Primeiro Exercício-Programa

Prof. Luciano Antonio Digiampietri

Prazo máximo para a entrega: 18/05/2025

1 Predição de Relacionamentos em Redes Sociais

Diferentes estruturas ou sistemas do mundo real podem ser representados utilizando-se grafos.

Neste EP trabalharemos com Redes Sociais representadas via grafos não direcionados e não ponderados, em que vértices representam pessoas e arestas representam a existência de um relacionamento (por exemplo, de amizade) entre pares de pessoas.

Um dos aspectos mais estudados na Análise de Redes Sociais em relação à dinâmica das redes é a predição de relacionamentos (*Link Prediction*). A predição de relacionamentos visa identificar/sugerir relacionamentos que não estão presentes na rede social.

Do ponto de vista da antropologia e sociologia, alguns aspectos sociais relevantes para a criação de relacionamentos são:[1]

1. Homofilia: quanto mais interesses em comum as pessoas possuem, como características ou preferências, maior as chances de se relacionarem.

2. Raridade: há mais chances de relacionamento entre pessoas com características ou preferências raras em comum. Pois características difíceis de encontrar tendem a se destacar em relação às outras características.

3. Influência Social: uma característica compartilhada com muitos amigos de uma determinada pessoa pode ser útil para encontrar potenciais relacionamentos.

4. Amizades em Comum (ou vizinhos em comum): quanto mais amigos em comum duas pessoas possuem, maiores são as chances de se relacionarem.

5. Proximidade Social: pessoas localizadas próximas em um grafo social possuem um relacionamento em potencial.

6. Conexão Preferencial: pessoas populares tendem a atrair mais pessoas quando comparadas às pessoas com poucos relacionamentos.

A ideia aqui é desenvolver funções que calculam, para um dado indivíduo (vértice) diferentes métricas que tentam mensurar a potencial relação com os demais indivíduos da rede. Cada uma dessas funções receberá o endereço de um grafo não direcionado e não ponderado

[1] Yin et al. (2010), Maruyama (2016)

(utilizando a representação de matriz de adjacências vista em aula, acrescida das informações de preferência de cada indivíduo), a indicação de um vértice `v` e o endereço de um arranjo que receberá a respectiva métrica calculada para cada um dos indivíduos da rede em relação ao vértice `v` (suas funções deverão preencher os valores desse arranjo).

Detalhamento: Para este EP, utilizaremos a mesma estrutura de dados usada nas aulas para representar grafos não direcionados e não ponderados com matrizes de adjacências, acrescentada de uma informação sobre as características (ou preferências) dos usuários.

Esta modificação na estrutura trata-se de uma matriz adicional de inteiros, chamada *caracteristicas*, cujo número de linhas é igual ao número de vértices (uma linha por vértice) e o número de colunas é fixo e igual a 10, correspondendo a 10 características do perfil do usuário. Para cada vértice haverá 10 números inteiros indicando sua preferência em relação a cada característica. Por exemplo, assuma que a primeira coluna corresponda à *comida favorita* e o valor zero corresponde a *feijoada*, enquanto o valor um corresponde a *strogonoff* e assim por diante. O valor -1 indicará que o usuário não preencheu esta informação em seu perfil. Cada característica terá seu valor entre -1 (o usuário não preencheu) e 99.

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;
  int** caracteristicas;
} Grafo;
```

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

Todas as funções recebem três parâmetros: o ponteiro/endereço para o grafo g, o vértice alvo v (para o qual serão calculadas as métricas em relação a todos os vértices do grafo) 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 e que v sempre corresponderá a um vértice do grafo g.

- *void homofilia(Grafo* g, int v, int* valores)*: função que calcula o número de características compartilhadas entre o vértice v e cada um dos vértices do grafo. Isto é, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir o número de características em comum entre o vértice v e o vértice 0. Especificamente na posição

valores[v] deverá atribuir o número de todas as características preenchidas por v (com valores diferentes de -1), isto é, todas as características de v são características que ele compartilha com ele mesmo. Note que uma característica com valor igual a -1, indica que o usuário não preencheu aquele valor de característica então se outro usuário também tiver colocado o valor -1 para a mesma característica, isso **não** deve ser considerado uma característica compartilhada pelos dois usuários.

- *void raridade(Grafo* g, int v, double* valores)*: função que calcula o número de características compartilhadas entre o vértice v e cada um dos vértices do grafo, ponderando cada uma dessas características pelo número total de pessoas que possuem essa característica (que possuem o mesmo valor para essa característica). Isto é, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir a soma da taxa de raridade de cada característica, sendo essa taxa igual a 0 (zero) se os vértices v e 0 não possuem o mesmo valor para essa característica ou possuem valor igual a -1, ou o valor $1/r$, sendo r o número total de vértices que possuem o mesmo valor de v para a respectiva característica (incluindo o próprio v nessa conta). Novamente, especificamente na posição *valores*[v] você deverá considerar que todas as características preenchidas por v (isto é, com valores diferentes de -1) são compartilhadas por ele mesmo. Lembre-se que uma característica com valor igual a -1, indica que o usuário não preencheu aquele valor de característica então se outro usuário também tiver colocado o valor -1 para a mesma característica, isso **não** deve ser considerado uma característica compartilhada pelos dois usuários.

- *void influenciaSocial(Grafo* g, int v, int* valores)*: função que calcula a soma ponderada das características de cada um dos vértices do grafo em relação às características dos vizinhos de v. Isto é, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir a soma da taxa de influência de cada característica, sendo essa taxa igual a 0 (zero) se o vértice 0 possui valor igual a -1 para a respectiva característica ou igual ao número de vizinhos de v que possuem o mesmo valor do vértice 0 para essa característica. Se 0 for vizinho de v, ele também deve ser considerado nessa conta.

- *void amizadesEmComum(Grafo* g, int v, int* valores)*: função que calcula o número de vizinhos em comum (ou adjacentes em comum) do vértice v com cada um dos vértices do grafo. Isto é, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir o número de vizinhos em comum entre o vértice v e o vértice 0. Especificamente na posição *valores*[v] deverá atribuir o número de vizinhos de v (isto é, todos os vizinhos de v são vizinhos em comum com ele mesmo).

- *void proximidadeSocial(Grafo* g, int v, int* valores)*: função que calcula a distância entre o vértice v e os demais vértices do grafo. Isto é, na posição 0 (zero) do arranjo *valores* (isto é, *valores*[0]) a função deverá atribuir a distância (em termos de número de arestas) entre o vértice v e o vértice 0. Se não for possível alcançar 0 a partir de v, você deverá atribuir o valor n, onde n é igual ao número de vértices do grafo.

Especificamente na posição $valores[v]$ deverá atribuir o valor 0 (zero). Para esta função, se quiser, você pode se basear no algoritmo de Busca em Largura fornecido no site da disciplina (lembrando que, se você for usar uma busca em largura ou outro algoritmo do tipo, você deve implementá-lo dentro do código de seu EP).

- *void conexaoPreferencial(Grafo* g, int v, int* valores)*: função que atribui a cada posição do arranjo *valores* o grau do respectivo vértice. Isto é, na posição 0 (zero) do arranjo *valores* (isto é, $valores[0]$) a função deverá atribuir o grau do vértice 0.

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 Predição de Relacionamentos em Redes Sociais, você pode acessar os seguintes slides: `http://www.each.usp.br/digiampietri/garsc/disciplina/07_PredicaoDeRelacionamentos.pdf` ou a seguinte videoaula: `https://youtu.be/vS5OokgSfko?si=DQfi757E5ZVQaX2i`.

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

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

Caracteristicas:
[ 0]  -1   1   2  -1  -1  -1  -1  -1  -1  -1
[ 1]  -1   1   2   3  -1  -1  -1  -1  -1  -1
[ 2]  -1  -1   2  -1  -1  -1  -1  -1  -1  -1
[ 3]  -1  -1   2  -1  -1  -1  -1  -1  -1  -1
[ 4]  -1  -1   5   3   4  -1  -1  -1  -1  -1

Realizando analise em relacao ao vertice 0.
Homofilia:
 v0  v1  v2  v3  v4
  2   2   1   1   0

Raridade:
  v0   v1   v2   v3   v4
0.75 0.75 0.25 0.25 0.00

Influencia Social:
 v0  v1  v2  v3  v4
  3   4   2   2   1

Amizades em Comum:
 v0  v1  v2  v3  v4
  2   0   0   2   1

Proximidade Social:
 v0  v1  v2  v3  v4
  0   1   1   2   2

Conexao Preferencial:
 v0  v1  v2  v3  v4
  2   3   2   2   1
```

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 ache 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 em sua nota, são:

- Este exercício-programa deve ser elaborado individualmente;
- Não será tolerado plágio;
- Exercícios com erro de sintaxe (ou seja, erros de compilação), receberão nota ZERO.

Referências

1 MARUYAMA, William Takahiro. Predição de coautorias em redes sociais acadêmicas. 2016. Dissertação (Mestrado em Sistemas de Informação) - EACH-USP, São Paulo, 2016. doi:10.11606/D.100.2016.tde-07052016-232625.

2 YIN, Z. et al. A unified framework for link recommendation using random walks. In: Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on., 2010. p. 152–159.