Daily Coding Problem #32

Problem

This problem was asked by Jane Street.

Suppose you are given a table of currency exchange rates, represented as a 2D array. Determine whether there is a possible arbitrage: that is, whether there is some sequence of trades you can make, starting with some amount A of any currency, so that you can end up with some amount greater than A of that currency.

There are no transaction costs and you can trade fractional quantities.

Solution

In this question, we can model the currencies and the exchange rates as a graph, where the nodes are the currencies and the edges are the exchange rates between each commodity. Since our table is complete, the graph is also complete. Then, to solve this problem, we need to find a cycle whose edge weights product is greater than 1.

This seems hard to do faster than brute force, so let's try to reduce it down to a problem we already know we can solve faster than brute force. Hint: `log(a * b) = log(a) + log(b)`. So if we take the negative log of the edge weights, the problem of finding a cumulative product that's greater than 1 turns into the problem of finding a negative sum cycle.

The Bellman-Ford algorithm can detect negative cycles. So if we run Bellman-Ford on our graph and discover one, then that means its corresponding edge weights multiply out to more than 1, and thus we can perform an arbitrage.

As a refresher, the Bellman-Ford algorithm is commonly used to find the shortest path between a source vertex and each of the other vertices. If the graph contains a negative cycle, however, it can detect it and throw an exception (or, in our case, return true). The main idea of Bellman-Ford is this:

Since the longest path in any graph has at most |V| - 1 edges, if we take all the direct edges from our source node, then we have all the one-edged shortest paths; once we take edges from there, we have all the two-edged shortest paths; all the way until |V| - 1 sized paths.

If, after |V| - 1 iterations of this, we can still find a smaller path, then there must be a negative cycle in the graph.

```
from math import log

def arbitrage(table):
    transformed_graph = [[-log(edge) for edge in row] for row in graph]

    # Pick any source vertex -- we can run Bellman-Ford from any vertex and
    # get the right result
    source = 0
    n = len(transformed_graph)
    min_dist = [float('inf')] * n

    min_dist[source] = 0
```

```
    # Relax edges |V - 1| times
    for i in range(n - 1):
        for v in range(n):
            for w in range(n):
                if min_dist[w] > min_dist[v] + transformed_graph[v][w]:
                    min_dist[w] = min_dist[v] + transformed_graph[v][w]

    # If we can still relax edges, then we have a negative cycle
    for v in range(n):
        for w in range(n):
            if min_dist[w] > min_dist[v] + transformed_graph[v][w]:
                return True

    return False
```

Because of the triply-nested foor loop, this runs in O(N^3) time.