Fundação CECIERJ - Vice Presidência de Educação Superior a Distância
Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação com Interfaces Gráficas
AD1 2^o semestre de 2019.
Professores: Mario Benevides e Paulo Roma

AD1: Grafos Sociais

1 Introdução

Este projeto lhe dará a oportunidade de escrever uma classe Python que constrói um grafo dirigido ponderado, para modelar amizades em uma rede social. Grafos são úteis em várias aplicações, como jogos de computador (por exemplo, modelar objetos de jogos com relações espaciais), mapas de cidades e rotas, redes de computadores (por exemplo, encontrar um caminho mais curto para enviar dados entre computadores), apenas para citar uns poucos.

2 Resumo

Você deve implementar duas classes: DiGraph.py e SocialGraph.py.

- A classe *DiGraph* é uma classe genérica. Esta classe requer métodos para manipular um grafo dirigido ponderado, tais como, para adicionar vértices e arestas, remover vértices e suas arestas associadas ou computar o caminho mais curto de um dado vértice para todos os outros vértices. Veja a API (*Application Programming Interface*) de referência desta classe, com todos os métodos a serem implementados [1]. A classe utiliza um dicionário Python para armazenar vértices, que são associados às suas arestas com pesos, na forma de conjuntos (*sets*) [2] de *Edges*. O código para a classe *Edge* está descrito na API, e a implementação de

[1] `http://orion.lcg.ufrj.br/python/ADs/AD1_2019-1_refman.pdf`
[2] `https://docs.python.org/2/library/sets.html`

vários métodos é apresentada nos comentários. Um objeto *Edge* deve ser *hashable* [3].

- *SocialGraph* é uma classe que lê cada linha de um arquivo de entrada, com o nome do arquivo especificado na linha de comando, e recebido em sys.argv[1] (sys.argv[0] contém o nome do programa). Se o nome do arquivo de entrada não for fornecido ($len(sys.argv) < 2$), imprima uma mensagem de erro na tela e termine o programa. Cada linha no arquivo descreve um comando e seus argumentos. Os comandos servem para estabelecer uma amizade entre nomes, ou mostrar amigos de um nome dado. O método principal chama o método associado a cada comando e passa seus argumentos. Veja o arquivo de entrada infile.txt e os resultados no arquivo de saída em outfile.txt. Assuma que o formato do arquivo de entrada está correto.

Não é necessário submeter código de teste que use o módulo unittest na AD1.

3 Construção de um Grafo Social

A Figura 1 apresenta o grafo após execução do comando "add James Larry 1", depois que o objeto *DiGraph* for construído. James é um amigo de Larry. O peso indica a distância da amizade baseada na percepção de James (ou seja, quão perto Larry está de James, com base na percepção de James). A distância está entre 1 e 5 inclusive, onde 1 indica a menor distância (ou seja, um amigo melhor); 5 indica a distância máxima. Note-se que sem um comando "add Larry James 1" explícito, no arquivo de entrada, Larry não será amigo de James. Além disso, se mais tarde, uma aresta de Larry para James é adicionada, não é necessário que o peso dessa aresta seja o mesmo que o de James para Larry. Em outras palavras, a distância da amizade não é necessariamente simétrica.

Na Figura 2, a aresta entre James e Larry já está lá. A linha "add James Larry 2" sobrescreve o peso existente entre James e Larry. Os outros nós são adicionados de acordo. O resultado de cada linha está mostrado.

[3] `http://blog.lerner.co.il/is-it-hashable-fun-and-games-with-hashing-in-python/`

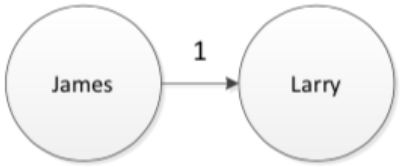

Figura 1: Grafo resultante da execução de "add James Larry 1" no arquivo de entrada; o número de vértices no grafo é 2 e o número de arestas é 1.

```
showFriends James
{<Liam, 1>, <Jim, 2>, <Larry, 2>}
showFriends Larry
{<Kaith, 1>}
showFriends Liam
{<Jim, 1>, <Keith, 1>, <Kaith, 3>}
showFriends Jim
{<Keith, 1>}
showFriends Keith
{<Jim, 1>}
showFriends Kaith
set()
```

O comando $recommendFriends < name >< dist|weightedDist >< topK >$ possui duas opções: "dist", para encontrar os candidatos a amigos usando as distâncias de amizade e "weightedDist", para encontrar candidatos a amigos usando distâncias de amizade ponderadas pelo número de arestas incidentes no vértice. Quanto mais arestas incidentes, mais a pessoa é querida. $< name >$ deve ser substituído pelo nome de interesse e $< topK >$ deve ser substituído pelo número máximo de nomes a serem retornados (quando não houver empates em termos das distâncias/distâncias ponderadas).

```
recommendFriends James dist 1
[<Keith, 2>]
recommendFriends James weightedDist 5
[<Keith, 16>, <Kaith, 21>]
```

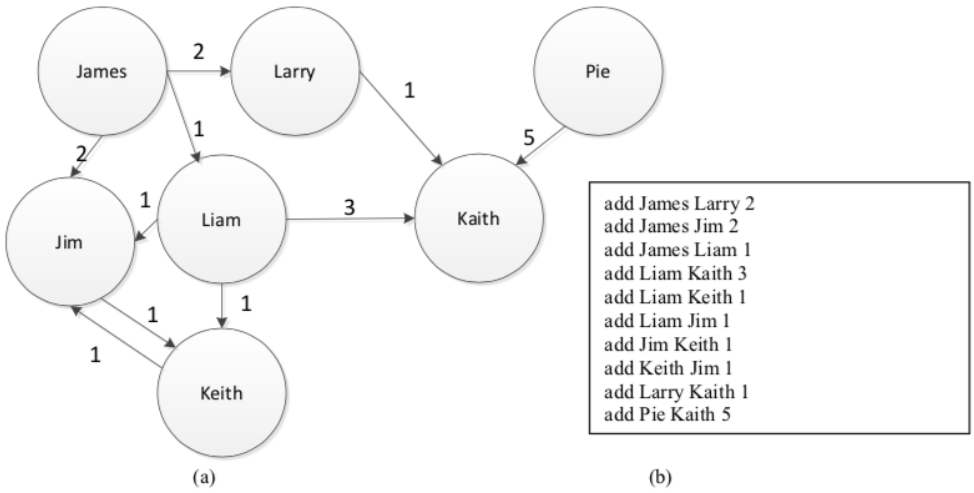

Figura 2: (a) Resultado das linhas mostradas em (b), no arquivo de entrada, sobre o grafo da Figura 1. O número de vértices no grafo é 7 e o número de arestas é 10.

- "recommendFriends James dist 1"

  Encontre um candidato a amigo para James, usando apenas a distância da amizade. O caminho mais curto de James para Keith possui uma distância de $1 + 1 = 2$ (James, Liam, Keith). A distância de James a Jim também é 2; no entanto, Jim já é amigo de James. Portanto, não listamos Jim como candidato.

- "recommendFriends James weightedDist 5"

  Encontre os cinco melhores candidatos a amigos para James, usando a amizade com a distância mais curta, multiplicada pelo número total de arestas no grafo, menos o número de arestas incidentes neste nó. Esta é uma função de distância ponderada simples. Quanto menor a distância ponderada, melhor o candidato. O cálculo de distância ponderada favorece os candidatos que são amigos mais próximos dos amigos (distâncias menores) e são bem quistos.

Dado o grafo na Figura 2, o número total de arestas no grafo é 10. A distância mais curta de amizade de James para Keith é 2. Keith possui 2 arestas incidentes; portanto, a distância ponderada para Keith é $2*(10-2) =$

16. A distância de amizade mais curta entre James e Kaith é 3 (James, Larry, Kaith). Kaith possui 3 arestas incidentes; a distância de amizade ponderada é $3 * (10 - 3) = 21$. Não recomendamos Larry ou Jim para James, já que eles já são amigos de James. Da mesma forma, não recomendamos Pie, já que Pie não é alcançado por algum amigo existente de James. Quando existem vários candidatos com a mesma distância / distância ponderada, do último candidato na lista, inclua todos esses candidatos na lista, mesmo que o número de nomes retornados possa ser maior que o especificado pelo valor$< topK >$. A Figura 3 mostra o grafo quando a linha "remove Jim" é processada. O vértice "Jim" e suas arestas incidentes e as arestas de saída de Jim são todas removidas.

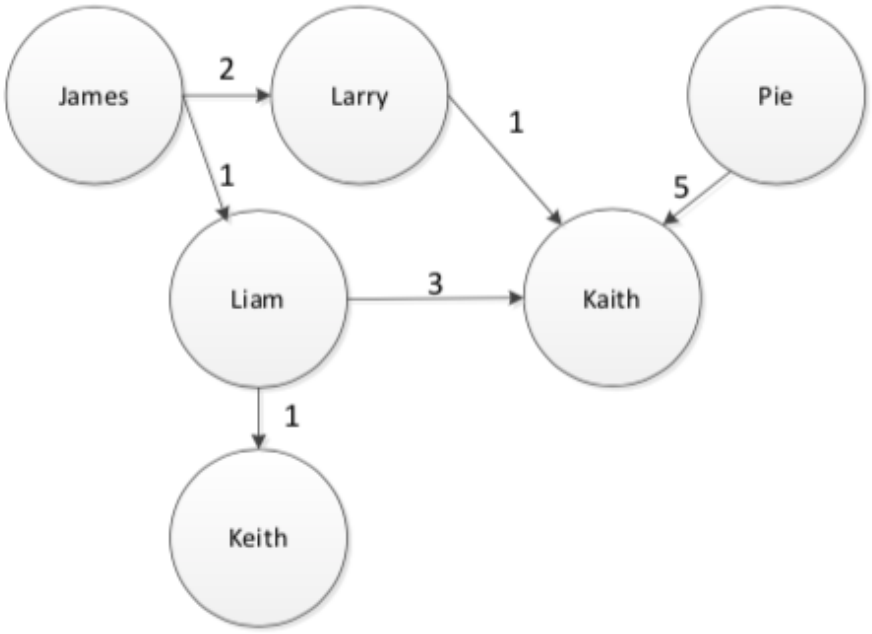

Figura 3: O grafo resultante quando é aplicado a linha "remove Jim" no grafo da Figura 2. O vértice "Jim" é removido juntamente com suas arestas incidentes e de saída do vértice. Um total de 4 arestas são removidas. O número total de arestas no grafo passa a ser 6.

4 Algoritmo Heap queue

Uma fila de prioridades, do módulo *heapq* [4], pode ajudar na implementação do algoritmo de Dijkstra [5]. Porém, use a variável de objeto *graph* fornecida, para manter vértices e objetos *Edge* corretamente e atualizar o valor de *numEdges*. É permitido usar variáveis de objeto adicionais, ou escrever métodos privados, conforme necessário. Cuidado para não para introduzir um número excessivo de variáveis.

5 Documentação

Todos os métodos públicos devem ter comentários no estilo Doxygen [6], incluindo descrições de cada parâmetro e quaisquer exceções que possam ser lançadas. Adicione a descrição sobre como os métodos foram implementados. Quaisquer outros métodos ou classes auxiliares que forem introduzidos, devem possuir comentários Doxygen *in-line*, explicando o que eles fazem. Adicione o tag @author, seguido do seu nome completo, em cada arquivo fonte Python.

6 Submissão

Coloque todos os arquivos fonte Python no projeto AD1. Lembre-se de salvar a estrutura de diretórios (por exemplo, AD1) em um arquivo zip. Entregue apenas o arquivo zip, que não deve conter qualquer arquivo .pyc. O arquivo zip deve ser nomeado Nome_Sobrenome_AD1_PIG.zip, onde Nome e Sobrenome devem ser substituídos pelo seu primeiro nome e sobrenome, respectivamente.

Não é necessário enviar código de teste unittest na AD1, e é possível compartilhar qualquer código de teste na plataforma. Você pode enviar uma versão preliminar do seu código, antes do prazo final de entrega, para ver se há algum problema no envio para a plataforma. Consideraremos apenas o último envio.

[4] `https://docs.python.org/2/library/heapq.html`
[5] `http://pt.wikipedia.org/wiki/Algoritmo_de_Dijkstra`
[6] `https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html`