Pontifícia Universidade Católica de Minas Gerais
Curso de Engenharia de Software
Disciplina: Teoria de Grafos e Computabilidade
Prof. Gabriel Fonseca 2025/2

Trabalho prático

O trabalho pode ser feito em grupos de **até** 6 alunos. As implementações podem ser feitas nas linguagens **Java ou Python**, e o relatório deve ser feito em LaTeX. Ele deverá ser entregue no Canvas, até as 23:59 da data limite estipulada (data mostrada na tarefa do Canvas). Cópias serão sumariamente zeradas. A entrega deve ser realizada da seguinte forma: **um** arquivo *.zip* ou *.rar* contendo o código fonte do trabalho, o fonte do relatório (.tex) e o PDF do relatório. Além disso, indiquem no relatório as responsabilidades e o que foi feito por cada membro do grupo. As entregas intermediárias devem ser feitas através do Github Classroom ou Verde, conforme definido nos enunciados de cada ponto de controle.

Objetivo

Desenvolver uma ferramenta computacional que processe dados estruturados como grafos, aplicando conceitos da teoria dos grafos e boas práticas de engenharia de software. O projeto envolve o estudo de um repositório real de algum projeto de código aberto e a análise das interações dos colaboradores, utilizando grafos como estrutura central para modelagem, processamento e análise dos dados.

Etapas do Trabalho

Etapa 1 – Modelagem e Planejamento da Solução

Escolha de um repositório público no GitHub com um número significativo de estrelas (Acima de 5.000 estrelas), de modo a garantir uma comunidade ativa e rica em interações. A partir deste repositório, o aluno deve realizar a extração e levantamento de dados, com foco especial nas interações entre usuários, incluindo:

- Comentários em *issues*;
- Fechamento de *issues*;
- Comentários em *pull requests*;
- Abertura, revisão, aprovação e merge de *pull requests*;

Modelagem do Grafo: Cada usuário deve ser representado como um nó. As interações entre usuários são representadas como arestas. O grafo criado deve ser **simples e direcionado**, e caso seja necessária a representação de uma relação bidirecional, deve-se utilizar arestas **anti-paralelas**.

O aluno deverá realizar a modelagem em duas etapas:

1. Construir grafos separados, um para cada tipo de relação:
    - Grafo 1: comentários em *issues* ou *pull requests*;
    - Grafo 2: fechamento de *issue* por outro usuário;
    - Grafo 3: revisões/aprovações/merges de *pull requests*;

2. Construir um grafo integrado, no qual cada usuário é um nó e cada aresta representa uma combinação ponderada de todas as interações. Os pesos devem refletir a intensidade de cada tipo de relação, conforme sugerido abaixo:

    - Comentário em *issue* ou *pull request*: peso 2;
    - Abertura de *issue* comentada por outro usuário: peso 3;
    - Revisão/aprovação de *pull request*: peso 4;
    - Merge de *pull request*: peso 5.

O grafo consolidado deve permitir analisar a rede de colaboração do repositório considerando as diferentes naturezas das interações.

Essas regras podem ser adaptadas pelos alunos, mas refletindo as interações mais relevantes para a colaboração técnica (como revisões e merges) tenham maior peso do que interações mais leves (como reações).

Entregável 1: Documento com a descrição do problema, justificativa da escolha do repositório, a estratégia de coleta de dados (incluindo as interações analisadas e como foram transformadas em arestas), a proposta de modelagem do grafo com pesos e o plano de desenvolvimento da solução.

Entregável 2: Submissão do código de mineração de dados na plataforma VERDE[1].

Etapa 2 – Desenvolvimento da Ferramenta

Implementação da estrutura de grafos, aplicação dos algoritmos e construção de uma interface mínima de uso.

Entregável: Protótipo funcional com código versionado no Github Classroom.

Nesta etapa, os alunos devem desenvolver a ferramenta de acordo com a seguinte organização e requisitos:

- **Estrutura de classes:**
    - Classe abstrata `AbstractGraph`, definindo a API comum, atributos compartilhados (rótulos e pesos de vértices), e métodos auxiliares (ex.: verificação de índices).
    - Classe concreta `AdjacencyMatrixGraph`, implementando a API utilizando **matriz de adjacência**.
    - Classe concreta `AdjacencyListGraph`, implementando a API utilizando **listas de adjacência**.
- **Construtores:**
    - `AdjacencyMatrixGraph(int numVertices)`
    - `AdjacencyListGraph(int numVertices)`
- **API obrigatória:**
    - `int getVertexCount();`
    - `int getEdgeCount();`
    - `boolean hasEdge(int u, int v);`
    - `void addEdge(int u, int v);`
    - `void removeEdge(int u, int v);`
    - `boolean isSucessor(int u, int v);`
    - `boolean isPredessor(int u, int v);`
    - `boolean isDivergent(int u1, int v1, int u2, int v2);`

[1] Acessível em: `http://verde.icei.pucminas.br`

  - `boolean isConvergent(int u1, int v1, int u2, int v2);`
  - `boolean isIncident(int u, int v, int x);`
  - `int getVertexInDegree(int u);`
  - `int getVertexOutDegree(int u);`
  - `void setVertexWeight(int v, double w);`
  - `double getVertexWeight(int v);`
  - `void setEdgeWeight(int u, int v, double w);`
  - `double getEdgeWeight(int u, int v);`
  - `boolean isConnected();`
  - `boolean isEmptyGraph();` e
  - `boolean isCompleteGraph()`.

- **Restrições:**
  - Grafos devem ser **simples**: não permitir laços nem múltiplas arestas.
  - `addEdge(u,v)` deve ser idempotente (não duplicar arestas).
  - Deve-se lançar exceções para índices inválidos e para operações inconsistentes.

- **Adicional:**
  - Método `void exportToGEPHI(String path)` para exportar o grafo em um dos formatos aceitos pelo software de visualização GEPHI[2]

A avaliação será baseada na corretude da API, na utilização de herança e abstração, e na clareza do código.

Etapa 3 – Análise do repositório baseada em dados

Análise dos repositórios utilizando algoritmos em grafos aprendidos na disciplina, e algoritmos e métricas de redes complexas, como:

- Métricas de Centralidade:
  1. Grau (degree centrality): mede quantas conexões diretas cada colaborador tem (ex.: número de interações com outros membros). Indica quem participa mais ativamente de revisões, discussões ou coedições.
  2. Centralidade de intermediação (betweenness centrality): mostra quais colaboradores atuam como "pontes" entre diferentes grupos ou áreas do projeto.
  3. Centralidade de proximidade (closeness centrality): identifica quem está mais "próximo" de todos os outros, ou seja, quem tem acesso rápido à informação no grafo.
  4. PageRank / Eigenvector centrality: mede a influência de um colaborador, ponderando não só suas conexões, mas também a importância de quem está conectado a ele.

- Métricas de Estrutura e Coesão
  1. Densidade da rede: proporção entre o número de conexões existentes e o número máximo possível. Indica o quão colaborativa é a rede como um todo.
  2. Coeficiente de aglomeração (clustering coefficient): mede a tendência de colaboradores formarem "clusters" (pequenos grupos muito conectados).
  3. Assortatividade: mostra se colaboradores com muitas conexões tendem a se conectar entre si (rede centralizada) ou se interagem mais com colaboradores menos conectados.

[2]Formatos aceitos disponíveis em: `https://gephi.org/users/supported-graph-formats/`

- Métricas de Comunidade

    1. Detecção de comunidades (ex.: modularidade): permite identificar grupos de colaboradores que trabalham mais frequentemente juntos (ex.: times informais dentro do projeto).
    2. Bridging ties: análise de quem conecta diferentes comunidades, atuando como elo entre grupos que, de outra forma, seriam isolados.

Entregáveis: Relatório técnico em LaTeX usando o template oficial da SBC[3] (obrigatório); apresentação oral com demonstração da ferramenta, podendo ou não ser necessária a utilização de slides; repositório Git contendo o histórico de desenvolvimento (commits individuais serão avaliados).

Relatório:

- Entre 7 e 15 páginas;
- Contextualizar as escolhas de modelagem e desenvolvimento, incluindo escolha de repositório;
- Apresentar a modelagem proposta;
- Demostrar todos artefatos produzidos, como modelagem, diagramas, resultados obtidos e telas produzidas;

Apresentação em vídeo (demonstração) e presencial (arguição):

- Demostrar os resultados;
- Todos os membros do grupo devem participar;
- Slides podem ser solicitados para a apresentação;
- Duração da apresentação entre 10 e 15 minutos; e
- Duração do vídeo entre 5 e 10 minutos.

Obs.: Somente serão avaliadas entregas que tenham todos os itens; a falta de um deles causa nota igual a zero.

[3] Acessível em: `https://www.overleaf.com/latex/templates/sbc-conferences-template/blbxwjwzdngr`