Trabalho Prático LI3

(Versão 1)

A Wikipedia[1] é uma enciclopédia web global de livre acesso. Um dos aspetos mais interessantes desta plataforma é que os artigos são escritos pelas próprias pessoas que utilizam o serviço para procurar informação. Este serviço é hoje em dia usado massivamente por milhões de utilizadores todos os dias. De facto, olhando apenas para a Wikipedia Inglesa, existem mais de 5 milhões de artigos, 880 milhões de edições em artigos, e 30 milhões de colaboradores. Ainda, em Junho de 2015, o tamanho de um backup da Wikipedia Inglesa que continha todos os artigos e edições feitas aos mesmos tinha mais de 10 terabytes.

Esta quantidade enorme de dados contém informação muito útil não só no texto dos artigos mas nos próprios metadados dos mesmos. Perceber características da Wikipedia como quais os colaboradores que são mais ativos ou quais os artigos que mais são revistos é fundamental para melhor compreender o funcionamento desta plataforma. No entanto, analisar 10 terrabytes de dados e metadados e conseguir cruzar os mesmos para efetuar interrogações interessantes não é uma tarefa simples. Tal requer um sistema desenhado cuidadosamente de forma a não só a conseguir carregar esta quantidade enorme de dados mas também a utilizar as melhores estruturas de dados e algoritmos para depois conseguir responder às interrogações necessárias em tempo útil.

Objetivos

Este trabalho prático tem como objetivo construir um sistema que permita analisar os artigos presentes em backups da Wikipedia, realizados em diferentes meses, e extrair informação útil para esse período de tempo como, por exemplo, o número de revisões, o número de novos artigos, etc.

Requisitos:

O sistema a desenvolver deve ser implementado em C e terá de suportar as seguintes funcionalidades:

[1] https://www.wikipedia.org

Interface

Todos os projetos deverão definir um ficheiro “interface.h” com o seguinte tipo abstrato de dados e funções:

```
/* interface.h */
typedef struct TCD_istruct * TAD_istruct;

TAD_istruct init();
TAD_istruct load(TAD_istruct qs, int nsnaps, char* snaps_paths[]);
long all_articles(TAD_istruct qs);
long unique_articles(TAD_istruct qs);
long all_revisions(TAD_istruct qs);
TAD_istruct clean(TAD_istruct qs);
```

Esta interface será importada por um ficheiro program.c que deverá poder chamar qualquer uma das funções referidas em cima. È esperado que a primeira função a ser chamada seja a função *init()*, que irá inicializar a estrutura *TAD_istruct.*

Carregamento dos dados

O passo seguinte do programa será chamar a função *load(TAD_istruct qs, int nsnaps, char* snaps_paths[]).* Que recebe como argumentos extra o número de backups e o caminho onde estes estão armazenados.

Serão disponibilizados inicialmente 3 backups parciais da Wikipedia Inglesa que contêm a versão mais atual de um conjunto de artigos correspondente aos meses de Dezembro de 2016, Janeiro e Fevereiro de 2017.

Cada backup corresponde a um ficheiro distinto no formato XML. Um exemplo muito simples destes ficheiros é o seguinte:

```
<mediawiki (...)>
  <siteinfo>
      (...)
```

```
</siteinfo>
<page>
  <title>AmoeboidTaxa</title>
  <ns>0</ns>
  <id>24</id>
  <redirect title="Amoeba" />
  <revision>
    <id>627604809</id>
    <parentid>625443465</parentid>
    <timestamp>2014-09-29T22:26:03Z</timestamp>
    <contributor>
      <username>Invadibot</username>
      <id>15934865</id>
    </contributor>
    <minor />
    <comment>Bot: Fixing double redirect to [[Amoeba]]</comment>
    <model>wikitext</model>
    <format>text/x-wiki</format>
    <text xml:space="preserve">Text example for this article….</text>
    <sha1>afkde9noo6ive9c3gr5pq9sqlqf6w64</sha1>
  </revision>
</page>
<page>
  (...)
</page>
  (...)
</mediawiki>
```

Cada ficheiro começa com a tag <mediawiki> e <siteinfo>, no entanto, para este trabalho apenas nos interessa analisar o conteúdo dentro da tags <page> e </page>. A informação contida entre estas tags corresponde a um artigo da Wikipedia distinto que tem um título (<title>) e identificador único (<id>). Cada <page> apenas apresenta detalhes da última revisão daquele artigo antes de o backup ter sido efetuado (<revision>). A revisão contém um identificador único (<id>), a data em que foi efetuada (<timestamp>), a identificação e nome do colaborador (<contributor> <id> <username>) e, finalmente, o texto do artigo (<text xml:space="preserve">)

A leitura dos dados e extração de informação relevante deve ser feita para cada um dos backups, seguindo a ordem temporal em que estes foram efetuados. Convém notar que entre backups sucessivos, é possível que novos artigos tenham sido adicionados, que artigos existentes tenham ou não sido revistos, e que artigos tenham sido apagados.

Para processar cada um dos ficheiros deve ser utilizada a biblioteca libxml2[2], a qual possui já as ferramentas necessárias para ler ficheiros no formato XML de forma eficiente.

Interrogações

De momento o ficheiro “interface.h” define 3 interrogações iniciais:

long all_articles(TAD_istruct qs);
long unique_articles(TAD_istruct qs);
long all_revisions(TAD_istruct qs);

A primeira interrogação retorna todos os artigos encontrados nos backups analisados. Para esta interrogação, artigos duplicados em backups sucessivos bem como novas revisões de artigos devem ser contados como um novo artigo.

A segunda interrogação apenas pretende saber quais os artigos únicos encontrados nos vários backups analisados. Ou seja, artigos duplicados ou revisões dos mesmos que estejam presentes em backups distintos não devem ser contabilizados.

A última interrogação pretende saber quantas revisões fora efetuadas naqueles backups. O valor retornado deve incluir quer a versão base do artigo bem como as revisões feitas ao mesmo.

Exemplo de resultados para a query:

	Backup 1	Backup 2	Backup 3
Artigo 1	ID revisão - 1	ID revisão - 1	ID revisão - 1
Artigo 2	Não existe	ID revisão - 1	ID revisão - 2
Artigo 3	ID revisão - 5	ID revisão - 6	ID revisão - 6

A primeira interrogação retornaria o valor 8, a segunda o valor 3, a terceira o valor 5.

[2] http://xmlsoft.org

Falta por fim falar na função *clean()* que deverá libertar todos os recursos associados à estrutura *TAD_istruct*. Depois de um *clean()*, caso se pretendam fazer novas interrogações, terão de ser chamados de novo os métodos *init() e load().*

Modularidade

O código produzido deverá ser modular visto que o número de interrogações e até os próprios dados extraídos no carregamento de dados irão evoluir ao longo do projeto. Ou seja, no futuro, o projeto terá de suportar outros tipos de interrogações que precisarão de estruturas auxiliares de dados diferentes que podem até ter um impacto na abordagem utilizada ao carregar os dados.

Desta forma, é muito importante que o projeto esteja organizado de forma modular e permita que novas estruturas de dados e novas funcionalidades sejam adicionadas ao código sem obrigar a uma re-implementação de grande parte do mesmo.

Sempre que o conjunto de interrogações evoluir, será atualizado o ficheiro “interface.h” com a definição das funções respetivas.

Avaliação

O projeto será avaliado através das seguintes vertentes:

Desenho da solução

A modularidade da solução, bem como as abordagens utilizadas para responder de forma eficiente ao carregamento de dados e às diferentes interrogações propostas serão fatores importantes na avaliação do trabalho.

Os alunos deverão ser capazes de argumentar a razão pela qual escolheram uma certa abordagem (estrutura de dados, travessia, etc) e serem críticos em relação à mesma. A escolha das estruturas de dados a utilizar deve focar-se não só na rapidez de resposta às interrogações mas também nos recursos de CPU e memória que estas estruturas requerem para suportar as interrogações.

Como explicado anteriormente, a solução desenvolvida terá de ser modular visto que quer o carregamento de dados quer as interrogações a efetuar podem mudar ao longo do tempo. Os

alunos deverão ser capazes de justificar de que forma a sua implementação possibilita estas mudanças.

Desempenho da solução

Cada grupo irá ter acesso a um repositório GIT onde o seu projeto será armazenado. Estes projetos serão avaliados experimentalmente por uma ferramenta que irá testar qual a latência das operações *init(), load() e clean()* e de cada interrogação. Os resultados serão publicados no repositório de cada grupo num ficheiro chamado “resultados”.

Para ser possível realizar estes passos, em cada repositório deverá existir uma Makefile que gere uma aplicação “program” através de um ficheiro “program.c”. Estes três ficheiros têm de estar incluídos na raiz do repositório. Os testes serão sempre corridos no branch “master” do repositório.

O ficheiro “program.c” será substituído por uma versão da ferramenta e o único requisito é que este ficheiro possa importar o ficheiro “interface.h” e possa chamar as funções definidas neste.

Cada grupo deverá escrever o seu próprio ficheiro “program.c” para testar o seu projeto e ter certeza que cumpre os requisitos necessários.

Contribuição para o projeto

Através dos repositórios GIT é possível identificar as contribuições para o projeto. É obrigatório que cada aluno faça “commits” com o seu utilizador e que as mensagens destes sejam claras. Este ponto não só é importante para avaliar o trabalho mas também para o grupo saber exatamente o que foi alterado em cada “commit” e quem o alterou.