Projecto DB-UFP

Digital Bibliography UFP: Aplicação de Gestão de bibliografia digital

Programação Orientada aos Objetos (POO) com Recurso e Estruturas de Dados Lineares e Não Lineares

Linguagens Programação II	**Algoritmos e Estruturas de Dados II**
Rui Silva Moreira rmoreira@ufp.edu.pt João Viana jviana@ufp.edu.pt	José Torres jtorres@ufp.edu.pt Célio Carvalho celio.carvalho@ufp.edu.pt

Fevereiro 2024 (Versão 1.0)

Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia

1. Definição Geral do Problema

Neste projeto pretende-se que os alunos modelizem, implementem, testem e documentem uma aplicação, seguindo uma abordagem OOP em Java, para manipular e gerir informação sobre artigos (papers) de cariz académico/científico e respectivos autores.

Um artigo científico (paper) pode ser publicado numa edição de uma *Conferência* ou num *Journal* e possui tipicamente uma lista de autores e uma lista de referências bibliográficas de outros papers citados no corpo do texto (cf. referências). Pretende-se utilizar um grafo dirigido para representar as relações de citação entre papers, ou seja, cada vértice (node) representa um paper e cada ligação (edge) representa uma citação de um paper para outro paper que é citado no primeiro.

A título de exemplo, existem várias plataformas, como o Google scholar (https://scholar.google.com/) e a Semantic Scholar (https://www.semanticscholar.org/) que armazenam informações sobre paper e respectivas citações entre os papers, num grafo de citações. Estas ferramentas são importantes para gerir e fazer pesquisas de artigos. A plataforma independente (https://dblp.org) mantém também uma base de dados de artigos (nomeadamente da área de *Computer Science*) e dos seus autores, e integra as citações e referências de cada artigo com informação carregada de outras plataformas (http://opencitations.net, https://www.crossref.org, https://www.semanticscholar.org/).

Pretende-se ainda utilizar um grafo não dirigido para representar as relações entre os autores dos papers, onde cada vértice (node) representa um autor e cada ligação (edge) corresponde a uma relação entre dois autores que colaboraram na escrita de papers.

As entidades mais relevantes neste domínio são: **Autor** (cf. nome, nome curto/científico, nome científico, filiação, ORCID, Ciência ID, Google Scholar ID, Scopus Author ID, etc.); **Artigo** (cf. título, palavras chave, abstract, tipo de publicação (e.g. conferência, journal), ano, nº de downloads, nº visualizações em cada dia, nº de likes por dia, lista de autores, lista de referências/citações a outros artigos, etc.); **Publicação** (cf. **Journal** e respectivo nome, publisher, ano, periodicidade, JCR IF, Scopus IF, etc. ou **Conferência** e respectivo nome, ano, número da edição, local, etc.).

Devem utilizar-se estruturas do tipo grafo para representar associações entre algumas das entidades anteriores. Por exemplo, deve utilizar-se um **Grafo de Artigos** (grafo dirigido entre artigos que se citam uns aos outros - os vértices representam os Artigos e as aresta representam as citações) e um **Grafo Autores** (grafo não dirigido entre co-autores de artigos - os vértices representam os Autores e as aresta representam as co-autorias de 1+ artigos).

Os grafos de artigos e de autores devem estar relacionados entre si, uma vez que um artigo pode ter um ou mais autores e um autor pode ter um ou mais artigos.

O sistema implementado deverá permitir que um utilizador faça a gestão da sua biblioteca bibliográfica de acordo com as suas necessidades. Em particular, um utilizador pode carregar um grafo com vários nós (artigos) e respectivas ligações entre os nós (citações). Pode ainda carregar um grafo de autores com vários nós (autores) e respectivas ligações entre nós (colaborações de escrita entre autores). As ligações entre cada par de vértices do grafo de Artigos podem ter vários tipos de pesos (e.g. diferença temporal entre os dois artigos, diferença entre número de citações, etc.). Similarmente, as ligações entre vértices do grafo de Autores podem ter vários pesos (e.g. número de artigos em co-autoria, lista de artigos em co-autoria, etc.). Estes grafos podem depois ser utilizados para explorar conhecimento e relações entre os vários tipos de dados.

Adicionalmente, pretende-se que os alunos combinem a utilização de estruturas de dados orientadas a objetos (e.g. colecções lineares de dados, tabelas de símbolos, etc.) para armazenar e gerir toda a informação necessária, i.e. todas as entidades representadas.

As estruturas do tipo *Symbol Table* (e.g. *Redblack Binary Search Trees ou Hashmaps*) deverão permitir armazenar e gerir informação relativa a várias entidades, como por exemplo: Autores, Artigos, Publicações. Para cada entidade deverão ser guardados os atributos relevantes. No caso dos Autores, por exemplo, deverá registar-se o perfil com as suas áreas de investigação e formação. Deverá também ser possível consultar o histórico de todos os artigos publicados pelo autor.

As estruturas do tipo grafo deverão ser utilizadas para armazenar a informação bibliográfica relativa à rede de citações entre artigos e à rede de autores.

Para facilitar a abordagem ao problema e a sua implementação, irá utilizar-se uma arquitetura *standalone*, ou seja, o desenvolvimento de uma app que deverá funcionar individualmente em cada PC. Os alunos deverão utilizar pacotes de software pré-existentes (cf. algs4) que oferecem estruturas de dados genéricas (cf. grafos, árvores, tabelas de símbolos etc.). Estas estruturas OO devem ser reutilizadas na modelização do problema proposto e na respectiva implementação. Desta forma não terão que implementar as estruturas de dados básicas, podendo concentrar-se na lógica e requisitos funcionais da aplicação proposta.

2. Requisitos funcionais

Pretende-se que os alunos sigam uma abordagem orientada aos objetos (OO) na modelização e implementação do problema proposto. Em concreto deverão desenhar os diagramas de classes necessários que permitam representar as entidades e associações do

problema, reutilizando classes pré-existentes (cf. *graphs*, *trees*, *hashmaps*, etc.) através de herança ou composição.

Pretende-se que desenvolvam *packages* de classes com várias funcionalidades úteis à implementação do problema proposto e que satisfaçam os requisitos listados a seguir. Pretende-se ainda que implementem casos de teste (*Test Cases*) das *packages* para cada um dos requisitos. As funções de teste devem ser responsáveis por chamar todos os métodos necessários para demonstrar a correta funcionalidade de cada um dos requisitos. Cada caso de teste deverá ser devidamente documentado numa função *static* que deverá ser caracterizada pelo conjunto de funções a testar, pelos valores de input a utilizar no teste (preferencialmente provenientes de ficheiros ou definidos programaticamente nas funções de teste), e por valores de output/resultado do teste enviados para a consola e/ou escritos em ficheiros.

Estão previstos **dois milestones** de entrega do projeto, contudo, está prevista **apenas uma apresentação/defesa final presencial** (de “viva voz”), a acontecer no final do semestre, em datas a anunciar pelos docentes, com a presença obrigatória de todos os elementos de cada grupo. Os projetos entregues fora do prazo ou que não sejam apresentados de “viva voz”, não serão considerados para classificação.

2.1. Milestone 1

R1. Modelizar num diagrama UML as classes e respectivas associações para representar as **entidades** do problema proposto (e.g. Autor, Artigo, Publicação - Journal ou Conference, etc.). O modelo de dados deve caracterizar as entidades, recorrendo à reutilização por herança ou composição, com eventual redefinição (@Overriden) de estruturas de dados base da package algs4 (e.g. grafos, árvore, *hashmap*, etc.) e respetivos métodos/operações (cf. propriedades, pesquisas, etc.).

R2. Para armazenar toda a informação das entidades suportadas, deverá existir uma Base de Dados (BD) com várias coleções de dados. Devem utilizar vários tipos de *Symbol Tables* (ST) com chaves ordenáveis e não ordenáveis. Por exemplo, deverão utilizar *Binary Search Trees* balanceadas (RedBlack BST) para armazenar informação ordenável por chaves comparáveis (e.g. datas, nomes de autores, etc.); deverão utilizar STs do tipo *HashTable/HashMap* quando se pretende utilizar etiquetas para caracterizar as propriedades das entidades (e.g. vértices e arestas dos grafos). A BD deve expor várias interfaces para gerir as entidades necessárias, i.e., suportar todas as funções para inserir, remover, editar e listar a informação de cada uma das várias coleções consideradas.

R3. Em particular na remoção de informação das várias estruturas de dados, deve proceder-se ao arquivamento da informação necessária. Por exemplo, ao remover

um autor da base de dados deve garantir-se a atualização de todas as estruturas associadas (e.g. artigos), mantendo, para o caso dos autores, apenas uma string com o nome curto/científico do autor, e respectivo arquivamento em ficheiros ou em estruturas auxiliares. Deve ainda validar-se a consistência de toda a informação armazenada como, por exemplo, que todas as referências para os autores que são mencionados noutras STs, existam na ST principal de autores.

R4. Em particular nas pesquisas de informação, devem implementar diversas funções sobre a base de dados como, por exemplo, determinar:
    1. todos os artigos escritos por um autor num dado período;
    2. todos os artigos que não foram descarregados ou visualizados num dado período;
    3. todos os autores que já citaram uma dada lista de artigos e num dado período;
    4. o Top-3 dos Artigos que foram mais usados num dado período;
    5. as citações de todos os artigos de um journal para um determinado período temporal;

R5. Relativamente às listagens de informação, deverão existir métodos para gerar um relatório global do sistema que inclua: lista de artigos; lista de autores e lista de ligações entre artigos. Deverá ainda ser possível listar a utilização (vezes que é descarregado ou visualizado) mensal e anual dos artigos, assim como o número de votos (likes);

R6. Deverão criar unidades de teste, recorrendo a funções *static*, para validar todas as estruturas e funcionalidades pretendidas. As funções devem popular as diversas STs da aplicação para a realização dos testes e apresentar os resultados na consola para todos os requisitos solicitados;

R7. Todas as classes devem ser devidamente comentadas e anotadas para ser possível gerar a respectiva documentação, via Javadoc, incluindo todos os atributos e funções implementadas.

2.2. Milestone 2

R8. Acrescentar ao modelo de dados a representação de dois grafos: grafo de Artigos e grafo de Autores e atualizar as entidades necessárias. Devem reutilizar uma estrutura do tipo **grafo dirigido** (cf. algs4) para definir a rede global de Artigos e respectivas citações. O grafo deverá permitir organizar e relacionar todos os Artigos *e* Citações, com os respectivos atributos. As Citações correspondem às ligações entre Artigos existentes. As Citações podem ter vários atributos/pesos (e.g. diferença temporal entre os dois artigos, diferença entre número de citações, etc.); esta

informação poderá ser armazenada na ligação com recurso a pares chave/valor. Deverá ser utilizado um **grafo não dirigido** (cf. algs4) para definir a rede de Autores e respectivas colaborações entre Autores na escrita de artigos. Cada ligação entre dois autores deve representar as co-autorias na escrita de um ou mais artigos.

R9. Com base no grafo de Artigos deverão ser exploradas tarefas de pesquisa/gestão de informação, como por exemplo, usar o grafo para:

1. listar vértices (Artigos) publicados num determinado Journal ou Conferência e num determinado período temporal (entre duas datas);
2. calcular as citações de 1ª ordem, i.e. o número de papers que citam o artigo;
3. calcular as citações de 2ª ordem, i.e. o número total de citações de todos os artigos que citam um dado artigo;
4. calcular as autocitações de um dado artigo, i.e. o número de citações a outros artigos dos mesmos autores;
5. calcular caminhos mais curtos entre dois artigos, e.g. usando distâncias temporais;
6. Selecionar um sub-grafo contendo, por exemplo, apenas os Artigos de Journals ou de Conferências. A esses subgrafos poderão aplicar-se os mesmos algoritmos ou funcionalidades descritas anteriormente;
7. Verificar se o grafo principal ou qualquer um dos sub-grafos é ou não conexo;

R10. Com base no grafo de Autores deverão ser exploradas tarefas de pesquisa/gestão de informação, como por exemplo, usar o grafo para:

1. listar vértices ou arestas com base em múltiplos critérios úteis, como por exemplo, os vértices (Autores) que estão associados a determinadas instituições (combinando operadores && ou ||);
2. calcular para um dado autor, com quantos autores trabalhou;
3. calcular o número de artigos escritos entre dois autores;
4. calcular caminhos mais curtos entre dois autores, e.g. número de saltos (hops) mínimo entre dois autores;
5. Selecionar um sub-grafo contendo, por exemplo, apenas os Autores pertencentes a um conjunto de instituições. A esses subgrafos poderão aplicar-se os mesmos algoritmos ou funcionalidadesdescritas anteriormente;
6. Verificar se o grafo principal ou qualquer um dos sub-grafos é ou não conexo;

R11. Deverá ser criada uma interface gráfica (GUI) para visualização e manipulação dos grafos e entidades, nomeadamente:

1. Visualizar e editar a informação das entidades do problema proposto;
2. Visualizar os grafos de artigos e de autores. Deverão distinguir visualmente os diferentes tipos de Artigos (e.g. usando cores ou ícones diferentes);
3. Adicionar e remover elementos/entidades à base de informação e ao grafo;

4. Realizar todas as pesquisas definidas em requisitos anteriores bem como visualizar os respectivos resultados, recorrendo à GUI;

R12. Utilizar ficheiros de texto para popular e tornar persistente toda a informação das diversas entidades organizadas nas coleções de dados (cf. ST, Digraph, etc.). Deve ser possível carregar os dados (*input*) ou gravar (*output*) em ficheiros de texto. Os resultados da execução das pesquisas deverão poder ser também gravados (*output*) em ficheiros de texto para consulta posterior dos utilizadores.

R13. Deverá ser ainda possível importar e exportar os dados das entidades armazenados nas coleções de dados da aplicação (cf. Digraph, ST, etc.) recorrendo a ficheiros binários.

2.3. Cotação dos Requisitos

REQ	1	2	3	4	5	6	7	8	9	10	11	12	13
AED2	0	2	2	2	1	1	1	2	3	3	0	2	1
LP2	1	2	1	2	1	1	0.5	2	2	2	2.5	2	1

(NB: cotação 0-20)

3. Ficheiros e documentos a entregar

O projeto proposto deve ter uma implementação orientada aos objetos em Java. Todo o código (algoritmos e estruturas de dados) deve ser complementado com comentários apropriados, que facilitem a compreensão do mesmo e a respectiva geração automática de documentação.

Deve ser incluída uma explicação dos algoritmos implementados e uma menção ao desempenho dos mesmos assim como dos testes efetuados/implementados num ficheiro **README.md** escrito em markdown language (https://www.markdownguide.org/) e localizado na raiz do seu repositório do projeto. Devem ainda ser realizados testes unitários que demonstrem o bom funcionamento das classes desenvolvidas.

Todos os elementos necessários à avaliação deverão estar disponíveis através de um repositório **git** (https://github.com/) partilhado com os docentes desde o início do trabalho. No git, o **nome do repositório** escolhido deve codificar os números dos alunos, as UCs e ano letivo, como por exemplo: **1234_5678_aed2_lp2_202324**. Este repositório deve incluir os seguintes componentes complementares:

i) **Modelos de classes** (ficheiros **zargo** e **PNG** dos diagramas de classes);
ii) **Código fonte** de toda a implementação do projecto;
iii) **Ficheiros de teste** contendo os respectivos dados *input/output* utilizados;
iv) **Documentação** do código (páginas HTML geradas com *JavaDoc*).

4. Links para consulta

- https://algs4.cs.princeton.edu/home/ (biblioteca algs4 de princeton)