Relatório da Fase 2 – Grupo 25

Introdução

Este relatório visa discutir a segunda fase do projeto da Unidade Curricular Laboratórios de Informática III do ano letivo 2023/2024. Continuando o trabalho iniciado na primeira fase, esta etapa avançou para a implementação de todos os requisitos iniciais, além da inclusão de novas funcionalidades e otimizações significativas.
Entre os principais avanços desta fase, destacam-se a implementação da totalidade das queries propostas, a introdução de um modo de operação interativo com um sistema de menu e paginação para a apresentação de resultados maiores e a realização de testes rigorosos para análise de desempenho da aplicação, tanto em termos de tempo de CPU quanto de uso de memória.
Neste relatório, serão abordadas as escolhas técnicas e estratégicas realizadas pelo grupo para atender a estes desafios, enfatizando as metodologias de desenvolvimento adotadas, as estruturas de dados utilizadas e as técnicas de otimização implementadas. Além disso, discutiremos os resultados obtidos nos testes de desempenho e funcionalidades, evidenciando como a solução proposta se comporta sob diferentes condições e cargas de trabalho.

Principais diferenças entre a primeira fase e a segunda fase do projeto

A evolução do nosso projeto de Laboratórios de Informática III da primeira para a segunda fase foi fortemente influenciada pelas valiosas indicações e feedbacks recebidos dos professores. Os conselhos foram importantes para identificar as áreas que precisavam de melhoria. Abaixo, apresentamos as principais diferenças e melhorias implementadas, refletindo o impacto positivo dessas orientações:

1. *Reestruturação das Estruturas de Dados*
   Inicialmente, as estruturas de dados como as structs e hash tables estavam localizadas nos arquivos header (.h) correspondentes a cada classe, como user.h e reservas.h. Na segunda fase, optamos por uma abordagem mais encapsulada, movendo estas estruturas para os arquivos de implementação propriamente ditos, como user.c e reservas.c. Esta mudança fortaleceu o encapsulamento e a ocultação de dados, alinhando-se melhor com as práticas de programação.

2. *Todas as queries foram completas*
   Um dos objetivos desta fase era implementar todas as queries delineadas no projeto. Conseguimos alcançar este marco, o que demonstra uma expansão significativa da funcionalidade do nosso sistema em comparação com a fase anterior, onde apenas uma parte das queries tinha sido implementada.

3. *Otimização e gestão de memória*
   Com a utilização do Valgrind, uma ferramenta de análise de memória, conseguimos identificar e corrigir os diversos memory leaks presentes na primeira versão do projeto. Esta melhoria não só otimizou o uso da memória, mas também contribuiu para a estabilidade e eficiência do sistema.

4. *Melhoria na modularidade e divisão dos ficheiros*
   Na segunda fase, fizemos uma reorganização dos ficheiros do nosso projeto, proporcionando uma estrutura mais modular. Essa reestruturação permitiu uma separação mais clara das diferentes partes do código, facilitando a manutenção, compreensão e expansão futura do sistema.

5. *Foco na modularidade e encapsulamento*
   Considerando que o encapsulamento e a modularidade eram critérios de avaliação nesta fase, dedicamos especial atenção a esses aspectos. As mudanças realizadas visaram não apenas atender a esses critérios, mas também melhorar a qualidade geral do código e a eficiência do sistema.

Análise de complexidade e explicação das queries em falta.

Visto que já abordamos a maioria das queries no primeiro relatório e que nenhuma destas sofreu alterações significativas para a segunda fase, o que iremos fazer é falar sobre a complexidade de cada query e quando alcançarmos uma query que não tenha sido abordada procederemos então à sua explicação.

Query 1

A. *Operações de Recuperação de Dados*
   A função q1 começa por recuperar informações de três tabelas hash diferentes (users, voos e reservas). Cada operação de recuperação (RetrieveUser, RetrieveVoo, RetrieveReserva) tem uma complexidade média de O(1) devido à eficiência das tabelas hash em localizar elementos. No entanto, essa complexidade pode piorar para O(n) em casos de colisões excessivas, onde n é o número de elementos num 'bucket' específico da tabela hash.

B. *Acesso a Dados e Cópia de Strings*
   Após a recuperação dos dados, a função faz várias chamadas a métodos auxiliares para aceder e copiar strings (como userGetAccountStatus, vooGetAirline, reservaGetHotelId, etc.). Cada uma dessas operações é O(1), porque envolvem acesso direto a campos de structs e operações de cópia de strings.

C. *Escrita de Dados*
   A função q1 também envolve a escrita de informações formatadas num ficheiro ou terminal, dependendo do contexto. Essa operação é proporcional ao tamanho dos dados a serem escritos. No entanto, visto que a quantidade de dados para cada user, voo ou reserva é fixa e limitada, podemos considerar essa operação como O(1) para cada entidade.

A complexidade total da função q1 pode ser maioritariamente considerada como O(n) no pior caso. No entanto, sob condições normais e com uma função de hash eficaz que minimiza colisões, a complexidade tende a se aproximar de O(1).

Query 2

A complexidade desta query está intimamente ligada à eficiência da recuperação do user na tabela hash, seguida pelo processamento sequencial da lista ligada Q2 associada a cada user. Dada a natureza das tabelas hash, a recuperação do user é geralmente rápida, aproximando-se de O(1) na maioria dos casos, embora possa degradar para O(n) no pior cenário com colisões significativas como mencionado anteriormente. Uma vez recuperado, a função itera sobre a lista ligada Q2, que armazena as atividades do user, aplicando filtros conforme necessário. Esta iteração é linear em relação ao número de entradas na lista, resultando assim numa complexidade de O(n) para essa parte do processo.
Em suma, a complexidade geral da função q2 pode ser vista como O(n), onde n é o número de entradas na lista Q2 do user.

Query 3

A complexidade desta query está intrinsecamente ligada à eficiência da recuperação do hotel na tabela hash e ao subsequente processamento das reservas associadas a esse hotel. A recuperação do hotel é geralmente eficiente, aproximando-se de O(1) na maioria dos casos devido à natureza das tabelas hash, embora possa variar para O(n) em cenários com colisões significativas. Uma vez que o hotel é recuperado, a função itera sobre a lista ligada ReservaResumo, acumulando e calculando a média das classificações. Esta iteração é linear em relação ao número de entradas na lista, resultando numa complexidade de O(n) para essa parte do processamento.
Portanto, a complexidade total da função q3 pode ser considerada O(n) no pior caso, onde n representa o número de reservas vinculadas a um hotel específico. Esta análise destaca a eficiência da recuperação de dados em tabelas hash e a necessidade de considerar o impacto do processamento sequencial em listas ligadas, especialmente em operações de agregação como o cálculo de médias.

Query 4

Analisando a complexidade desta query, vemos que ela depende principalmente do número de reservas associadas a um hotel específico. A função começa por ir buscar o hotel da tabela hash, uma operação que geralmente é eficiente, com uma complexidade aproximada de O(1), especialmente se as colisões na tabela hash forem mínimas. Uma vez recuperado o hotel, a query itera sobre a lista ligada ReservaResumo e processa cada reserva individualmente.
Durante a iteração, a query executa várias operações de recuperação de dados e formatação de saída, mas todas elas são realizadas em tempo constante para cada reserva. Portanto, a complexidade total da função q4 é O(n), onde n é o número de reservas ligadas ao hotel dado no input. Esta complexidade reflete a natureza linear da iteração sobre a lista ligada de reservas.

Query 5

A query começa por encontrar um aeroporto específico da tabela hash de aeroportos, o que é uma operação eficiente (O(1)). Depois disso, a query itera sobre a lista ligada de resumos de voo associados a esse aeroporto, filtrando aqueles que estão dentro do intervalo de datas especificado. Esta parte da query tem uma complexidade linear (O(n)) em relação ao número de resumos de voo do aeroporto, pois cada resumo deve ser verificado individualmente.
É importante destacar que a eficiência desta função depende significativamente do número de voos associados a um aeroporto específico e do intervalo de datas fornecido. Se o aeroporto tiver um grande número de voos e/ou o intervalo de datas for amplo, a função poderá levar mais tempo para executar devido à necessidade de processar uma grande quantidade de dados. Por outro lado, se o aeroporto tiver poucos voos ou o intervalo entre datas for pequeno, a função pode executar mais rapidamente.
No geral, a complexidade de q5 é O(n), onde n é o número de voos do aeroporto. Esta complexidade reflete a abordagem linear adotada para filtrar os voos.

Query 6

A query começa por converter os argumentos fornecidos (ano e N) de strings para inteiros usando a função 'atoi'.
Em seguida, a função 'criarListaSomaPassageirosAno' é chamada. Esta função itera sobre todos os voos na tabela hash de voos, agrupando-os pelo ano do input. Para cada aeroporto, ela soma o número total de passageiros dos voos desse ano.
A lista resultante é então ordenada com base no total de passageiros, e os primeiros N aeroportos (conforme especificado pelo argumento N) são selecionados para incluir no resultado final.
A função itera sobre a lista de aeroportos criada, enquanto vai imprimindo as informações sobre cada aeroporto (nome e total de passageiros) no ficheiro de saída.

A conversão dos argumentos de strings para inteiros (atoi) é uma operação de complexidade O(1), pois é independente do tamanho dos dados de entrada.
A função 'criarListaSomaPassageiros' envolve iterar sobre todos os voos na tabela hash de voos, agregando o número total de passageiros por ano para cada aeroporto. A

complexidade desta operação é O(n), onde n é o número total de voos, assumindo que o acesso aos voos na tabela hash é eficiente (aproximadamente O(1) para cada acesso).
Após a agregação, a lista de aeroportos é ordenada com base no total de passageiros. A complexidade da ordenação é O(m log m), onde m é o número de aeroportos distintos na lista. A ordenação é a parte mais demorada em termos de tempo de execução.
Imprimir os resultados envolve percorrer toda a lista ordenada de aeroportos até um máximo de N elementos (fornecido pelo user). Esta operação tem uma complexidade de O(N).
Portanto, combinando todas essas etapas, a complexidade total da query q6 pode ser expressa como O(n + m log m + N).

Query 7

A query 7 analisa os dados do aeroporto para calcular as medianas.
O processo começa com a função 'GetMedianaAeroportos', que percorre a tabela hash de aeroportos, calculando a mediana para cada um. Esta etapa é central para a função, pois é ela que controla a organização e análise dos dados de cada aeroporto.
Posteriormente, a função itera sobre uma lista de aeroportos, mas limita-se a N entradas conforme especificado pelo parâmetro de input. Em cada iteração, ela devolve o nome do aeroporto e a mediana correspondente, imprimindo essas informações no ficheiro de saída.

A complexidade da query 7 no contexto do projeto foi analisada em diferentes etapas.
A primeira parte significativa da complexidade surge durante a operação GetMedianaAeroportos, que é responsável por calcular a mediana para cada aeroporto dentro da tabela hash. Esta operação envolve percorrer cada lista ligada dentro dos buckets da tabela hash, que armazenam os aeroportos e calcular a mediana dos dados associados a cada aeroporto. A complexidade desta etapa é, portanto, dependente do número de aeroportos (N_a) e da distribuição deles nos buckets da hash. No caso médio, essa complexidade é O(N_a).
A segunda parte da complexidade está no loop da query q7, que itera sobre a lista resultante até um máximo de N elementos, sendo N o número de aeroportos para os quais a mediana é solicitada. Esta parte da função tem uma complexidade linear O(N), pois percorre cada elemento da lista uma vez até o limite definido.
Portanto, a complexidade total da função q7 é maioritamente a complexiade da primeira parte, ou seja, O(N_a).

Query 8

A query 8 é uma query que calcula o lucro total de um hotel específico num intervalo de datas. Esta função depende da operação GetLucro, que percorre a lista de reservas de um hotel e soma o valor total das reservas que se encontram dentro do intervalo de datas.

A complexidade da query 8 está diretamente ligada à quantidade de reservas associadas ao hotel específico. A função precisa de iterar sobre cada reserva do hotel, verificando se a data da reserva está dentro do intervalo de datas passado como input. Esta verificação é feita em tempo constante para cada reserva. Portanto, a complexidade da função GetLucro é O(n), onde n é o número de reservas do hotel.

Uma vez calculado o lucro, a função q8 simplesmente imprime o resultado. Esta parte da função tem uma complexidade constante O(1), pois envolve apenas algumas operações de impressão.
Assim sendo, a complexidade total da query 8 é dominada pelo processo de cálculo do lucro, que é O(n).

Query 9

A complexidade da query 9 é determinada principalmente pela função 'GetUserPrefix', que itera sobre as entradas na tabela hash de users para encontrar aqueles cujos nomes começam com o prefixo dado como input.
Dado que a tabela hash está projetada para distribuir as entradas uniformemente, a operação de busca por um prefixo específico, em teoria, teria uma complexidade média de O(1) para cada acesso individual. No entanto, a necessidade de percorrer as listas ligadas dentro de cada bucket da tabela hash para comparar os prefixos implica uma complexidade que pode chegar a O(n) no pior caso, onde n é o número total de users na tabela. Isto ocorre especialmente se muitos users compartilharem o mesmo prefixo, o que se verifica para alguns casos, mesmo usando o dataset pequeno.
Uma vez encontrados os users com o prefixo desejado, a complexidade das operações subsequentes (como a obtenção de IDs e nomes, e a impressão dos resultados) é constante para cada user. Portanto, o tempo total de execução da query 9 é dominado pela busca inicial dos usuários com o prefixo correspondente, sendo O(n) no pior caso.

Query 10

A query 10 é uma análise estatística detalhada que se adapta dinamicamente para fornecer informações sobre users, voos e reservas num ano, mês ou dia com base nos argumentos passados como input. Inicialmente, a função determina se deve processar dados para um ano específico, um mês específico de um ano ou para uma data específica. Na ausência de um ano específico, ela percorre um intervalo fixo de anos, devolvendo para cada ano, o número total de users, voos, passageiros, passageiros únicos e reservas. Se um ano específico for fornecido, a função analisa todos os meses daquele ano. Da mesma forma, se um mês específico dentro de um ano for fornecido, a função devolve as informações para cada dia daquele determinado mês.

Quanto à complexidade da query 10, ela é diretamente influenciada pela quantidade de dados processados em cada nível que tem de ser analisado. Na análise anual, a complexidade é proporcional ao número de anos multiplicado pelo número de dados de cada ano, resultando numa complexidade linear O(n). Para a análise mensal, a complexidade aumenta ligeiramente, tornando-se O(m * n), com m representando os 12 meses e n o número de dados por mês. Finalmente, na análise diária, a complexidade é O(d * n), onde d pode ir até 31 (o número máximo de dias num mês) e n representa o número de dados num determinado dia.

Modo interativo

Inicialização e Processamento de Dados

- main_interativo: A função responsável pela inicialização das tabelas hash e pelo processamento dos ficheiros CSV. Ela lê os dados de quatro arquivos CSV diferentes e os armazena em estruturas de dados apropriadas. Se ocorrer algum erro durante o processamento dos arquivos (indicado pelo retorno 1 de qualquer função de processamento), uma mensagem de erro é exibida e a função retorna 1, indicando uma falha no carregamento dos dados.

Exibição e Paginação de Dados

- display_page: Esta função gere a exibição de uma página de resultados no ecrã. Lê o conteúdo do arquivo linha por linha e exibe as linhas no ecrã, começando de um ponto específico (start) até um ponto final (end). Esta função é crucial para exibir dados de forma paginada.
- move_pages: Esta função facilita a navegação entre diferentes páginas de resultados. Ela permite ao user mover-se para a próxima ou para a página anterior de resultados, utilizando as teclas de navegação.

Interação do User e Execução de Comandos

- programa_interativo: É a função principal que controla a interação do user com o programa. É apresentado um menu com três opções: inserir o caminho dos arquivos CSV, inserir uma query para processamento, ou sair do programa. Dependendo da escolha do user, diferentes ações são realizadas:
    - Caminho dos CSV: Solicita ao user que insira o caminho para a pasta contendo os arquivos CSV e chama main_interativo para processar os dados.
    - Inserir Query: Permite ao user inserir uma query. Se os dados ainda não foram carregados, exibe uma mensagem de erro. Caso contrário, processa a query inserida.
    - Sair: Encerra o programa e liberta os recursos alocados.

- interativo: A função configura a interface gráfica do user utilizando como recurso à biblioteca ‘ncurses’. Cria uma janela e gere a interação do user com o menu, capturando as escolhas do user e chama o programa_interativo para realizar as ações correspondentes.

- comando_interativo: Esta função lida com a execução das queries inseridas pelo users. Ela divide a linha de comando inserida em argumentos como no modo batch, decide qual query chamar com base no comando inserido e escreve os resultados num arquivo de texto auxiliar. Após a execução do comando, retorna à diretoria original do trabalho.

O nosso modo interativo oferece uma forma eficiente, flexível e simples de aceder e manipular os conjuntos de dados. Através de uma interface de user clara e funcionalidades como paginação e execução de queries, o programa consegue atender às diversas necessidades dos users de forma intuitiva.

Análise de Desempenho

Computadores utilizados

	Computador 1	Computador 2	Computador 3
CPU	**Intel i7-1165G7 4-Core**	**Intel i5 1,8GHz Núcleo Duplo**	**Intel i9-9880H 8-core**
RAM	**16GB DDR4 3200MHz**	**8GB DDR3 1600MHz**	**16GB DDR4 2667 MHZ**
Disco	**1 TB SSD**	**128 GB SSD**	**1 TB SSD**
Sistema Operativo	**Windows 11 WSL 1.0**	**macOS Monterey 12.6.6**	**Ventura 13.4.1**
Compilador	**GCC 9.2.0 (Ubuntu 20.04)**	**Clang 14.0.0**	**Clang 14.0.3**

Desempenho das queries

Query	Computador 1 (s)	Computador 2 (s)	Computador 3 (s)
1 JéssiTavares910	0.000066	0.000092	0.000014
1F JéssiTavares910	0.000050	0.000086	0.000016
2 JéssiTavares910 reservations	0.000065	0.000075	0.000008
2F JéssiTavares910 reservations	0.000050	0.000070	0.000011
3 HTL1001	0.000104	0.000109	0.000036
3F HTL1001	0.000096	0.000118	0.000027
4 HTL1001	0.002665	0.00300	0.000668
4F HTL1001	0.004697	0.005332	0.001125
5 LIS "2021/01/01 00:00:00" "2022/12/31 23:59:59"	0.000222	0.000289	0.000052
5F LIS "2021/01/01 00:00:00" "2022/12/31 23:59:59"	0.000553	0.00640	0.000106
6 2023 10	0.002698	0.004286	0.010361
6F 2023 10	0.003412	0.003900	0.014218
7 10	0.001273	0.000704	0.001562
7F 10	0.000662	0.000736	0.001060
8 HTL1001 2023/05/02 2023/05/02	0.000126	0.000164	0.000053
8F HTL1001 2023/05/02 2023/05/02	0.000148	0.000147	0.000040
9 J	0.008053	0.009172	0.037891
9F J	0.008574	0.009611	0.028944
10 2023 03	1.538278	1.923562	1.761591
10F 2023 03	1.528872	1.913287	1.704245

Leitura e Validação do Dataset

Dataset pequeno			
Utilizadores	Reservas	Passageiros	Voos
0.027104	**0.258065**	**1.162790**	**0.004279**

Gestão de memória

Dataset pequeno	
Memória Alocada	Leaks
53.016 MB	**0.004 MB**

Conclusão

Nesta segunda fase do projeto da Unidade Curricular Laboratórios de Informática III, do ano letivo 2023/2024, houve um foco particular na melhoria da modularidade e do encapsulamento, seguindo as orientações dos docentes e as necessidades identificadas na primeira fase. Este aprimoramento traduziu-se numa organização mais eficaz do código, com uma divisão mais clara e lógica dos ficheiros e numa gestão de memória mais eficiente, especialmente graças ao uso da ferramenta Valgrind para detetar e resolver os problemas de memória.
Um dos grandes objetivos desta fase foi a implementação do modo interativo. Esta nova funcionalidade elevou significativamente a usabilidade do sistema, oferecendo uma experiência de utilizador mais dinâmica e interativa. Através de um menu intuitivo e de um sistema de paginação, foi possível gerir e apresentar grandes volumes de dados de forma eficaz, melhorando a interação do utilizador com o sistema.
A colaboração e o empenho do grupo foram fundamentais para atingir os objetivos propostos. A equipa trabalhou de forma coesa, com cada membro a contribuir com as suas competências e conhecimentos. Este ambiente colaborativo foi crucial para o desenvolvimento de uma solução integrada e eficaz.
Em conclusão, esta fase do projeto destacou-se pelo aprimoramento na experiência do utilizador e na eficiência do código.

Notas

Inicialmente, um dos objetivos era adaptar a solução para lidar eficazmente com um conjunto de dados de grande dimensão. Contudo, ao longo do processo, percebemos que esta adaptação exigiria uma reestruturação aprofundada do programa e um investimento de tempo que poderia comprometer outras áreas do projeto. Assim, após uma cuidadosa ponderação e avaliação das prioridades e dos recursos disponíveis, decidimos não prosseguir com essa implementação específica. Esta escolha permitiu-nos concentrar os nossos esforços na melhoria e aperfeiçoamento das funcionalidades existentes, assegurando a entrega de um trabalho completo e de qualidade dentro do prazo estabelecido.

Adicionalmente, é relevante mencionar a questão da Query 10. Durante os testes, constatou-se que esta query apresentava um tempo de execução significativamente mais longo em comparação com as demais, impactando no desempenho geral do sistema. Apesar deste desafio, optámos por mantê-la como parte do projeto. A inclusão desta query foi uma decisão consciente para demonstrar o processo de aprendizagem e evolução pelo qual passámos, bem como para evidenciar a nossa capacidade de enfrentar desafios complexos, mesmo quando estes afetam o desempenho global do sistema.
Estas notas refletem as escolhas e os desafios enfrentados pelo grupo ao longo do desenvolvimento do projeto. Apesar dos obstáculos, acreditamos que as decisões tomadas foram as mais acertadas para o contexto e os recursos que tínhamos disponíveis, permitindo-nos entregar um trabalho consistente e alinhado com os objetivos inicialmente propostos.