Projeto de LI3 - Wikipedia
Grupo 69

Sérgio Jorge (A77730) Vítor Castro (A77870) Marcos Pereira (A79116)

Resumo

Neste relatório faremos uma análise do primeiro projeto de Laboratórios de Informática III, cujo objetivo era desenvolver um programa, na linguagem de programação C, que fizesse a leitura de *backups* em XML da *Wikipedia*, respondendo a diversas interrogações propostas. Assim, este documento apresenta detalhadamente a abordagem tomada, quer na sua generalidade, quer ao nível das interrogações individuais.

Conteúdo

1 Introdução

Este projeto foi realizado com o objetivo de construir um programa capaz de fazer uma análise dos artigos que estão nos *backups* da *Wikipedia* fornecidos pela equipa docente, para que seja possível determinar informações úteis acerca desses mesmos *backups*. Assim, foram propostas pelos professores 10 tarefas computacionais, às quais o nosso programa, construído em C, deverá responder com sucesso. A realização das 10 tarefas permitiram, por um lado, melhorar e consolidar os conhecimentos adquiridos nas UCs de Programação Imperativa, Algoritmos e Complexidade e Arquitetura de Computadores. Por outro, incentivaram à exploração de bibliotecas como a *libxml2* e a *GLib*, enquanto desafiaram à modularidade estritamente necessária para a participação capaz no desenvolvimento em grupo de um projeto como este. Dado o tamanho dos *backups* e da

necessidade de fazer procuras rápidas, só *hashtables* fizeram sentido, pelo que é com elas que vamos fazer todo o armazenamento de dados. Assim, de modo a facilitar a compreensão do projeto, o relatório está dividido da seguinte forma:

Secção 2 : Problema;

Secção 3 : Solução;

Secção 4 : Conclusão.

2 Problema

Neste projeto de LI3, é-nos pedido para a partir de *backups* da *Wikipedia*, fornecidos pelos professores, fazermos a leitura dos dados e a extração de informação que a equipa docente considera relevante. Assim, a informação que devemos gerar é:

1 - All articles
devolver o número de artigos analisados nos *backups*.

2 - Unique articles
devolver o número de artigos únicos (com ID único) encontrados nos vários *backups* analisados.

3 - All revisions
devolver quantas revisões foram efetuadas nos *backups*.

4 - Top 10 contributors
devolver um *array* com os identificadores dos 10 autores que mais contribuíram, quer em artigos únicos, quer em revisões de artigos já existentes. O resultado deve ser ordenado pelos autores com mais contribuições, sendo que se existirem autores com o mesmo número de contribuições, o resultado deve apresentar primeiro os autores com um identificador menor.

5 - Contributor name
devolver o nome do autor com um determinado identificador, ou **NULL** caso não exista.

6 - Top 20 largest articles
devolver um *array* com os identificadores dos 20 artigos que possuem textos com um maior tamanho em bytes. Para cada artigo deve ser contabilizado o maior texto encontrado nas diversas versões (revisões) do mesmo. O resultado deve ser ordenado pelos artigos com maior tamanho. Se existirem artigos com o mesmo tamanho, o resultado deve apresentar primeiro os artigos com um identificador menor.

7 - Article title
devolver o título do artigo com um determinado identificador, ou **NULL** caso não exista.

8 - Top N articles with more words
devolver um *array* com os identificadores dos N (passado como argumento) artigos que possuem textos com o maior número de palavras e o resultado deve ser ordenado pelos artigos com maior número de palavras.

9 - Titles with prefix
devolver um *array* de títulos de artigos que começam com um prefixo passado como argumento e o resultado deve ser ordenado por ordem alfabética, ou **NULL** caso não existam artigos com esse prefixo.

10 - Article timestamp
devolver o *timestamp* para uma certa revisão de um artigo, ou **NULL** caso não haja essa revisão.

3 Solução

A nossa solução foi implementada com base em diferentes módulos dos quais destacamos os quatro principais:

- Parser
- Hashtable
- Users
- Articles

3.1 Parser

Módulo que funciona com base na biblioteca *libxml2*. A *libxml2* é uma biblioteca para a linguagem C que nos permite trabalhar com ficheiros .xml e, em específico para o trabalho, percorrer o ficheiro de nodo em nodo, retirando o seus conteúdos ou tratando-os de forma útil. Assim, é-nos então possível percorrer o ficheiro, iterar pelas páginas, e fazer o parse extraindo elementos como: *Title*, *ID* e *Revision* (a cada *Revision* está associado: *ID*, *ParentID*, *Timestamp*, *ContributorID*, *ContributorUsername*, *Text*). A informação vai sendo transferida para a memória, que se concretiza na forma de *hashtables* criadas com a *GLib*, estrutura que a seguir invocamos.

3.2 Hashtables

Módulo que atráves do uso da biblioteca *GLib*, armazena os dados transmitidos pelo *parser*. Optamos pela uso de 2 hashtables principais (*articles* e *users*), sendo que a primeira tem uma *hashtables revisions* para cada artigo.

Hashtable *articles* Esta tabela armazena a ID de cada artigo, o tamanho do texto em *bytes*, o número de palavras do texto, o título, e além disso cria, caso o artigo não exista ainda, uma *hashtables revisions* onde é inserida informação relativamente à revisão que está a ser processada.

Hashtable *revisions* Cada artigo tem a ele associado uma estrutura de dados deste tipo com as revisões.

Hashtable *users* Esta estrutura de dados armazena o ID de cada contribuidor, o *username* do contribuidor e o seu número de contribuições.

3.3 Users

Este módulo trata de calcular as respostas a todas as *queries* relativas aos contribuidores, fazendo uso das *hashtables* já preenchidas. Além disso, faz a ligação entre o módulo ***Parser*** e o módulo ***Hashtable***, processando toda a informação obtida no *parsing* e enviando para a *hashtable users* apenas informação relevante.

3.4 Articles

Muito semelhante ao ***Users***, este módulo trata de calcular as respostas a todas as queries relativas aos artigos, fazendo uso das *hashtable* de artigos e da *hashtable* de revisões presente em cada entrada da anterior. Faz também a ligação entre o módulo ***Parser*** e o módulo ***Hashtable*** aquando do parse inicial dos dados, passando também apenas a informação relevante para as *hashtables articles* e *revisions*.

3.5 Implementação

3.5.1 All articles, Unique articles e All revisions

Para resolvermos estas interrogações, optamos por utilizar dois pointers (*articleWasFound* e *articleWasUpdated*), que são alterados em runtime quando se decide que campos das hashtable são criados ou alterados. Assim, sempre que se recebe um artigo, verifica-se se este já existe na estrutura de dados: Se não existe, $articleWasFound = 0$ e $articleWasUpdated = 1$ e, cria o novo artigo na hashtable de artigos, cria a hashtable de revisões e adiciona a revisão à hashtable de revisões. Apesar de não ser encontrado o artigo, é feito o update do mesmo, o que nos permite aumentar o contador allRevisions, presente na *hashtables articles*. Se o artigo já existe na estrutura de dados, $articleWasFound = 1$ e verifica-se o que surgiu de diferente no artigo. Então, se de facto surgiram alterações ao artigo, $articleWasUpdated = 1$. Senão, $articleWasUpdated = 0$. Além disso, a revisão é introduzida na hashtable de revisões do artigo em questão, caso o artigo tenha algo diferente. Neste caso, em que adicionamos um novo artigo à tabela, incrementamos o contador uniqueArticles, presente na *hashtables articles*. Portanto, ***All articles*** é alcançado a partir de um contador que é incrementado sempre que é encontrado um artigo, independentemente dos apontadores *articleWasFound* e *articleWasUpdated*. ***Unique articles*** é devolvido a partir de um contador que é incrementado se e só se o *articleWasFound* está com valor 0. ***All revisions*** é calculado também a partir de um contador que é incrementado quando o $articleWasUpdated = 1$. Termos estes contadores na *hashtable* foi a maneira que nos pareceu mais eficiente de colher tais informações, uma vez que o custo seria apenas de mudar o valor de um apontador, contrariamente ao que seria o custo de, por exemplo, percorrer toda a *hashtable* no fim e incrementar os mesmos contadores.

3.5.2 Top 10 contributors

Definimos um *array* com 10 posições para colocar o top. Percorremos a *hashtable* de utilizadores e, para cada user, vamos buscar o seu número de contribuições e comparamos se tem mais que o utilizador que está na última posição do top (índice 9). Em caso afirmativo, comparamos com as sucessivas posições, encontramos a posição e deslizamos o resto dos utilizadores para haver lugar para o que está a ser tratado. Para os casos em que temos igual número de contribuições na comparação, faz-se a comparação dos ID dos users para sabermos qual o com menor ID. Escolhemos fazer a comparação sempre pelo último lugar do top pois evitamos que, na maioria das vezes, se percorra o top sem necessidade.

3.5.3 Contributor name

É feita a procura passando o ID a uma função getUser do módulo hashtable que, caso exista tal ID na *hashtable users*, retorna uma estrutura de utilizador com todos os dados correspondentes àquele utilizador em específico. Caso não exista, retorna **NULL**. Depois, é feito o acesso ao campo username daquele user, na função **getContributorName** do módulo ***Users***.

3.5.4 Top 20 largest articles

Definimos um *array* com 20 posições para colocar o top. Percorremos a *hashtable* de artigos e, para cada artigo, vamos buscar o número do texto em bytes e comparamos se tem mais que o artigo que está na última posição do top (índice 19). Em caso afirmativo, comparamos com as sucessivas posições, encontramos a posição e deslizamos o resto dos artigos para haver lugar para o autal. Para os casos em que temos igual número de contribuições na comparação, faz-se uma compaação dos IDs dos articles para sabermos qual o menor ID. Tal como na solução **Top 10 contributors**, fizemos a comparação a partir do último elemento do top. O tamanho em bytes é obtido através da função **wordCounter**, presente em articles, que atualiza um pointer com o valor do tamanho da *string* analisada. Em **Top N articles with more words** explica-se um pouco melhor esta função.

3.5.5 Article title

É feita a procura passando o ID a uma função **getArticle** do módulo *hashtable* que, caso exista tal ID na *hashtable articles*, retorna uma estrutura de artigo com todos os dados correspondentes àquele artigo em específico. Caso não exista, retorna **NULL**. Depois, é feito o acesso ao campo *title* daquele *article*, na função **get_article_title** do módulo articles.

3.5.6 Top N articles with more words

Definimos um *array* com n posições para colocar o top. Percorremos a *hashtable* de artigos e, para cada artigo, vamos buscar o número de palavras e comparamos se tem mais que o artigo que está na última posição do top (índice n-1). Em caso afirmativo, comparamos com as sucessivas posições, encontramos a posição e deslizamos o resto dos artigos para haver lugar para o atual. Para os casos em que temos igual número de contribuições na comparação, faz-se um string compare dos dois ID dos articles para sabermos qual o menor ID. Faz-se a comparação a partir do último lugar do top pelos mesmos motivos que em **Top 10 contributors** e em **Top 20 largest articles**. Para ver o número de words usou-se a função **wordCounter**, do módulo *articles*, que tinha *flags* para que fosse possível controlar os espaços entre palavras e carateres especiais, de modo a que se obtivesse números fiáveis. O contador usado para avançar de char em char é também usado para fornecer, no final, o número total de *bytes* do artigo, o que resulta num poupar de trabalho significativo, sendo esta função uma 2 em 1 que também é importante no **Top 20 largest articles**.

3.5.7 Titles with prefix

A forma que encontramos para resolver este problema foi implementar a função **isTitlePrefix** que verifica se determinada *string* é prefixo de outra *string*. Assim, percorremos a *hashtable* de artigos e chamamos a função implementada e verificamos se os títulos são prefixo. Em caso afirmativo, inserimos o título num *array* definido para o efeito de armazenar todos os títulos com o prefixo dado como argumento. Posteriormente, o *array* de títulos é ordenado por ordem alfabética.

3.5.8 Article timestamp

É feita a procura passando o ID a uma função getArticle do módulo *hashtable* que, caso exista tal ID na *hashtable articles*, retorna uma estrutura de artigo com todos os dados correspondentes àquele artigo em específico. Caso não exista, retorna **NULL**. Depois, é feito o acesso ao campo *timestamp* daquele *article*, na função **get_article_timestamp** do módulo ***Articles***.

3.6 Resultado Final

```
all_articles() -> 59593
unique_articles() -> 19867
all_revisions() -> 40131
top_10_contributors() -> 28903366 13286072 27823944 27015025 194203 212624 7852030 7328338 7611264 14508071
contributor_name(28903366) -> Bender the Bot
contributor_name(194203) -> Graham87
contributor_name(1000) -> (null)
top_20_largest_articles() -> 15910, 23235, 11812, 28678, 14604, 23440, 26847, 25507, 26909, 18166, 4402, 14889, 23805, 25391, 7023, 13224,
12108, 13913, 23041, 18048,
article_title(15910) -> List of compositions by Johann Sebastian Bach
top_N_articles_with_more_words(30) -> 15910, 25507, 23235, 11812, 13224, 26847, 14889, 7023, 14604, 13289, 18166, 4402, 12157, 13854,
23805, 25401, 10186, 23041, 18048, 16772, 22936, 28678, 27069, 9516, 12108, 13913, 13890, 21217, 23440, 25391,
article_title(25507) -> Roman Empire
article_title(1111) -> Politics of American Samoa
titles_with_prefix(Anax) -> Anaxagoras, Anaxarchus, Anaximander, Anaximenes of Lampsacus, Anaximenes of Miletus,
article_timestamp(12,763082287) -> 2017-02-01T06:11:56Z
article_timestamp(12,755779730) -> 2016-12-20T04:02:33Z
article_timestamp(12,4479730) -> (null)

real    0m9.448s
user    0m8.800s
sys     0m0.632s
```

4 Conclusões

Este projeto serviu para aprofundarmos o conhecimento da linguagem C, assim como as bibliotecas que lhe estão associadas. Achámos que, com a realização de um trabalho deste tipo permite uma consolidação proveitosa da linguagem, não só em termos teóricos como também em termos práticos e acaba por ser uma forma diferente de cimentar os conteúdos de algumas das UCs que nos surgiram neste curso. Permite também melhorar as habilidades na resolução de problemas. Concluímos também que:

- O uso da estrutura de dados adequada (*hashtables*) foi essencial de forma a obter um programa eficiente, que faça a manipulação necessária dos dados em tempo útil;
- A separação do trabalho em módulos permite uma fácil divisão do trabalho pela equipa, aliando-se ao uso do ***Git*** numa organização eficiente do trabalho do grupo.