

1.

Descrição breve de quais foram as principais dificuldades que o grupo teve ao longo do desenvolvimento do trabalho e como é que as resolveram.

(Limite de 500 palavras nesta descrição.)

Durante o desenvolvimento deste trabalho, as principais dificuldades que o grupo enfrentou foram:

- ***Decidir na escolha de uma estrutura de dados que fosse a mais adequada para armazenar a informação do grafo e responder às tarefas requeridas ao longo do trabalho:***

Optámos por usar um dicionário onde cada chave irá conter a chave dos nós do grafo e o valor correspondente a esses nós serão novamente dicionários com as seguintes chaves e valores (que inicialmente no momento de criação do nó estarão vazios):

```
{  
    "connections": [],  
    "sent_to": {},  
    "received_from": {},  
    "number_of_transactions_sent_to": {},  
    "number_of_transactions_received_from": {},  
}
```

Desta forma a manipulação da estrutura do grafo e a implementação das tarefas pedidas ficaram mais acessíveis e compreensíveis de codificar.

- ***Realizar a função de visualização do grafo, através da biblioteca networkx:***

Esta etapa também apresentou desafios. Um dos maiores entraves foi a forma como a biblioteca networkx estava a mostrar o grafo com as 10 mil transações, com todos os nós desenhados em cima uns dos outros de forma que não favorecia uma leitura e interpretação claras do grafo. Pesquisando sobre soluções ao problema considerámos outras bibliotecas como o Jaal (<https://mohitmayank.com/jaal/>). E chegámos a implementar as visualizações com esta biblioteca. Porém, foi nos dito e aconselhado em aula de apoio que esse problema era normal e que não haveria constrangimentos em usar a biblioteca networkx.

- ***Implementação dos métodos `top_transacting_accounts()` e `most_freq_txs()`:***

O método **`top_transacting_accounts()`** revelou-se mais difícil do que inicialmente tínhamos previsto, uma vez que conceptualmente após leitura do enunciado, pensámos que apenas seria necessário verificar quais dos nós teriam mais **`out_degree`**. Porém, tendo em consideração que poderão existir várias transações de um determinado sender para um determinado receiver, e tendo em conta que nestas situações os valores transacionados são somados (opção escolhida por nós), verificamos então que o **`out_degree`** ou grau de saída (isto é, o nº de setas que sai) de um determinado nó não necessariamente implica que esse nó seja o nó com mais transações realizadas.

Para resolver este entrave, tivemos de guardar esta informação no dicionário que contém a estrutura do grafo. Através da chave **`number_of_transactions_sent_to`** foi possível verificar o nº total de transações realizadas de cada um dos nós. E daí ordenar todos eles e extrair o top 3.

Em relação ao método **`most_freq_txs()`**, o desafio foi encontrar a lógica necessária para descobrir os valores certos.

Porém, uma vez mais com a adição das chaves "number_of_transactions_sent_to" e "number_of_transactions_received_from" conseguimos encontrar a lógica que nos permitiu somar os nº de transações relacionadas entre duas pessoas.

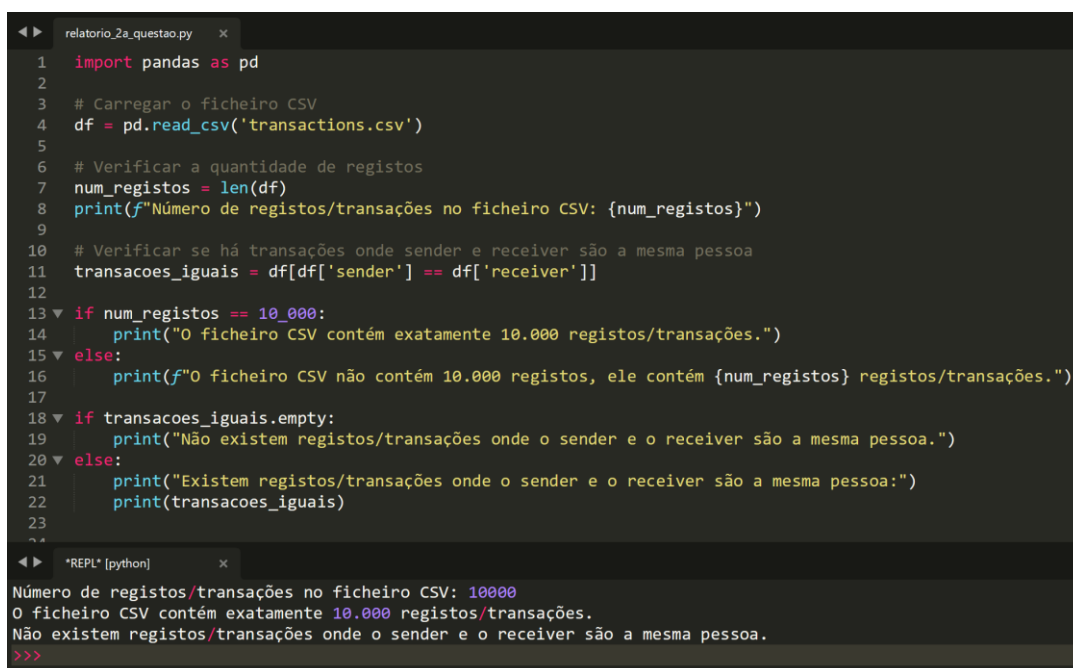
- **Implementação do algoritmo de Dijkstra adaptado ao contexto pedido:**

Esta foi outra dificuldade principal. A modificação da lógica do algoritmo para considerar pesos máximos (opção escolhida por nós) apresentou desafios. A solução encontrada foi implementar uma versão adaptada utilizando uma fila para gerir a procura do caminho mais pesado, explorando todos os caminhos potenciais para encontrar aquele com o peso máximo, de forma cumulativa.

2.

Mostrem/Provem **através de código Python ou com funções do Excel** (para fins de avaliação, **irá ser valorizado quem fizer com código**) **que o vosso ficheiro .csv tem 10.000 (dez mil) registos/transações** e que **não existe nenhuma transação em que o sender e o receiver são a mesma pessoa**.

(Nesta parte devem recorrer a prints do código que desenvolveram e do seu output (ou das funções do Excel) para demonstrar os resultados. Não queremos que expliquem cada linha de código desenvolvida, basta explicarem qual foi o processo que fizeram para chegar a estas confirmações.)



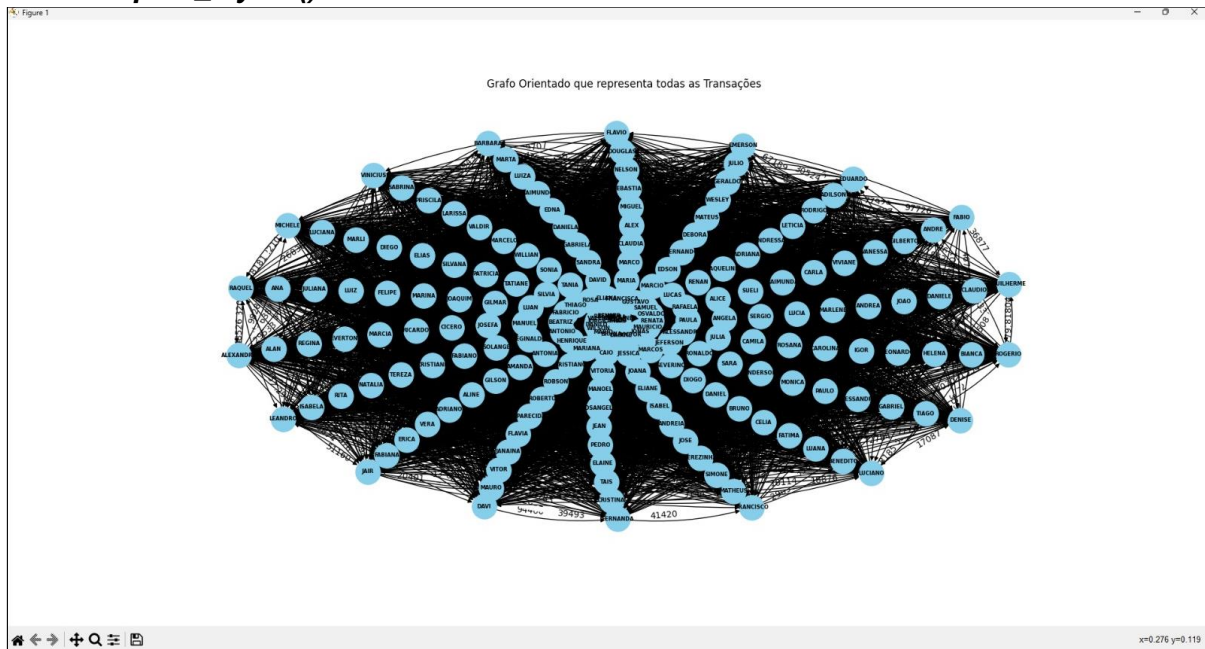
```
relatorio_2a_questao.py x
1 import pandas as pd
2
3 # Carregar o ficheiro CSV
4 df = pd.read_csv('transactions.csv')
5
6 # Verificar a quantidade de registos
7 num_registos = len(df)
8 print(f"Número de registos/transações no ficheiro CSV: {num_registos}")
9
10 # Verificar se há transações onde sender e receiver são a mesma pessoa
11 transacoes_iguais = df[df['sender'] == df['receiver']]
12
13 if num_registos == 10_000:
14     print("O ficheiro CSV contém exatamente 10.000 registos/transações.")
15 else:
16     print(f"O ficheiro CSV não contém 10.000 registos, ele contém {num_registos} registos/transações.")
17
18 if transacoes_iguais.empty:
19     print("Não existem registos/transações onde o sender e o receiver são a mesma pessoa.")
20 else:
21     print("Existem registos/transações onde o sender e o receiver são a mesma pessoa:")
22     print(transacoes_iguais)
23
24
25 *REPL* [python] x
Número de registos/transações no ficheiro CSV: 10000
O ficheiro CSV contém exatamente 10.000 registos/transações.
Não existem registos/transações onde o sender e o receiver são a mesma pessoa.
>>>
```

Este pequeno script em Python faz todas as verificações requeridas neste ponto do relatório. Lê o ficheiro CSV de transações e armazena todos os registos num Dataframe, de seguida conta o número total de registos, onde depois na linha 13 verifica se o número de registos é 10.000. Na linha 11 identifica transações onde o remetente e o destinatário são a mesma pessoa, e depois na linha 18 verifica e informa os resultados dessas verificações.

3.

Coloquem o plot do vosso grafo.

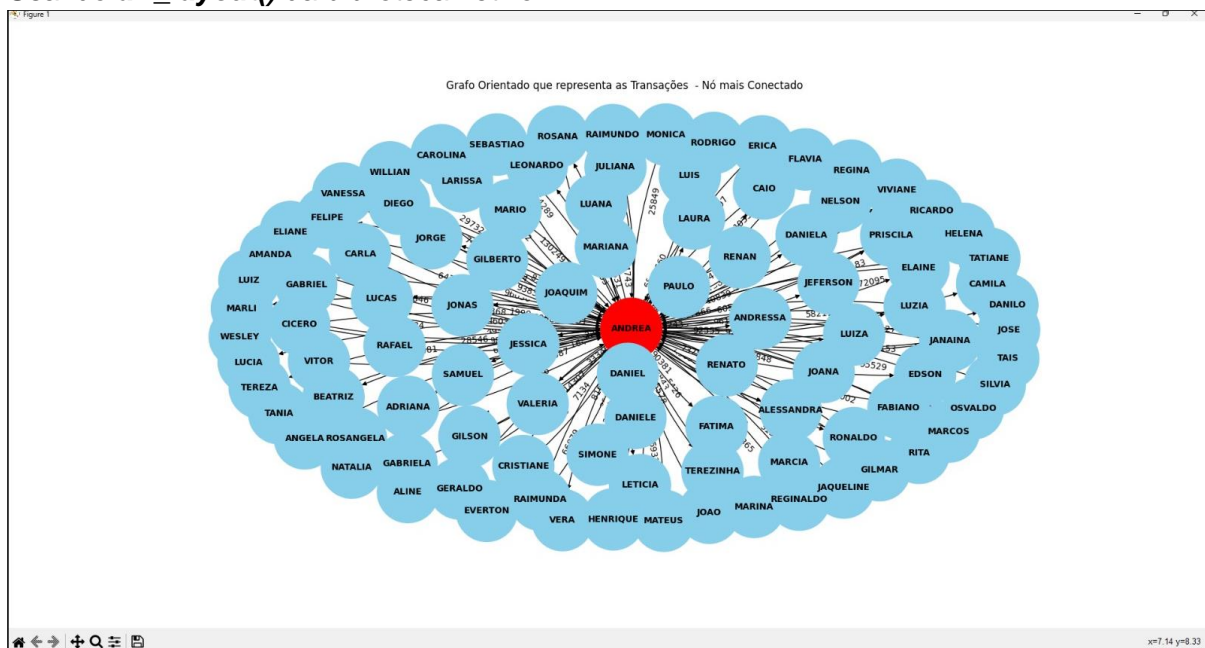
Usando *spiral_layout()* da biblioteca networkx:



4.

Coloquem o plot do grafo para um dos n s.

Usando *arf_layout()* da biblioteca networkx:



5.

Mostrem o resultado das vossas análises e das estatísticas que foram pedidas.

(Esta parte não deverá ter mais do que 250 palavras)

Análises e Estatísticas

Número total de nós no grafo	200
Número total de arestas no grafo	8840
Número médio de arestas dos nós do grafo	44.2
Nó mais conectado	ANDREA
Nó menos conectado	BIANCA
Transação com maior volume (euros)	('ELAINE', 'ANTONIO', 253503)
Transação com menor volume (euros)	('DIOGO', 'RENAN', 4)
3 contas com mais transações	[(ANDREA, 72), (SIMONE, 70), (MAURO, 67)]
3 contas que transacionaram maior volume em euros	[(ANDREA, 3836811), (ROSANGELA, 3544608), (LARISSA, 3493456)]
3 pares de nós que tenham efetuado o maior número de transações entre si	['CLAUDIA', 'RAQUEL', 5], ['ANDREA', 'GILMAR', 4], ['TANIA', 'ADILSON', 4]
3 pares de nós que tenham transacionado maior volume (euros) entre si	['ANDRESSA', 'MARINA', 286158], ['VERA', 'ERICA', 279445], ['ADRIANA', 'RENAN', 271778]

6.

Na 3ª parte do trabalho é pedido para implementarem uma aproximação do algoritmo de Dijkstra para calcular qual o caminho que usa as ligações mais fortes entre duas pessoas. Indiquem que conceito (frequência de transações ou volume de transações) usaram para definir uma ligação forte entre duas pessoas.

Expliquem qual é a principal diferença entre o algoritmo de Dijkstra e a implementação que tem de ser feita na parte final do trabalho.

(Nesta parte não devem usar mais do que 250 palavras)

No nosso trabalho, decidimos usar o volume de transações para definir uma ligação forte. O termo “mais forte” pode ser interpretado de várias maneiras, mas no contexto que estamos a trabalhar, acreditamos que o caminho “mais forte” se refere ao caminho com o maior valor total transacionado ao longo das arestas.

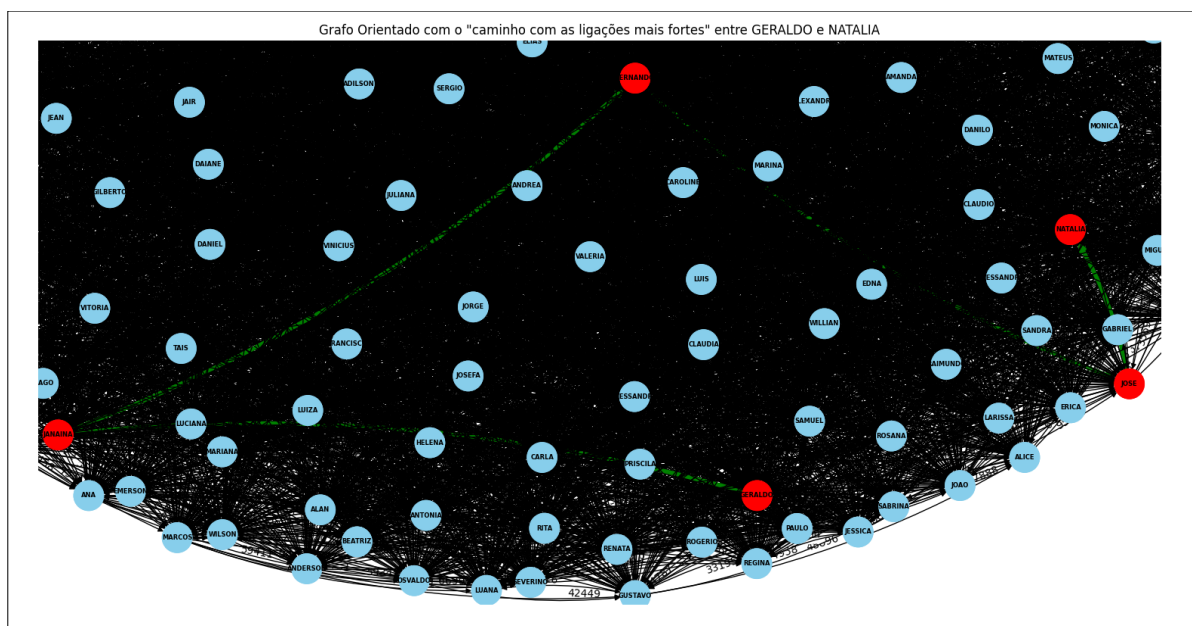
Desta forma, ao usarmos este critério conseguimos refletir melhor a intensidade e importância das interações financeiras entre os nós do grafo. Em cenários reais, a quantidade de dinheiro movimentada pode ser um indicador mais significativo do que a frequência das transações. E assim permite-nos uma análise mais precisa dos caminhos economicamente mais relevantes, alinhando-se com o objetivo de encontrar as conexões mais influentes no grafo. A principal diferença entre o algoritmo de Dijkstra tradicional e a implementação que tem de ser feita neste ponto, é que o Dijkstra convencional encontra o caminho “mais leve/barato” entre dois nós enquanto que a nossa implementação calcula precisamente o oposto - o caminho “mais pesado/carro”.

Esta abordagem permitiu-nos calcular o caminho que utiliza as liga es mais fortes, conforme definido pelo volume de transa es, garantindo que o caminho encontrado maximiza o valor total transacionado entre as duas pessoas.

7.

Mostrem que o vosso c digo que encontra o caminho funciona com o recurso aos plots. Devem descrever quais foram as altera es que fizeram ao algoritmo de Dijkstra **sem uma explica o desnecess ria de cada linha de c digo que implementaram**.
(Nesta parte, n o devem usar mais do que 250 palavras)

O caminho mais pesado de GERALDO para NATALIA  :
['GERALDO', 'JANAINA', 'FERNANDO', 'JOSE', 'NATALIA'] com peso total de 349662



O nosso m todo **dijkstra_max_path(self, graph, start, end)** retorna um tuplo: (**max_weight**, **max_path**) onde **max_weight**   o peso m ximo encontrado e o **max_path**   uma lista que possui os n s que corresponder o ao caminho direcionado entre o n o inicial e o n o final. As principais altera es foram: substituir a fila de prioridade por uma lista simples para armazenar os n s a serem explorados, mudar o crit rio de sele o de n s onde   tida em conta a ordem com que foram adicionados   fila, atualizar o caminho para focar no maior peso acumulado, e adicionar uma verifica o expl cita para evitar ciclos, garantindo que o caminho encontrado n o repetisse n s, o que   essencial n o s o em termos de efici ncia, mas t m-b m para a precis o do resultado encontrado.

Neste caso o n o inicial foi **GERALDO**, o n o final foi **NATALIA**, o **max_weight** foi **349662** e o **max_path** retornado foi:

['GERALDO', 'JANAINA', 'FERNANDO', 'JOSE', 'NATALIA']

E para desenharmos o grafo com este caminho a verde, como na imagem, cri mos um novo m todo dentro da classe chamado **draw_graph_with_path(self, path)** que receber  como argumento o **path** encontrado pela fun o anteriormente referida.