Trabalho 2 – versão 1.01 – atualização publicada a 16/Maio/2024

Rede de transações monetárias entre indivíduos: visualização e estudo da rede usando um grafo

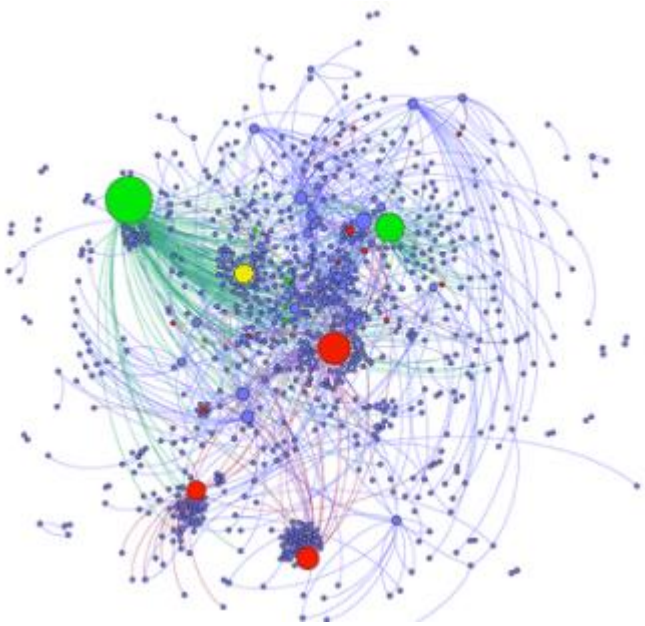

Figura 1 - Diagrama que representa uma rede de transações monetárias.

Este trabalho tem como objetivo desenvolver-se uma representação, visualização e subsequentes análises sobre um conjunto de transações monetárias (em €), entre indivíduos, usando a estrutura de grafo.

A implementação do grafo que representa a rede de transações monetárias entre indivíduos **será utilizada para fazer uma simulação** em Python **que permita estudar o "caminho" que ligue duas pessoas com o maior número de transações possível.** Para além do cálculo deste caminho, é também necessário que seja desenvolvido um método de representação visual do mesmo.

Para a realização deste trabalho, **os alunos devem seguir as indicações passadas no Guião** para preparação do 2º trabalho de EDA 23/24 (**aula de apoio de 9/Maio/2024**) para gerarem um conjuntos de dados (*datasets*). **Este ficheiro deve ter o formato ".csv", ter 10.000 (dez mil) transações** e o seguinte aspeto:

Sender	Receiver	Value
Artur	Marta	€ 100,000.00
Marta	Alice	€ 54,935.00
Rodrigo	Carolina	€ 3,187.00

A implementação do grafo que representa a rede de transações monetárias entre indivíduos **deve ser feita em Python com uma implementação da TAD Grafo** (tal como indicado no guião da aula de 9 de Maio).

Para a realização deste trabalho**, os alunos NÃO PODEM RECORRER À BIBLIOTECA NETWORKX, NEM A QUALQUER OUTRA BIBLIOTECA OPEN SOURCE DO PYTHON que substitua a implementação da TAD Grafo. A única excepção que se faz é para as tarefas de visualização** onde os alunos estão autorizados a recorrer às bibliotecas de visualização (NetworkX, Folium, Membgraph, etc...) **só podendo ser usadas para o momento da visualização.**

No desenvolvimento devem ser executadas as seguintes tarefas:

1. **Implementação de uma classe BankTransfersGraph:**

   a) deve ser implementada em Python, e ser uma implementação do TAD Grafo
   b) o seu atributo principal deve ser um grafo direcionado;
   c) deve ter as seguintes funcionalidades implementadas como métodos da classe:

      1. **__init__()**: Inicializar o grafo direcionado com todos os parâmetros necessários.
      2. **load_file(file_path)**: Ler a informação de cada linha do ficheiro de transações e adicionar ao grafo os elementos que se obtêm a partir desses dados.
      3. **add_node(id)**: Adicionar um nó (node) ao grafo.
      4. **create_node(id)**: Criar um objeto do tipo Node.
      5. **add_edge(sender_id, receiver_id, amount)**: Adicionar uma ligação (aresta/edge) ao grafo.
      6. **get_edges(node)**: Retornar todas as arestas (edges) incidentes num determinado nó.
      7. **nr_nodes()**: Devolver o número total de nós (nodes) no grafo.
      8. **nr_edges()**: Devolver o número total de arestas (edges) no grafo.
      9. **neighbours(node)**: Devolver todos os nós vizinhos de um dado nó.
      10. **avg_edges()**: Devolver o número médio de arestas (edges) dos nós do grafo.
      11. **plot_graph()**: Visualização do grafo (**tal como na Figura 1**), usando uma das seguintes bibliotecas:
          i) NetworkX (https://networkx.org/)

ii) Folium (https://python-visualization.github.io/folium/#).

iii) Memgraph (https://memgraph.com/blog/graph-visualization-in-python)

NOTA:
Para a classe **BankTransfersGraph** pode usar como ponto de partida a implementação do TAD Grafo fornecida em Graph.zip que se encontra no material relativo à semana 9 (29/Abril a 4/Maio).
No entanto tenha em consideração, que implementação usa uma representação por **listas de adjacências, mas em que a associação de cada vértice à sua lista de adjacências é implementada usando o tipo pré-definido do Python *dict*** (é, portanto, implementada com um mapa/dicionário).
Antes de avançar para uma implementação **considere que tipo de representação** de entre as várias que estudámos **é a mais adequada** para as várias tarefas deste trabalho.

2. Desenvolver métodos em Python para a classe BankTransfersGraph que permitam:

a) **visualize_node(node)**: Visualização de todas as contas que interagem com uma dada conta, usando o método e biblioteca de visualização que acharem melhor.
b) **top_transacting_accounts()**: Devolver as 3 contas com mais transações realizadas. (Não esquecer que estamos a trabalhar com um grafo direcionado)
c) **top_volume_accounts()**: Devolver as 3 contas que transacionaram maior volume em Euros. (Não esquecer que estamos a trabalhar com um grafo direcionado)
d) **most_freq_txs()**: Devolver os 3 pares de nós que tenham efectuado o maior número de transações entre si.
e) **highest_volume_txs()**: Devolver os 3 pares de nós que tenham transacionado maior volume (euros) entre si.

3. Desenvolver uma simulação em Python para calcular uma aproximação do algoritmo de Dijkstra

Contextualização:

Muitas vezes, para a análise de redes, é relevante saber como conectar nós percorrendo o caminho com o maior número de nós centrais, isto é, percorrer o caminho entre dois nós usando ligações "fortes", pois estas ligações irão ter uma menor probabilidade de deixarem de existir.

O objetivo desta parte final do trabalho é o desenvolvimento de uma simulação que, para qualquer par de nós, permita encontrar o caminho com as ligações mais fortes. Para tal, os alunos têm de desenvolver um método que siga os seguintes passos:

a) Escolher aleatoriamente dois nós do grafo (start_node, end_node)

b) Calcular qual o caminho que usa as ligações mais fortes entre essas duas pessoas (nós)
   i. Implementar em Python uma aproximação do algoritmo de Dijkstra
   Nota: Para esta implementação **poderá ser necessário** os alunos definirem outros métodos na classe **BankTransfersGraph** para além daqueles nos definidos em cima.

c) Desenvolver um método de visualização que mostra todo o grafo, os dois pontos gerados e ainda pinta o caminho entre o start_node e o end_node.

Qualidade do código Python

O desenvolvimento do código Python de respeitar as normas PEP 8.

Submissão do trabalho incluindo relatório

Este trabalho deve ser feito preferencialmente em grupos de 2 alunos. O trabalho **deve ser distribuído equitativamente** entre os 2 elementos do grupo e o relatório deve conter a identificação de ambos os alunos. Brevemente **será disponibilizado no Moodle um local para a inscrição dos grupos** de 2 elementos. Tenham em atenção que **só os alunos inscritos em grupos poderão submeter o trabalho**.

Cada grupo deve submeter o resultado do seu trabalho até à **data e hora limite de entrega: 28 de Maio de 2024, 23h59m**.

Entregas depois da hora prevista irão ter uma penalização e não será aceite qualquer trabalho que seja entregue com mais de 12h de atraso (11h59 do dia 29 de Maio).

A submissão do trabalho deve ser feita no **Moodle no local de submissão do trabalho 2**, fazendo o carregamento (upload) do ficheiro ZIP com o conteúdo descrito de seguida. Para a submissão, devem entregar um zip com:

- O relatório utilizando o *template* fornecido **em formato PDF**. **Qualquer relatório que não seja enviado em formato PDF será considerado nulo/inválido para avaliação**.
- Os seguintes ficheiros com o código Python desenvolvido, em formato (.py) pronto a ser utilizado/testado:
    - **graph.py**: Ficheiro que deve conter a implementação das classes **BankTransfersGraph** e **Node.**
    - **__main__.py**: Ficheiro que deve conter todos os processos que desenvolvam para gerar o grafo e efectuar as análises necessárias.
    - **source_file.py**: Ficheiro que deve ter o código que foi desenvolvido para gerar o .csv que usaram para o trabalho.
- O ficheiro .csv que criaram e usaram para o trabalho.

A partilha de código em trabalhos de grupos diferentes será devidamente penalizada. Serão usados os meios habituais de verificação de plágio. O código entregue deve ser integralmente da autoria dos membros do grupo.

Cronograma para o desenvolvimento do trabalho:

Até à aula de 16 de Maio: Já devem ter o vosso ficheiro .csv pronto e implementação do grafo feita.

Até 20 de Maio: Deverão ter todas as análises realizadas e o progresso no relatório deve ser feito até ao ponto 6.

Até à aula de 23 de Maio: Deverão desenvolver a implementação da aproximação ao algoritmo de Dijkstra.

Até ao dia 28 de Maio: Finalizar as visualizações do caminho no plot e ter os pontos 6 e 7 do relatório finalizados.