Segundo Exercício-Programa

Prof. Luciano Antonio Digiampietri

Prazo máximo para a entrega: 22/06/2025

1 Centralidade em Redes Sociais

Em Teoria dos Grafos, medidas de centralidade têm por objetivo quantificar o quão central é a posição que um vértice ocupa.

Em análise de redes sociais, a centralidade costuma estar ligada à importância, influência ou prestígio do indivíduo.

Existem diferentes formas de se medir a centralidade de um vértice, a seguir são descritas quatro das mais utilizadas.

1. **Centralidade de Grau** (*Degree Centrality*): corresponde ao grau de entrada de cada vértice em um grafo direcionado. Em redes sociais, indica quantas pessoas seguem um determinado indivíduo.

2. **Centralidade de Proximidade** (*Closeness Centrality*): medida para verificar o quão próximo um nó está em relação a todos os demais da rede. Identifica nós que estão mais próximos dos demais, possuindo maior capacidade de disseminar informações pela rede.

3. **Centralidade de Intermediação** (*Betweenness Centrality*): medida para verificar o quão presente um nó está nos caminhos mais curtos da rede. Identifica nós que estão nos caminhos mais curtos (que fazem as ligações mais rápidas) com os demais nós da rede. Nós com maior centralidade de intermediação estão mais presentes nos fluxos de informação mais rápidos/eficientes da rede.

4. **Centralidade Page Rank**: medida para verificar a importância de um nó com base na importância dos nós que apontam para ele. Indivíduos mais centrais são aqueles que são seguidos por pessoas centrais/importantes.

Neste EP trabalharemos com Redes Sociais representadas via grafos direcionados e não ponderados, em que vértices representam pessoas e arestas representam a existência de um relacionamento, indicando que uma pessoa segue a outra.

Detalhamento: Para este EP, utilizaremos a mesma estrutura de dados usada nas aulas para representar grafos direcionados e não ponderados com matrizes de adjacências.

O código fornecido para este EP já possui várias funções implementadas de gerenciamento básico de grafos, bem como para realizar alguns testes no EP.

A estrutura do grafo é dada a seguir:

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

Você deverá implementar/completar quatro funções, podendo, se julgar conveniente, implementar funções adicionais/auxiliares.

Todas as funções recebem ao menos dois parâmetros: o ponteiro/endereço para o grafo g e o endereço de um arranjo cujo tamanho é igual ao número de vértices do grafo (e o qual deve ser preenchido por sua função). Para este EP, você pode considerar que g sempre terá o endereço de um grafo válido.

- *void centralidadeDeGrau(Grafo* g, double* valores)*: função que calcula o grau de entrada de cada vértice (ignorando-se auto-laços) dividido pelo número de vértices menos um. Isto é, para um grafo com n vértices, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir o grau de entrada do vértice 0 (sem contar o auto-laço de 0 [aresta do vértice 0 para o próprio vértice 0], caso exista) dividido por $(n-1)$.

- *void centralidadeDeProximidade(Grafo* g, double* valores)*: função que calcula o quão próximo, na média, cada vértice está em relação demais. Este cálculo é feito de forma normalizada dividindo o número de vértices menos um pela soma das distâncias do vértice atual em relação aos demais. Isto é, para um grafo com n vértices, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir o valor $(n-1)$ dividido pela soma da distância do vértice 0 em relação a cada um dos demais $n-1$ vértices. Você pode usar o Algoritmo de Floyd-Warshall, já implementado no código do EP, para te ajudar a resolver esta função.

- *void centralidadeDeIntermediacao(Grafo* g, double* valores)*: função que calcula em quantos caminhos mais curtos entre vértices um dado vértice está presente. Neste EP, faremos uma simplificação, considerando apenas um caminho mais curto entre cada par de vértices (embora possam existir diversos caminhos com mesmo tamanho). Este cálculo será feito de forma normalizada dividindo o número de vezes que o vértice aparece nos caminhos mais curtos entre todos os outros pares de vértices produzido pelo Algoritmo de Floyd-Warshall dividido pelo total de combinações entre os demais pares de vértice. Isto é, para um grafo com n vértices, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir o número de vezes que o vértice 0 aparece

nos caminhos mais curtos entre os demais vértices (com base na matriz de predecessores produzida pela função $calculaDistanciaFloydWarshall$ que já está implementada no EP) dividido por $(n-1)*(n-2)$.

- *void centralidadePageRank(Grafo* g, double* valores, int iteracoes)*: função iterativa que calcula a centralidade de um dado vértice de acordo com a centralidade dos demais. O parâmetro *iteracoes* determina quantas iterações de cálculo da medida (entre 0 e 10.000) sua função deverá executar. Inicialmente, todos os vértices começam com o mesmo valor de centralidade (igual a 1 dividido pelo número de vértices) e a cada iteração a seguinte fórmula é aplicada para atualização da centralidade:

$$PR(x;0) = \frac{1}{N}$$

$$PR(x;t+1) = \frac{1-d}{N} + d * \sum_{y \in Seguem(x)} \frac{PR(y;t)}{Saida(y)}$$

sendo $PR(x;t)$ a centralidade Page Rank do nó x na iteração t, N o número de vértices, d o fator de amortização (considere $d = 0.85$ para este EP), $Seguem(x)$ o conjunto de nós que apontam para x (seguem x) e $Saida(y)$ o grau de saída de y, ignorando auto-laços.

Os arranjos que deverão ser preenchidos por suas funções podem **não estar zerados**, assim, certifique-se de preencher todas as posições do arranjo com os valores corretos.

Para saber um pouco mais sobre Medidas de Centralidade, você pode acessar os seguintes slides: `https://www.each.usp.br/digiampietri/garsc/disciplina/03_AnaliseEstrutural.pdf` ou a seguinte videoaula: `https://youtu.be/0ZnjQ9lIrVk?si=uwod6fEuzPSkeyIW`.

Exemplo: A seguir são apresentadas as soluções para um grafo específico.

```
Imprimindo grafo (v=5; a=8)
      0   1   2   3   4
  0   0   1   1   0   0
  1   0   0   1   1   1
  2   0   0   0   1   0
  3   0   0   0   0   1
  4   1   0   0   0   0

Centralidade de Grau:
  v0   v1   v2   v3   v4
0.250 0.250 0.500 0.500 0.500

Centralidade de Proximidade:
  v0   v1   v2   v3   v4
0.667 0.800 0.400 0.444 0.500

Centralidade de Intermediacao:
  v0   v1   v2   v3   v4
0.500 0.250 0.000 0.250 0.500

Centralidade Page Rank: 0 iteracoes
  v0   v1   v2   v3   v4
0.200 0.200 0.200 0.200 0.200

Centralidade Page Rank: 1 iteracao
  v0   v1   v2   v3   v4
0.200 0.115 0.172 0.257 0.257

Centralidade Page Rank: 10 iteracoes
  v0   v1   v2   v3   v4
0.241 0.130 0.167 0.212 0.251

Centralidade Page Rank: 100 iteracoes
  v0   v1   v2   v3   v4
0.240 0.132 0.169 0.211 0.247
```

Para facilitar o entendimento das medidas calculadas, seguem as matrizes de distância e predecessores produzidas pelo algoritmo de Floyd-Warshall:

```
Exibindo matriz de distancias.
    0   1   2   3   4
0   0   1   1   2   2
1   2   0   1   1   1
2   3   4   0   1   2
3   2   3   3   0   1
4   1   2   2   3   0

Exibindo matriz de predecessores.
    0   1   2   3   4
0   0   0   0   1   1
1   4   1   1   1   1
2   4   0   2   2   3
3   4   0   0   3   3
4   4   0   0   1   4
```

Observação 1: A *centralidadeDeIntermediacao* é baseada no número de vezes que um vértice aparece no caminho mais curto entre outros pares de vértices. Isso não é, exatamente, o número de vezes que o vértice aparece na matriz de predecessores. No exemplo dado, o caminho mais curto entre o vértice 4 e o vértice 3 é $4 \rightarrow 0 \rightarrow 1 \rightarrow 3$, assim, o vértice 0 está neste caminho mais curto, mas ao observar a posição [4][3] da matriz de predecessores, você verá o vértice 1 (pois a matriz não mostra os caminhos em si, mas sim o último predecessor).

Observação 2: Algumas medidas de centralidade indicariam a "importância" do nó apenas se considerássemos o grafo transposto. Por simplicidade, neste EP, consideraremos apenas o grafo "normal" para os cálculos solicitados.

1.1 Material a Ser Entregue

Um arquivo, denominado `NUSP.c` (sendo NUSP o seu número USP, por exemplo: 123456789.c), contendo seu código, incluindo todas as funções solicitadas e qualquer outra função adicional que achar necessário. Para sua conveniência, `completeERenomeie.c` será fornecido, cabendo a você então completá-lo e renomeá-lo para a submissão.

Atenção!

1. Não modifique as assinaturas das funções já implementadas e/ou que você deverá completar!
2. Para a avaliação, as diferentes funções serão invocadas diretamente (individualmente ou em conjunto com outras funções). Em especial, qualquer código dentro da função `main()` será ignorado.

2 Entrega

A entrega será feita única e exclusivamente via sistema e-Disciplinas, até a data final marcada. Deverá ser postado no sistema um arquivo .c, tendo como nome seu número USP:

```
seuNumeroUSP.c   (por exemplo, 12345678.c)
```

Não esqueça de preencher o cabeçalho constante do arquivo .c, com seu nome, número USP, turma, etc.

A responsabilidade da submissão é exclusivamente sua. Por isso, submeta e certifique-se de que o arquivo submetido é o correto (fazendo seu download, por exemplo). Problemas referentes ao uso do sistema devem ser resolvidos com antecedência.

3 Avaliação

A nota atribuída ao EP será baseada nas funcionalidades solicitadas, porém não esqueça de se atentar aos seguintes aspectos:

1. Documentação: se há comentários explicando o que se faz nos passos mais importantes e para que serve o programa (tanto a função quanto o programa em que está inserida);
2. Apresentação visual: se o código está legível, identado etc;
3. Corretude: se o programa funciona.

Além disso, algumas observações pertinentes ao trabalho, que influenciam na sua nota, são:

- Este exercício-programa deve ser elaborado individualmente (os códigos devem ser produzidos pelo aluno e não por *chatbots* ou outros sistemas geradores de código);
- Não será tolerado plágio;
- Exercícios com erro de sintaxe (ou seja, erros de compilação), receberão nota ZERO.