Trabalho Prático LI3

(Versão 4)

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:

```
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);

long* top_10_contributors(TAD_istruct qs);

char* contributor_name(long contributor_id, TAD_istruct qs);

long* top_20_largest_articles(TAD_istruct qs);

char* article_title(long article_id, TAD_istruct qs);

long* top_N_articles_with_more_words(int n, TAD_istruct qs);

char** titles_with_prefix(char* prefix, TAD_istruct qs);

char* article_timestamp(long article_id, long revision_id, 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. É esperado que a primeira função a ser chamada seja a função *init()*, que irá inicializar a estrutura *TAD_istruct.*

Compilação

Deverá existir uma Makefile que permitirá gerar um executável “program” através do comando “make program”. Este comando deverá compilar um ficheiro “program.c” que irá incluir o ficheiro “interface.h”.

Respeitar esta nomenclatura é importante para garantir que o projeto é validado pelos scripts que irão executar nos repositórios GIT. O ficheiro “program.c“ será substituído por uma versão alternativa para correr os testes de validação. Este ficheiro irá executar todas funções definidas no ficheiro “interface.h” pela ordem correta (init, load, interrogações, clean). As interrogações poderão ser feitas mais de uma vez com parâmetros diferentes.

O único requisito é que o ficheiro “program.c” possa incluir o ficheiro “interface.h” e que não necessite de incluir outros ficheiros do projeto ou bibliotecas externas para executar as funções corretamente.

O projeto deverá ser sempre compilado com a flag -Wall e -std=c11. Para utilizar bibliotecas externas (exemplo: libxml, glib, etc). A compilação deve sempre utilizar o comando pkg-config com os argumentos --cflags (para declarar o caminho para os ficheiros .h das bibliotecas) e --libs (para declarar o caminho para as bibliotecas).

Exemplo (muito simples) de uma Makefile para o projeto:

```
CC = gcc
CFLAGS = -Wall -std=c11 -g `pkg-config --cflags libxml-2.0` `pkg-config --cflags glib-2.0`
LIBS = `pkg-config --libs libxml-2.0` `pkg-config --libs glib-2.0`

$(CC) $(CFLAGS) program.c -o program $(LIBS)
```

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 um 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

O ficheiro “interface.h” define as seguintes interrogações:

1 - long all_articles(TAD_istruct qs);
2 - long unique_articles(TAD_istruct qs);
3 - long all_revisions(TAD_istruct qs);
4 - long top_10_contributors(TAD_istruct qs);*
5 - char contributor_name(long contributor_id, TAD_istruct qs);*
6 - long top_20_largest_articles(TAD_istruct qs);*
7 - char article_title(long article_id, TAD_istruct qs);*
8 - long top_N_articles_with_more_words(int n, TAD_istruct qs);*
*9 - char** titles_with_prefix(char* prefix, TAD_istruct qs);*
10 - char article_timestamp(long article_id, long revision_id, TAD_istruct qs);*

A interrogação 1 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 para o resultado da interrogação.

A interrogação 2 apenas pretende saber quais os artigos únicos (*i.e.*, com um identificador único) encontrados nos vários backups analisados. Ou seja, artigos duplicados ou revisões dos mesmos que estejam presentes em backups distintos não devem ser contabilizados.

A interrogação 3 pretende saber quantas revisões fora efetuadas naqueles backups (*i.e.*, o número total de versões diferentes encontradas nos backups). O valor retornado deve incluir quer a versão base do artigo bem como as revisões feitas ao mesmo.

Exemplo de resultados para a três primeiras interrogações:

	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

A interrogação 4 devolve um array com os identificadores dos 10 autores que contribuíram para um maior número de versões de artigos (*i.e.*, contar contribuições para artigos e respetivas revisões). O resultado deve ser ordenado pelos autores com mais contribuições. Se existirem autores com o mesmo número de contribuições, o resultado deve apresentar primeiro os autores com um identificador menor.

A interrogação 5 devolve o nome do autor com um determinado identificador. Caso não exista o autor a interrogação retorna o valor NULL.

A interrogação 6 devolve um array com os identificadores dos 20 artigos que possuem textos com um maior tamanho. 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.

A interrogação 7 devolve o título do artigo com um determinado identificador. Caso não exista o artigo a interrogação retorna o valor NULL. No caso de um determinado artigo ter várias versões (revisões) deve ser considerado o título da revisão mais recente.

A interrogação 8 devolve um array com os identificadores dos N (parâmetro da interrogação) artigos que possuem textos com um maior número de palavras. Para cada artigo deve ser contabilizado o texto com o maior número de palavras encontrado nas diversas versões(revisões) do mesmo. O resultado deve ser ordenado pelos artigos com maior número de palavras. Se existirem artigos com o mesmo número de palavras, o resultado deve apresentar primeiro os artigos com um identificador menor.

A interrogação 9 devolve uma lista de títulos de artigos que começam com um prefixo passado como argumento da interrogação. O resultado deve ser ordenado por ordem alfabética. Para artigos cujo título tenha evoluído com as revisões, só deve ser considerado o título da última revisão de cada artigo para esta interrogação.

A interrogação 10 devolve o timestamp para uma certa revisão de um artigo. Caso não exista a revisão daquele artigo a interrogação retorna o valor NULL.

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

Uma componente importante da avaliação é a estruturação do código da solução. Tal como tem vindo a ser falado nas aulas, um código bem estruturado permite a sua fácil evolução, manutenção e boa legibilidade. Assim, é importante que a solução apresentada esteja organizada de forma modular e que os seus autores sejam capazes de justificar o porquê dessa mesma organização. Mais ainda, deverá ser possível demonstrar que a solução permite evoluir (alterar) alguns dos seus componentes sem que seja necessária refazer uma percentagem significativa do código.

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.

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 em http://li3.lsd.di.uminho.pt/results/ numa pasta independente para cada grupo. O ficheiro “output.txt” contém informação sobre a compilação e execução do projeto. Erros de compilação, execução e nos resultados esperados serão reportados neste ficheiro. O ficheiro “times.txt” contêm informação sobre os tempos de execução das diferentes operações do projeto.

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.

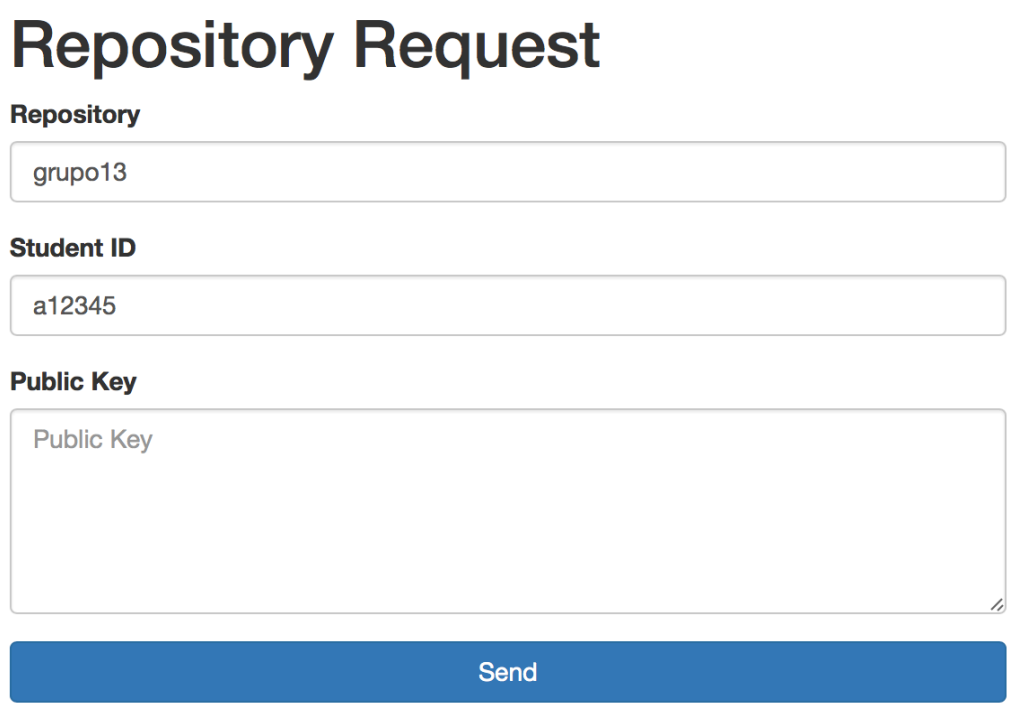
O ficheiro “program.c” será substituído por uma versão da ferramenta e o único requisito é que este ficheiro possa importar o ficheiro “interface.h” e possa chamar as funções definidas neste.

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.

Registo no repositório: http://li3.lsd.di.uminho.pt
Os alunos do mesmo grupo devem introduzir no campo “repository” a palavra “grupo” seguida do número. Por exemplo o grupo 13 deverá introduzir a seguinte informação:



No campo **Student ID** devem colocar o vosso número de aluno (exemplo: a12345). No campo **Public Key** deverão colocar a vossa chave pública. Para gerar uma chave, caso não a tenham já, podem utilizar o comando **ssh-keygen**.
Após o registo poderão fazer clone do vosso repositório para uma pasta local com o comando (exemplo para o grupo 13):

git clone git@li3.lsd.di.uminho.pt:grupo13

A partir deste momento o repositório está pronto a ser usado.