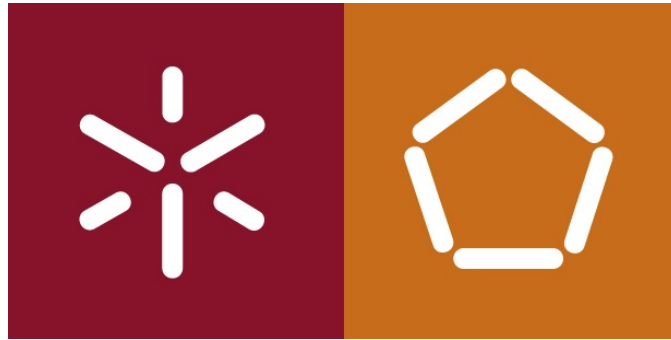


UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA



Stack Overflow

LABORATÓRIOS DE INFORMÁTICA III

GRUPO 51



Carla Cruz
A80564



Maria Dias
A81611



Pedro Freitas
A80975

June 12, 2018

Contents

Introdução	2
Leitura dos dados e Ferramentas usadas	3
Descrição das Coleções Utilizadas	3
Modularidade e Abstração de Dados	4
Encapsulamento	4
Interrogações	5
Interrogação 1	5
Interrogação 2	5
Interrogação 3	5
Interrogação 4	6
Interrogação 5	6
Interrogação 6	6
Interrogação 7	6
Interrogação 8	7
Interrogação 9	7
Interrogação 10	7
Interrogação 11	7
Resultados e Testes de Performance	8
Grafo de Dependências	8
Conclusão	9

Introdução

No âmbito da unidade curricular de Laboratórios de Informática III do segundo ano do Mestrado Integrado em Engenharia Informática foi proposto, numa segunda fase, o desenvolvimento de um projeto em linguagem Java, que visa o desenvolvimento de um sistema capaz de processar ficheiros XML que armazenam as várias informações utilizadas pelo Stack Overflow e a execução de um conjunto de interrogações de forma eficiente.

Este projeto já foi por nós abordado na linguagem C, e este surge agora no seguimento do anterior, com o objetivo de o converter (e, muito possivelmente, melhorar) para Java e, no final, comparar os resultados obtidos em termos de performance, facilidade de programação, entre outros.

Leitura dos dados e Ferramentas usadas

Como foi previamente referido, este projeto, desenvolvido em Java, podia ser auxiliado de diversas ferramentas.

Em relação aos ficheiros .xml que foram disponibilizados, foi necessário proceder ao parsing destes. Este parsing foi realizado por partes, em vez de carrega-lo na totalidade para a memória. Foi utilizada a biblioteca *stAX* (Streaming API for XML).

Para guardar os dados lidos após o parsing dos ficheiros .xml, o Java oferece um conjunto de classes que implementam as estruturas de dados mais utilizadas oferecem uma API consistente entre si, nomeadamente, *HashMaps* e *TreeMaps*.

Descrição das Coleções Utilizadas

Para a realização deste projeto, como referido acima, foi necessário proceder ao armazenamento de dados utilizando *HashMaps* e *TreeMaps*. Estes referidos, tem como vantagem, a sua fácil utilização que permite uma fácil procura através das suas keys. Estas coleções também provêm de métodos de ordenação, inserção e remoção já pré-definidos que facilita a sua utilização.

As principais diferenças entre estes são a sua organização dos dados em que um se baseia numa tabela hash enquanto que o outro se baseia numa árvore, sendo as *HashMaps*, as coleções mais usadas no desenvolvimento deste projeto.

Modularidade e Abstração de Dados

Como temos aprendido ao longo dos últimos tempos, é uma boa prática manter o código modular e abstrato em relação ao tipo de dados com que trabalhamos.

Encapsulamento

Seguindo o pensamento referido, separámos o nosso programa em diferentes *Classes*, deixando o mínimo de métodos visíveis para o exterior. Estes métodos definidos podem ser chamados fora destas Classes, como por exemplo, os "gets" e os "sets", e estes devolvem uma cópia da informação contida na Coleção, de modo a não comprometer o encapsulamento dos dados.

Interrogações

Após o desenho e a implementação das estruturas que servem de suporte ao desenvolvimento do programa pretendido, surge agora o momento de tirar partido de tais estruturas e começar a fazer interrogações ao programa, com o intuito de demonstrar o seu nível de praticidade para o problema em questão e também perceber o nível de fiabilidade que ele proporciona.

Antes de interrogar o programa, é feita uma inicialização da estrutura *TAD_community*, o que equivale a dizer que se inicializam as árvores dos Posts, Users e Tags, e a Hash Table dos Posts. Posteriormente, é efetuado o carregamento de dados para as estruturas, o que acontece, como já foi referido, no parser.

Depois da inicialização e do load da *TAD_community*, o programa está apto a responder às interrogações especificadas. Para isso, recorremos às estruturas criadas para nelas procurarmos a informação pretendida, através de funções definidas. Finalmente, seguido da resposta às interrogações, é necessário libertar o espaço em memória alocado pelas estruturas utilizadas. Para esse efeito, é definida uma função *clean* que limpa toda a informação da estrutura.

No ficheiro *interface.c* estão codificadas todas as interrogações definidas, as quais serão abordadas de seguida.

Interrogação 1

Esta interrogação retorna o título e o nome do autor de um post, dado o id do mesmo. Caso o post dado seja uma resposta, deverá retornar as informações (título e autor) da pergunta correspondente.

Estratégia: Para responder a este problema procuramos o Post na árvore dos Posts com esse id. De seguida verifica-se qual o tipo do Post. Se este for do tipo 1 (pergunta) através do *OwnerId* procuramos o User que é dono desse post e devolvemos um par com o título da pergunta e o username do dono. Se for do tipo 2 (resposta), através do *ParentId* encontramos a pergunta à qual esta resposta pertence e fazemos o mesmo procedimento do tipo 1.

Interrogação 2

Esta interrogação pretende obter o top N utilizadores com maior número de posts de sempre, tendo em conta todas as perguntas e respostas dadas por cada utilizador.

Estratégia: Para responder a esta interrogação, acedemos à árvore de Posts de cada utilizador, contando o número de nodos que cada árvore contém. Com base neste valor, é criada uma Heap que é ordenada com base nos valores lidos. No final é feito, N vezes, o pop desta Heap gerada e inserimos numa lista ligada os id's correspondentes a cada nodo retirado.

Interrogação 3

Esta interrogação procura, dado um determinado intervalo de tempo, obter o número total de posts (identificando perguntas e respostas separadamente) efetuados nesse período.

Estratégia: Dado a data de início e fim deste intervalo de tempo, foi possível consultar este intervalo de tempo na tabela de Hash, (usando a função *incrementaData* para avançar sucessivamente), em que cada dia continha o respetivo número de perguntas e respostas dadas. Estes valores

foram somados, obtendo assim, o valor todas de perguntas e respostas dadas no intervalo de tempo dado.

Interrogação 4

Esta interrogação retorna os ids de todas as perguntas contendo uma dada tag, por ordem cronológica inversa da data das perguntas.

Estratégia: Para responder a esta interrogação, acedemos a todas as posições da tabela de Hash de Posts que correspondem ao período de tempo dado, e em cada um dos arrays de Posts foram filtradas as publicações do tipo pergunta, cujo título continha a tag dada (percorrendo todas as tags da pergunta e comparando com a tag fornecida), e foram depois inseridas numa heap ordenada pela data mais recente.

Finalmente, removemos da Heap e inserimos numa lista ligada todos os Posts, resultando numa lista ordenada por ordem cronológica inversa, como era pedido.

Interrogação 5

Esta interrogação retorna a informação do perfil e as últimas dez publicações de um user do qual é dado o ID.

Estratégia: Dado o ID do user, procuramos na árvore dos users aquele que tinha esse mesmo ID e recorremos a duas funções auxiliares que, dado um User, retornam o seu perfil e a Heap dos seus Posts, ordenados por ordem cronológica inversa. Desta Heap, fazemos pop de dez publicações, ficando com os dez Posts mais recentes deste User.

Finalmente, criamos um USER com o perfil e posts que calculamos, e retornamos essa informação.

Interrogação 6

Esta interrogação procura, dado um determinado intervalo de tempo, obter os ids das N respostas com mais votos, por ordem decrescente do número de votos.

Estratégia: Acedemos à tabela de Hash de Posts e filtramos os arrays que contêm as publicações criadas no período de tempo em questão, sendo depois cada um desses arrays percorrido, filtrando as publicações do tipo resposta. Posteriormente, esses Posts foram inseridos numa Heap inicialmente vazia, ordenada pelo Score dos mesmos. Para obter a lista ligada das N respostas com mais votos (sendo que os votos são equivalentes ao Score da resposta), fizemos pop de N Posts da Heap, e inserimos os IDs correspondentes na lista a devolver, ficando assim com uma lista ordenada por ordem decrescente do número de votos das respostas.

Interrogação 7

Esta interrogação retorna os IDs das N perguntas com mais respostas num dado intervalo de tempo, por ordem decrescente do número de respostas.

Estratégia: Tal como foi feito para as restantes interrogações que envolvem um dado intervalo de tempo, acedemos à tabela de Hash de Posts e filtramos os arrays que contêm as publicações criadas nesse período. De seguida, cada um desses arrays foi percorrido, calculando o número de respostas de cada publicação e inserindo todos os Posts numa Heap inicialmente vazia, ordenada pelo maior número de respostas (efetuadas também no intervalo de tempo em questão). Para obter a lista ligada das N perguntas com mais respostas, fizemos pop de N Posts da Heap, e inserimos os IDs correspondentes na lista a devolver, ficando assim com uma lista ordenada por ordem decrescente do número de respostas.

Interrogação 8

Esta interrogação devolve os IDs das N perguntas cujos títulos contêm uma dada palavra, por ordem cronológica inversa destas.

Estratégia: Para responder a esta interrogação, inicializamos uma Max Heap e preenchemo-la com todos os posts que contêm uma dada palavra, à medida que percorremos a árvore de Posts. A heap é ordenada conforme a data mais recente. Para obter a lista ligada que se pretende retornar, fazemos pop de N Posts da heap e inserimos o seu ID na lista ligada, ficando assim com uma lista ordenada por ordem cronológica inversa.

Interrogação 9

Esta interrogação devolve as últimas N perguntas (por ordem cronológica inversa) em que participaram dois utilizadores específicos.

Estratégia: Nesta questão, para cada utilizador, percorremos a árvore de Posts respetiva a cada um. Caso o tipo dos Posts do ID 1 fosse 1, e encontrássemos um do tipo 2 correspondente ao ID 2, verificávamos se o Post comentado pelo ID 2 era correspondente ao publicado do ID 1 e vice-versa. Caso ambos fossem do tipo resposta, verificávamos se o Post comentado por ambos coincidia. Se estas verificações fossem verdadeiras, inserimos o Post numa Heap ordenada por ordem cronológica inversa. No final fizemos o pop da Heap, inserindo os ID's lidos, numa lista, sendo esta retornada.

Interrogação 10

Esta interrogação devolve a melhor resposta dada a uma certa pergunta, da qual é passada o ID.

Estratégia: Percorremos a árvore de Posts, e todas as publicações que são resposta à pergunta dada são inseridas numa Max Heap inicialmente vazia, e são ordenadas conforme a fórmula dada para calcular a sua classificação média. No final, é retirado o maior elemento da heap (que é do tipo Post), e é calculado e devolvido o seu ID.

Interrogação 11

Esta interrogação retorna, dado um intervalo de tempo, os IDs das N tags mais utilizadas pelos N utilizadores com melhor reputação, naquele período.

Estratégia: Dado um determinado número inteiro N criamos uma lista com esse tamanho e são feitos N pops na estrutura heapU que, ordenados pela reputação, nos preenche essa lista com os N users com mais reputação. Já obtida a lista o passo seguinte é percorrer a heap com todos os posts de cada user presentes nessa mesma lista. Nessa execução é utilizada uma estrutura auxiliar ATNum (array dinâmico de TNum - Pares de Tag-Número de ocorrência). Para cada post percorrido verifica-se se esse post pertence ao intervalo dado e, caso pertença, percorre-se todas as tags do mesmo, guardando na estrutura ATNum. Se a tag já estiver guardada nessa estrutura é incrementado o número de ocorrências dessa tag. Obtendo todas as tags e o número de ocorrências das mesmas, ordenamos o nosso ATNum de forma decrescente segundo o número de ocorrências de cada tag. Por fim para obter o id das N tags mais usadas, ao mesmo tempo que percorremos a estrutura ATNum, fazemos uma procura na estrutura TCD community em que procuramos a correspondência de cada tag da ATNum, obtendo assim o seu id. Para esta procura foi necessário uma função gboolean, onde o último parâmetro seria uma estrutura Duplos com a LONG list a ser preenchida e com um TNum com a tag a ser comparada. Cada id é guardado e depois retornado na LONG list.

Resultados e Testes de Performance

Após o desenvolvimento e codificação do projeto, foram realizados alguns testes de performance, que consistem na obtenção dos tempos de execução das interrogações, usando os backups *android* e *ubuntu*, dos quais lemos os ficheiros Users.xml, Posts.xml e Tags.xml, usando a biblioteca standard de C *time.h*.

Em seguida são demonstrados os resultados dos mesmos:

Figure 1: Tempos na dump android

Figure 2: Tempos na dump ubuntu

Uma vez que a quantidade de dados aumenta do backup android para o backup ubuntu, é aceitável que os tempos de execução para os carregar, ou seja, a fase de leitura e inserção de dados, também aumente consideravelmente. Comparando os valores de tempo de execução das interrogações, é possível observar nos gráficos que o tempo de execução não varia significativamente, uma vez que são registadas variações na ordem dos milissegundos, devido a uma estruturação dos dados independente da quantidade de dados acumulada.

Estes testes foram realizados numa máquina com um processador de 3,1GHz, com uma memória Ram de 8GB.

Grafo de Dependências

Figure 3: Grafo de dependências

Conclusão

Ao longo deste projeto trabalhamos com blocos de dados de tamanho muito maior ao que tínhamos feito até então, forçando à utilização de técnicas particulares, como as que foram mencionadas na introdução deste relatório. A estratégia seguida foi a de conceber o trabalho de forma a que fosse facilmente modificável e, apesar da complexidade, o mais otimizado possível.

As maiores dificuldades existiram na implementação da abstração de dados, com o objetivo de delimitar a vista do utilizador, pois era uma prática não conhecida até este ponto e exige um cuidado acrescido por parte do programador ; na limpeza das estruturas, devido ao facto de não haver conhecimento entre os módulos; na gestão das estruturas de forma eficiente, para obter uma boa performance.

Consideramos que este trabalho foi muito importante para o aprofundar do conhecimento no que toca ao uso da linguagem C e apreendimento das técnicas abordadas ao longo do relatório, sem esquecer o uso de algumas das ferramentas que nos foram apresentadas no desenvolver do projeto (como o valgrind), que nos ajudaram a progredir como programadores e a ter uma maior autonomia na correção de erros.