Trabalho Prático
Laboratórios de Informática
(Parte 2)

Grupo 6

Guilherme Viveiros e Gonçalo Faria

Departamento de Informática
Universidade do Minho

Conteúdo

Capítulo 1

Introdução

Em seguimento à primeira parte do projeto da Unidade Curricular de Laboratórios de Informática III[2], e com o intuito de melhor compreender as aplicações do paradigma de programação orientada aos objetos, foi proposto na segunda parte deste a re-implementação da primeira, no entanto na linguagem de programação java.

Capítulo 2

Carregamento dos dados

Após compreender as implicações do parser Libxml2[1] usado na primeira parte, decidimos, sendo que apenas é preciso aceder aos ficheiros uma vez, que usar um parser DOM[2] é tremendamente despendioso. Nos parsers DOM, é criada uma estrutura que representa o documento xml que possibilita o acesso a qualquer parte deste de forma extremamente eficiente. Estes mecanismos, não necessariamente precisos nesta aplicação, ocupam uma enorme quantidade de memória e tornam a análise de dados de enorme dimensão, como é o caso dos dados do StackOverflow, impraticável.

Em alternativa, optamos pelo uso de um parser SAX[3]. A interface SAX, funciona criando eventos à medida que o parser encontra os diferentes nós xml[1]. Esta é usada estipulando os métodos que lidam com estes eventos assim como os mecanismos que permitem recuperar os dados lidos.

Na implementação do parser foi criada uma hierarquia de classes para processar os diferentes ficheiros xml. No topo da hierarquia, encontra-se a classe abstracta SAXStackOverflow[4]. Esta classe implementa DefaultHandler[5] assim como, para além de definir os métodos em comum às classes de tratamento dos ficheiros xml, requer a todas estas que implementem a definição do método rowInspector[6].

Tendo em consideração que todas as entradas dos ficheiros encontram-se no atributo row, o que o método que analisa os atributos faz, no caso de um dado atributo ser do tipo row, é enviar o nó xml à função rowInspector ou descartá-lo caso contrário. Esta formulação possibilita às classes que estendem SAXStackOverflow, de simplesmente especificar quais os dados pretende recolher e proporcionar os mecanismos de os incorporar no modelo.

Para inicializar o parser de SAX com o respectivo 'descendente' de SAXStackOverflow foi também adiciona uma classe que simplifica todo este processo designada StackOverflowParser[7].

---

[1] http://www.xmlsoft.org/

[2] Document Object Model

[3] Simple API for XML

[4] **public abstract class** SAXStackOverflow **extends** DefaultHandler

[5] **public class** DefaultHandler **implements** EntityResolver, DTDHandler, ContentHandler, ErrorHandler

[6] **abstract public void** rowInspector(**Attributes** atts)

[7] **public class** StackOverflowParser

Capítulo 3

Hierarquia de classes

Sendo que uma grande parte da primeira parte deste projeto foi já dedicada a explicar em detalhe, as diferentes estruturas de dados usadas, assim como as ideias por detrás das soluções às interrogações, não usaremos este relatório para as repetir. Principalmente, esta decisão é devida ao facto que os algoritmos e ideias desenvolvidas na primeira parte, foram quase que inteiramente reutilizados devido ao seu desempenho na implementação em C.

Na realização desta parte, unicamente convertemos o já desenvolvido modelo para a linguagem java e o paradigma de programação orientada aos objetos.

3.1 Modelo

Em concordância com a estrutura do projeto definida pelos docentes, a classe principal, responsável pela implementação da interface TADCommunity, é a classe que designamos Comunidade.

Esta classe contém abstratamente as mesmas estruturas de dados que já a sua versão de C continha, três HashMaps e um TreeSet (em alternativa ao array intervalar).

O TreeSet destina-se a facilitar buscas intervalares de publicações sem ter de percorrer todo o conjunto destas, tal como o correspondente Array intervalar referido na primeira parte. Este TreeSet foi usado em alternativa ao Array definido em C por forma manter a compatibilidade que as Collections de java oferecem. Embora possivelmente menos eficiente devido a todo um conjunto de mecanismo para manter a árvore que este implementa balanceada. Apesar de tudo isto, acreditamos que a simplicidade que as collections de java oferecem compensam as possíveis imperfeições, quando aplicado esta tarefa, do TreeSet.

Os HashMaps indicados correspondem ao HashMap dedicado a fazer a correspondência entre identificador e publicação, identificador e utilizador assim como a do nome da tag e o objeto que a representa.

3.1.1 Tags

Para representar as tags foi criada uma classe com o mesmo nome que contém como variáveis privadas o nome assim como o código.

3.1.2 Publicação

Dados os mecanismos de hierarquia que java proporciona, em vez de fazer a classe publicação, como a classe que suporta todas os tipos de publicações tal como foi feito em C, foi antes definida como classe abstrata que apenas define tudo que há em comum entre os diferentes tipos de publicação. Este tipo de classe define um comportamento que permite às suas subclasses subscrever, por forma a outras estruturas poderem armazenar e operar em todas elas genericamente.

Como variáveis privadas a publicação contém um identificador, nome, classificação, número de comentários, conjunto de tags usadas, data de criação e o identificador do utilizador que a criou, assim como métodos para aceder e alterar todas estas.

Adicionalmente, para além de ser atribuída uma ordem natural a esta classe, foi também criado, através dos métodos e variáveis de classe, um mapa de comparadores que permite obter acesso facilitado a estes simplesmente indicando o seu nome, por exemplo, para obter o comparador que ordena as publicações por classificação basta usar o método getComparator[1] com argumento "MaiorScore". Esta pequena mudança, simplificou significativamente o projeto dispensando da necessidade de criar uma classe para cada um dos vários comparadores usados.

Pergunta

A classe pergunta estende publicação e adiciona a esta um conjunto que guarda os identificadores das respostas a essa pergunta assim como métodos para aceder a estes e alterá-los.

Resposta

Esta classe estende publicação e adiciona a esta o identificador da pergunta cuja resposta em questão se adressa, tal como métodos para aceder a estes e alterá-los.

3.1.3 Utilizador

Esta classe contém como variáveis privadas o identificador, nome, número de questões, respostas, pequena biografia a reputação e um mapa com as intervenções do utilizador em questão.

O mapa de intervenções tal como a variante implementada em C, é um conjunto pensado por forma a tornar a sua interseção extremamente eficiente. Para efetuar a interseção entre os mapa de intervenções de dois utilizadores foi criado um método para este efeito[2].

Tal como na publicação, ao utilizador está associado um mapa de comparadores, que é uma variável de classe que guarda os diferentes tipos de comparadores de utilizador e permite o rápido acesso a estes.

[1] **static public** Comparator<Publicacao>getComparator(**String** name)
[2] **public** public Set<String>mutualIntervention(**Utilizador** x)

3.2 Controlador

3.3 Interface gráfica

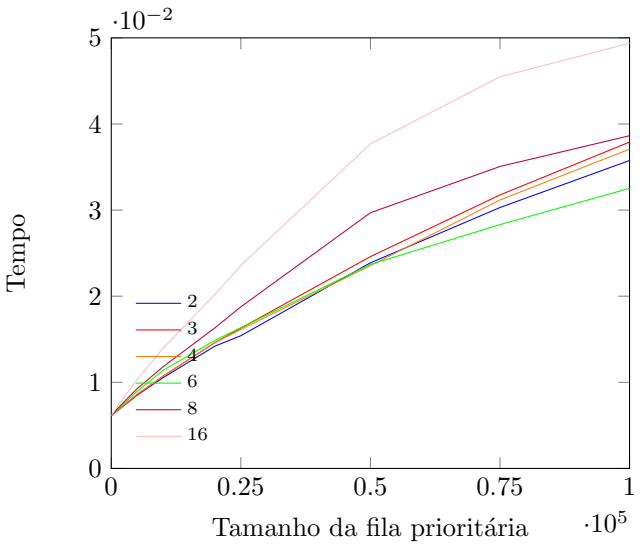
Capítulo 4

Fila prioritária limitada

Por forma a expandir o estudo iniciado na primeira parte, que tentava encontrar uma boa implementação de uma fila prioritária limitada, decidimos comparar os resultados da estrutura que criamos em C, para os seus diferentes parâmetros, com a que criamos para a substituir em java.

Denominada GeneralizedPriorityQueue[1], esta classe apresenta uma adição à PriorityQueue das collections de java que permite a esta manter-se limitada com os mecanismo já explicitados no relatório da primeira parte do projeto.

Ao gráfico já apresentado na primeira parte, tendo sido recriados os testes nas mesmas condições para GeneralizedPriorityQueue, adicionamos uma nova curva a este por forma a comparar as diferentes implementações.



(1)Os testes foram realizado na interrogação #2 os valores em cada entrada são o tempo médio nas 1000 vezes que a interrogação #2 foi repetida nos dados android, o eixo das abscissas corresponde à variação do argumento N. (2)Todos os testes foram realizado num portátil com : 8,00 GB (RAM) ,Intel(R) Core ,i7-4720HQ e CPU 2.60 GHZ. (3)Todos os testes usaram o parâmetro -Ofast no gcc.

[1] **public class** GeneralizedPriorityQueue<**E**>

Capítulo 5

Modelo MVC

O processo de construção da árvore, é naturalmente recursivo devido à natureza indutiva do tipo de dados.

Inicialmente, procura-se palavra a palavra, a sequência de caracteres que representa o operador no cimo da hierarquia. Enquanto esta operação é efetuada, a expressão inicial fica naturalmente dividida em duas expressões mais pequenas. A expressão da frente, que contém todas as palavras que já foram analisadas, e a expressão detrás, que contêm as palavras ainda por analisar.

Na eventualidade do operador em questão ser identificado, à expressão da frente, que como sabemos não contêm o operador, é aplicada a análise do próximo operador da hierarquia. À expressão detrás, é novamente aplicada a função com o mesmo operador, visto que ainda não foi analisada e é inteiramente possível que este ainda nela se encontre.

Adicionalmente, um nó é criado e é lhe associado o índice na tabela de sintaxe do operador que dividiu as expressões. À subárvore esquerda, é atribuido o resultado da árvore sintática da expressão da frente, e à subárvore da direita, o resultado da árvore sintática da expressão detrás.

Este processo vai se repetindo recursivamente, para todos os operadores da hierarquia, tornando as expressões cada vez mais pequenas, até que fiquem compostas unicamente por fórmulas atómicas.

Bibliografia

[1] Hristidis V. Farfán F. e Rangaswami R. ≪Beyond Lazy XML Parsing.≫ Em: *DEXA 2007* (2007), pp. 75–86.

[2] Gonçalo F. e Guilherme V. ≪Trabalho Prático Laboratório de Informática III≫. Em: Parte 1.(Grupo 6) (2018).