Universidade do Minho – Licenciatura em Engenharia Informática

Universidade do Minho
Escola de Engenharia

LABORATÓRIOS DE INFORMÁTICA III

Relatório da 2ª Fase

Grupo 09

a103993, Júlia Bughi Corrêa da Costa

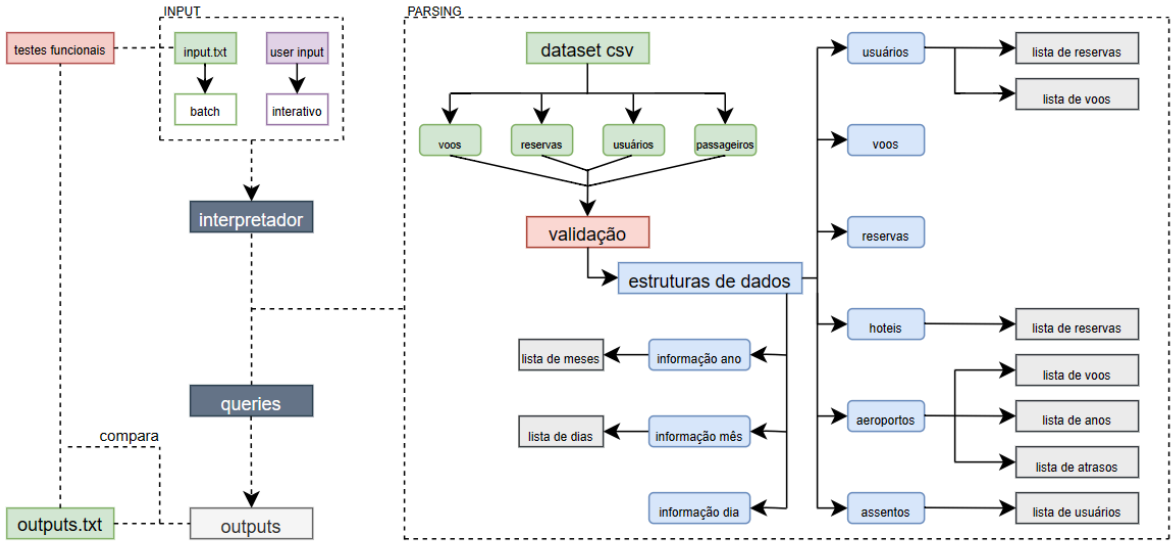
a104613, Luís Pinto da Cunha

Índice

Introdução ao Projeto

O objetivo do projeto é a consolidação dos conhecimentos da linguagem C, mas também uma avaliação da capacidade de gestão das ferramentas essenciais a todo programador, como, por exemplo, o encapsulamento e a modularidade, o uso adequado de estruturas de dados e medições de desempenho.

Estruturamos o trabalho de modo a maximizar a eficiência e a organização do código. Com o objetivo de facilitar a visualização da arquitetura do nosso projeto, elaboramos o esquema abaixo.



Guardamos os dados que julgamos importantes em nove tipos de estruturas, de modo a otimizar a execução das queries. Cada uma delas possui uma hashtable associada, como será explicado em maior detalhe na próxima seção.

Estruturas de Dados

Na primeira fase nós guardávamos os dados apenas em hashtables (users, voos, passageiros e reservas) e a realização da maioria das queries era feita percorrendo-se as hashtables, o que era totalmente ineficiente e contra a utilidade das hashtables.

Neste momento contamos com estruturas de dados: hashtables de hotéis e de aeroportos, lista de nomes e ids de usuários (ordem alfabética) e estruturas que guardam informações de anos, meses e dias.

A hashtables dos hotéis e dos aeroportos foram criadas para facilitar a realização das queries relativas a hotéis (3, 4 e 8) e as relativas a aeroportos (5, 6 e 7) . A hashtable dos hotéis é constituída por structs que guardam o id do hotel, nº de reservas desse hotel e uma lista das suas reservas. A hashtable dos aeroportos é constituída por structs que guardam o id do aeroporto, nº de voos, nº de passageiros, nº atrasos de voos, uma lista dos voos, uma lista dos atrasos e uma lista dos passageiros de cada ano.

A lista de nomes e ids ordenada por ordem alfabética foi criada para tornar a query 9 mais rápida e eficiente.

A estrutura dos anos, meses e dias foi criada para auxiliar na realização da query 10. Esta estrutura é uma lista de structs que com a informação de um ano: nº usuários criados, nº voos, nº passageiros, nº passageiros únicos, nº reservas e uma hashtable com os meses desse ano. Da mesma forma, as structs dos meses guardam os mesmos dados só que possuem uma hashtable de dias.

Modo interativo

Nesta fase também tivemos de elaborar um novo modo de execução do programa através de instruções dadas no terminal. Neste modo o programa não recebe argumentos e todas as informações necessárias são escritas pelo utilizador, havendo uma interação utilizador-programa.

O programa inicia-se escrevendo o caminho para a pasta que contém os ficheiros de dados, que é logo verificada para evitar erros. De seguida, o utilizador tem de escolher o número da query que quer realizar, o modo de escrita da resposta e escrever os argumentos para que a query possa ser realizada. Quando não quiser realizar mais queries basta clicar na tecla ‘q’ e o programa termina.

Quanto aos modos de apresentação de resposta, existem duas opções: imprimir as respostas num ficheiro txt ou imprimir diretamente no terminal. No primeiro caso o utilizador tem que escrever o caminho para onde quer que o ficheiro txt seja criado, já no segundo, caso a resposta seja extensa, é apresentado um sistema de paginação que permite ao utilizador escolher a página que quer ver de modo a tornar mais fácil a análise das respostas.

Foram criadas todas as precauções possíveis para reagir a possíveis erros de input do utilizador, sendo os erros assinalados e possibilitando sempre uma nova tentativa.

Análise de Desempenho

Para realizar a componente relativa aos testes funcionais do programa, elaboramos o "programa-testes". Decidimos optar por um formato simples, onde após cada query ser testada individualmente, o tempo de execução e a indicação se o teste passou são impressos no ecrã.

Através das mudanças que implementamos ao longo do tempo, fomos capazes de reduzir imensamente o tempo de execução. Apesar de algumas dessas alterações serem às custas de um maior uso da memória, de modo geral, chegamos a resultados muito positivos.

Realizamos 10 testes para cada query em cada máquina (especificadas abaixo) e obtivemos os seguintes dados:

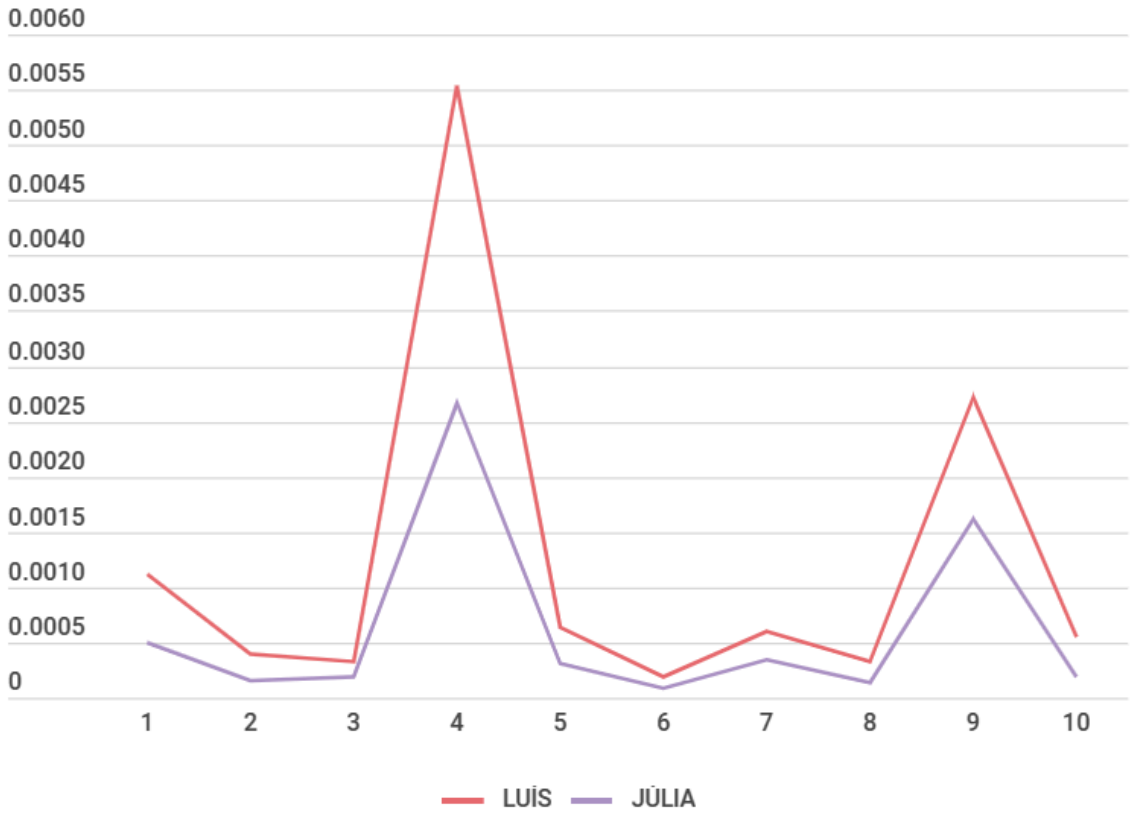
MÉDIA DE TEMPO EM SEGUNDOS

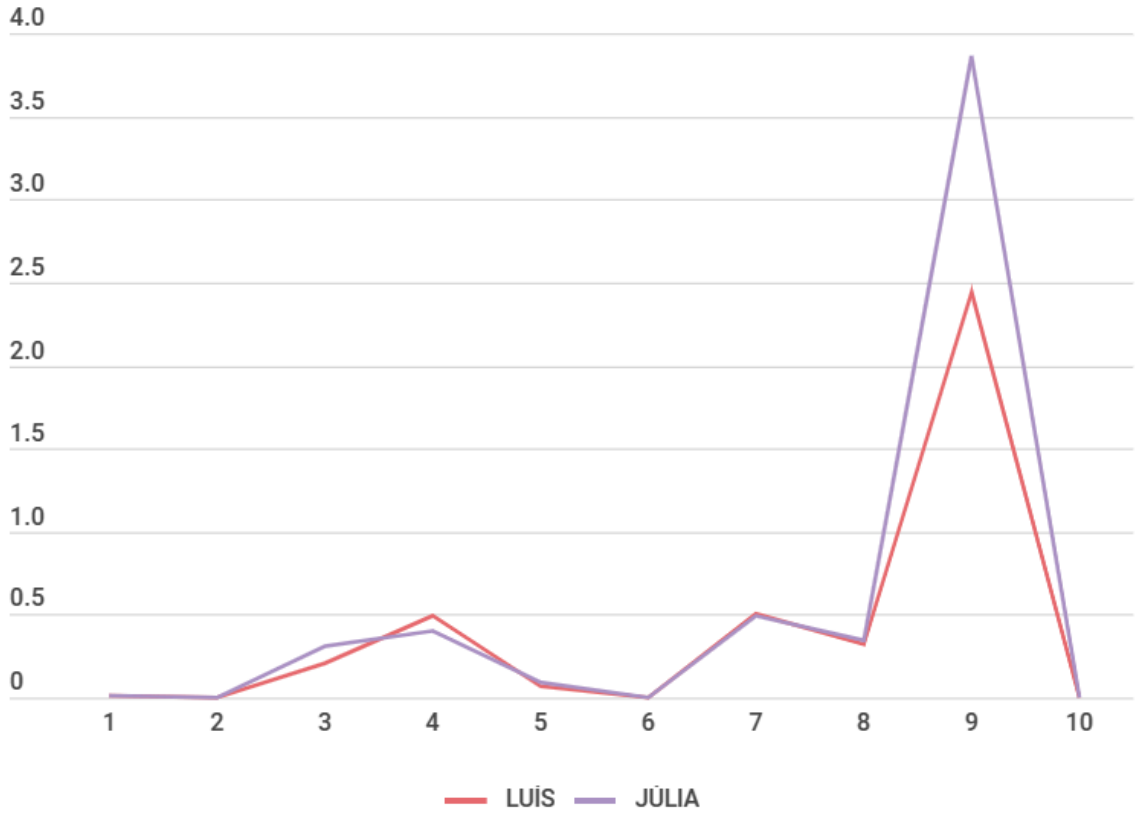
REGULAR DATASET

QUERIES	LUÍS	JÚLIA
1	0,001112	0,000494
2	0,000399	0,000157
3	0,000326	0,000188
4	0,005544	0,002665
5	0,000632	0,000302
6	0,000196	0,000083
7	0,000605	0,000349
8	0,00033	0,000145
9	0,002723	0,001617
10	0,000546	0,000193
TOTAL	0,233726	0,159672
MEM	28,56 MB	28,50 MB

LARGE DATASET

QUERIES	LUÍS	JÚLIA
1	0,014305	0,008615
2	0,002754	0,002305
3	0,203184	0,312861
4	0,492509	0,39625
5	0,071842	0,086027
6	0,002807	0,002473
7	0,499357	0,496043
8	0,315488	0,347022
9	2,441799	3,862501
10	0,000656	0,000324
TOTAL	43,598778	45,49482
MEM	3650,53 MB	3650,16 MB





* Luís : processador i7-7700HQ // ram 16GB
* Júlia : processador i5-1135G7 // ram 8GB

Através da análise das tabelas é notável que certas queries são realizadas mais rapidamente que outras. As queries 1 ,2 ,5, 6 e 10 são muito mais rápidas porque só requerem o acesso a valores de uma hashtable. As queries 3, 4, 7 e 8 demoram mais tempo porque, apesar de também irem buscar os valores a hashtables, têm de os processar posteriormente envolvendo cálculos e ordenações.

O pior caso é definitivamente a query 9, dado que envolve uma passagem por uma lista ligada que pode ser mais ou menos extensa dependendo do prefixo dado como argumento (tempo linear).

Outro caso particular é a query 4, que no dataset pequeno é a menos eficiente, porém no dataset grande, apesar de ainda ser uma das mais lentas, não apresenta um aumento linear como a query 9. Isso se deve ao facto de se tratar de uma query baseada em ordenação, com complexidade n.log(n).

Mudanças relativas à 1ª fase

Após a conclusão da 1ª fase e início da implementação do dataset maior, deparamo-nos com um problema. O nosso tempo de execução estava elevadíssimo, devido ao facto de utilizarmos append em todas as GLists. Como já sabíamos, dar append é uma operação muito custosa relativamente à prepend (detalhe o qual não tivemos em atenção inicialmente), e, por isso, efetuamos a substituição.

Seguidamente, fizemos algumas modificações sugeridas pelos professores que avaliaram o nosso trabalho na 1ª apresentação. Por exemplo, implementamos uma maior divisão de ficheiros (melhor encapsulamento e modularidade), o retorno de “consts” nas funções de get, uma melhor legibilidade nos ficheiros, uma documentação mais detalhada e tornamos a makefile um pouco mais elaborada.

Também por sugestão dos professores, fundimos as funções em modo normal e modo f, fazendo-se assim necessária a criação de uma função global de impressão de respostas. Essa mudança posteriormente facilitou a implementação da funcionalidade interativa do programa, para além de tornar o programa mais conciso e menos repetitivo.

Para melhorar o programa, criamos novas estruturas de dados já anteriormente apresentadas, porém o tempo de execução ainda não estava satisfatório. Suspeitamos que o problema consistia no facto de que estávamos a guardar as datas em strings e a realizar operações com esse formato. Chegamos a conclusão de que seria mais eficiente guardar esses dados numa estrutura de inteiros e isso provou-se verdade. O tempo de execução baixou drasticamente.

Por fim, precisávamos de aprimorar a utilização da memória, que estava por volta dos 5000 MB. Para isso, substituímos todas as GLists por GSLists, já que a presença de dois apontadores não nos era útil, tendo em conta que só percorremos as listas numa direção. Para potencializar ainda mais o uso da memória, passamos a guardar os ids de voos, reservas e hotéis em inteiros em vez de strings. Com essas mudanças, conseguimos reduzir o uso para aproximadamente 3600 MB.

Conclusão

Nesta segunda fase do projeto percebemos a importância de ter atenção aos pequenos detalhes que podem ter um grande impacto. A maior parte das dificuldades que tivemos derivaram de pequenos aspetos que considerávamos irrelevantes, mas que depois no dataset grande se tornaram decisivos em termos de tempo de execução e gasto de memória.

De um modo geral, foi um grande processo de aprendizagem onde pusemos em prática aquilo que já tínhamos abordado e onde aprendemos mais sobre estruturas de dados, modularidade, encapsulamento e consumo de memória.