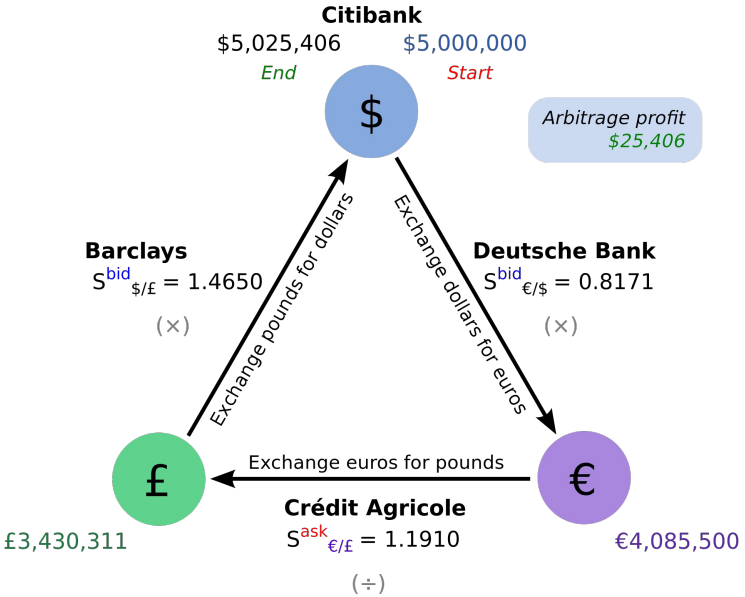
Arbitrage Opportunity

A Program by Alexis Dalforno

What is it?

There are multiple types, but the type that I have used for this program is the risk-free Forex trading. There tend to be pricing discrepancies when exchanging foreign currency. Though the gain is small (seen to the right), it is a perfect problem to use to find a shortest path. In this project I find through which exchanges you will make the fastest profit depending on the original currency.



Challenges

```
// INSTANCE VARIABLES
private Map<Vertex, Map<Vertex, Edge>> graph; // the cheaters way of making a multi-key Map object :)
```

- Making the project actually useful in modern time.
- Finding a way to make a graph that has a weight between every single vertex in both directions.
- Getting specific information from my graph object for Dijkstra's algorithm.
- Actually making Dijkstra's algorithm work for what I needed.

```
// FULL CONSTRUCTOR
public Graph(List<Vertex> vertices, List<Edge> edges)
{
    int index1 = 0;
    int index2 = 0;

    // sets the lists of vertices and edges
    this.vertices = vertices;
    this.edges = edges;

    // gets the number of vertices in the graph
    this.numV = this.vertices.size();

    this.graph = new HashMap<Vertex, Map<Vertex, Edge>>();

    // sets all the vertices
    for(int i = 0; i < numV; i++)
    {
        this.graph.put(this.vertices.get(i), new HashMap<Vertex, Edge>());
    }

    // sets all the possible edges to be made between the vertices
    for(int i = 0; i < numV * numV; i++)
    {
        this.graph.get(this.vertices.get(index1)).put(this.vertices.get(index2), null);

        index2++;
```

Algorithm

```
/**
 * This method finds the shortest way to make a profit through foreign currency exchange.
 * To make the method the most accurate and efficient, the method will only look at three
 * transactions, the third transaction being the transaction back to original currency.
 * This method is also specific to the program at hand because instead of setting any empty edges
 * (technically there are none), it will set any edge connected to itself as weight 1.0 because
 * any vertex converted to itself is one dollar. (ex. 5 us dollars is 5 us dollars)
 *
 * @param graph The weighted graph to be searched
 * @param pred The output array to contain the predecessors in the shortest path
 * @param dist Output array to contain the distance in the shortest path
 */
public static double dijkstrasAlgorithm(Graph graph, Map<Vertex, Vertex> pred, Map<Vertex, Double> dist)
```

In the end I chose Dijkstra's Algorithm for my project because it did work for what I needed. The dist Map helped me keep track of the weights while the pred Map let me view the fastest path. Each time it found the first fastest path it would remove the predecessor from the list and run through again until two additional transactions were made.