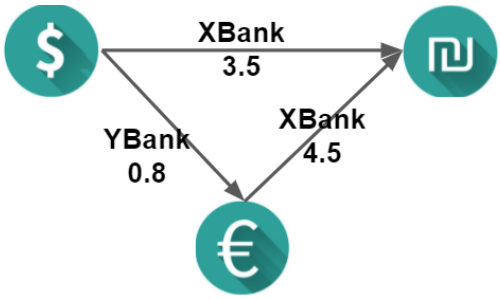
Given the differences in exchange rates of different banks in different currencies, it is possible to find an 'exchange route' which will give the best exchange rate from one currency to another.
For example, the cheapest Bank (XBank) the exchange rate from 1 Dollar to Shekel is 3.5 This means that for 1 Dollar you can get 3.5 Shekels.
However, you can get a better deal. The best rate for exchanging Dollar to Euro is 0.8 (in a bank called YBank) and the best rate from euro to Shekel is 4.5 (in XBank). If you exchange your 1 Dollar to 0.8 Euro and then exchange 0.8 Euro to Shekels you will end up with 3.6 Shekels which is better than the 3.5 you would have gotten by directly exchanging Dollar to Shekel.
This can be viewed as a path in a weighted graph where the final exchange rate is the multiplication of all the weights in the path:



A. Algorithm implementation

You are in charge of developing the backend of the app - XchangeR8

The app holds the current exchange rate graph.

The app receive queries with the wanted currency exchange as a dict:
{'from': 'Dollar','to': 'Shekel','amount':1527}

The app returns the best exchange route as a list of dicts:
[{'from': 'Dollar','to': 'Euro','bank': 'Ybank'},
{'from': 'Euro','to': 'Shekel','bank': 'Xbank'}]
The app receive updates when some bank changes its rate for a specific transaction and updates the graph accordingly:
[{'from': 'Dollar','to': 'Euro','bank': 'Ybank','rate': 0.9},
{'from': 'Rupee','to': 'Shekel','bank': 'Zbank','rate': 0.041}]

- For each pair of 'from',' to' currencies, the graph itself only needs to hold the data from the bank with the **current** minimum exchange rate from these currencies. However, you need to keep the data from all banks because exchange rate might change
- Use networkx package in python for building a graph and using graph algorithms

- The functions for shortest paths in networkx find the minimum route when the route length is the sum of the weights on the edges. We need the minimum multiplication product of the weights. Also, we are actually looking for the heaviest path because we want to get as much money in the end. Apply -log on the weights before, due to these reason
- When the app starts, it will read a csv file containing the initial rates of exchanging currencies from many banks. Use pandas.read_csv() to read the csv file and build the initial graph from the dataframe you have
- If there is no rout the app will return an empty string

Example input dicts (use in this order):
{'from': 'Yen','to': 'Shekel','amount':1527}
[{'from': 'Euro','to': 'Dollar','bank': 'WBank','rate': 0.35}]
{'from': 'Yen','to': 'Shekel','amount':1527}
{'from': 'Shekel','to': 'Ruble','amount':350}
[{'from': 'Shekel','to': 'Ruble','bank': 'WBank','rate': 0.14}]
{'from': 'Shekel','to': 'Ruble','amount':350}
[{'from': 'Dollar','to': 'Ruble','bank': 'XBank','rate': 0.28}]
{'from': 'Shekel','to': 'Ruble','amount':350}
[{'from': 'Dollar','to': 'Ruble','bank': 'YBank','rate': 0.5}]
{'from': 'Shekel','to': 'Ruble','amount':350}

B. Architecture

This is a more open task to see how you plan architecture so you should make your own assumption on any unknown issue. Try to make assumptions that are relevant to the real world.
The app will receive many requests and we want to give higher priority to large transactions, i.e., transactions with higher amounts (regardless of the currency).
While we want to give higher priority to large transactions, we do not want to starve the small transactions. Eventually we want to handle each transaction in no longer than 1 second.
The exchange calculator that you wrote in A will run on many machines. The amount of machines will be determined with an auto scale rule so you can assume that you will have enough machines when you have many requests.
Design and write a request handler that takes into account the rate of requests and the amount of transaction. In this task write **only the skeleton** (the classes, function signatures and class attributes)

- Give priority to larger transactions
- All requests must be answered in maximum 1 second

- Think about costs (we do not want to have many machines running when we can use less)
- Because the handler is responsible for the instances of the exchange route calculators, the handler will need to hold the current exchange rates of the banks and init/updated the instances when relevant
- You can assume that processing a request will never take longer than 10 ms

- When you write the code pay attention to design and dividing to functions/classes/files

Good luck!