Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III - Relatório do Projeto de C

Mestrado Integrado em Engenharia Informática

Ana Paula Carvalho (A61855)
Joana Arantes (A57810)
João Pires Barreira (A73831)
Miguel Cunha (A78478)

Ano letivo 2016/2017
Grupo 52

abril 2017

Índice

1 Introdução

No âmbito da unidade curricular de Laboratórios de Informática III, do 2.º ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho, elaboramos um projeto em C que consiste na construção de um sistema que permita analisar e extrair informação dos artigos presentes em *backups (snapshots)* da Wikipedia, realizados em diferentes meses.

Com esta aplicação, podemos perceber características fundamentais da Wikipedia em cada *snapshot*, para melhor compreender o seu funcionamento.

A fonte de dados consiste, inicialmente, em três *snapshots* parciais da Wikipedia Inglesa, que contêm a versão mais atual de um conjunto de artigos, correspondente aos meses de dezembro de 2016 e de janeiro e fevereiro de 2017. Cada *snapshot* corresponde a um ficheiro distinto no formato XML.

Ao longo deste relatório, explicaremos, com detalhe, a nossa abordagem e solução, bem como a resolução de cada interrogação *(query)* proposta para este projeto.

2 Importância da Modularidade e do Encapsulamento dos Dados na nossa abordagem

Sendo que o objetivo principal desta Unidade Curricular é garantir o encapsulamento dos dados e a modularidade do código, mantivemos estes dois aspetos como fundamentais na elaboração do nosso projeto.

2.1 Modularidade do Código

2.1.1 O que é? Para que serve?

A nossa codificação foi dividida em módulos de acordo com a sua função na globalidade do projeto. Desta forma, é assegurada uma maior legibilidade do código, uma melhor manutenção do projeto e melhor desempenho da nossa solução.

2.1.2 Exemplos de aplicação no nosso projeto

- Módulo do *parsing* separado do módulo da interface, módulo da AVL separado das *queries*, etc.

2.2 Encapsulamento dos Dados

2.2.1 O que é? Para que serve?

A nossa codificação também teve como objetivo principal o encapsulamento dos dados, ou seja, cada módulo possui estruturas e funções privadas que não podem ser acedidas por outros módulos. Para isso, existe uma API visível a esses outros módulos, que serve para que estes consigam utilizar as funcionalidades de outro módulo sem que, para isso, tenham que conhecer as suas informações privadas. Tudo isto, faz com que a nossa solução seja mais robusta e imune a situações inesperadas e irregulares, como o acesso a informação que deveria estar protegida.

2.2.2 Exemplos de aplicação no nosso projeto

- Os *getters* devolvem uma cópia do campo pedido e não um apontador para o mesmo.
- Os *setters* recebem o valor, efetuam uma cópia do mesmo e colocam esse novo valor (a cópia e não o valor recebido como parâmetro) no campo respetivo.
- Dar preferência (ou exclusividade) à passagem de cópias de valores em vez da passagem por referência

3 Estruturação dos módulos

A estruturação do projeto foi pensada de modo a dividir o código em módulos de acordo com a função de cada um destes na globalidade do projeto. No entanto, além deste foco na modularidade do código, foi também sempre tida como prioridade a necessidade do encapsulamento dos dados e da criação de um solução genérica, nomeadamente nas estruturas de dados usadas (módulo da AVL).

3.1 Módulo do programa

Este módulo, chamado *program*, é o módulo principal e responsável pelo chamamento da interface gráfica do menu ou, em alternativa, por correr o programa sequencialmente. Ou seja, é o módulo que chama qualquer uma das 10 *queries* pedidas para esta aplicação e também as funções *init()*, *load()* e *clean()*, que inicializa, carrega e liberta, respetivamente, a estrutura `TAD_istruct`.

3.2 Módulo do menu

O *menu* é o módulo responsável pela interface gráfica do programa, permitindo assim a sua facilidade no teste e execução.

Este módulo está feito e pensado para que, quando se quiser executar uma query, apenas com o número da query, o *menu* chame a query certa, executando-a e mostrando os seus resultados no ecrã, bem como quanto tempo demorou.

3.3 Módulo da interface

O módulo *interface* contém a definição de todas as funções pertencentes à interface do programa: *init()*, *load()*, *clean()* e as respetivas *queries*.

3.3.1 Estruturas de dados

- `TAD_istruct`: Estrutura principal do programa, que contém todos os dados lidos dos *snapshots* disponibilizados.

3.3.2 API

- **init**: Inicializa a principal estrutura de dados do programa. Esta função deve ser a primeira a ser executada, pois só após a chamada desta é que a estrutura de dados estará pronta a ser carregada.
- **load**: Carrega a principal estrutura de dados do programa, sendo que a estrutura de dados deve ser primeiro inicializada. O carregamento é feito a partir dos ficheiros dos *snapshots*, cujo caminho é passado como parâmetro.
- `all_articles`: Corresponde à *querie* 1, que calcula o número total de artigos encontrados, incluindo artigos duplicados e novas revisões de artigos já existentes.

- `unique_articles`: Corresponde à *querie* 2, que calcula o número de artigos com um identificador único encontrados, ou seja, não inclui artigos duplicados nem novas revisões de artigos já existentes.
- `all_revisions`: Corresponde à *querie* 3, que calcula o número total de revisões de artigos encontradas, quer da versão base do artigo, como das restantes revisões feitas ao mesmo, posteriormente.
- `top_10_contributors`: Corresponde à *querie* 4, que devolve um *array* com os IDs dos 10 colaboradores que contribuíram para um maior número de revisões de artigos, ordenado decrescentemente. Caso existam colaboradores com um número igual de contribuições, estes aparecem ordenados pelo seu ID, do menor para o maior.
- `contributor_name`: Corresponde à *querie* 5, que devolve o *username* de um contribuidor cujo ID é passado como parâmetro. Caso não exista nenhum contribuidor com esse ID, é devolvido o valor *NULL*.
- `top_20_largest_articles`: Corresponde à *querie* 6, que devolve um *array* com os IDs dos 20 artigos que possuem textos com um maior tamanho em bytes, ordenado decrescentemente. Para cada artigo, é apenas contabilizado o maior tamanho de todas as revisões do mesmo. Caso existam artigos com um tamanho igual, estes aparecem ordenados pelo seu ID, do menor para o maior.
- `article_title`: Corresponde à *querie* 7, que devolve o título de um artigo cujo ID é passado como parâmetro, sendo apenas considerados os títulos das últimas revisões dos artigos. Caso não exista nenhum artigo com esse ID, é devolvido o valor *NULL*.
- `top_N_articles_with_more_words`: Corresponde à *querie* 8, que devolve um *array* com os IDs dos N artigos que possuem textos com um maior número de palavras, ordenado decrescentemente. Para cada artigo, é apenas contabilizado o texto com o maior número de palavras de todas as revisões do mesmo. Caso existam artigos com um número de palavras igual, estes aparecem ordenados pelo seu ID, do menor para o maior.
- `titles_with_prefix`: Corresponde à *querie* 9, que devolve um *array* com os títulos dos artigos que começam pelo prefixo passado como parâmetro, sendo apenas considerado o título da última revisão de cada artigo.
- `article_timestamp`: Corresponde à *querie* 10, que devolve o *timestamp* de uma dada revisão de um dado artigo, ambos passados como parâmetro. Caso a revisão e/ou o artigo não existam, é devolvido o valor *NULL*.
- **clean**: Liberta a principal estrutura de dados do programa, isto é, liberta o apontador para a estrutura passada como parâmetro bem como todo e qualquer espaço previamente ocupado pela mesma.

3.4 Módulo do parsing XML

Este módulo, chamado *xmlparser*, contém as funções de leitura e *parsing* dos *snapshots* disponibilizados (correspondentes a ficheiros no formato XML).

3.4.1 API

- **parseContributor**: Faz a leitura e o *parsing* de uma *tag "contributor"* de um *snapshot*, ou seja, a função lê as *tags* hierarquicamente inferiores a *"contributor"*, retirando a informação necessária e colocando-a nas estruturas previamente criadas.
- **parseRevision**: Faz a leitura e o *parsing* de uma *tag "revision"* de um *snapshot*, ou seja, a função lê as *tags* hierarquicamente inferiores a *"revision"*, retirando a informação necessária e colocando-a nas estruturas previamente criadas.
- **parsePage**: Faz a leitura e o *parsing* de uma *tag "page"* de um *snapshot*, ou seja, a função lê as *tags* hierarquicamente inferiores a *"page"*, retirando a informação necessária, criando as estruturas necessárias e inserindo-as na estrutura de dados recebida.
- **parseFile**: Faz a leitura e o *parsing* de um ficheiro XML, ou seja, a função lê o *snapshot*, valida-o, faz o seu *parsing* e carrega a estrutura *WikiData* recebida.

3.5 Módulo das queries

O módulo das *queries*, chamado *queries*, contém todo o código fundamental para a execução das 10 *queries* pedidas para esta aplicação.

Este módulo contém também as estruturas que guardam todos os contribuidores (*contributors*) e os artigos (*articles*), para que o acesso ao mesmo seja mais fácil durante a execução destas *queries*.

3.5.1 Estruturas de dados

- **ContributorSet**: Estrutura que guarda todos os contribuidores lidos dos *snapshots*. É constituída por um *array* de 10 AVLs para guardar todos os contribuidores lidos; cada índice do *array* corresponde a uma AVL dos contribuidores cujo ID começa pelo número do índice do *array*.
- **ArticleSet**: Estrutura que guarda todos os artigos lidos dos *snapshots*. É constituída por um *array* de 10 AVLs para guardar todos os artigos lidos; cada índice do *array* corresponde a uma AVL dos artigos cujo ID começa pelo número do índice do *array*.
- **wikidata**: Estrutura que guarda toda a informação (artigos e contribuidores) lida dos *snapshots*. É constituída por um apontador para a estrutura que contém as AVLs dos contribuidores e por um apontador para a estrutura que contém as AVLs dos artigos.

3.5.2 API

- **initArticleSet**: Inicializa o apontador para uma nova estrutura de artigos e para cada uma das 10 posições do *array*.

- **initContributorSet**: Inicializa o apontador para uma nova estrutura de contribuidores e para cada uma das 10 posições do *array*.
- **initWikiData**: Inicializa o apontador para uma nova estrutura wikidata, inicializando também os *arrays* dos contribuidores e dos artigos.
- **freeArticleSet**: Liberta o apontador para o conjunto dos artigos, libertando também cada uma das 10 posições do *array*.
- **freeContributorSet**: Liberta o apontador para o conjunto dos contribuidores, libertando também cada uma das 10 posições do *array*.
- **freeWikiData**: Liberta o apontador para a estrutura WikiData, libertando também os *arrays* dos contribuidores e dos artigos.
- **insertArticle**: Insere um artigo na estrutura WikiData (no seu *array* de artigos).
- **insertContributor**: Insere um contribuidor na estrutura WikiData (no seu *array* de contribuidores).
- **query1**: Calcula o número total de artigos encontrados, incluindo artigos duplicados e novas revisões de artigos já existentes.
- **query2**: Calcula o número de artigos com um ID único encontrados, ou seja, não inclui artigos duplicados nem novas revisões de artigos já existentes.
- **query3**: Calcula o número total de revisões de artigos encontradas. Inclui quer a versão base de um artigo, quer as restantes revisões feitas ao mesmo, posteriormente.
- **query4**: Devolve um *array* com os IDs dos 10 colaboradores que contribuíram para um maior número de revisões de artigos, ordenado decrescentemente. Caso existam colaboradores com um número igual de contribuições, estes aparecem ordenados pelo seu ID, do menor para o maior.
- **query5**: Devolve o *username* de um contribuidor cujo ID é passado como parâmetro. Caso não exista nenhum contribuidor com esse ID, é devolvido o valor *NULL*.
- **query6**: Devolve um *array* com os IDs dos 20 artigos que possuem textos com um maior tamanho em bytes, ordenado decrescentemente. Para cada artigo, é apenas contabilizado o maior tamanho de todas as revisões do mesmo. Caso existam artigos com um tamanho igual, estes aparecem ordenados pelo seu ID, do menor para o maior.
- **query7**: Devolve o título de um artigo cujo ID é passado como parâmetro, sendo apenas considerados os títulos das últimas revisões dos artigos. Caso não exista nenhum artigo com esse ID, é devolvido o valor *NULL*.
- **query8**: Devolve um *array* com os IDs dos N artigos que possuem textos com um maior número de palavras, ordenado decrescentemente. Para cada artigo, é apenas contabilizado o texto com o maior número de palavras de todas as revisões do mesmo. Caso existam artigos com um número de

palavras igual, estes aparecem ordenados pelo seu ID, do menor para o maior.

- **query9**: Devolve um *array* com os títulos dos artigos que começam pelo prefixo passado como parâmetro, sendo apenas considerado o título da última revisão de cada artigo.
- **query10** - Devolve o *timestamp* de uma dada revisão de um dado artigo, ambos passados como parâmetro. Caso a revisão e/ou o artigo não existam, é devolvido o valor *NULL*.

3.6 Módulo da AVL genérica

Este módulo, chamado *avl*, contém a definição da AVL genérica e as respetivas funções.

3.6.1 Estruturas de dados

- **node**
  - **info**: Informação guardada no nodo.
  - **height**: Altura do nodo na árvore.
  - **node (left)**: Apontador para o raíz da sub-árvore esquerda.
  - **node (right)**: Apontador para o raíz da sub-árvore direita.
- **avl**
  - **total**: Número total de nodos na árvore.
  - **root**: Apontador para a raíz (nodo) da árvore.
  - **cmp**: Apontador para a função de comparação dos nodos da árvore.
  - **delete**: Apontador para a função de *free* dos nodos da árvore.

3.6.2 API

- **initNode**: Inicializa um nodo de uma árvore.
- **initAvl**: Inicializa uma árvore. Recebe como parâmetro um apontador para a função de comparação dos nodos da árvore e retorna um apontador para a AVL criada.
- **freeNode**: Liberta o espaço ocupado por um nodo de uma árvore.
- **freeAvl**: Liberta o espaço ocupado por uma árvore. Retorna um apontador para a AVL a NULL.
- **insert**: Efetua a inserção de um nodo numa árvore. Retorna um apontador para a AVL.
- **getTotalNodes**: Devolve o número total de nodos de uma árvore.
- **incrementCounters**: Incrementa os contadores de um nodo de uma árvore.

- **mapAVL**: Aplica uma função a todos os nodos de uma árvore.
- **findAndApply**: Aplica uma função a um nodo específico de uma árvore. Primeiro, procura na árvore o nodo recebido como parâmetro e, de seguida, aplica-lhe a função recebida como parâmetro. Esta função retorna o que retornar a função que lhe é passado como argumento.

3.7 Módulo dos artigos

Este módulo, chamado *articles*, contém a definição das estruturas que definem o que é um artigo e uma revisão do mesmo. Contém também algumas funções a serem aplicadas a ambas as estruturas.

3.7.1 Estruturas de dados

- **revision**: Estrutura responsável por guardar a informação relativa a uma revisão e constituída por:
  - **id**: Identificador único de uma revisão.
  - **timestamp**: Data e hora em que a revisão foi submetida.
  - **title**: Nome da revisão do artigo.
  - **textsize**: Tamanho do texto da revisão do artigo.
  - **wc**: Número de palavras do texto da revisão do artigo.
- **article**: Estrutura responsável por guardar a informação relativa a um artigo e constituída por:
  - **id**: Identificador único de um artigo.
  - **revcount**: Número de revisões de um artigo, isto é, posições ocupadas do seu *array* de revisões.
  - **revcapacity**: Capacidade atual (tamanho) do *array* de revisões de um artigo.
  - **revisions**: *Array* (dinâmico) das revisões de um artigo.
  - **occurences**: Número total de ocorrências de um artigo, incluindo duplicados e novas revisões do mesmo artigo.

3.7.2 API

- **initRevision**: Inicializa a estrutura responsável por guardar uma revisão, ou seja, inicializa o apontador para uma nova revisão e os seus campos. Para o ID, o *timestamp* e o *title*, o espaço é alocado aquando a execução das funções *setRevisionID*, *setTimestamp* e *setTitle*. No fim, retorna o apontador gerado para a revisão.
- **initArticle**: Inicializa a estrutura responsável por guardar um artigo, ou seja, inicializa o apontador para um novo artigo e os seus campos. Para o ID, o espaço é alocado aquando a execução da função *setArticleID*. O *array* de revisões é inicializado alocando espaço para duas revisões (número médio de revisões para cada artigo único). No fim, retorna o apontador gerado para o artigo.

- **freeRevision**: Liberta o espaço ocupado por uma revisão, ou seja, liberta os campos que eventualmente tinham sido alocados manualmente na memória através de um *malloc*, colocando o apontador para a revisão passado como parâmetro com o valor *NULL*.
- **freeArticle**: Liberta o espaço ocupado por um artigo, ou seja, liberta os campos que eventualmente tinham sido alocados manualmente na memória através de um *malloc*, colocando o apontador para o artigo passado como parâmetro com o valor *NULL*.
- **getRevisionAt**: Retorna uma cópia da revisão de um artigo situada na posição recebida com parâmetro do seu *array* de revisões.
- **duplicateArticle**: Função que trata dos casos de inserção de um artigo que já existia na árvore. Quando o artigo é repetido, é verificado se se está a adicionar uma nova revisão ou não. Caso seja uma revisão nova, esta é colocada no *array* de revisões de artigo que já estava na árvore. Caso contrário, apenas se adiciona a *flag* que é recebida como parâmetro e que sinalizará que não é necessário colocar o contribuidor correspondente na árvore dos contribuidores (pois não houve, de facto, nenhuma nova contribuição).
- **addFirstRevision**: Adiciona a primeira revisão (previamente alocada) a um artigo. No fim, retorna o apontador para o artigo.
- **setArticleID**: Altera o valor do campo ID de um artigo. É alocado espaço e efetuada uma cópia do ID recebido. No fim, retorna o apontador para o artigo com o campo alterado.
- **setRevisionID**: Altera o valor do campo ID de uma revisão. É alocado espaço e efetuada uma cópia do ID recebido. No fim, retorna o apontador para a revisão com o campo alterado.
- **setTimestamp**: Altera o valor do campo *timestamp* de uma revisão. É alocado espaço e efetuada uma cópia do *timestamp* recebido. No fim, retorna o apontador para a revisão com o campo alterado.
- **setTitle**: Altera o valor do campo *title* de uma revisão. É alocado espaço e efetuada uma cópia do *title* recebido. No fim, retorna o apontador para a revisão com o campo alterado.
- **setTextSize**: Altera o valor do campo *textsize* de uma revisão. É alocado espaço e efetuada uma cópia do *textsize* recebido. No fim, retorna o apontador para a revisão com o campo alterado.
- **setWordCount**: Altera o valor do campo *wordcount* de uma revisão. É alocado espaço e efetuada uma cópia do *wordcount* recebido. No fim, retorna o apontador para a revisão com o campo alterado.
- **getArticleID**: Retorna, alocando o espaço necessário, uma cópia do valor do campo ID de um artigo. O espaço alocado para a cópia é posteriormente libertado.

- **getOccurrences**: Retorna, alocando o espaço necessário, uma cópia do valor do campo *occurrences* de um artigo. O espaço alocado para a cópia é posteriormente libertado.
- **getRevCount**: Retorna, alocando o espaço necessário, uma cópia do valor do campo *revcount* de um artigo. O espaço alocado para a cópia é posteriormente libertado.
- **getTitle**: Retorna, alocando o espaço necessário, uma cópia do valor do campo *title* de uma revisão. O espaço alocado para a cópia é posteriormente libertado.
- **getTimeStamp**: Retorna, alocando o espaço necessário, uma cópia do valor do *timestamp*. O espaço alocado para a cópia é posteriormente libertado.
- **getLastRevisionTitle**: Retorna, alocando o espaço necessário, uma cópia do valor do campo *title* da última revisão de um artigo. O espaço alocado para a cópia é posteriormente libertado.
- **getArticleRevisionTimestamp**: Retorna, alocando o espaço necessário, uma cópia do valor do campo *timestamp* de uma revisão de um artigo. Se não existir revisão, retorna o valor *NULL*. O espaço alocado para a cópia é posteriormente libertado.
- **getBiggestRevisionSize**: Retorna o tamanho máximo das revisões de um artigo.
- **getBiggestRevisionWC**: Retorna o *wordcount* máximo das revisões de um artigo.

3.8 Módulo dos contribuidores

Este módulo, chamado *contributors*, contém a definição das estruturas que definem o que é um contribuidor e as respetivas funções.

3.8.1 Estruturas de dados

- **contributor**: Estrutura responsável por guardar a informação relativa a um contribuidor e constituída por:
  - **id**: Identificador único de um contribuidor.
  - **username**: Nome do contribuidor.
  - **revisions**: Número de revisões únicas feitas por um contribuidor.

3.8.2 API

- **initContributor**: Inicializa a estrutura responsável por guardar um contribuidor, ou seja, inicializa o apontador para um novo contribuidor e os seus campos. Para o ID e o *username*, não é alocado espaço ainda (o espaço para estes é alocado aquando a execução das funções *setContributorID* e *setUsername*) e o número de *revisions* é colocado a 1. No fim, retorna o apontador gerado para o contribuidor.

- **freeContributor**: Liberta o espaço ocupado por um contribuidor na memória. Começam por serem libertados os campos ID e *username* (o campo *revision* não é necessário, pois não é feito *malloc* para um inteiro) e só depois é feito *free* do apontador da estrutura.

- **duplicateContributor**: Trata dos casos de inserção de um contribuidor que já existe na árvore. Quando o contribuidor é repetido, apenas se incrementa o número de revisões do contribuidor original e faz-se *free* do duplicado. Um apontador para esta função é passado como parâmetro aquando da inserção de um contribuidor numa árvore. Assim sendo, tem de obedecer à API do módulo da AVL e, por isso, tem de receber a *flag* como parâmetro, mesmo não precisando dela.

- **setContributorID**: Altera o valor do campo ID de um contribuidor. É alocado espaço e efetuado uma cópia do ID recebido. No fim, retorna o apontador para o contribuidor com o campo alterado.

- **setUsername**: Altera o valor do campo *username* de um contribuidor. É alocado espaço e efetuado uma cópia do *username* recebido. No fim, retorna o apontador para o contribuidor com o campo alterado.

- **getContributorID**: Retorna uma cópia do conteúdo do campo ID de um contribuidor, alocando o espaço necessário que será, posteriormente, libertado.

- **getUsername**: Retorna uma cópia do conteúdo do campo *username* de um contribuidor, alocando o espaço necessário que será, posteriormente, libertado.

- **getRevisions**: Retorna uma cópia do conteúdo do campo *revisiona* de um contribuidor, alocando o espaço necessário que será, posteriormente, libertado.

4 Informações relevantes para trabalho futuro

Num caso hipotético em que uma outra equipa receba o nosso projeto e, a partir dele, queira continuar a trabalhar, poderá fazê-lo facilmente pois todos os módulos estão exaustivamente comentados e explicados.

Todas as funções de cada módulo possuem uma descrição genérica do seu funcionamento e o seu corpo está ainda comentado nos pontos fulcrais, de modo a que não seja preciso perder tempo a analisar o código e a tentar perceber o que ele faz.

Além disso, o facto dos dados estarem encapsulados, do código estar feito de uma forma modular e de se terem usado estruturas genéricas para guardar os dados (AVL), contribui para que uma futura equipa de desenvolvimento tenha o seu trabalho facilitado.

5 Conclusão

Com a realização deste trabalho, pudemos pôr em prática os conceitos adquiridos, sobretudo, nas UCs de Programação Imperativa e de Álgoritmos e Complexidade.

Percebemos a importância da modularidade do código e do encapsulamento dos dados na elaboração de um projeto sólido e bem fundamentado.

Achámos que era importante fazermos as nossas próprias estruturas de dados em vez de usarmos estruturas que já foram implementadas noutras bibliotecas (e.g. GLib). Assim sendo - e apesar de termos gastado mais tempo a implementar estruturas que já estavam feitas -, achámos que foi algo enriquecedor, que nos deu boas bases para trabalho futuro e que fez com que desenvolvessemos capacidades que, por outra via, não teriam sido desenvolvidas.