Universidade do Minho
Escola de Engenharia

Universidade do Minho

PROJETO LABORATÓRIOS DE INFORMÁTICA III
Ano letivo 2023/2024
Fase 2

Grupo 77

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Afonso Sousa (A104262), João Carvalho (A104533) e Rui Cruz (A104355)

Conteúdo

Capítulo 1

Introdução

Este projeto, parte integrante da disciplina Laboratórios de Informática III do ano letivo 2023/2024, foi concebido para introduzir aos alunos conceitos essenciais da Engenharia de Software, como design, modularidade e encapsulamento. O desafio inicial foi desenvolver uma base de dados em memória para armazenar dados fornecidos pelos professores. Com a primeira fase focada no parsing dos dados e implementação do modo batch, avançamos para cumprir os requisitos restantes. Esta interface permite a introdução de queries pelo usuário e o carregamento dos dados a partir de um diretório especificado. A segunda fase do projeto também testou a robustez das estruturas e algoritmos previamente implementados, principalmente devido ao aumento significativo no tamanho dos dados.

Capítulo 2

Módulos e estruturas de dados

2.1 Users

Para armazenar todos os users usámos uma hash table que associa uma chave(calculada com o nome de cada user) a uma estrutura de dados que chamamos users. Um user permite armazenar, para um dado utilizador, todas as suas informações (id, nome, data de nascimento, sexo, passaporte, morada, data de criação de conta , estado da conta, um apontador para outro possível user, um array com os flights do user, o número de flights do user atual, o máximo de flights do user atual, um array com as reservations do user, o número de reservations do user atual e o número máximo de reservations do user atual), tendo assim a nossa definição de user:
O módulo user permite realizar operações de inserção, pesquisa e atualização de dados.
Como se vai reparar mais tarde este módulo é o que mais informação contem, isto irá permitir melhorar a performance de várias queries como se verá mais tarde.

```
6   typedef struct users {
7       char *id;
8       char *name;
9       //char *email; //n se usa
10      //char *phone_number; //n se usa
11      char *birth_date;
12      char sex;
13      char *passport;
14      char *country_code;
15      //char *address; //n se usa
16      char *acc_creation;
17      //char *payment; //n se usa
18      char *status;
19      struct users *next;
20
21      Flights **user_flights;
22      int current_user_flight;
23      int max_user_flights;
24
25      Reservations **user_reservations;
26      int current_user_reservation;
27      int max_user_reservations;
28  } Users;
```

2.2 Reservations

Para armazenar as reservations a estratégia utilizada é similar à estratégia usada para os users, sendo essa uma hash table, mas no caso das reservations associamos uma chave(calculada com o hotel id de cada reservation) a uma estrutura de dados que chamamos de reservations, onde, tal como nos users, podemos armazenar todas as informações de uma reserva (id, id do utilizador, identificador do hotel, nome do hotel, estrelas do hotel, imposto da cidade em percentagem, data de início, data de fim, preço por noite, inclusão de pequeno almoço , rating e um apontador para outra possível reservation), tendo assim a nossa definição de reservation:
O módulo reservations permite realizar operações de inserção, pesquisa e atualização de reservations.

```c
typedef struct reservations {
    char *id;
    char *user_id;
    char *hotel_id;
    char *hotel_name;
    unsigned int stars;
    unsigned int tax;
    //char *address; //n se usa
    char *begin_date;
    char *end_date;
    unsigned int price_per_night;
    char *breakfast;
    //char *details; //n se usa
    unsigned int rating;
    //char *comments;//n se usa
    struct reservations *next;
} Reservations;
```

2.3 Flights

Para armazenar os flights foi utilizada a mesma estratégia utilizada para os users e as reservations, utilizando uma estrutura de dados à qual chamamos flights(key será calculada com o próprio id) onde podemos armezenar as informações de cada voo (id, airline, modelo do avião, total de lugares, origem, destino, data de partida estimada e real, data de chegada estimada e real e um apontador para outro possível flight), obtendo assim a nossa definição de flight:
O módulo flights permite realizar operações de inserção, pesquisa e atualização de flights.

```c
typedef struct flights {
    char *id;
    char *airline;     // companhia aérea
    char *model;       // modelo do avião
    int sits;
    char *origem;
    char *destino;
    char *dep;         // data de partida prevista
    char *arr;        // data de chegada prevista
    char *rdep;        // data de partida real
    char *rarr;      // data de chegada real
    //char *pilot; //n se usa      --- nome do piloto
    //char *copilot; //n se usa    --- nome do copiloto
    //char *notes;  // n se usa    --- notas sobre o voo
    struct flights *next;
} Flights;
```

2.4 Passengers

Para armazenar os passengers, a estratégia utilizada a mesma das restantes bases de dados, usando uma hash table onde guardamos uma estrutura de dados que nomeamos passengers(key será com o id do flight). Essa estrutura de dados permite armazenar um id de utilizador, que no caso é o passageiro, o id do voo e um apontador para outro possível passenger.

O módulo passengers permite realizar operações de inserção, pesquisa e atualização de passengers.

```
5
6   typedef struct passengers {
7       char *flight_id;
8       char *user_id;
9       struct passengers *next;
10  } Passengers;
11
```

Capítulo 3

Queries

Um dos nossos objetivos para esta segunda fase foi terminar a implementação das queries. Este objetivo foi concluído com a ajuda de algumas otimizações que foram realizadas e que serão posteriormente referidas na secção “Otimizações”. Para algumas das queries, era imprescindível um algoritmo de ordenação, diferente para cada contexto. Um dos algoritmos de ordenação implementados foi o ***merge sort***, que foi utilizado por exemplo na query 9 onde era necessário ordenar os utilizadores de acordo com o prefixo, uma vez que a sua complexidade em tempo é dada como constante, O(n log n) para o pior, médio e melhor caso. As queries 1,2 e 10 exigiram um pouco mais do nosso esforço uma vez que se tratam de definições mais geral e completas no que toca à recolha, cálculo e apresentação de dados.

Capítulo 4

Otimizações

O dataset da segunda fase veio a tornar-se um desafio já que possuía um tamanho muito superior comparado com o da primeira fase. Uma das otimizações importantes que surgiu em combate a este problema, foi a ordenação e o modo como os users estão dispostos na hashtable. Foi entendido que ordenando os users de acordo com um id, facilitaria a procura tanto de reservas como passageiros. Acrescentando ainda a esta métrica, agora é possível englobar as reservas e voos de um passageiro na sua própria estrutura. Foi também entendido que remover os emails, números de telemóvel, métodos de pagamento, os endereços, os comentários das reservas e as notas dos voos nas respectivas structs tornaria mais fácil a execução de várias funções uma vez que estes parâmetros não se consideram em nenhuma das queries.
Nas querys 1 e 2, o facto de termos passado quase tudo o que pede para as base de dados em si tornou possível que estas ficassem com os tempos mais rápudo até agora como por exemplo a query 1 que passou de demorar 0.5 segundos por iteração para 0.000125, para além disto a query2(bem como todas as outras que precisavam de um sort) passou a usar um sort(merge) muito mais eficiente que o que usava antes(selection). A query3 ficou melhor por a base de dados ter as reservations separados pelos hoteis que lhe correspondem. A query4 ficou melhor por se guardar as reservations e/ou flights de cada user dentro da propria struct users. Na query5 não sentimos a necessidade de a melhorar mais do que ja estava antes. Na 6 e 7 em vez de primeiro ver a base de dados dos flights para ver todos os aeroportos existentes(só origens na 7 e origens e destinos na 6) e depois ver a base de dados de novo para fazer o que era pedido, os dois precessos foram juntos, o que fez uma pequena mas satisfatória otimização. A query8 é apenas uma cópia da 3(embora o facto do ano ter 31*12 dias em vez de 365 deunos um grande na 2 fase). A query9 é a menos otimizada em relação a base de dados pois tem que se percorrer todos os users por cada input quando com um pouco de otimização da nossa parte podiamos mete-la a ver so um grupo de users para certos inputs. A query10 embora seja a mais lenta acabou por ser umas das que está melhor otimizada para a grande tarefa que pede, pois so tem que percorrer a base de dados dos users e depois com as informações contidas dentro de cada user podemos deduzir as informações que poderiam necessitar de ver todos os flights, passengers e reservations, o que acaba por não acontecer.

Capítulo 5

Testes de desempenho

Para medir o tempo de execução do projeto, recorremos à biblioteca time.h, e obtivemos tempos relativamente similares a estes (em segundos):

Queries	Regular Dataset	Large Dataset
1	0.000046	0.000344
2	0.000067	0.000117
3	0.000027	0.001895
4	0.003288	0.454472
5	0.000382	0.018369
6	0.000194	0.027462
7	0.000759	0.146962
8	0.000236	0.040551
9	0.002516	0.250408
10	0.016181	4.675059
Total	0.602857	276.47333

Para além dos testes de tempo de execução também foi realizado um teste de memória com o auxílio da libraria sys/resource.h para ver a memória usada pelo programa, sendo esta 52224 kilobytes no dataset regular e 4238360 kilobytes no dataset grande.

Capítulo 6

Modo Interativo

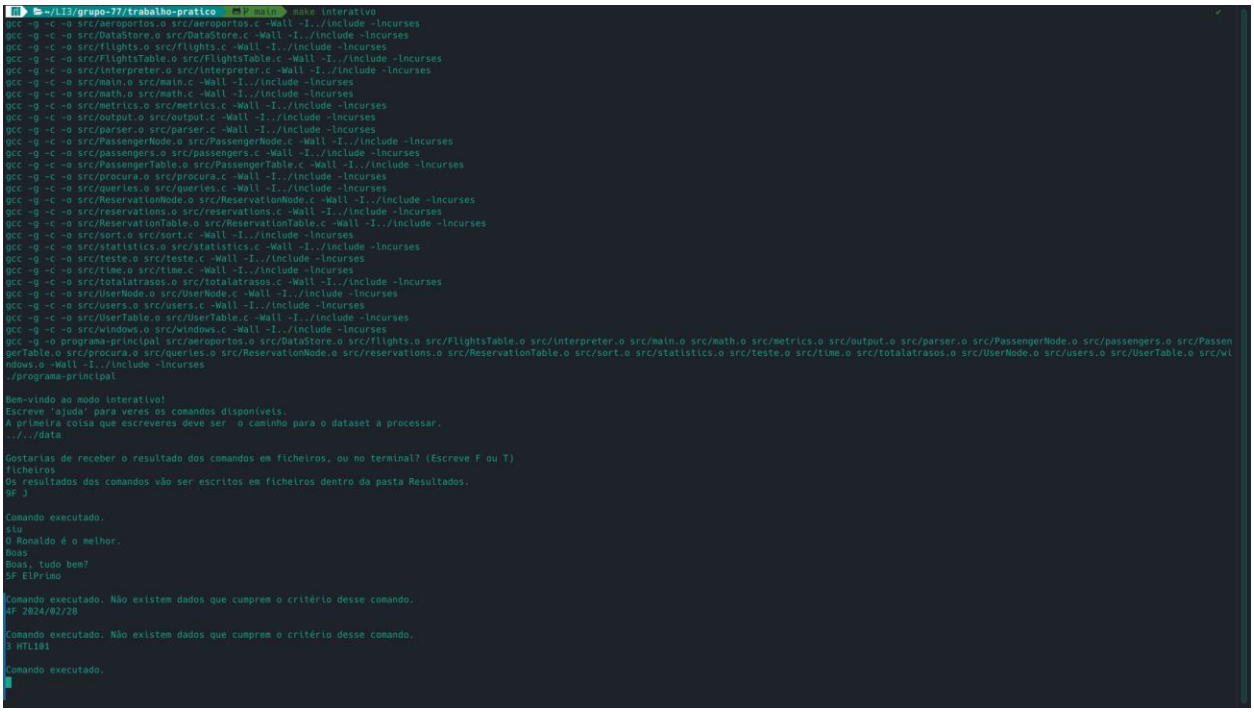
Um dos requisitos essenciais desta fase foi a criação de um modo interativo com interface gráfica no terminal. Para alcançar isso, optamos pela biblioteca ncurses, pois ela simplifica a implementação de um ambiente gráfico, além de padronizar e tornar mais fácil nossa abordagem e também porque já nos era conhecida. Este modo gráfico apresenta as seguintes funcionalidades:

Capacidade de processar consultas (queries) e fornecer seus resultados.

Em situações onde os resultados são extensos demais para serem exibidos simultaneamente na tela, eles são organizados em páginas, permitindo que o utilizador navegue entre elas.

Um menu de ajuda que explica detalhadamente a função de cada consulta.

Seguem-se alguns esboços do modo interativo:



```
Bem-vindo ao modo interativo!
Escreve 'ajuda' para veres os comandos disponíveis.
A primeira coisa que escreveres deve ser  o caminho para o dataset a processar.
../siu
O caminho não existe. Por favor indica um caminho válido.
../../data

Gostarias de receber o resultado dos comandos em ficheiros, ou no terminal? (Escreve F ou T)
terminal
Os resultados dos comandos vão ser reproduzidos no terminal.
siahudbwq
Comando não reconhecido. Escreve 'ajuda' para ver os comandos disponíveis.
h
Comando não reconhecido. Escreve 'ajuda' para ver os comandos disponíveis.
a
Comando não reconhecido. Escreve 'ajuda' para ver os comandos disponíveis.
ajuda
Lista de comandos disponíveis:
1 <ID>  |  (Lista o resumo de um utilizador, voo, ou reserva.)
2 <ID> [flights|reservations]  |  (Lista os voos ou reservas de um utilizador. Se não houver 2º argumento, serão listados os voos e reservas.)
3 <ID> | (Apresenta a classificação média de um hotel, a partir do seu ID)
4 <ID> | (Lista as reservas de um hotel, ordenadas por data de início)
5 <"nome"> <"data_início"> <"data_fim"> | (Lista os voos com origem num dado aeroporto, num intervalo de tempo)
6 <ano> <N> | (Lista os N aeroportos com mais passageiros num dado ano)
7 <N> | (Lista os N aeroportos com a maior mediana de atrasos)
8 <ID> <"data_início"> <"data_fim"> | (Lista a receita total de um hotel, num intervalo de tempo)
9 <"prefixo"> | (Lista os utilizadores cujo nome começa por um dado prefixo)
10 [ano[mês]] | (Apresenta várias métricas como o número de novos utilizadores registados, número de voos, número de passageiros, número de passageiros únicos e número de reservas de acordo com o ano e/ou mês fornecidos)
No fim de cada número de comando, podes escrever F para uma forma de output diferente.
Sair ou q | (Sai do programa)
1

Comando executado. Não existem dados que cumprem o critério desse comando.
1F JéssiTavares910

Comando executado.
10

Comando executado.
```

```
--- 1 ---
id: JaiAlves1726
name: Jaime Alves

--- 2 ---
id: JaiAlves1885
name: Jaime Alves

--- 3 ---
id: JaimAmaral
name: Jaime Amaral

--- 4 ---
id: JAnjos
name: Jaime Anjos

--- 5 ---
id: JAzevedo
name: Jaime Azevedo

--- 6 ---
id: JaimBarros
name: Jaime Barros

--- 7 ---
id: JaBrito
name: Jaime Brito

--- 8 ---
id: JaCarvalho-Vieira349
name: Jaime Carvalho-Vieira

--- 9 ---
id: JCastro476
name: Jaime Castro

--- 10 ---
id: JCoelho
name: Jaime Coelho

--- 11 ---
id: JCosta1289
name: Jaime Costa

--- 12 ---
id: JaCruz-Branco
name: Jaime Cruz-Branco

--- 13 ---
id: JaiGaspar
```

Capítulo 7

Dificuldades Sentidas

Nesta fase, apesar dos desafios apresentarem uma complexidade significativamente maior, sentíamos-nos mais organizados e preparados, já que tínhamos uma arquitetura base e uma ideia clara da estruturação do projeto. Contudo, isso não significa que não enfrentamos dificuldades ao longo do desenvolvimento.

Um dos desafios encontrados foi no encapsulamento do programa. No início, acreditávamos que nosso programa estava completamente encapsulado, mas depois, para além das sugestões dos nossos professores, descobrimos que nossas estruturas de dados estavam a ser acessadas diretamente pelos módulos de estatísticas e queries, comprometendo o encapsulamento. Esse problema foi resolvido definindo funções específicas dentro do módulo de cada estrutura. Além disso, sempre houve desafios relacionados à modularidade do programa.

Uma outra dificuldade encontrada e que se notou até ao fim do prazo limite foi o tempo de execução do programa. Tentamos reduzir ao máximo, no entanto o nosso projeto demora cerca de 8 segundos. Acreditamos que um trabalho mais aprofundado resultaria num melhor desempenho.

Capítulo 8

Conclusões

Para concluir, apesar deste projeto, e em particular esta segunda fase, representarem um grande desafio devido à introdução de muitos conceitos novos e complexidade na implementação, sentimo-nos confiantes de que estávamos à altura desse desafio. Isso deve-se em grande parte ao trabalho sólido realizado durante a primeira fase e as boas bases que estabelecemos.

Temos a percepção de que aprimoramos o nosso trabalho em todos os aspectos em relação à fase anterior. Importante destacar que aplicamos de forma adequada os fundamentos essenciais deste projeto, como a modularidade e o encapsulamento.