Projeto de Laboratórios de Informática 3
Grupo 4

Daniel Costa (A81302) Carlos Castro (A81946) Luís Macedo (A80494)

5 de Maio de 2018

1 Introdução

No sentido de analisar o funcionamento do site mais procurado pelos estudantes Engenharia Informática, *Stackoverflow*, a equipa docente requesitou aos alunos de LI3 que criassem um programa que respondesse a 11 querys, sobre a atividade do site.

O próprio centra-se nas perguntas e respostas da comunidade sobre qualquer problema na área da Informática, ou seja, após a criação de uma pergunta, qualquer utilizador que veja essa pergunta pode responder e receber upvotes, se a resposta ajudar na resolução do problema, ou downvotes, caso aconteça o contrário.

Para tal, são nos fornecido ficheiros xml com toda a informação dos usuários, perguntas/respostas, tags, etc.

2 Tipo Concreto de Dados

A fim de guardar a diferente informação obtida pela análise dos ficheiros fornecidos, foi necessário fazer um estrutura de dados.

Essa estrutura é constituida por:

- *tables*, um *array* de *TTable*, com 3 elementos, que guarda a informação sobre os Utilizadores, as Perguntas e as Respostas;
- *maps*, um *array* de *TRowMap*, com 2 elementos, que guarda pares chave-valor do tipo id-*TRow* dos Utilizadores/Posts;
- *mapWords*, um *TRowMapList* que guarda pares palavra-*TRow's* das Perguntas que têm a palavra no título;
- *tags*, uma *GHashTable**, a implementação do glib de *hash table*, que guarda pares tag-id;

3 Estruturas de Dados

As estruturas de dados está implementada no módulo *database*.

3.1 TTable

Esta estrutura armazena de forma organizada as *TRow's* e é organizada do seguinte modo:

- *columns*, um *array* de *TColumn*;
- *numCol*, um *int* que guarda o número de colunas que existe na tabela;

O *array columns* contem todas as *TRow's* inseridas na tabela. Cada posição do *array columns* tem uma *TColumn* que tem as *TRow*'s ordenadas pela respetiva coluna.

A *TTable* tem funções associados tais como:

- *tNew*, cria uma nova tabela;
- *tDelete*, apaga uma tabela já criada;
- *tSort*, ordena todas colunas;

3.2 TColumn

Esta estrutura armazena o tipo dos dados, ordena as *TRow's* e é organizada do seguinte modo:

- *índices*, um *GSequence** de *TRow's*;
- *type*, um *Type* que guarda o tipo dos dados dessa coluna;

O *GSequence* índices* é uma árvore AVL de *TRow's* com apontador para o pai. A árvore é ordenada pelos valores dessa coluna na *TRow*.

A *TRow* tem função associada:

- *tColumnSort*, ordena uma coluna;

3.3 TRow

Esta estrutura armazena o tipo os dados e é organizada do seguinte modo:

- *iters*, um *array* de *GSequenceIter**;
- *data*, um *array* de *void** que guarda apontadores para a informação a ser guardada;

O *array GSequenceIter* índices* permite encontrar a *TRow* anterior ou a próxima de uma determinada coluna. O *array data* tem em cada elemeto um apontador para a informação a ser guardada. O índice do *array* é o id da coluna.

A *TColumn* tem as seguinte funções associadas:

- *tGetRow*, retorna uma *TRow* que cumpre todas as condições;
- *tGetRowEquals*, retorna uma row que valor igual ao parametro passado;
- *tGetRows*, retorna as *TRow's* que cumprem todas as condições;
- *tCounTRows*, conta o número de *TRow* que cumprem todas as condições;
- *tApplyRows*, aplica uma função a todas as *TRow* que cumprem todas as condições;
- *tFirstRow*, retorna a *TRow* com o menor valor;
- *tLastRow*, retorna a *TRow* com o maior valor;
- *tNextRow*, retorna a *TRow* imediatamente maior;
- *tPrevRow*, retorna a *TRow* imediatamente menor;
- *tGetData*, retorna o apontador da coluna da *TRow*;
- *tGetDataEquals*, retorna o apontador da coluna da *TRow*;

3.4 TQuery

Esta estrutura armazena uma condição e é organizada do seguinte modo:

- *colId*, um *unsigned int* que indica a coluna a que se aplica a condição;
- *opr*, um *Operator* que indica a operação da condição;
- *value*, um *void** que aponta para o valor a ser comparado;

A *TQuery* tem funções associados tais como:

- *tNewQuery*, cria uma nova *TQuery*;
- *tDeleteQuery*, apaga uma *TQuery*;

3.5 TRowList

Esta estrutura armazena uma lista de *TRow* e é organizada do seguinte modo:

- *array*, um *GPtrArray**, a implementação de um *array* dinámico do glib, que guarda uma lista de *TRow's*;

A emphTRowList tem funções associados tais como:

- *tListNew*, cria uma nova *TRowList*;
- *tListPosition*, retorna a *TRow* numa determinada posição;
- *tListAdd*, adiciona uma *TRow* ao final da lista;
- *tListLength*, retorna o tamanho da lista;
- *tListSort*, ordena a lista;
- *tListSortWithData*, ordena a lista;
- *tListForEach*, aplica uma função a cada elemento da lista;
- *tListDelete*, apaga uma *TRowList*;

3.6 TRowMap

Esta estrutura armazena um par chave-*TRow* e é organizada do seguinte modo:

- *map*, uma *GHashTable**, a implementação de uma *hash table* do glib, que guarda uma *Row*;
- *type*, tipo das chaves;

A *TRowList* tem funções associados tais como:

- *tNewMap*, cria uma nova *TRowMap*;
- *tMapSetRowByKey*, insere uma *TRow*;
- *tMapSetRowByCol*, insere uma *TRow*;
- *tMapGetRow*, retorna uma *TRow* dado a chave;
- *tMapDelete*, apaga uma *TRowMap*;

3.7 TRowMapList

Esta estrutura armazena um par chave-*TListRow* e é organizada do seguinte modo:

- *map*, uma *GHashTable**, a implementação de uma *hash table* do glib, que guarda uma *TRowList*;
- *type*, tipo das chaves;

A *TRowList* tem funções associados tais como:

- *tNewMapList*, cria uma nova *TRowMapList*;
- *tMapListSetRowByKey*, insere uma *TRow*;
- *tMapListSetRowByCol*, insere uma *TRow*;
- *tMapListGetRows*, retorna uma *TRowList* dado a chave;
- *tMapListDelete*, apaga uma *TRowMapList*;

4 Melhoramento de Performance

De forma a melhorar o desempenho do programa, foi preciso estruturar certas estrategias para que ao fazer parse dos ficheiro xml, o tempo que este demora seja o mais pequeno possivel. Para tal, optou-se usar uma AVL, com apontador para o pai, de modo a ser possivel inserir, pesquisar e iterar rapidamente. Esta AVL tem de ser balanceada, pois melhora o pior caso em relação às árvores binárias de procura e tem uma complexidade de O(log n), em vez de O(n).

Para além das AVL, também usamos Hash Tables. Estas são eficientes para inserção, retirada e busca, com custo de pesquisa de O(1) para o caso médio. Estas são especialmente usadas para as querys que não necessitam de iterar.

5 Estratégias das Interrogações

1. Procura a *Row* do post pelo id no *TRowMap* de posts. Vai à *Row* buscar os dados pedidos e guarda num *STR_pair*.
2. Percorre a coluna ordenada por reputação na *TTable* de users e guarda o id dos top N.
3. Percorre as colunas ordenadas por tempo nas *TTable* de perguntas e respostas, no intervalo dado, e conta o número de *Rows*.
4. Percorre a coluna ordenada por tempo na *TTable* de perguntas, no intervalo dado, e verifica em cada *Row* se a hash table de tags contém a tag indicada. Se a tag corresponder à *Row* é guardada num array dinâmico que depois é convertido para *LONG_list*.
5. Procura a *Row* do user pelo id no *TRowMap* de Users, onde vai buscar os dados pedidos. O array de posts é ordenado por data e depois convertido para *LONG_list*.
6. Percorre a coluna ordenada por tempo na *TTable* de respostas, no intervalo dado, e guarda as *Rows* num array que é ordenado pelo número de votos. Este array é depois copiado para uma *LONG_list* de tamanho máximo N.
7. Percorre a coluna, ordenada por tempo, na *TTable* de perguntas, no intervalo dado, e guarda as *Rows* num array que é ordenado pelo número de respostas dentro do intervalo. Este array é depois copiado para uma *LONG_list* de tamanho máximo N.

8. Procura a *TRowList* de perguntas, pela palavra dada, num *TRowMap*. Ordena a lista por data e copia-a para uma *LONG_list* de tamanho máximo N.

9. Procura as *Rows* dos users pelo id no *TRowMap* de users e vai buscar as suas listas de posts. Percorre as listas convertendo as respostas nas suas perguntas correspondentes e depois ordena-as por data. Percorre as listas ordenadamente avançando a *Row* da lista com data maior até encontrar uma data em comum nas duas. Se encontrada uma data em comum faz uma busca linear de ids iguais em todas as *Rows* com essa data. Se encontrar, adiciona o id a um array dinâmico. Este array é depois conpiado para uma *LONG_list* de tamanho máximo N, filtrando elementos repetidos no processo.

10. Procura a *Row* de um post (que neste caso é uma pergunta) pelo id no *TRowMap* de posts onde vai buscar a lista de respostas. Percorre a lista de respostas e guarda a que tiver melhor pontuação com os critérios pedidos.

11. Procura as *Rows* das perguntas dentro de um intervalo de tempo. Percorre a coluna ordenada de reputação pelos N utilizadores com mais reputação e adiciona o seu Id a uma tabela de hash. Depois, percorre as perguntas encontradas e verifica se o seu *ownerId* está na tabela de hash com os top N utilizadores, se assim for, procura a tag numa tabela de hash e incrementa o seu valor. Posteriormente, a hash table com o número de utilização da tag é convertido para um array dinâmico e ordenado pelo número de utilização. Por fim, procura-se o id de cada uma das tags na tabela de hash *tags* e adiciona-se numa *LONG_list*.

6 Notas

Devido a um lapso do membro Daniel Costa, o qual enganou-se ao configurar o email no git do terminal, exitem varios commits no repositorio de um user diferente, portanto vimos informar que tais commits pertençem ao Daniel.