dados e nas complexidades algorítmicas contribuem de forma significativa para o desempenho global do sistema, assegurando uma resposta ágil e eficaz na execução das *queries* propostas.

5.1 Valgrind

Começamos por resolver os 100 *bytes* de *memory leaks* do sistema. A partir desse ponto, mantivemos a libertação de memória total na evolução da segunda fase, resultando na ausência de *leaks* tanto no programa principal quanto no programa de teste, conforme pode ser verificado nas imagens em anexo. Este progresso não apenas resolveu desafios específicos, mas também reflete o nosso compromisso contínuo com a eficiência operacional.

```
marianna@trabalho-pratico$ valgrind ./programa-principal dataset/dataset/input.txt
==14106== Memcheck, a memory error detector
==14106== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==14106== Using Valgrind-3.18.1 and LlVdEX; rerun with -h for copyright info
==14106== Command: ./programa-principal dataset/dataset/input.txt
==14106==
==14106==
==14106== HEAP SUMMARY:
==14106==   in use at exit: 18,804 bytes in 9 blocks
==14106==   total heap usage: 1,781,413 allocs, 1,781,404 frees, 54,773,618 bytes allocated
==14106==
==14106== LEAK SUMMARY:
==14106==   definitely lost: 0 bytes in 0 blocks
==14106==   indirectly lost: 0 bytes in 0 blocks
==14106==   possibly lost: 0 bytes in 0 blocks
==14106==   still reachable: 18,804 bytes in 9 blocks
==14106==   suppressed: 0 bytes in 0 blocks
==14106== Rerun with --leak-check=full to see details of leaked memory
marianna@trabalho-pratico$ valgrind ./programa-testes dataset/dataset/input.txt outputs/
==14383== Memcheck, a memory error detector
==14383== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==14383== Using Valgrind-3.18.1 and LlVdEX; rerun with -h for copyright info
==14383== Command: ./programa-testes dataset/dataset/input.txt outputs/
==14383==
==14383== HEAP SUMMARY:
==14383==   in use at exit: 0 bytes in 0 blocks
==14383==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==14383==
==14383== LEAK SUMMARY:
==14383==   in use at exit: 0 bytes in 0 blocks
==14383==   total heap usage: 1,876,937 allocs, 1,876,928 frees, 0 bytes allocated
==14383==
==14383== LEAK SUMMARY:
==14383==   definitely lost: 0 bytes in 0 blocks
==14383==   indirectly lost: 0 bytes in 0 blocks
==14383==   possibly lost: 0 bytes in 0 blocks
==14383==   still reachable: 0 bytes in 0 blocks
==14383==   suppressed: 0 bytes in 0 blocks
==14383== Rerun with --leak-check=full to see details of leaked memory
```

Figura 6: Quantidade de *memory leaks* do programa-principal e do programa-testes

Ao longo do desenvolvimento, a atenção proativa à resolução de *leaks* iniciais contribuiu para melhorar a estabilidade do sistema e estabelecer uma base sólida para a gestão eficiente de recursos.

Em resumo, ao superar desafios iniciais e manter uma abordagem diligente na gestão de memória, conseguimos criar um sistema mais robusto e eficiente, agora livre de qualquer tipo de *memory leak*.

6 Dificuldades Sentidas

O desenvolvimento do modo interativo e a preparação para avaliação do sistema não estiveram isentos de desafios. À medida que exploramos a implementação detalhada do modo interativo, desde a criação da página inicial até o sistema de outputs flexíveis, identificamos obstáculos que exigiram soluções engenhosas. Simultaneamente, a etapa de preparação para a avaliação do sistema envolveu desafios específicos relacionados aos testes funcionais e de desempenho, incluindo monitoramento de memória e otimização de tempo de execução. Este capítulo discute esses desafios, apresentando uma perspectiva abrangente sobre como foram abordados, proporcionando uma base para as conclusões finais.

6.1 Modo Interativo

O desenvolvimento do modo interativo envolveu superar desafios significativos. A aprendizagem da biblioteca *NCurses*, a gestão eficaz de memória, a implementação de encapsulamento modular e a lógica de negócios associada à manipulação de catálogos e apresentação de *outputs* requereram esforços específicos para garantir um sistema robusto e eficiente. Estas adversidades contribuíram para o aprimoramento das habilidades de desenvolvimento, resultando numa solução que atende aos padrões mais elevados de qualidade e usabilidade.

6.2 Avaliação do Sistema

A fase de testes funcionais e de desempenho, embora mais acessível por causa do código fornecido pelos professores, não esteve isenta de desafios. Enfrentamos uma complexidade adicional relacionada ao sistema operativo, especialmente no ambiente *MacOS*. Um problema peculiar surgiu ao calcular a utilização de memória, onde, mesmo com um uso real de aproximadamente 32 MB, o *MacOS* indicava erroneamente um consumo de 25 GB, uma discrepância que, até o momento, permanece sem uma solução clara. Este obstáculo destacou a importância de considerar as nuances dos diferentes sistemas operativos para garantir uma avaliação precisa do desempenho do sistema.

7 Conclusão

Ao longo deste projeto, embarcamos numa jornada para criar um sistema robusto e eficiente de gestão de dados. Desde a fase inicial, onde delineamos a arquitetura e realizamos adaptações estruturais, até à implementação do modo interativo com foco em otimizações de *queries* e experiência do usuário, buscamos constantemente a excelência. A eliminação de módulos desnecessários e a reestruturação eficiente das *hash tables* refletem a nossa dedicação à lógica de negócios e a conceitos teóricos como a modularidade e o encapsulamento.

A introdução de um modo interativo, desenvolvido utilizando a biblioteca *NCurses*, apresenta uma interface intuitiva e amigável para interação com o sistema. Os menus, abas de instruções e configurações proporcionam uma experiência coesa e simplificada, refletindo a atenção cuidadosa dada à qualidade de vida do utilizador.

As dificuldades enfrentadas durante o desenvolvimento, especialmente na criação do modo interativo e na preparação para a avaliação do sistema, destacaram a nossa busca incessante pela perfeição. A abordagem sistemática nos testes funcionais e de desempenho permitiu-nos garantir não apenas a funcionalidade correta, mas também a eficiência operacional.

Olhando para o futuro, este projeto não é apenas um sistema bem sucedido, mas também um exemplo de escalabilidade e adaptabilidade. As alterações estratégicas implementadas ao longo do desenvolvimento e o cuidado constante na gestão de recursos estabelecem uma base sólida para futuras expansões e melhorias.