Student Name: Ali Albayrak 1/11/2020
Student Number: 19978497

DSA Assignment 1 – Crypto Currency Report

Information for Use

1.Introduction

This program is coded for a solution to analyse crypto-currency trading data. Program has two mode to run. First of them is interactive mode. In this mode; program can import the data by using JSON. And prints some selection options for the user. Then user can select the desired option and see the relevant information about any crypto-currency or trading paths between them. Also it is possible to see overall information about assets or paths. User can save the entire graph as a serialized file and load it back any time. Second mode the program has is; report mode. In this mode program displays general information about assets and paths; like top ten volume paths etc..

2.Installation

To run the program user needs to use -cp <json package> flag. Because the program uses JSON package, we need to implement it this way. Full command for usage of program with JSON;
Compile : -javac -cp .:<Json package> cryptoGraph.java
Run: -java -cp .:<Json package> cryptoGraph

Also, the program uses lots of recursion while calculating the possible paths between assets. For this reason when we are saving/reading serialised file, program need more memory and memory error occurs. To solve this we use the -Xss4m flag before we run the code. Full terminal command to use:
Run: -java -Xss4m -cp .:<Json package> cryptoGraph <run argument>

User also needs to use argument for the selection of the mode. Program has two different modes, interactive mode and report mode.
Interactive mode argument is -i .
Report mode argument is -r.

Terminal command for interactive mode:
java -Xss4m -cp .:<Json package> cryptoGraph -i

Terminal command for report mode:
java -Xss4m -cp .:<Json package> cryptoGraph -r <asset_file> <trade_file>

<asset_file> : File that contains asset information such as, baseAsset , quoteAsset and trade path status. This is the file supplied to us as Exchange info.
<trade_file>: File that contains trade information such as volume,weighted price, count vs. This is the file supplied to us as 24hr trade information.

3.Terminology/Abbreviations

Asset: this implies a single crypto-currency such as ETH,BTC etc..
Trade / trade path: this implies a single transfer path from one asset to another such as ETHBTC.
Base asset: The asset that is being sold to get another asset. Also start point of a path. For example, ETHBTC; Eth is the base asset.
Quote asset: The asset that is being bought, end point of a path. For example, ETHBTC; BTC is the quote asset.
Vertex: Graph points that hold asset information.
Edge: Graph connections between vertices, holds trade information.

4. Walkthrough

Interactive Mode:

Start the program with this command after compiling:
java -Xss4m -cp .:<json package> cryptoGraph -i

```
 Welcome to cryptoGraph Interactive!


Please select an option:
1.Load Data
2.Display an asset
3.Display a trade path
4.Find potential trade paths
5.Asset filter
6.Asset Overview
7.Trade Overview
8.Save Data
9.Exit
```

We will see some options to select. To use all of the program specifications; first we need to import asset and trade data.
To do this select the option 1.

```
1.Load Asset data
2.Load Trade data
3.Load from Serialized data
```

And then first load the asset data. Program hardcoded the filename. So for the interactive mode, user doesn't need to enter a filename.

```
1.Load Asset data
2.Load Trade data
3.Load from Serialized data
1
1091 trade paths loaded.
```

After we loaded asset data, for full functionality(optional) we need to load trade data.

```
1.Load Asset data
2.Load Trade data
3.Load from Serialized data
2
Trade data loaded.
```

Now we are ready to use all of functionalities of the program.

-To display a particular asset information, simply select the “2.Display an asset” and enter the asset label which is crypto currency name. For demonstration, I will use “ETH”.

```
Enter a asset label:
ETH
Asset Label:ETH Direct trade path to: 149 Direct trade path from: 16 different currencies
```

So, this option displays;
asset label
number of direct trade paths to that asset(paths that include ETH as quote)
number of direct trade paths from that asset(paths that include ETH as base)

-To display a particular trade path, select the “3.Display a trade path” and enter the trade label when asked. For demonstration, I will use the “ETHBTC”.

```
Enter a trade label:
ETHBTC
Trade label: ETHBTC
trade from : ETH
trade to : BTC
Avg. Weight: 0.03273027
Volume: 299478.446
Status: TRADING
Total count so far : 121437
```

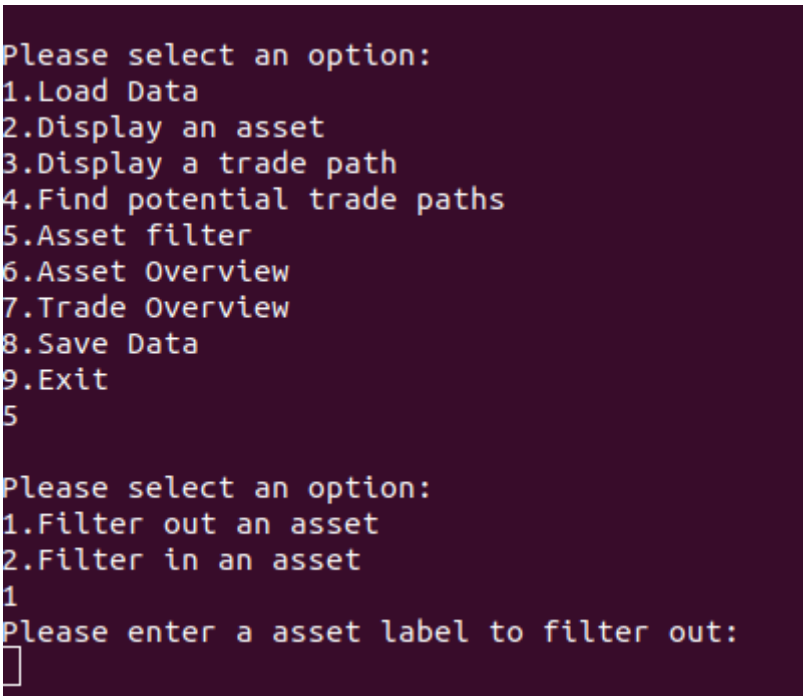
So, this option displays;
Trade label and base and quote labels.
Avg.Weight is the average price occurred in last 24 hrs.
Volume of the trade, and current status.

-To display potential trade paths between two asset, simply select the option 4.
First enter the base asset, and after enter the quote asset. For demonstration, I will use ETHEUR.

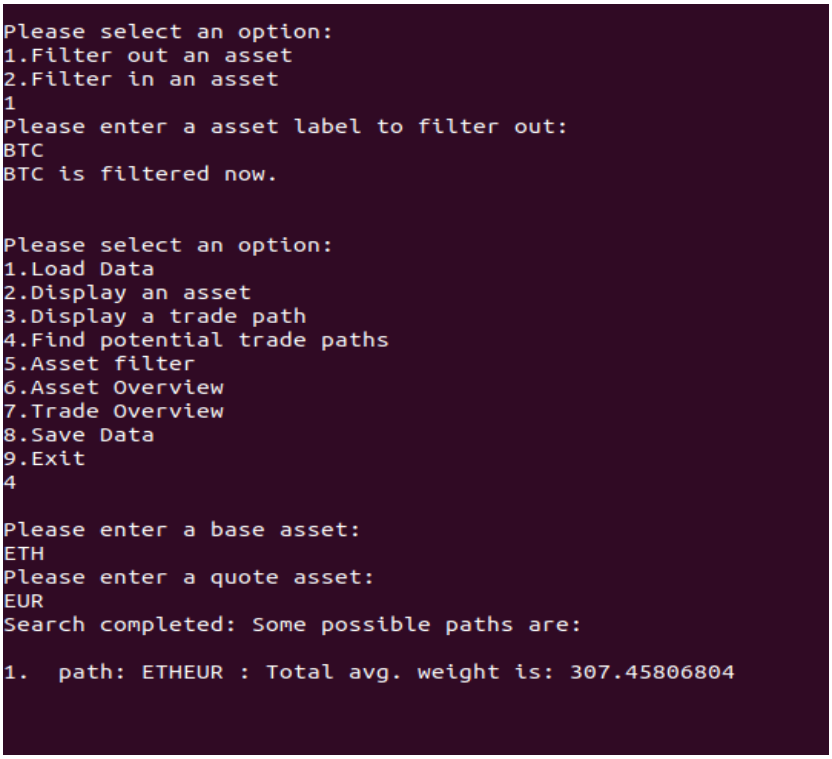
```
Please enter a base asset:
ETH
Please enter a quote asset:
EUR
Search completed: Some possible paths are:

1.  path: ETHEUR : Total avg. weight is: 307.45806804


2.  path: ETHBTC >> BTCEUR : Total avg. weight is: 307.68812750453486
```

Program found 2 different possible paths those statuses are active. And their avg.price as well.
Note: This functionality does not working properly due to design of its algorithm. The algorithm may not be able to find all possible paths. It can find most of them, but may fail at finding some of 2-3 step possible paths that goes through same paths as previous trade path.
Note 2: This function only return path that are active. If there is a path its status “BREAK”, it wont return this path as it is impossible to trade through it.

-To filter out any particular asset, select the option 5. And select the option “1.filter out an asset” from the submenu. After that enter the asset label you want to filter out.



I chose the BTC for the demonstration. So we can see that second path from ETHEUR will not be available anymore(see the example above for possible paths).



- To include the filtered out asset again, select the 5.Asset filter from main menu, and then select the 2.Filter in an asset from submenu. And after enter the asset you wish to include in the graph again.

```
Please select an option:
1.Load Data
2.Display an asset
3.Display a trade path
4.Find potential trade paths
5.Asset filter
6.Asset Overview
7.Trade Overview
8.Save Data
9.Exit
5

Please select an option:
1.Filter out an asset
2.Filter in an asset
2
Please enter an asset label to filter back in:
BTC
BTC is back in the graph now.
```

-To overview all of assets, select the 6.Asset overview from the main menu.

```
Total Asset Number: 315

The Asset has the most direct trade path from : BTC.
BTC  can be directly traded for : 254 different crypto currencies.

The Asset has the most direct trade path to : BNB.
BNB can be directly traded from: 19 different crypto currencies.
```

This will display the total asset number. And the asset has the most direct trade paths as Base asset. Also displays the asset has the most direct trade paths as Quote asset.

-To overview all of trade paths, select the 7. Trade overview from the main menu.
This will display some top-ten trade paths.

```
Total Direct Trade Path Number :1091

Top-ten Trade Paths by Volume:
1.BTTUSDT   Volume: 3.0919119885E10
2.WINUSDT   Volume: 1.2022472935E10
3.NPXSUSDT   Volume: 6.819657047E9
4.WINTRX   Volume: 3.3535063665E9
5.BTTTRX   Volume: 1.9849255995E9
6.VETUSDT   Volume: 1.864392125E9
7.NPXSETH   Volume: 1.823267491E9
8.TRXUSDT   Volume: 1.6852966641E9
9.MFTUSDT   Volume: 1.658499476E9
10.OSTBTC   Volume: 1.417318393E9

Top-ten Trade Paths by Count:
1.BTCUSDT   Count: 750364
2.ETHUSDT   Count: 286988
3.YFIIUSDT   Count: 240664
4.YFIUSDT   Count: 213665
5.LINKUSDT   Count: 200777
6.BNBUSDT   Count: 195093
7.BTCBUSD   Count: 163367
8.UNIUSDT   Count: 146072
9.BUSDUSDT   Count: 133945
10.ETHBTC   Count: 121437

Top-ten Trade Paths by Avg.Weight:
1.BTCBIDR   Avg.Weight: 1.6460237925E8
2.BTCIDRT   Avg.Weight: 1.6320723708E8
3.BTCBKRW   Avg.Weight: 1.282409521086428E7
4.ETHBIDR   Avg.Weight: 5375707.2
5.BTCNGN   Avg.Weight: 5137466.834558
6.BTCRUB   Avg.Weight: 858991.80632272
7.BNBBIDR   Avg.Weight: 418330.02
8.ETHBKRW   Avg.Weight: 414801.29325028
9.BNBIDRT   Avg.Weight: 413548.34
10.BTCUAH   Avg.Weight: 315732.88932617
```

-To save the entire graph as serialised data, please select the 8. Save Data from the main menu. That will ask you to enter a filename to save. Enter the filename. Then program will serialise the graph and save it as a serialised data. The file will be located at the programs folder location.

```
Please select an option:
1.Load Data
2.Display an asset
3.Display a trade path
4.Find potential trade paths
5.Asset filter
6.Asset Overview
7.Trade Overview
8.Save Data
9.Exit
8

Please enter a file name to save the graph:
testserial
```

-To Exit the program select the option 9.Exit from the main menu.

Also, to import data from serialised data, select the 1.Load Data from sub-menu and select the 3.Load from serialized data from the submenu. Enter the file name, and the graph will be loaded into your program.
Note: Loading form serialized file makes an assumption that both trade and asset data are present and loaded in. This may be wrong! But I have to make that assumption for full functionality.

Report Mode:
To run the program with report mode, use the following command;
java -cp .:json-20200518.jar cryptoGraph -r assetdata.json tradedata.json (or any other relevant json file with same format)

```
Total Asset Number: 315

The Asset has the most direct trade path from : BTC.
BTC  can be directly traded for : 254 different crypto

The Asset has the most direct trade path to : BNB.
BNB can be directly traded from: 19 different crypto cu


Total Direct Trade Path Number :1091

Top-ten Trade Paths by Volume:
1.BTTUSDT   Volume: 3.0919119885E10
2.WINUSDT   Volume: 1.2022472935E10
3.NPXSUSDT   Volume: 6.819657047E9
4.WINTRX   Volume: 3.3535063665E9
5.BTTTRX   Volume: 1.9849255995E9
6.VETUSDT   Volume: 1.864392125E9
7.NPXSETH   Volume: 1.823267491E9
8.TRXUSDT   Volume: 1.6852966641E9
9.MFTUSDT   Volume: 1.658499476E9
10.OSTBTC   Volume: 1.417318393E9

Top-ten Trade Paths by Count:
1.BTCUSDT   Count: 750364
2.ETHUSDT   Count: 286988
3.YFIIUSDT   Count: 240664
4.YFIUSDT   Count: 213665
5.LINKUSDT   Count: 200777
6.BNBUSDT   Count: 195093
7.BTCBUSD   Count: 163367
8.UNIUSDT   Count: 146072
9.BUSDUSDT   Count: 133945
10.ETHBTC   Count: 121437

Top-ten Trade Paths by Avg.Weight:
1.BTCBIDR   Avg.Weight: 1.6460237925E8
2.BTCIDRT   Avg.Weight: 1.6320723708E8
3.BTCBKRW   Avg.Weight: 1.282409521086428E7
4.ETHBIDR   Avg.Weight: 5375707.2
5.BTCNGN   Avg.Weight: 5137466.834558
6.BTCRUB   Avg.Weight: 858991.80632272
7.BNBBIDR   Avg.Weight: 418330.02
8.ETHBKRW   Avg.Weight: 414801.29325028
9.BNBIDRT   Avg.Weight: 413548.34
10.BTCUAH   Avg.Weight: 315732.88932617
```

4.Future Work

For the finding possible paths section, my search algorithm is not working properly. It can not be able to find all of possible paths. I would try to optimise it in future. Also, additionally instead of importing the trade and asset data manually, we can implement a design that takes the live data automatically from the website. Also, additional data can be added into the vertices. This data is given to us, but I did not have enough time to implement it in my code.
Moreover, we can add additional specification that calculates the most profitable path from one currency to another. Furthermore, we do not need to stick with only one website, we can search through all of the trading websites, and find the best possible profitable paths.
At least but not last, display can be much more user friendly.

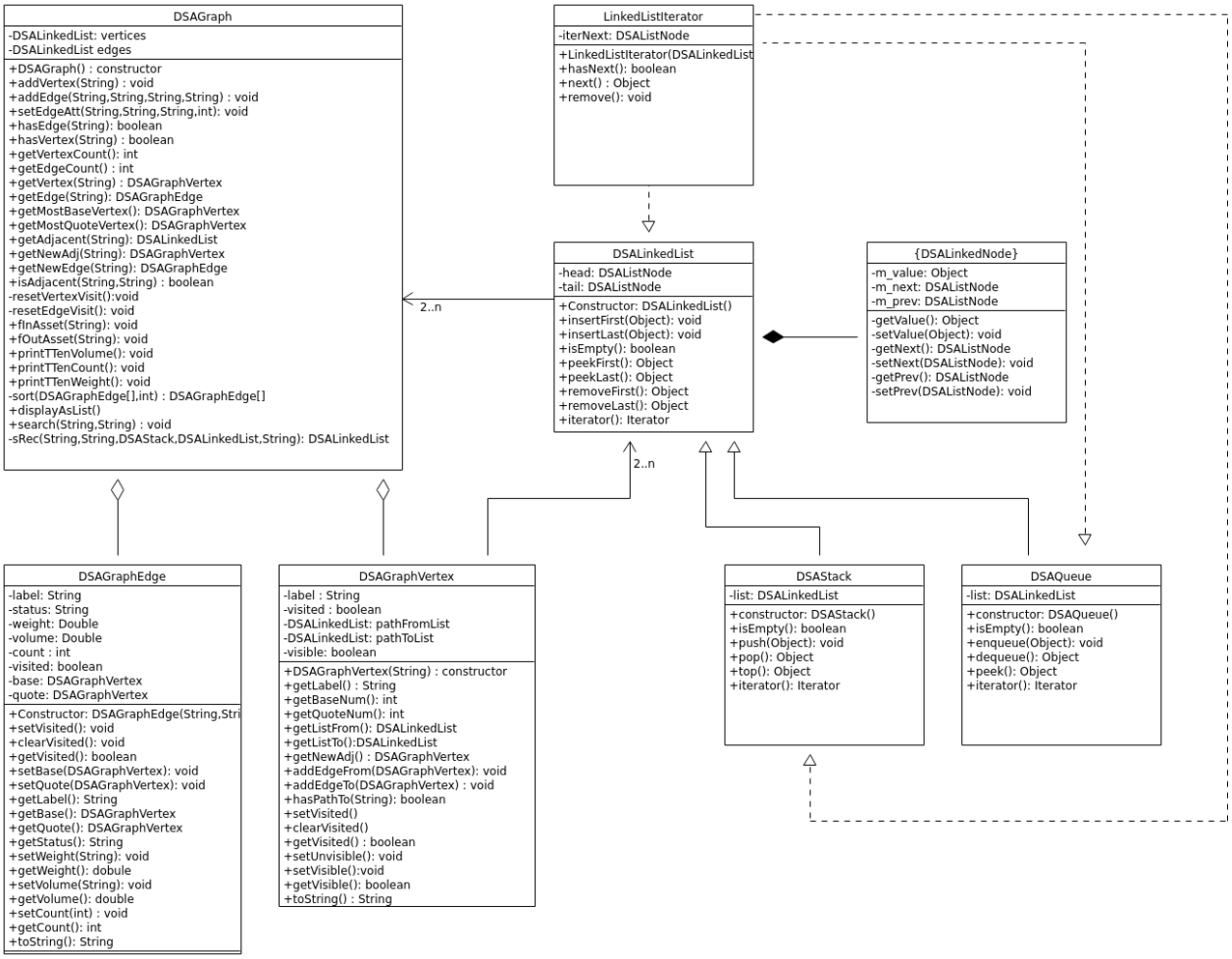
Traceability Matrix

	Requirements	Design/Code	Test & More information
1.Driver/Menu & Modes	1.1. System displays usage if called without arguments	cryptoGraph.main()	More info at Justification
	1.2. System displays menu if called with -i argument	CryptoGraph.main()	More info at Justification
	1.3.In interactive mode user selects a menu option	CryptoGraph.main() & UserInterface.userInput()	UnitTestUserInterface.java
	1.4 System return appropriate error messages from wrong user input.	UserInterface.userInput()	More info at Justification
	1.5 System display report mode if called with -r argument and two datafile	CryptoGraph.main()	More info at Justification
	1.6 System displays usage if called without enough arguments	cryptoGraph.main()	More info at Justification
2. Load Data	2.1 System loads data from assetdata.json	ParseJSON.assetdata()	UnitTestParseJson.java
	2.2 System displays an error message if parse json fails	ParseJSON.assetdata()	UnitTestParseJson.java
	2.3 System loads data from tradedata.json	ParseJSON.tradedata()	UnitTestParseJson.java
	2.4 System displays an error message if parse json fails	ParseJSON.tradedata()	UnitTestParseJson.java
	2.5 System load data from serialised file	FileIO.readSerFile()	UnitTestfileIO.java
	2.6 System displays error message if fileIO fails to read	FileIO.readSerFile()	UnitTestfileIO.java
3. Display an Asset	3.1 System ask for a asset name	UserInterface.userInput()	UnitTestUserInterface.java
	3.2 System displays asset information if found	Graph.getVertex() & DSAGraphVertex.toString()	UnitTestDSAGraph.java
	3.3 System displays an error message if not found	CryptoGraph.main()	More info at Justification
	3.4 System displays an error message if data is not loaded	Cryptograph.main()	More info at Justification
4. Display a trade path	4.1 System asks for a trade label	UserInterface.userInput()	UnitTestUserInterface.java
	4.2 System displays trade path information	Graph.getEdge() & DSAGraphEdge.toString(	UnitTestDSAGraph.java

		)	
	4.3 System displays an error message if not found	CryptoGraph.main()	More info at Justification
	4.4 System displays an error message if datas are not loaded	CryptoGraph.main()	More info at Justification
5.Potential Trade Paths	5.1 System asks for user input for base asset	UserInterface.userInput()	UnitTestUser Interface.java
	5.2 System asks for quote asset name	UserInterface.userInput()	UnitTestUser Interface.java
	5.3 System finds some possible paths	Graph.search()	UnitTestDSA Graph.java
	5.4 System displays possible paths	Graph.search()	UnitTestDSA Graph.java
	5.5 System displays an error message if no path is found.	Graph.search()	UnitTestDSA Graph.java
	5.6 System display an error message if datas are not loaded.	CryptoGraph.main()	More info at Justification
6. Filter in/out an asset	6.1 System displays two option: filter in or out	CryptoGraph.main()	More info at Justification
	6.2 User selects an option	UserInterface.userInput()	UnitTestUser Interface.java
	6.3 if user selects filter out; system will ask for a asset label to filter out	UserInterface.userInput()	UnitTestUser Interface.java
	6.4 System will filter out the selected asset	Graph.fOutAsset()	UnitTestDSA Graph.java
	6.5 If asset can not found system will display an error message	Graph.fOutAsset()	UnitTestDSA Graph.java
	6.6 If user selects filter in, system will ask for a label name to filter back in	UserInterface.userInput()	UnitTestUser Interface.java
	6.7 System will filter in the selected asset	Graph.fInAsset()	UnitTestDSA Graph.java
	6.8 If asset can not found system will display an error message	Graph.fInAsset()	UnitTestDSA Graph.java
	6.9 System will display an error if asset and trade datas are not loaded	cryptoGraph.main()	More info at Justification
7. Asset Overview	7.1 System will display general information about assets	Graph.getVertexCount() & graph.getMostBaseVertex	UnitTestDSA Graph.java

		() & graph.getMostQuoteVertex()	
	7.2 System will display an error message if asset and trade datas are not loaded	cryptoGraph.main()	More info at Justification
8. Trade Overview	8.1 System will display general information about trade paths.	Graph.getEdgeCount() & graph.printTTenVolume() & graph.printTTenCount() & graph.prinTTenWeight()	UnitTestDSAGraph.java
	8.2 System will display an error message if asset and trade data are not loaded	cryptoGraph.main()	More info at Justification
9. Save file	9.1 System will ask a file name to save	UserInterface.userInput()	UnitTestUserInterface.java
	9.2 System will save the file as binary file	FileIO.writeSerializedFile()	UnitTestfileIO.java
	9.3 System will display an error message if serialisation unsuccessful	FileIO.writeSerializedFile()	UnitTestfileIO.java
	9.4 System will display an error message if throws stack over flow error	cryptoGraph.main()	More info at Justification
	9.5 System will display an error message if asset data is not loaded	cryptoGraph.main()	More info at Justification
10. Exit	10.1 System will display good-bye message and end the program	cryptoGraph.main()	More info at Justification
11. Report Mode	11.1 System displays general information about assets and trade paths	Graph.getVertexCount() & graph.getMostBaseVertex() & graph.getMostQuoteVertex() & Graph.getEdgeCount() & graph.printTTenVolume() & graph.printTTenCount() & graph.prinTTenWeight()	UnitTestDSAGraph.java

Class Diagrams:



Note: I did not include the main method, fileIO and parseJson classes into the UML diagram. They have just methods and have use relationship between main and fileIO&parseJson. Also a use relationship arrow would be added between graph and main.

Class Descriptions:

fileIO class:

This class handles the serialisation and deserialisation of the data. This class contains four method; writeSerFile and readSerFile and their wrappers. I chose to create this class to handle file read and write out of the main method. Otherwise main method would have messy structure and would be look bad.

NOTE: I used this class on my previous submitted work on Practical 07.

ParseJSON class:

This class handles the reading data from json files into the graph. Takes graph and filename as input and returns updated graph. This class has two different method. AssetData for reading the asset data(baseAsset, quoteAsset and status) and tradeData for reading tradeData(weighted avg. price, volume and count).

I chose to create this class to handle JSON parsing outside of the main method to keep it clear and simple. These methods would have been included inside fileIO.java as well but I preferred make it separately to have more clear structure since fileIO has nothing related with Json. So I can compile it without JSON package as well.

UserInterface class:

This class handles the all of the input/output for the UI. Also have lots of input/output combinations and error checks. I chose to make it separately from my main to keep main method clean.

NOTE: I used this class on my previous submitted work on PDI class last term.

DSALinkedList class:

This class contains the structure of a double ended-doubly linked list. It has DSAListNode as an inner class. It also has iterator to be able to iterate through the nodes. Also have constructor, insertFirst/last, removeFirst/last and isEmpty methods.

NOTE: I used this class on my previous submitted work on Practical 06.

DSAListNode class:

This is a private inner class inside the DSALinkedList class. This class contains value as an object, and pointer to next and prev nodes. Also have getters/setters and constructor. I created this class inside the DSALinkedList because, nodes do not have meaning by themselves in my design. So they will remain inside the linkedlist and work inside it as private class.

DSAQueue class:

This class contains some of the linked list class methods to imitate queue behaviour as a list. I created this class separately from the linked list because, I have many queue,stacks and linked lists in my code, so this way is much better for code readability.

Note: I used this class on my previous submitted work on Practical 06.

DSAStack class:

This class contains some of the linked list class methods to imitate stack behaviour as a list. I created this class separately from the linked list because of code readability. As I stated earlier, my code has lots of linked list. This way anyone who inspects the code can see and guess linked list behaviour and duty.

Note: I used this class on my previous submitted work on Practical 06.

DSAGraph class:

This class created for a graph representation and contains two Linked lists. One for vertices and one for edges. Those linked lists holds the information about vertex and edges – assets and trade paths respectively-. The reason I chose linked list to store information about vertices and edges is that we do not know how many assets or trade path we would have before storing them. Linked lists are flexible and very useful for this problem(Maxville.2019.DSA Lecture 04 Slide pg.28). The graph is formed by these vertices and edges. This class handles most of the operation related to program. Each vertex is a node(`s value object) in the vertices linkedlist. And each edge is a node in the edges linkedlist. Also this class have lots of methods/functions to operate through graph.

Note: I used this class on my previous submitted work on Practical 06. I added and changed some of the methods for this assignment.

DSAGraphVertex class:

This class created for holding the asset information. It has private variables and getters/setters. It would be possible to create this class as a inner-class of DSAGraph. But in that case, I wont be able to create a vertex outside of the graph. Vertex have a meaning by itself which is an asset so it is not logical to create it as an inner class. Also this class is stored as a node in the vertices linked list in graph.

Note: I used this class on my previous submitted work on Practical 06. I added,changed or deleted some of the methods for this assignment.

DSAGraphEdge class:

This class created for holding the trade paths information. Similar to vertex class it has private variables and getters/setters. Each edge represents a path between two asset. Like vertex class, I created this class outside of the graph, because this class have meaning by itself which is a path. And I would like to create/use it freely without going through graph.

CryptoGraph class:

This class contains main method and some menu options.

Justification:

I implemented directional graph for crypto-currency trading. The reason I chose a graph is, the trade paths and assets form a graph with points(vertex) and paths(edges). Graphs can grow/shrink and store information on points and on paths as well(Maxville.2020.DSA-lecture06 slide.pg.4). Graph were a perfect solution for this project. Also, all of the direct trade paths are directional. For example, while “ETHBTC” is a valid trade path, that doesn’t mean there is a path such as “BTCETH”.

Interaction mode load data: I preferred to hard-code the file names for asset and trade data json files. Because, it would be ambiguous if I ask the user to enter the file name for asset data. I find it a bit confusing, since both of the data files includes information about the assets, I may have called the other file as asset data. Hard coding will help to prevent confusion. And I will supply the data files with the submission.

Display asset: I chose to add more variables into the vertex, for displaying the asset. At first it had only label to display. Then I added a list that holds the paths ends in this vertex, and another list for the paths start from this vertex. So I can display number of paths comes and goes from the vertex.

Display Trade Path: I chose to display label,volume,count and weight avg. price for the information about a trade path. I chose weighted avg. price over askPrice, because ask price is the amount of price that people asks for, it is not guaranteed to be happened. But weighted avg. price is calculated with volume,high price,low price and close price altogether. I found it more consistent. Also, volume and count number can give more information about the trade popularity and stability so I chose to use these two section on my trade display as well.

Potential trade paths: I used depth first search algorithm to find potential trade paths. Please see below, search function section for more information. My algorithm is not able to find all possible trade paths, because at some point I had to mark to visited paths to prevent cycles. When I marked those paths, the algorithm will not be check again those paths to find more paths going through it. It may be fixed by better algorithm design.

Filter an asset in/out: I chose to create another functionality for the graph, and added filter asset back in option. If users change their mind and want to add the filtered out asset back in, they will be able to do it without restarting the program. Please see the filtering in/out section below for more information.

Asset Overview:Since we didn't have to many fields in our vertices/assets, I chose to display general statistics about the assets. So I chose to display total asset number in the graph, The asset with the most direct trade path starts from itself, the asset with the most direct trade path end on itself, and number of those paths.

Trade Overview: I chose to form a top ten list for every field a trade path/edge has. So they have volume,count and weighted avg. price, I created top ten list for those. Also I chose to print information about total path number in the list. Please see printTen functions section below for more informaiton about top ten lists and their algorithm.

DSAGraph.java: I used two DSALinkedList to hold information about vertices and edges. Reasons I choose Linked lists are that they are flexible, can shrink and grow. Since we do not know how many assets and trade paths in the data, linked list are useful to store all the information.

Some graph methods:

Filtering in/out: I add another field for vertex as visible to filter them in or out when wanted. It works just like visited function. When filtering a vertex,I simply set its visibility to 0. This way it is possible to make them filtered back in. I did not prefer to change the imported data as we need to reset the filter back.

PrintTen functions:
I used an array of DSAGraphEdge to store top ten values. Because it is much easier to sort the data after each insert. We can always know how many data inside and we know that we only need total of 10 top datas.
So, it is easier to insert. Since we only have 10 variables time complexity of insertion 0(n) is not much important. Also its easy to access(O(1)) and sort.
I may have use the heap class instead of array, It can be sorted through trickleUp and down. For the top ten variable, I may use length of 11 heap array. So the smaller one will keep changing after each comparison. But in this case, program need to add every single data assest`s attribute to heap array. And array will sort(trickle) after each one. This will increase the required time. And implementation of the heap array is a bit more complicated. And also we can only remove the root node from heap array, I would need to implement a min heap and I would had to reverse it when printing and probably would have to use another adt for this process.

Sort:
I used bubble sort to sort my array for top ten values. Because my array will be always almost-sorted array. Bubble sort is one of the best when sorting almost-sorted arrays (Maxville.2019.DSA Lecture 01 Slide. pg78). This means best case scenario for bubble sort will occur most of the time with O(N) time complexity which is the minimum we can get for a sorting algorithm.

Search function:
I used recursive deep first search to find possible paths. Using breadth first search may be more effective for finding shortest possible paths, but I find it a bit complicated to implement.
I used a linkedList to store all possible paths found in recursive algorithm. And I stored possible paths as a stack in recursion. Because we can not possibly know how many path will be found, it is logical to use a linked list. And using stack will keep them in order, and easily remove the last visited path if we go back in the graph. After recursive operation completed, I printed the possible paths by iterating the linked list and getting stacks one by one and popping each of them until they are empty.
I also added another field to edge class, visited. I run my algorithm by checking edge`s visited fields. Because if I was using vertex visited status, I may not be able to find many possible paths because there may be different paths going through same vertex . My algorithm does not only go through an edge if it is visited before. So this prevents the repetition and cycles.

ParseJSON: I prefered to take only symbol,labels and statuses. And create an edge with those, and add it to graph by using graph.addEdge(). This way, I created the edge and vertices if they don't exit, with minimal variables. Then additionally, I set the edge's attributes by reading trade data, and get weight,volume,count from there. I created another method for this as setEdgeAtt. Finds the related path and sets its attributes.
I used avg weighted price to calculate overall exchange rate between assets. I used status as a checking method if the trade is still active. If it is not active, I did not include that path when finding possible trade paths. I used volume and count just as additional informations.

Additional Data:
I did not prefer to add additional asset info into my graph as my time is limited. If I would do it, I would take name,symbol,market cap and price by Json or reading from csv. Also, I would add more fields into the DSAGraphVertex class such as markep cap, price and name. After get the values from file, I can find the related vertex by getVertex(label) function, and edit its fields accordingly. And I can use this information when displaying an asset. Also, I can display top ten expensive assets and market cap.

UnitTesting:
I did not create separate Unit testing for main method, stack and queue. Because, main method inludes the functions already tested in other Unit Tests. Also, functions ofstack and queue linked lists are tested in linked list Unit Test file.

Expected Performance
When user run the program with interactive mode by using;
-java -Xss4m -cp .:json20200518.jar cryptoGraph -i
Program should display the 9 menu options to the user. User should enter a integer to select an option.
When user selects option;
Case 1.Load data
Program should display a sub-menu with 3 options to select.

1.Load Asset data: When user select first option, program will execute ParseJSON.assetData(graph,"assetdata.json"); command. This will read the data from assetdata.json file and return it as updated graph. Program will go through the json file and save its related fields into the variables and lastly add them as edge in the graph for each symbol in the file. Time complexity of this function is about the number of symbols in the json file, so we can say that it is O(n). And memory usage would be related to the number of symbols as well.
2.Load Asset data: Same as first option, this time program will execute the tradeData(graph,"tradedata.json"); and return an updated graph. Program will go through Json array and get the objects from it. And it should get the related parts from object then call the setEdgeAtt() function and update related parts of the edges in the graph. So program needs to go one by one for every object in the json arrray, this will make the time complexity O(n).

3.Load Serialized data:
When user selects the third option, user will be asked to enter a file name. After file name received, program will execute readSerializedFile(filename), and will read the data from binary data.

Case 2: Display an asset
When user selects second option from main menu, program will ask user for a asset label. When entered, program will call the graph.getVertex(assetLabel);
This function will go through every vertex in vertices list until it finds the correct label. So time complexity will be best O(1), worst O(n) and avg O(n). It will return the vertex and print the information about it. Lets use BTC for demonstration. Program should print it like this;

```
Enter a asset label:
BTC
Asset Label:BTC Direct trade path to: 254 Direct trade path from: 18 different currencies
```

Case 3: Display a trade path
When user selects case 3, program will ask user for a path name via UserInterface class. When entered, program will call the graph.getEdge(tradelabel);
This function will go through every edge in the edges list until it finds the correct label. Time complexity would be best O(1), worst O(n) and avg O(n). Then it will return the edge and print the information about it. Lets use ETHBTC for demonstration. Program should print it like this;

```
Enter a trade label:
ETHBTC
Trade label: ETHBTC
trade from : ETH
trade to : BTC
Avg. Weight: 0.03273027
Volume: 299478.446
Status: TRADING
Total count so far : 121437
```

If not found in the graph output should be like this;

```
Enter a trade label:
BTCETH

Could not find: BTCETH
```

Case 4: Potential trade paths

When user selects case 4, program will ask user for base and quote labels. When they are entered, program will call the graph.search(baseLabel, quoteLabel);
I used depth first search algorithm for search function. Normally, O(n+m) is the time complexity of the depth first search where n is number of vertex and m is number of edges, but we are not stopping after a path found, we will continue to check other paths as well. Also in my design searching can pass through a vertex more than once, this may make the worst case scenario $O(n^2 + m)$ but this is very very unlikely as it needs every vertex connected to each other. Avg case would be still O(n+m). For demonstration we can search paths from eth to try;

```
Please enter a base asset:
ETH
Please enter a quote asset:
TRY
Search completed: Some possible paths are:

1.  path: ETHTRY : Total avg. weight is: 2864.70101961


2.  path: ETHBTC >> BTCTRY : Total avg. weight is: 2862.683743112092


3.  path: ETHBTC >> BTCBUSD >> BUSDTRY : Total avg. weight is: 2862.222427451353


4.  path: ETHBTC >> BTCUSDT >> USDTTRY : Total avg. weight is: 2864.1052056034637
```

Note: Keep in mind, program doesn't include any commission taken by the trading website. So although some longer paths may look more profitable, they can be less effective because of commissions of each transaction.
Note2 : Also these paths do not include the trades that are on break status.
Note 3: This function does not work properly, since it wont be able to find more path that contains, BUSDTRY and USDTTRY edges.

If we enter a trade without any possible path, program will display this message;

```
Search completed: Some possible paths are:
There is no available path.
```

Case 5: Filter out/in

When user selects case 5 from main menu, a submenu will appear and ask the user select filter in or out an asset. User will make a selection and program will call fOutAsset or fInAsset. These functions will use the getVertex function to get the correct vertex and set their visibility accordingly. GetVertex function have O(n) time complexity.

For demonstration we can filter out BTC and re-search possible paths from ETH to TRY again. In this case program should only return the paths without BTC.

```
Please select an option:
1.Filter out an asset
2.Filter in an asset
1
Please enter a asset label to filter out:
BTC
BTC is filtered now.
```

Then search paths from ETH to TRY again;

```
Please enter a base asset:
ETH
Please enter a quote asset:
TRY
Search completed: Some possible paths are:

1.  path: ETHTRY : Total avg. weight is: 2864.70101961


2.  path: ETHBUSD >> BUSDTRY : Total avg. weight is: 2850.5137304077225


3.  path: ETHUSDT >> USDTTRY : Total avg. weight is: 2858.0571352556635
```

If user wants to include BTC again, should select filter in option and enter the BTC.

```
Please select an option:
1.Filter out an asset
2.Filter in an asset
2
Please enter an asset label to filter back in:
BTC
BTC is back in the graph now.
```

Then our search algorithm will include btc again;

```
Please enter a base asset:
ETH
Please enter a quote asset:
TRY
Search completed: Some possible paths are:

1.  path: ETHTRY : Total avg. weight is: 2864.70101961


2.  path: ETHBTC >> BTCTRY : Total avg. weight is: 2862.683743112092


3.  path: ETHBTC >> BTCBUSD >> BUSDTRY : Total avg. weight is: 2862.222427451353


4.  path: ETHBTC >> BTCUSDT >> USDTTRY : Total avg. weight is: 2864.1052056034637
```

Case 6: Asset Overview

This option will display general information about assets in the graph. It will use getVertexCount() to print the number of assets in the graph. This function will iterate through vertices list and count them. So it has O(n) time complexity. Then it will get the asset with most number of being a base in a edge via getMostBaseVertex(). This function will iterate through every vertex in vertices list so it has O(n) time complexity as well. Similarly, program will call getMostQuoteVertex() and iterate through every vertex in the vertices list, with O(n) time complexity. Then it will print this information.

```
Total Asset Number: 315

The Asset has the most direct trade path from : BTC.
BTC  can be directly traded for : 254 different crypto currencies.

The Asset has the most direct trade path to : BNB.
BNB can be directly traded from: 19 different crypto currencies.
```

Case 7: Trade Overview

This option will display general information about trade paths in the graph. It will use getEdgeCount() to get the number of the edges in the graph and print it. This function will iterate through every edges in the edges list so it has O(n) time complexity. And then it will call three of top ten functions for volume,count and weight. These functions will go through every edges and have time complexity of O(n) as well. They will print the vertices with top ten values of related field.

```
Total Direct Trade Path Number :1091

Top-ten Trade Paths by Volume:
1.BTTUSDT   Volume: 3.0919119885E10
2.WINUSDT   Volume: 1.2022472935E10
3.NPXSUSDT   Volume: 6.819657047E9
4.WINTRX   Volume: 3.3535063665E9
5.BTTTRX   Volume: 1.9849255995E9
6.VETUSDT   Volume: 1.864392125E9
7.NPXSETH   Volume: 1.823267491E9
8.TRXUSDT   Volume: 1.6852966641E9
9.MFTUSDT   Volume: 1.658499476E9
10.OSTBTC   Volume: 1.417318393E9

Top-ten Trade Paths by Count:
1.BTCUSDT   Count: 750364
2.ETHUSDT   Count: 286988
3.YFIIUSDT   Count: 240664
4.YFIUSDT   Count: 213665
5.LINKUSDT   Count: 200777
6.BNBUSDT   Count: 195093
7.BTCBUSD   Count: 163367
8.UNIUSDT   Count: 146072
9.BUSDUSDT   Count: 133945
10.ETHBTC   Count: 121437

Top-ten Trade Paths by Avg.Weight:
1.BTCBIDR   Avg.Weight: 1.6460237925E8
2.BTCIDRT   Avg.Weight: 1.6320723708E8
3.BTCBKRW   Avg.Weight: 1.282409521086428E7
4.ETHBIDR   Avg.Weight: 5375707.2
5.BTCNGN   Avg.Weight: 5137466.834558
6.BTCRUB   Avg.Weight: 858991.80632272
7.BNBBIDR   Avg.Weight: 418330.02
8.ETHBKRW   Avg.Weight: 414801.29325028
9.BNBIDRT   Avg.Weight: 413548.34
10.BTCUAH   Avg.Weight: 315732.88932617
```

Case 8: Save graph as serialised file

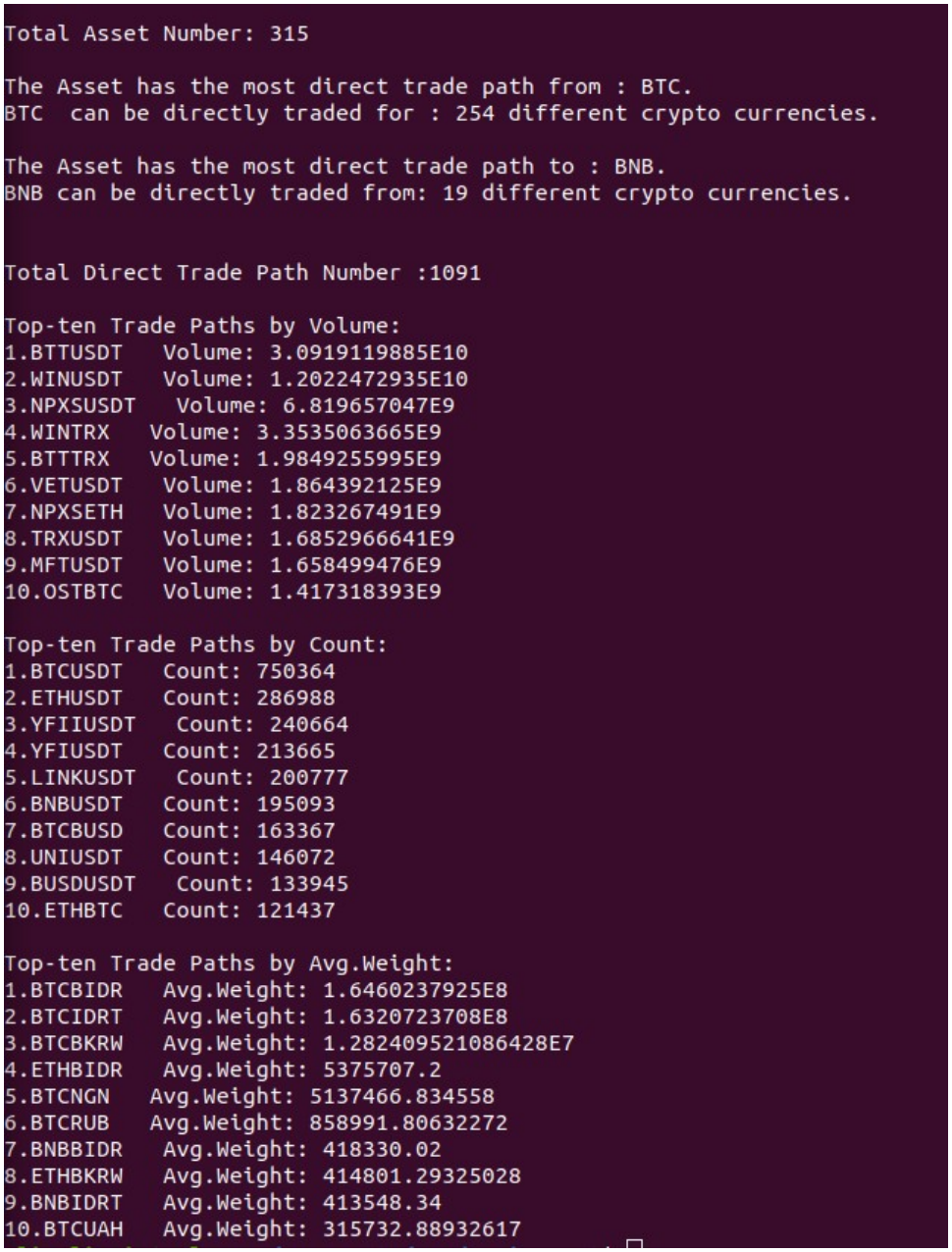
This option will serialise the graph and its contents and save it to binary file. Since my program have lots of iterations with linked lists, without -Xss4m flag, it throws stackoverflow error. To prevent this -Xss4m flag should be used when executing the file. This will enable program to use more memory and prevent the stack overflow error. File will be saved at location of program folder.

Case 9: Exit

This option will exit the program by exiting the menu while loop.

Report Mode:
User should use the following command to run the program with report mode;
-java -cp .:json-20200518.jar cryptoCurrency -r assetdata.json tradedata,json
assetdata.json is the file that contains exchangeInfo. It can be any other named file, but should be in the same format.
Tradedata.json: is the file that contains last 24hr trade information. It can be any other named file, but should be in the same format as 24hr trade data.

Then program will read data from two json files and will update the graph accordingly. After that similar to interactive mode, program will call some functions such as; getMostBaseVertex, getMostQuoteVertex, getVertexCount, getEdgeCount,printTTenVolume, printTTenCount, prinntTTenWeight. As I mentioned in the interactive mode section, these functions have O(n) time complexity each.



Invalid Argument number:

Program will print the required usage if argument number is wrong.

```
Usage:
For Interactive testing mode: run with -i.
For use of serial files run with -Xss4m flag.
Recommended usage for full functionality:
 java -Xss4m <other arguments> -cp <json package> cryptoGraph -i
For Report mode: java -cp <json package> cryptoGraph -r <asset_file> <trade_file>
```

Error handling:

All error handling by user input taken care of inside the UserInterface class. Other than this invalid or non-exist file name errors taken care of inside the fileIO class.

```
1.Load Asset data
2.Load Trade data
3.Load from Serialized data
3

Please enter a file name to read from:
wrongfilename
File not found!
```

```
Please select an option:
1.Load Data
2.Display an asset
3.Display a trade path
4.Find potential trade paths
5.Asset filter
6.Asset Overview
7.Trade Overview
8.Save Data
9.Exit
erer
 Error: Please enter a valid value between 1 and 9
```

Reference List:
1. Valerie Maxville.2019.DSA-Lecture01 Intro and Sorting.ppt Pg.78
2. Valerie Maxville.2019.DSA-Lecture04 Lists and Iterators.ppt Pg.28
3. Valerie Maxville 2020.DSA-Lecture06 Graphs.ppt. Pg.4