Universidade do Minho

Processamento de xml

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática 3

(2º Ano, 2º Semestre, 2017/2018)

A79003 Pedro Mendes Félix da Costa

A80453 Bárbara Andreia Cardoso Ferreira

Braga
Maio 2018

Índice

Introdução

Este trabalho foi feito no âmbito da unidade curricular laboratórios de informática 3 e tem como objetivo o processamento de qualquer *dump* da base de dados do site StackOverflow para responder a queries de forma eficiente, aplicando conhecimentos de algoritemia e programação imperativa.

Resolução inicial do Problema

Fazendo uma análise às queries, decidimos que seria necessário representar as seguintes entidades:

Posts

Para representar os **posts** dividimos os atributos por duas sub estruturas, cujos atributos de cada uma das sub estruturas são:

Questões

- Id
- Score
- Data
- Título da questão
- Id do autor da questão
- Nome do autor da questão
- Número de respostas
- Lista das respostas
- Tags

Respostas

- Id
- Score
- Data
- Número de comentários
- Id do autor da resposta
- Id da questão a que responde
- Nome do autor da resposta
- Referência da questão a que responde

Além dos dados fornecidos diretamente pelos ficheiros xml, decidimos também guardar na **questão** a lista das **respostas** de cada questão, bem como uma contagem destas. Na **resposta**, inversamente, guardamos uma referência para a pergunta a que esta responde. Com estas informações extra as pesquisas que envolvem relacionar estas duas entidades tornam-se mais eficientes.

Post

Ambas estas estruturas, questões e respostas, podem ser representadas de forma abstrata por um **Post**.

Esta estrutura é necessária, pois o utilizador precisa de guardar todas as questões e respostas, sem ser necessário uma distinção entre os dois tipos.

Utilizadores

Para representar os **utilizadores** guardamos os seguintes atributos:

- Id
- Biografia
- Nome
- Reputação
- Número de posts
- Lista dos posts

Mais uma vez foram guardadas mais informações para além das disponibilizadas diretamente pelo xml. Foi guardado o número de **posts** do utilizador (para determinar os utilizadores mais ativos de forma mais rápida) e a lista destes para permitir pesquisas mais rápidas.

Estruturas de Dados

Tipo concreto de dados

Para armazenar as entidades descritas acima foi implementado um TCD que as armazena de diferentes formas.

```
struct TCD_community{
    QUESTIONS_HTABLE questions;
    ANSWERS_HTABLE answers;
    SO_USERS_HTABLE users;
    TAGS_HTABLE tags;
    CALENDARIO calendarioQuestions;
    CALENDARIO calendarioAnswers;
};
```

Hashtables

Todas as entidades são armazenadas numa tabela de hash pois para todas são necessárias pesquisas por id (ou nome no caso das tags).

Tags

A tabela de hash das *tags* serve para criar uma associação $Nome \rightarrow Id$ visto que as questões guardam uma lista com os nomes das tags e para responder à query 11 é necessário obter os ids das mesmas.

Calendário

Para ser possivel manter as questões e respostas ordenadas por data foi pensada a utilização de árvores binárias ordenadas por data, mas isto foi considerado ineficiente quando comparada à solução escolhida. Foi então concebida uma estrutura à qual demos o nome de **Calendário** que permite acessos em tempo constante $O(1)$ a todos os elementos associados a uma determinada data. Guardamos assim duas instâncias desta na TCD, uma para perguntas e outra para respostas.

Esta estrutura permite:

- Guardar qualquer objeto desde que seja passada uma data associada ao mesmo.
- Iterar sobre os objetos, dado um intervalo de tempo, por ordem cronológica normal ou inversa, conforme a ordem dos argumentos.

A estrutura em si consiste numa matriz de quatro dimensões com um lista ligada em cada célula desta.

No primeiro nível temos uma lista de **anos**. Cada um destes, é constituído por uma lista de **meses** que são constituídos por uma lista de **dias**, cujo tamanho varia entre 29 e 31. Cada **dia** é constituido por 24 **horas**, e cada uma destas horas é uma lista ligada de elementos ordenada por data.

A utilização de uma lista ligada para representar uma hora é mais eficiente, pois é necessário fazer inserções ordenadas, que são, em quase todos os casos, feitas à cabeça devido aos posts serem inseridos quase cronologicamente.

DateTime

Para a implementação da estrutura **Calendário** foi necessário extender o **Date** para incluir a hora, minuto, segundo e milissegundo garantindo assim uma ordenação mais consistente. Foi então criado o **DateTime**.

```
struct _dateTime{
    int year, month, day;
    int hours, minutes, seconds, milisseconds;
};
```

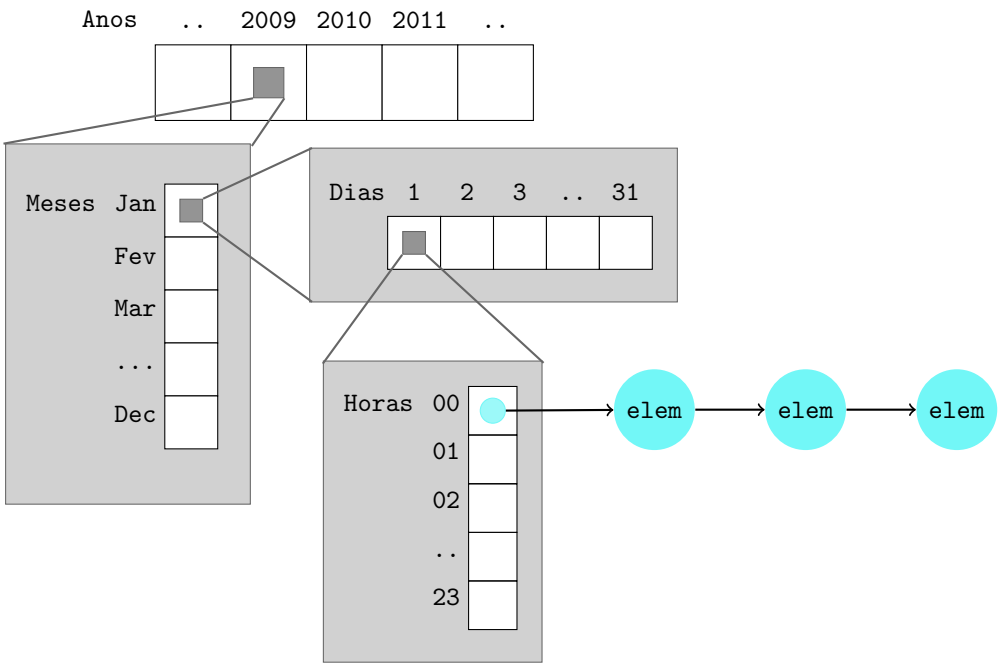

Figura 1: Representação gráfica do Calendario

String Rose Tree

Para auxiliar à resolução da query 11 foi implementada uma estrutura para contar *strings*. Esta, consiste numa árvore n-ária em que qualquer string representa um caminho único sobre a árvore. Assim, quando é inserida uma string, efetivamente, é guardado o número de vezes que esse caminho é percorrido.

Modularização Funcional e Resolução das queries

Para aceder aos dados da estrutura principal foi definida uma API simples que permite:

1. Pesquisas por id de **questões**, **respostas** e **utilizadores**.
2. Pesquisas de ids de **tags** dada a designação.
3. Pesquisa de listas, ordenadas por qualquer critério, de **utilizadores**, **questões** e **respostas**.
4. Pesquisas de **questões** filtradas por qualquer critério.
5. Pesquisas genéricas de **questões**/**respostas** num determinado intervalo de tempo.

Com estas funções a resolução da maioria das queries mostrou-se trivial.

Para as queries que necessitam de pesquisas por id (queries: 1, 5, 9 e 10) são resolvidas por **1**.

Para as queries que necessitam de pesquisas de utilizadores ordenados (queries: 2 e 11) são conseguidas através de **3**, sendo que assim basta fornecer uma função de comparação para definir o critério de ordenação.

Para as queries que necessitam de pesquisas de **perguntas**/**respostas** num intervalo de tempo ordenadas (queries: 6 e 7) são também resolvidas através de **3**. No caso de não ser necessária a filtragem do intervalo de tempo, simplesmente passamos as datas máximas definidas pelas macros `dateTime_get_epoch()` e `dateTime_get_year2038()` em *dateTime.h*.

Para as queries que necessitam de listas de questões que obedeçam a um determinado critério (queries: 4 e 8), são conseguidas através de **4**.

Nos casos em que estes métodos especializados não são necessários (query: 3) fizemos uso de uma iteração genérica através de **5**.

Como referido anteriormente, (3.4) para a resolução da query 11 foi implementada uma estrutura para contar tags, para que esta contagem fosse eficiente. Com isto, o único trabalho necessário foi obter as **tags** das **questões** dos **utilizadores** com melhor reputação por **1**. Em seguida, obtemos a lista ordenada por ocorrência das tags e através de **2** obtemos os ids das mesmas.

Conclusões e Trabalho Futuro

Em suma, o grupo considera que o trabalho foi realizado na sua totalidade de forma eficiente e correta, respondendo a todas as queries.

Um aspeto que poderia ser melhorado é a ordenação de utilizadores. Estes, foram guardados apenas numa tabela de hash e quando é necessária um lista ordenada dos mesmos, esta, tem de ser percorrida na sua totalidade. Esta decisão, centrou-se no facto de que nenhuma única ordenação se apresenta particularmente vantajosa, face às demais.