Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Vig	Student ID:	19983530
Other name(s):	Muskan		
Unit name:	Data Structures and Algorithms	Unit ID:	COMP1002
Lecturer / unit coordinator:	Valerie Maxville	Tutor:	Ben Thursday 2-4 pm
Date of submission:	Due by 1 November, 2020 ,6 pm	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Date of 30 October
2020 Muskan

Signature:signature:

(By submitting this form, you indicate that you agree with all the above text.)

Project Report: cryptoGraph

Information for Use

- **Introduction:**
  This assignment is about analyzing the crypto-currency trading data. The data can be retrieved from www.binance.com.
  The main entry point program is cryptoGraph with command line arguments:
  "No arguments": Displays what arguments are valid with functionality.
  "-i": interactive testing environment
  "-r <asset_file> <trade_file>": report mode

  The functionality of the program begins with loading the crypto data either from JSON files or loading the serialized data. Furthermore, assets and trades can be enquired. Menu options give range of options such as finding a particular asset and trade, displaying details regarding it and giving an overview of assets and trades. The data can be saved as serialized data and potential direct and indirect trade paths between two assets.

  The Data Structures and Algorithms investigated and chosen for this task are linked list, stack, queue, graph, breadth-first search Algorithm, depth-first search Algorithm, Warshall's Algorithm, insertion sort Algorithm and a few more. More details on choices and decisions of DSA are discussed later in the report.

- **Installation:**
  You will need JDK installed. The files included are:
  - cryptoGraph.java
  - json-20200518.jar
  - ParseJSON.java
  - DSAGraph.java
  - DSAGraphVertex.java
  - DSAGraphEdge.java
  - DSALinkedList.java
  - DSAStack.java
  - DSAQueue.java
  - TradeInformation.java
  - UnitTestDSAGrap.java
  - UnitTestDSALinkedList.java
  - UnitTestDSAStack.java
  - UnitTestDSAQueue.java
  - UnitTestTradeInformation.java
  - README.txt
  - CoverSheet.pdf

- ProjectReport.pdf

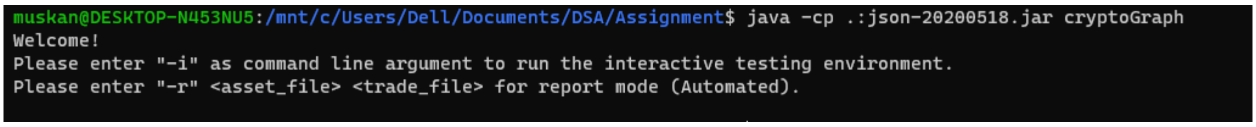
Further information about each class and file is discussed later in this report.

Compiling- To compile the program, use the command:

Javac -cp .:json-20200518.jar *.java

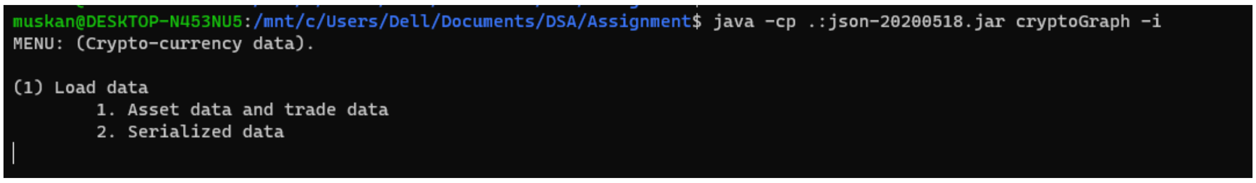
Run- To run the program use the following command for different modes.

Java -cp .:json-20200518.jar cryptoGraph



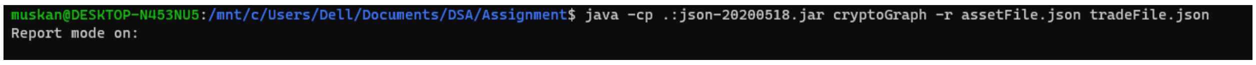
```
muskan@DESKTOP-N453NU5:/mnt/c/Users/Dell/Documents/DSA/Assignment$ java -cp .:json-20200518.jar cryptoGraph
Welcome!
Please enter "-i" as command line argument to run the interactive testing environment.
Please enter "-r" <asset_file> <trade_file> for report mode (Automated).
```

Java -cp .:json-20200518.jar cryptoGraph -i



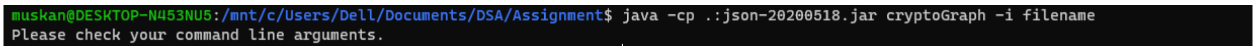
```
muskan@DESKTOP-N453NU5:/mnt/c/Users/Dell/Documents/DSA/Assignment$ java -cp .:json-20200518.jar cryptoGraph -i
MENU: (Crypto-currency data).

(1) Load data
        1. Asset data and trade data
        2. Serialized data
|
```

Java -cp .:json-20200518.jar cryptoGraph -r <asset_file> <trade_file>



```
muskan@DESKTOP-N453NU5:/mnt/c/Users/Dell/Documents/DSA/Assignment$ java -cp .:json-20200518.jar cryptoGraph -r assetFile.json tradeFile.json
Report mode on:
```

The command line arguments are validated and displays meaningful error messages.



```
muskan@DESKTOP-N453NU5:/mnt/c/Users/Dell/Documents/DSA/Assignment$ java -cp .:json-20200518.jar cryptoGraph -i filename
Please check your command line arguments.
```

- **Terminology/Abbreviations:**

  - **Complexity:** **Algorithm complexity** is a measure which evaluates the order of the count of operations, performed by a given or **algorithm** as a function of the size of the input **data**. To put this simpler, **complexity** is a rough approximation of the number of steps necessary to execute an **algorithm**.
  -

- **Walkthrough:**

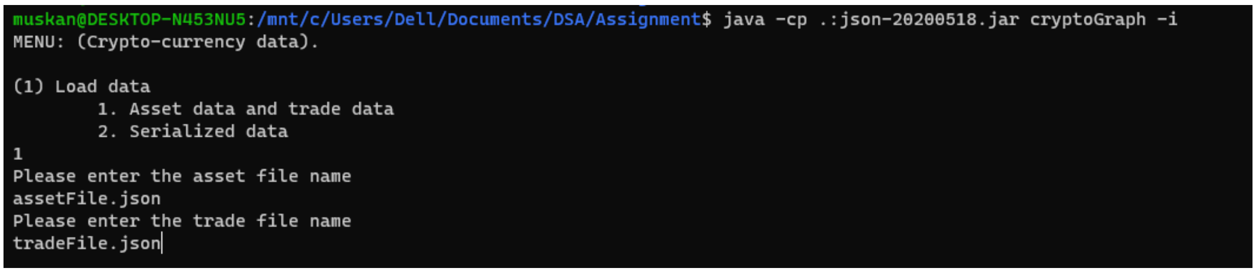
  The functionality and implementation of the options in the program is as below-

  (1) load data

  - Asset data

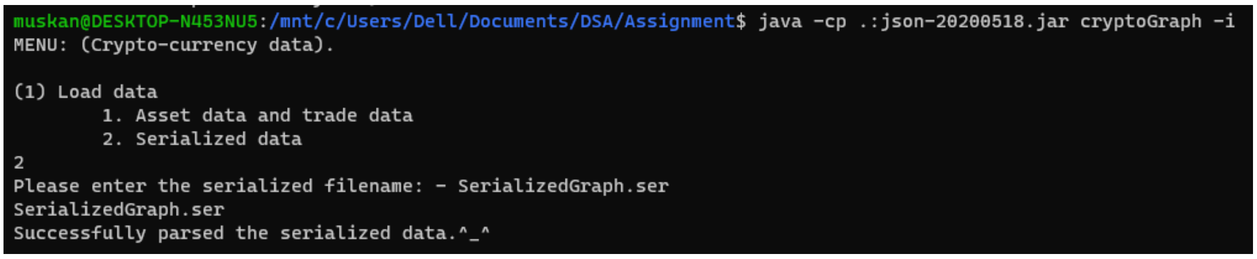
  -Trade data

  This menu option loads the data from the saved JSON files: assetFile.json and tradeFile.json. The files get parsed and data is stored in the program for further functionality of the menu.



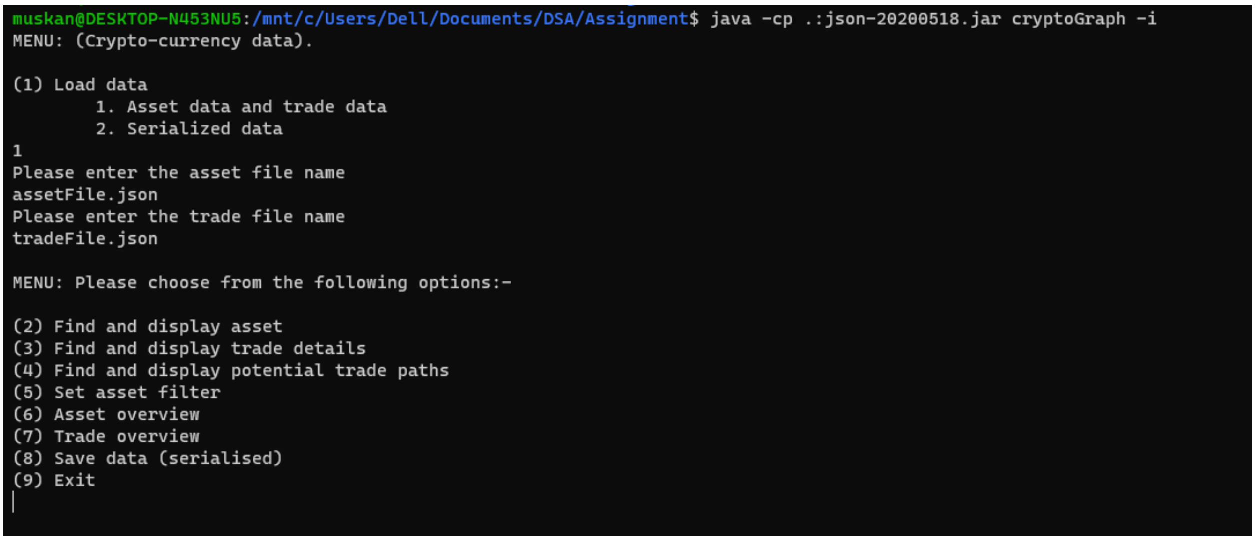
```
muskan@DESKTOP-N453NU5:/mnt/c/Users/Dell/Documents/DSA/Assignment$ java -cp .:json-20200518.jar cryptoGraph -i
MENU: (Crypto-currency data).

(1) Load data
        1. Asset data and trade data
        2. Serialized data
1
Please enter the asset file name
assetFile.json
Please enter the trade file name
tradeFile.json
```

- **Serialized data:** This option loads the serialized data which is further used in other options' functionality.

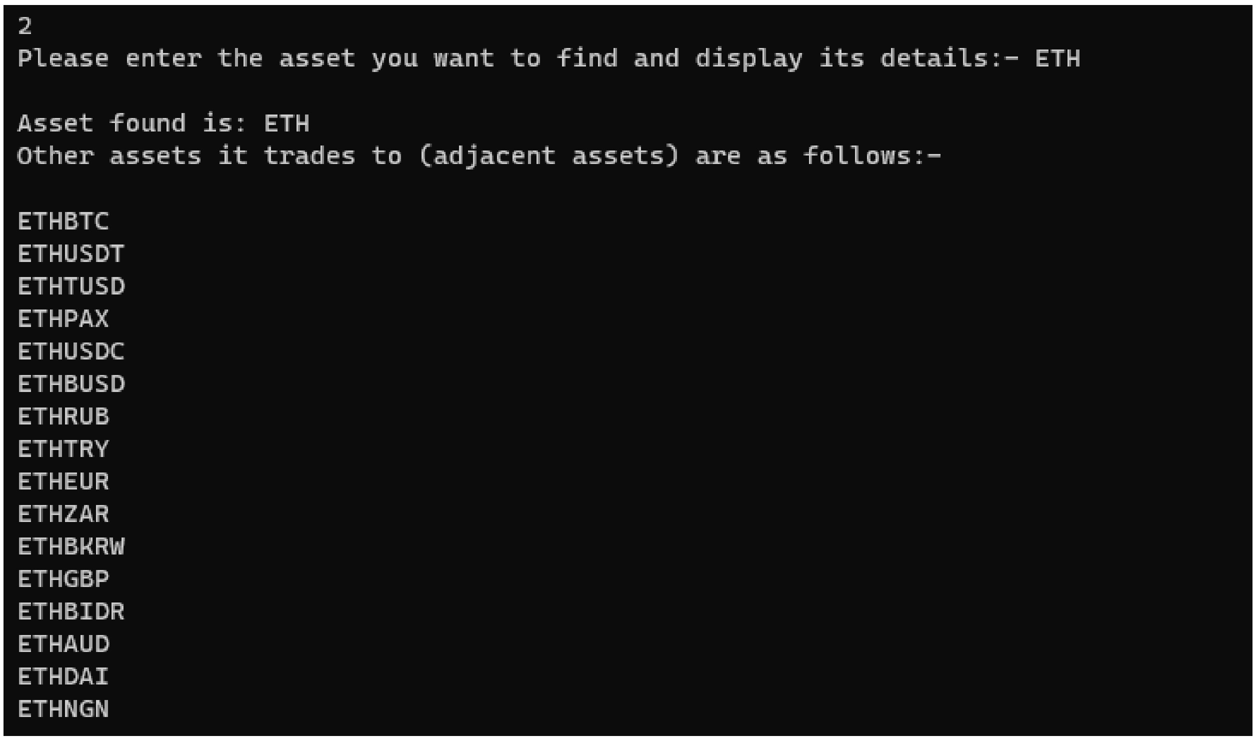


```
muskan@DESKTOP-N453NU5:/mnt/c/Users/Dell/Documents/DSA/Assignment$ java -cp .:json-20200518.jar cryptoGraph -i
MENU: (Crypto-currency data).

(1) Load data
        1. Asset data and trade data
        2. Serialized data
2
Please enter the serialized filename: - SerializedGraph.ser
SerializedGraph.ser
Successfully parsed the serialized data.^_^
```

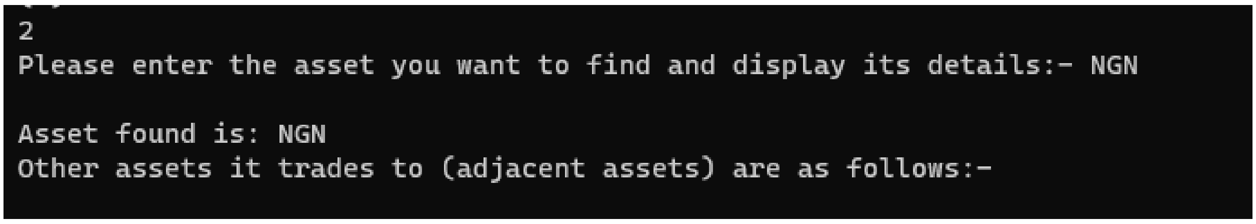
(2) Find and display asset: This menu option finds the particular asset and display its Relevant information.



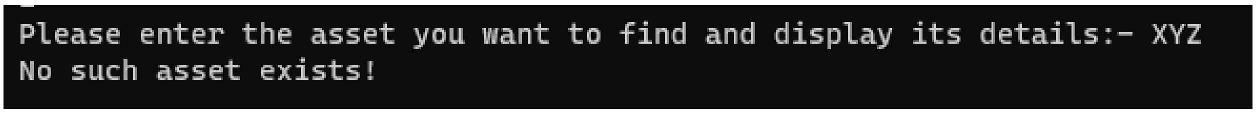
```
muskan@DESKTOP-N453NU5:/mnt/c/Users/Dell/Documents/DSA/Assignment$ java -cp .:json-20200518.jar cryptoGraph -i
MENU: (Crypto-currency data).

(1) Load data
        1. Asset data and trade data
        2. Serialized data
1
Please enter the asset file name
assetFile.json
Please enter the trade file name
tradeFile.json

MENU: Please choose from the following options:-

(2) Find and display asset
(3) Find and display trade details
(4) Find and display potential trade paths
(5) Set asset filter
(6) Asset overview
(7) Trade overview
(8) Save data (serialised)
(9) Exit
```



```
2
Please enter the asset you want to find and display its details:- ETH

Asset found is: ETH
Other assets it trades to (adjacent assets) are as follows:-

ETHBTC
ETHUSDT
ETHTUSD
ETHPAX
ETHUSDC
ETHBUSD
ETHRUB
ETHTRY
ETHEUR
ETHZAR
ETHBKRW
ETHGBP
ETHBIDR
ETHAUD
ETHDAI
ETHNGN
```



```
2
Please enter the asset you want to find and display its details:- NGN

Asset found is: NGN
Other assets it trades to (adjacent assets) are as follows:-
```

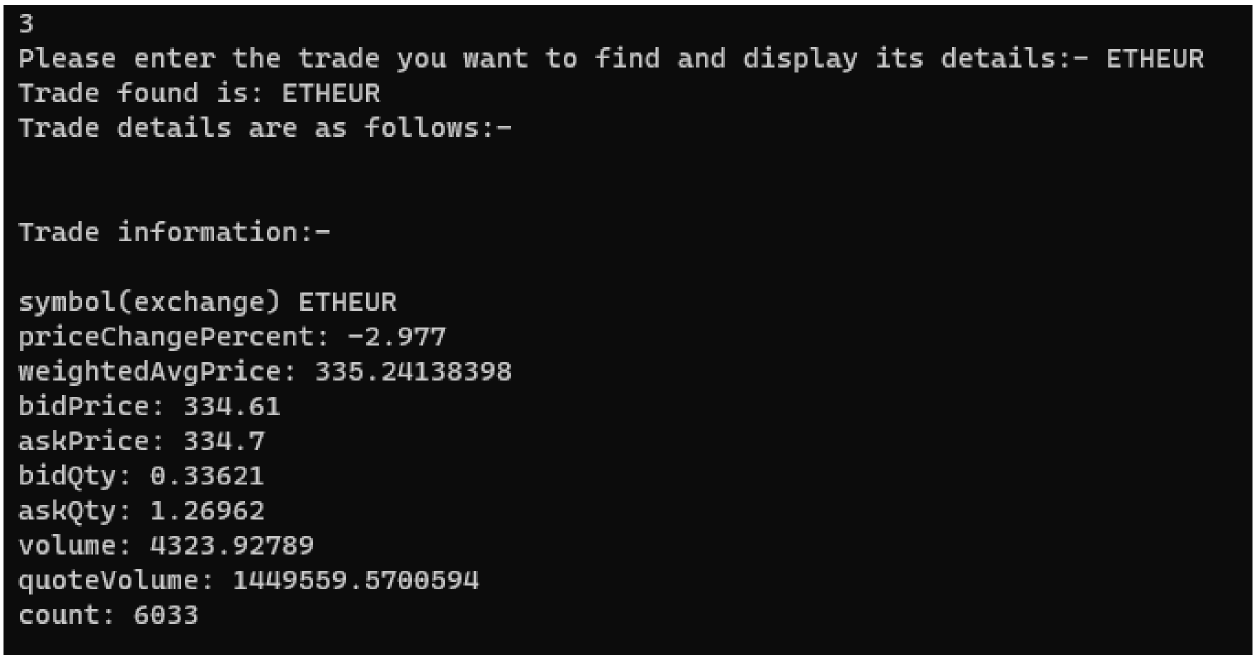


```
Please enter the asset you want to find and display its details:- XYZ
No such asset exists!
```

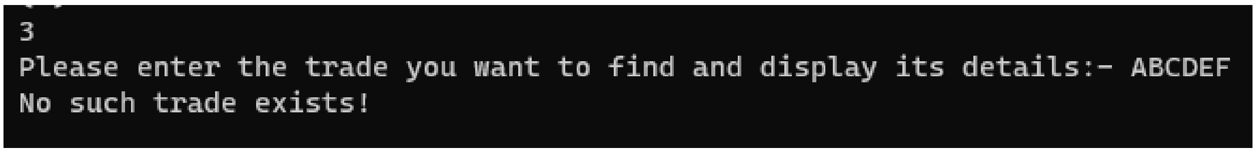
Here we see that NGN asset is not the base asset for any trade.

No XYZ asset exists.

(3) **Find and display trade details:** This menu option will display details of the trade entered.



```
3
Please enter the trade you want to find and display its details:- ETHEUR
Trade found is: ETHEUR
Trade details are as follows:-


Trade information:-

symbol(exchange) ETHEUR
priceChangePercent: -2.977
weightedAvgPrice: 335.24138398
bidPrice: 334.61
askPrice: 334.7
bidQty: 0.33621
askQty: 1.26962
volume: 4323.92789
quoteVolume: 1449559.5700594
count: 6033
```



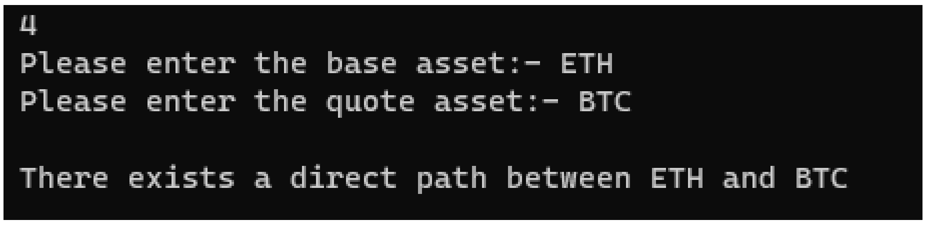
```
3
Please enter the trade you want to find and display its details:- ABCDEF
No such trade exists!
```

No ABCDEF trade exits.

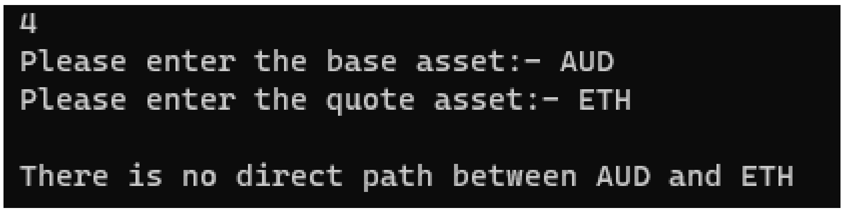
(4) Find and display potential trade paths: This option shows all the paths from base Asset to quote asset. There might be direct as well as indirect paths.

This option isn't implemented fully because of issues encountered while working on it.

This graph is directional as I assumed that here trades are going one way.



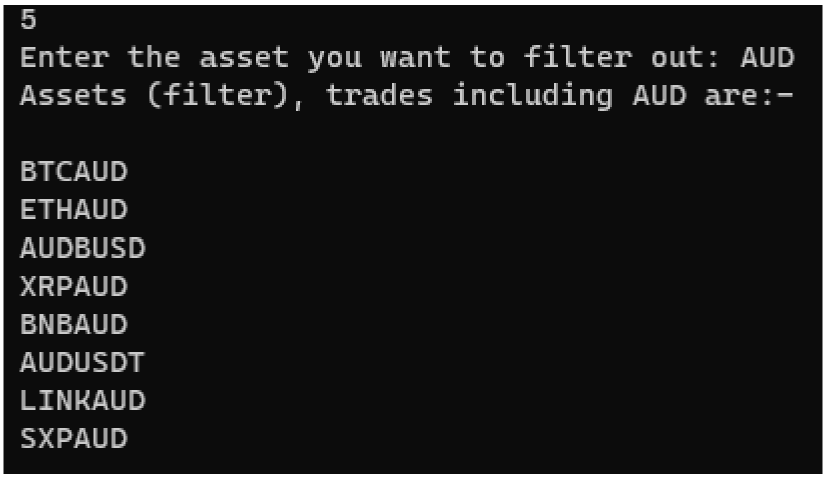
```
4
Please enter the base asset:- ETH
Please enter the quote asset:- BTC

There exists a direct path between ETH and BTC
```



```
4
Please enter the base asset:- AUD
Please enter the quote asset:- ETH

There is no direct path between AUD and ETH
```

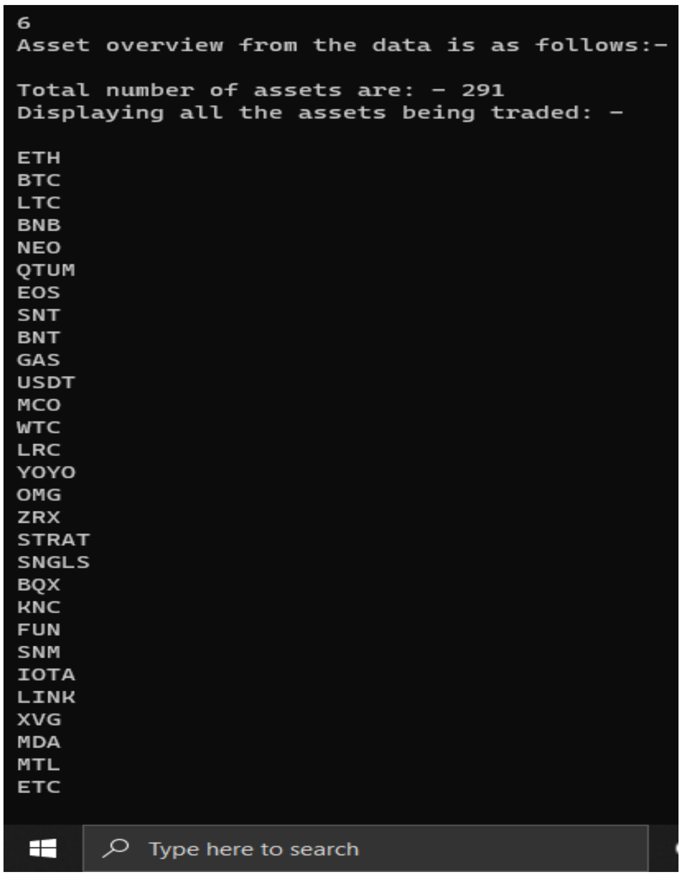
The details of displaying indirect paths are discussed in justification part.

(5) Set asset Filter: This option filters out the trades which includes the certain asset Chosen by the user and displays those filtered trades (exchanges).



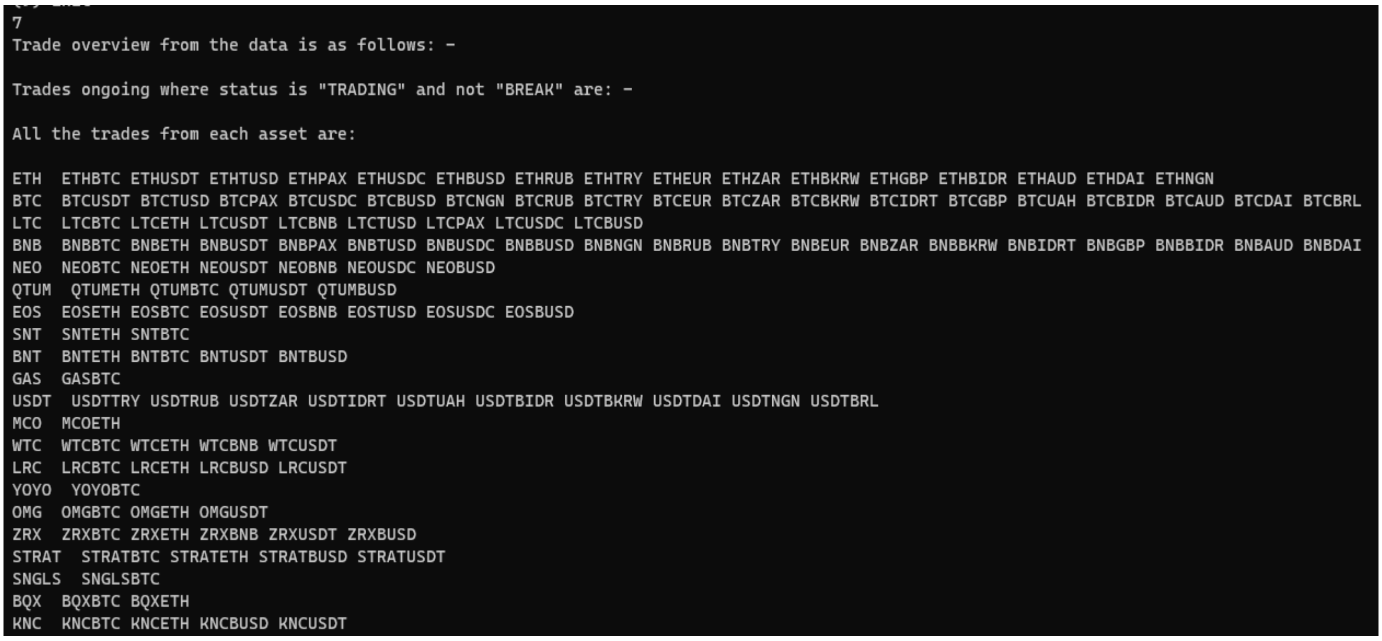
```
5
Enter the asset you want to filter out: AUD
Assets (filter), trades including AUD are:-

BTCAUD
ETHAUD
AUDBUSD
XRPAUD
BNBAUD
AUDUSDT
LINKAUD
SXPAUD
```

(6) Asset overview: This option displays the information about assets in the dataset Such as total number of assets and their symbols.



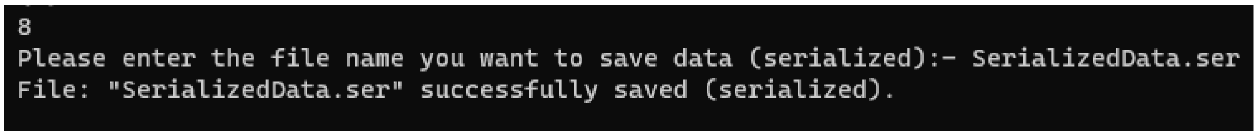
```
6
Asset overview from the data is as follows:-

Total number of assets are: - 291
Displaying all the assets being traded: -

ETH
BTC
LTC
BNB
NEO
QTUM
EOS
SNT
BNT
GAS
USDT
MCO
WTC
LRC
YOYO
OMG
ZRX
STRAT
SNGLS
BQX
KNC
FUN
SNM
IOTA
LINK
XVG
MDA
MTL
ETC
```

(7) Trade overview: This option gives the overview of the trades in the data with number of trades, trades going from each asset and statistics like top trades with respect to volume, count.



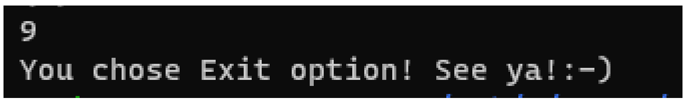
```
7
Trade overview from the data is as follows: -

Trades ongoing where status is "TRADING" and not "BREAK" are: -

All the trades from each asset are:

ETH  ETHBTC ETHUSDT ETHTUSD ETHPAX ETHUSDC ETHBUSD ETHRUB ETHTRY ETHEUR ETHZAR ETHBKRW ETHGBP ETHBIDR ETHAUD ETHDAI ETHNGN
BTC  BTCUSDT BTCTUSD BTCPAX BTCUSDC BTCBUSD BTCNGN BTCRUB BTCTRY BTCEUR BTCZAR BTCBKRW BTCIDRT BTCGBP BTCUAH BTCBIDR BTCAUD BTCDAI BTCBRL
LTC  LTCBTC LTCETH LTCUSDT LTCBNB LTCTUSD LTCPAX LTCUSDC LTCBUSD
BNB  BNBBTC BNBETH BNBUSDT BNBPAX BNBTUSD BNBUSDC BNBBUSD BNBNGN BNBRUB BNBTRY BNBEUR BNBZAR BNBBKRW BNBIDRT BNBGBP BNBBIDR BNBAUD BNBDAI
NEO  NEOBTC NEOETH NEOUSDT NEOBNB NEOUSDC NEOBUSD
QTUM  QTUMETH QTUMBTC QTUMUSDT QTUMBUSD
EOS  EOSETH EOSBTC EOSUSDT EOSBNB EOSTUSD EOSUSDC EOSBUSD
SNT  SNTETH SNTBTC
BNT  BNTETH BNTBTC BNTUSDT BNTBUSD
GAS  GASBTC
USDT  USDTTRY USDTRUB USDTZAR USDTIDRT USDTUAH USDTBIDR USDTBKRW USDTDAI USDTNGN USDTBRL
MCO  MCOETH
WTC  WTCBTC WTCETH WTCBNB WTCUSDT
LRC  LRCBTC LRCETH LRCBUSD LRCUSDT
YOYO  YOYOBTC
OMG  OMGBTC OMGETH OMGUSDT
ZRX  ZRXBTC ZRXETH ZRXBNB ZRXUSDT ZRXBUSD
STRAT  STRATBTC STRATETH STRATBUSD STRATUSDT
SNGLS  SNGLSBTC
BQX  BQXBTC BQXETH
KNC  KNCBTC KNCETH KNCBUSD KNCUSDT
```

(8) Save data (serialized): This menu option saves the serialized data for further usage.



```
8
Please enter the file name you want to save data (serialized):- SerializedData.ser
File: "SerializedData.ser" successfully saved (serialized).
```

(9) Exit: The exit the program.



```
9
You chose Exit option! See ya!:-)
```

"-r" mode: When ran in report mode, it is automated and does not require user interaction. It displays necessary details on the console.

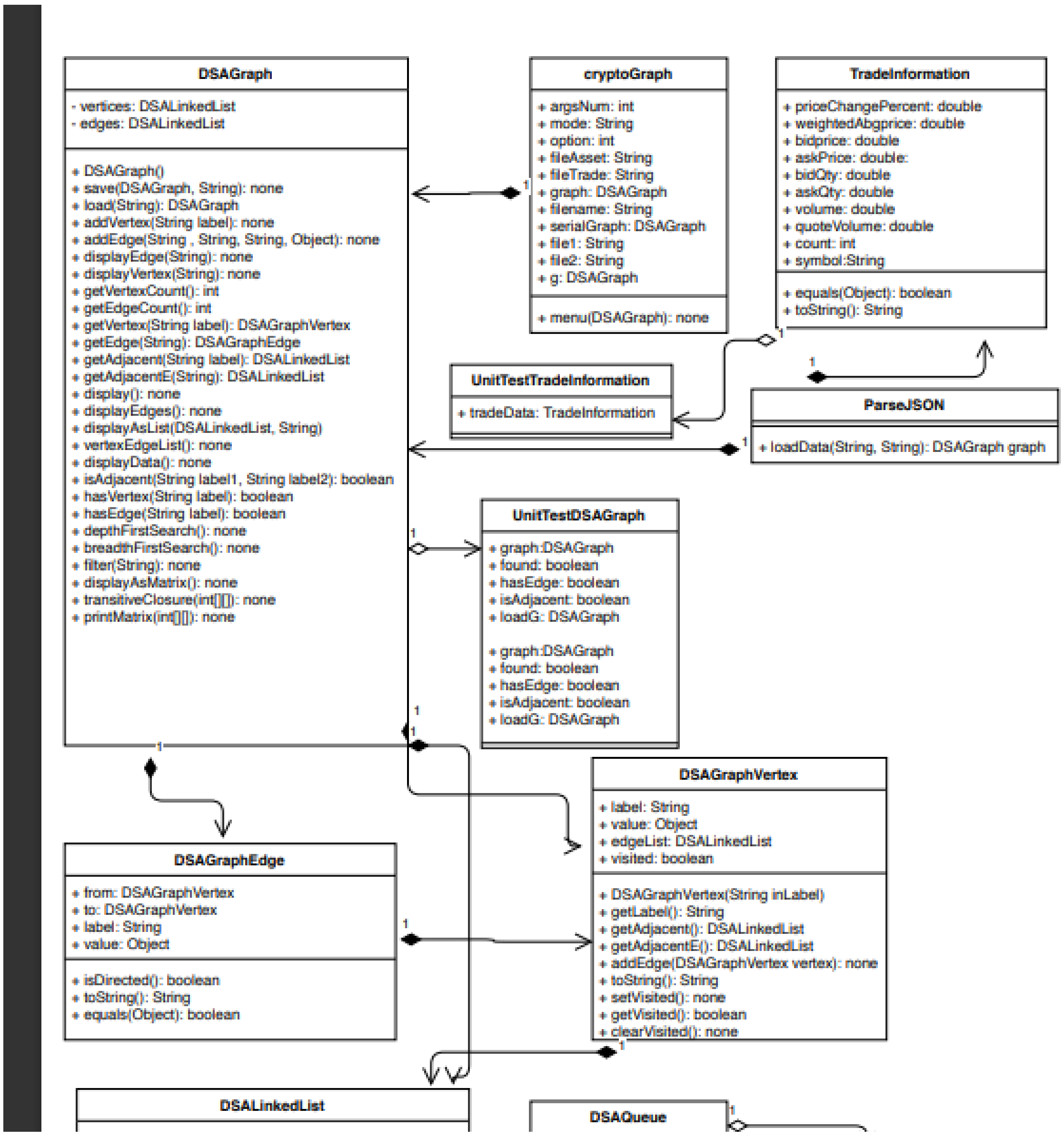
- **Future Work:** In future I plan to work on getting data from APIs and make this interesting project more useful. My assets stored in vertices just have the symbol so far as asset data. However, using assetfile.csv which has all the data about assets can be stored as vertices' value which will be more informative. To calculate trade rates, optimizing trade paths, finding profitable trade loops, I will use Dijkstra's Algorithm. Visualization and plotting graphs will enhance this project and I wish to use JPanel or swing component for it. The issue in finding indirect paths will be looked upon and will be fixed for other functionalities.
  More web scarping will be done to optimize the solution and make the algorithms more efficient.

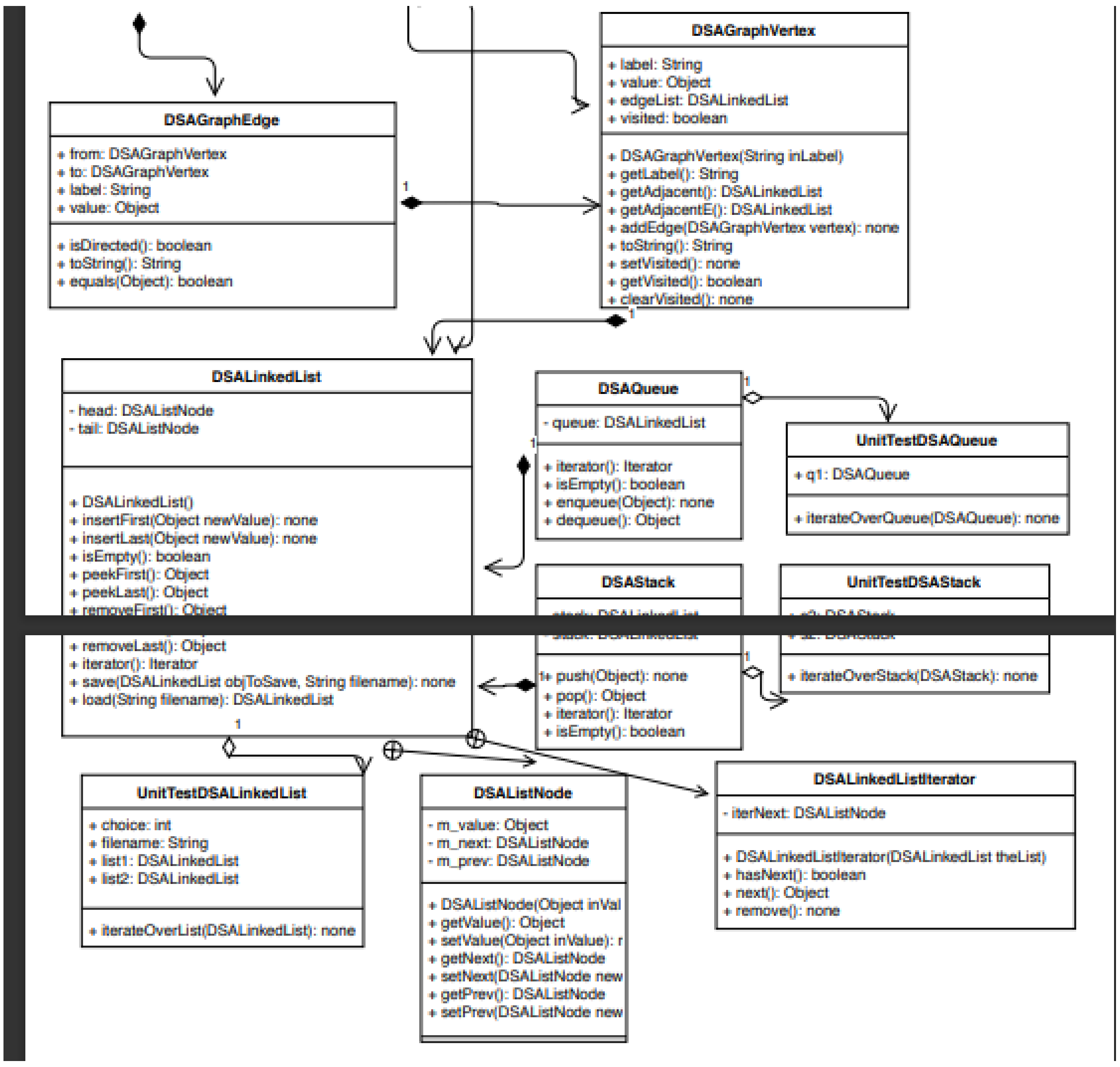
<u>Traceability Matrix:</u>

S no.	Functionality	Requirements	Design/Code	Test
1.	Command line arguments	1.1 Programs displays if no arguments passed. 1.2 Program displays interactive menu with "-I" 1.3 Program runs in report mode with "-r <asset_file> <trade_file>" 1.4 program displays meaningful error messages when invalid arguments are passed.	cryptoGraph.java	cyptoGraph.java Program functions correctly according to the command line arguments.
2.	Menu	Prompts various menu options of different functionality.	cryptoGraph.java	cryptoGraph.java Menu options are tested in ADT's test harnesses
3.	FileIO	3.1 To read the file, program asks for asset, trade and serialized file information. 3.2 After reading "SerializedData.ser", it says successfully read.	ParseJSON.java	ParseJSON.java Correct graph is being created and files are being read correctly.
4.	Asset functions	Displaying asset and details.	DSAGraph.java	UnitTestDSAGraph.java All the functions are tested and works.
5.	Trade functions	Displaying trade and trade overview.	DSAGraph.java DSAEdge.java TradeInformation.java	UnitTestDSAGraph.java UnitTestDSALinkedList.java UnitTestDSAStack.java UnitTestQueue.java UnitTestTradeInformation.java
6.	Trade information (Edge's weight)	Display's trade's information.	TradeInformation.java	UnitTestTradeInformation.java
7.	Serialization	7.1 Loads and saves serialized data.	DSAGraph.java	UnitTestDSAGraph.java

		7.2 Prompts the user for file names for save and load.	cryptoGraph. java	Tested and serialization works fine.

Class Diagram:





<u>Class Descriptions:</u>

TradeInformation class: - This class is designed to hold the information of the trades. Since trade happens base asset and quote asset and, in our program, assets are vertices and trades are exchanges. The object of this class will be stored as value (weight) in the edge.

This class has equals() and toString() method which act as helping functions for edge's weight. Information such as weighted average price, bid price, ask price, bid quantity, ask quantity, volume, count, symbol and price change percent is extracted and encapsulated in the edge's value as object.

UnitTestTradeInformation: - This class is designed for unit testing. This class holds a good significance as its object is the weight of our edges. Testing each class is for sanity check and helps in not dealing with debugging a whole set of files where one doesn't know what went wrong.

cryptoGrpah class: - This class is the entry point of the program. It deals with different command line arguments with validation and performs each mode's function. This class contains menu() method which display the interactive menu to the user and do necessary calls.

ParseJSON class: - This class is to deal with FileIO operations. It reads the parsed json files and parses data to the required data structures (here it forms the graph). It reads the assetFile and creates the vertices by extracting “base asset” and “quote asset” key and trade file is parsed to add edges with “symbol” keyword. TradeInformation object is created to pass the information from this file into edges’ value.

Both the files together form the graph structure. However, we notice that in the trade file data, which is from past 24 hours, certain trades have “BREAK” status in asset file and have 0 values in trade file for the same. So, we choose to exclude them from our graph.

DSAGraph class: - This class is the implementation of graph data structure since it is the main pillar for our program. Using graph data structure made for sense to me due to the connections between different values and their relation. If I didn’t have any knowledge of DSA, I might have thought of arrays but again, amount of data is not predicted in advance here. Different ADTs are also used to implement graph such as stack, queues and all these structures use linked list which fairly have advantage over other data structures.

Different helping functions are involved in it and save(), load() for serialization are included as well.

UnitTestDSAGraph class: - This class is designed for unit testing of DSAGraph class. This class tests the working our graph in all aspects and gives us a sanity check before dealing with a very huge graph.

DSAGraphVertex class, DSAGraphEdge class, DSALinkedList class: - This class is designed for the implementation of DSAGraph since DSAGraph shares composition relationship with all these classes.

DSAStack and DSAQueue class: - These abstract data structures implementation is used for certain algorithms to traverse the graph such as bfs, dfs.

UnitTestDSALinkedList, UnitTestDSAStack, UnitTestDSAQueue: - These classes are made for unit testing. It makes sure that there is no logical error in our ADTs implementation which will not affect the implementation of our program in any negative manner. Both black box as well as white box testing are equally important.

Private inner classes in DSALinkedList are also included but only this classes use them, so they are designed as private inner classes. Those classes include **DSALinkedListNode, DSALinkedListIterator.**

<u>Justification:</u>

Files for loading data in the program are saved as json files from www.binance.com. Asset file is saved with the name assetFile.json and trade file is saved as tradeFile.json.

Asset file- Tradeable tokens: https://www.binance.com/api/v3/exchangeInfo

Trade file- Trade information for last 24 hours: https://www.binance.com/api/v3/ticker/24hr/

The **graph** is implemented as **directional with edge class implementation to store weight**. We have created graph from the trades where status is “**Trading**” in asset file since there are trades in past 24 hours which are not being traded and have status as “**Break**”. **Linked list data structure** has been used for this ADT implementation which stores **vertex objects** and **edges object** for graph. Insertion in list takes O (1) storage which is fast. Reasons for choosing linked list **instead of**

arrays are whatever amount data is there, we don't have predict in advance. This data structure expands to fill out all the available memory, hence we don't have to care about how many edges between the vertices will be there before hand.

A stack ADT is implemented for our **depth-first search algorithm**. Stack follows LIFO (Last in First Out) structure which is helpful for DFS. Similarly, **queue ADT** is used for **breadth-First search algorithm** which follows First in First Out (FIFO) structure. Again, these ADTs can make use of arrays and linked list, but we decided to use linked list as amount of data can not be predicted. Stacks and Queues **have O (1) complexity for both insertion as well as deletion.**

The depth-first search and breadth-first search run in **O (V^2) time**, where V is the number of vertices, for **adjacency matrix representation**. They run in **O (V + E) time**, where E is the number of edges, for **adjacency list representation.**

The menu option (4) Find and display potential trade paths show the direct and indirect paths from one vertex to another (base asset to quote asset via different paths). At first, I approach this problem by using **Warshall's Algorithm**. It lets you quickly find out whether one vertex is reachable from another vertex which basically lets you know that if there is indirect path or not. This algorithm creates a matrix called transitive closure which look up at the adjacency matrix of the graph too. This table lets you know instantly and hence takes up O (1) time. **This algorithm does a lot in a few lines of code.** The idea is based on a simple approach: If you can get from vertex L to vertex M, and you can get from M to N, then you can get from L to N.

While implementing it, mid-way I realized that this will not be very much useful in finding **all the indirect paths** or may be the logic behind this solution didn't click me from this algorithm. So, this time I **approached it in another manner**. This approach **involves BFS**. Starting from node 0, we add in list of labels of assets in our queue representing the paths we have taken up to that point. As we dequeue from the queue, the last node in the list is the node we must continue our BFS on. If the last node in the list is the destination node, then we have successfully found a path going from our source to target and we add that list inside of our **resulting list of paths.**

The problem I encountered was since I created a directional graph with "Trading" status, there might be a possibility that the node, which is currently consumed, **don't have a sink node** which breaks the traversal path and affects the BFS algorithm for finding paths. The time complexity for this solution is O ($N^2 * 2^N$) where N is number of nodes in our graph. In future, I will find another way around to implement it with optimization.

For trade overview, it is a good idea to sort the edges in a list based on edge's weight's object value. For example, trades with maximum count or volume or average price. This can be implemented with **insertion sort algorithm** by creating a new list and comparing the object's values by comparing top 10 in descending manner when we are trying to get top 10 count which can be more efficient as well.

Serialization (saving and loading of serialized data) is implement in DSAGraph class which are interface of Serializable in all the classes used such as DSALinkedList, TradeInformation (Object in edges).

While walking the assignment, I learnt many new algorithms and their efficiency with the usage of proper data structures to accomplish the task. Unit test harnesses are included so that the quality of the code can be tested easily as quality matters over quantity.

References: Abstract data types implementation is taken from submitted labs Linked list, Stack, Queue and Graph (Practical) and are referenced inside file’s block comments too.