Page Count – 14 (9 excluding headers and UML)
Word Count – 2214

Data Structures and Algorithms Assignment 1.0 Semester 2, 2020 19779797 Curtin University

Contents

Information for use 3

- Introduction 3
- Installation 3
- Terminology 4
- Walkthrough 4
- Future Work 8

Traceability Matrix 9

Class diagrams 10

Class descriptions 13

Justification 14

References 14

Information for use

Introduction

This program is able to read in JSON files and puts all of this data into a graph. The purpose of this program is to implement multiple different ADTs, instead of using the built in ADTs that Python provides us with. The program uses ADTs to provide a simple solution to analyse crypto-currency trading data. From this, we can query different currencies, to get relevant data, as well as trade paths, from one currency to another. Another functionality of the program is to provide an overview of the dataset, giving a visual implementation of the graph. This can be done in '-r' (report) mode, one of the 2 modes in the program. The second mode '-i' (interactive) mode allows the user to select from an assortment of options, giving data on all of the assets, as well as trade details.

Installation

In order for this program to work correctly, the user must have a python IDE installed on their computer, allowing them to open .py files and run them. There are a total of 7 .py files that work in tandem and are required for this program to run correctly. They can be found below accompanied by a short description.

cryptoGraph.py

The very simple 'front page' of the program. This is what users will interact with when using the program, hence its simplicity. It calls a function from graphMenu.py, which houses the program. This means the user does not need to know how the program works, and only requires them to run the program with command line parameters, using this file.

graphMenu.py

This file is the main menu of the program. It supplies the user with different items based on the chosen mode, and when inside of interactive mode, this file gives the user all of the options and calls functions from other files to apply them.

graphLinkedList.py

The main file in creating a graph based on the data being supplied to it. It creates a custom graph with fields specific to the program, like priceChange and lastPrice for edges, and price and volume for the vertices. The graphs is directional, meaning a trade link ETHBTC only goes from ETH to BTC not the other way round. This means we are able to follow trade paths to find different routes from one asset to another. An additional feature of this file is that it makes use of other files, such as DSAStack.py and BinarySearchTree.py to do certain calculations. For example, graphLinkedList.py makes use of the sorting feature of a binary search tree by adding all the assets to it and recalling the top 10 assets in certain categories to show the user.

linkedList.py

graphLinkedList.py makes use of linkedList.py to create a graph. The two elements of the graph, vertices, and edges, have separate linked lists, that are used to store the assets and trade pairs, along with their relevant data. Alongside all the basic linked list functions, there is a function called findNthNode, which will find a node/vertex/edge in the linked list and can be used to alter and use the values inside of that node. To save processing power and time later on, the linked list inserts elements in an alphabetically sorted order, which makes other calculations easier.

binarySearchTree.py

This file is used as a 'helper' file for graphLinkedList.py. A binary search tree is a data structure that represents hierarchical relationships once implemented, it can quickly and non-computer-intensively recall all the elements in order, making it simple to find the largest n elements. This is used to find the top 10 assets in certain categories and display them to the user.

DSAStack.py

A stack is used as an alternative to lists or arrays in Python. Its implementation in this program is within finding trade paths. When another node is found in between, it can easily be added to the stack, and popped off when required. We can then print a stack out, being a trade path, and clear the stack for a new path.

allTestHarness.py

This file houses all of the test harnesses for this program. It calls them one by one, allowing the user to test all elements of the program at the same time. It provides the user with results of the test, to see if the program passed or failed.

Terminology

Throughout this report, the term '.py' is used repeatedly. This is an abbreviation for Python, and is the name given to a Python code file.

ADT is short for Abstract Data Type. It is a mathematical model of a data structure that specifies the type of data stored, the operations supported by them and the parameters of the operations[1].

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language[2].

Walkthrough

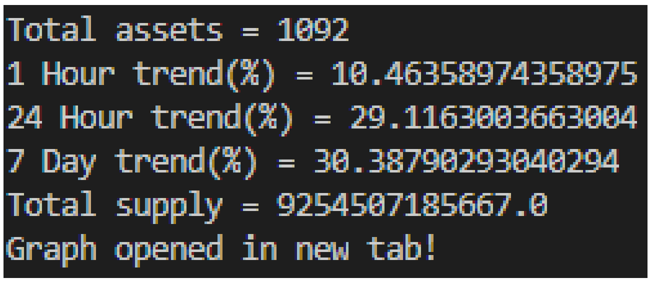
To show usage information

```
python .\cryptoGraph.py
```

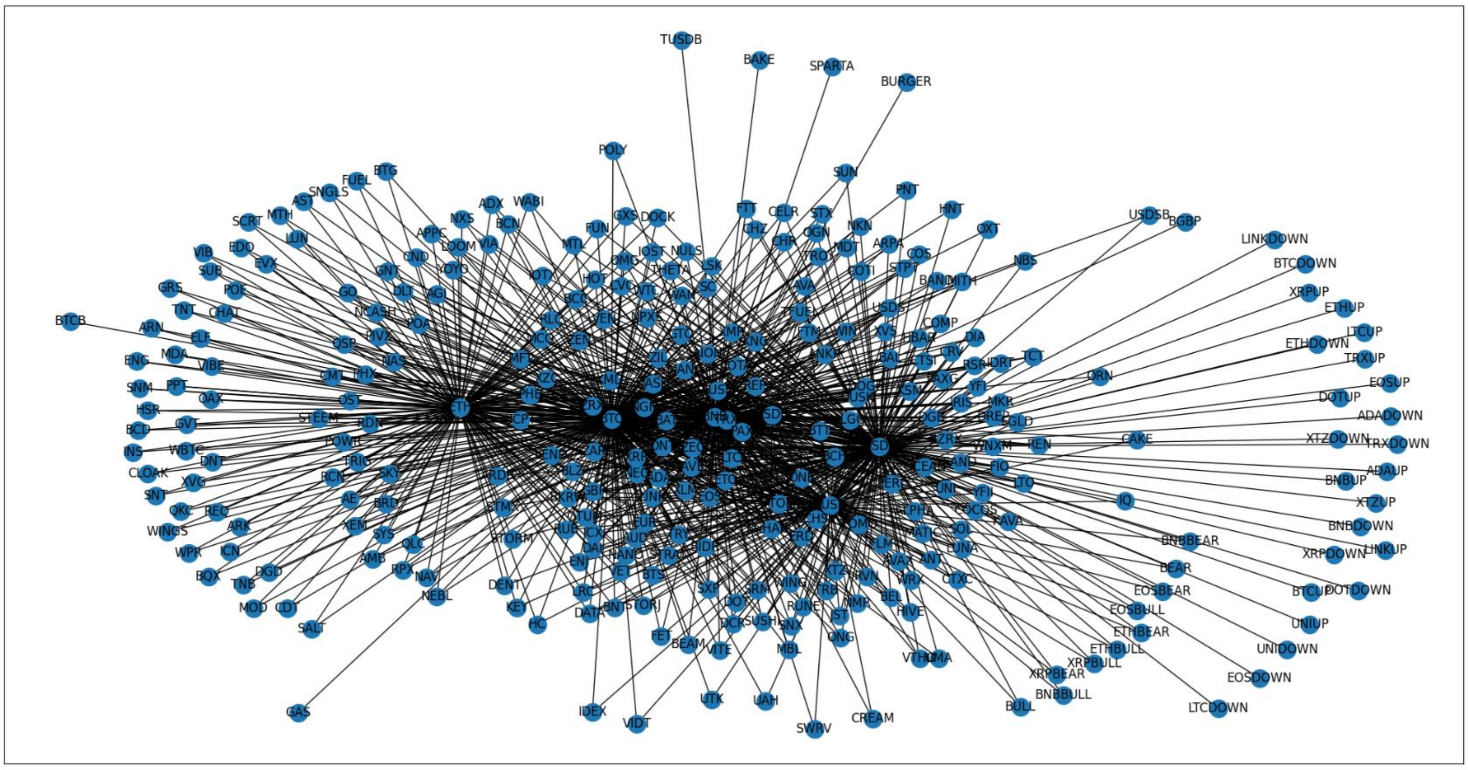
To open in report mode

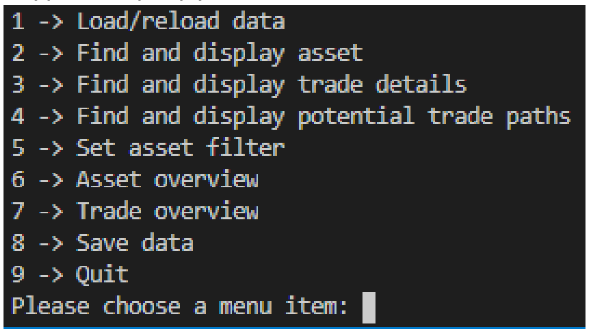
```
python .\cryptoGraph.py -r .\assetFile.json .\24hrTrades.json .\assetInfo.json
```

These JSON files may be stored under different filenames on different computers, so these command line parameters may have to be altered



```
Total assets = 1092
1 Hour trend(%) = 10.463589743589 75
24 Hour trend(%) = 29.1163003663004
7 Day trend(%) = 30.38790293040294
Total supply = 9254507185667.0
Graph opened in new tab!
```



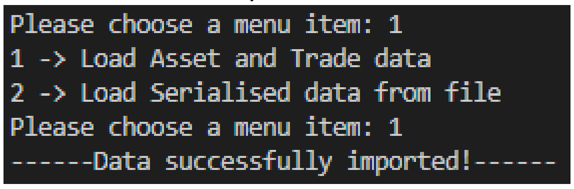


To open in interactive mode

python .\cryptoGraph.py -i

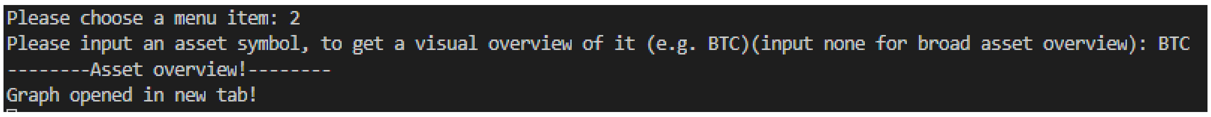
```
1 -> Load/reload data
2 -> Find and display asset
3 -> Find and display trade details
4 -> Find and display potential trade paths
5 -> Set asset filter
6 -> Asset overview
7 -> Trade overview
8 -> Save data
9 -> Quit
Please choose a menu item: 
```

Option 1:

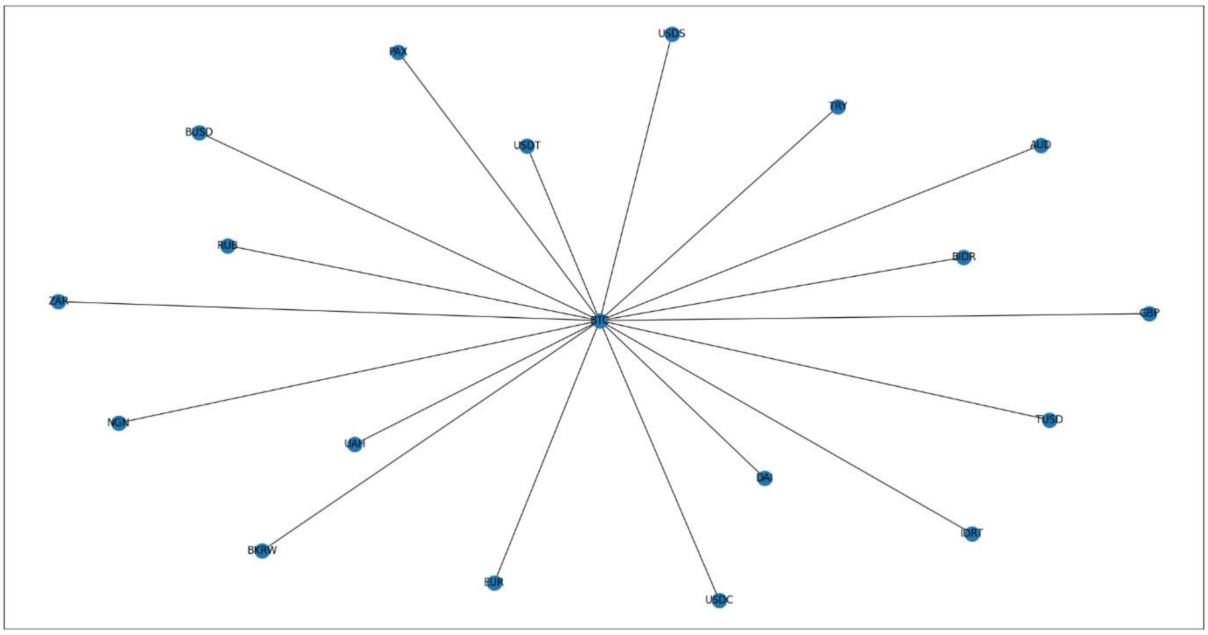


```
Please choose a menu item: 1
1 -> Load Asset and Trade data
2 -> Load Serialised data from file
Please choose a menu item: 1
------Data successfully imported!------
```

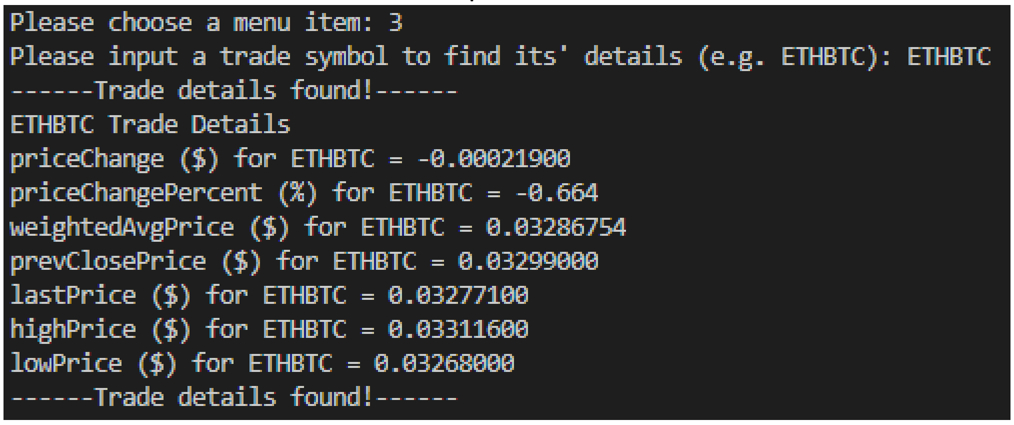
Option 2:



```
Please choose a menu item: 2
Please input an asset symbol, to get a visual overview of it (e.g. BTC)(input none for broad asset overview): BTC
--------Asset overview!--------
Graph opened in new tab!
```

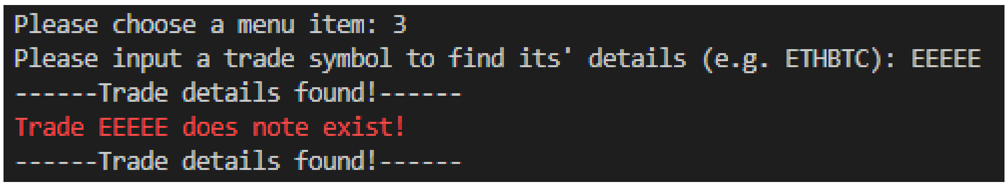


Option 3:



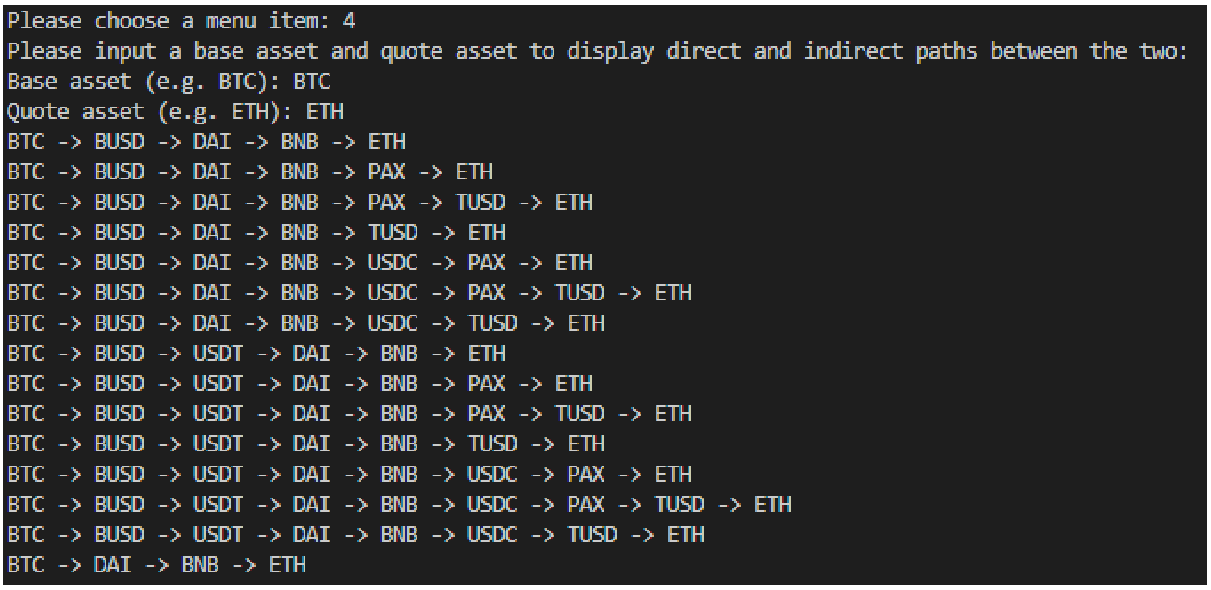
```
Please choose a menu item: 3
Please input a trade symbol to find its' details (e.g. ETHBTC): ETHBTC
------Trade details found!------
ETHBTC Trade Details
priceChange ($) for ETHBTC = -0.00021900
priceChangePercent (%) for ETHBTC = -0.664
weightedAvgPrice ($) for ETHBTC = 0.03286754
prevClosePrice ($) for ETHBTC = 0.03299000
lastPrice ($) for ETHBTC = 0.03277100
highPrice ($) for ETHBTC = 0.03311600
lowPrice ($) for ETHBTC = 0.03268000
------Trade details found!------
```

If at any point the user enters an asset or trade that does not exist, they are greeted with the following message.



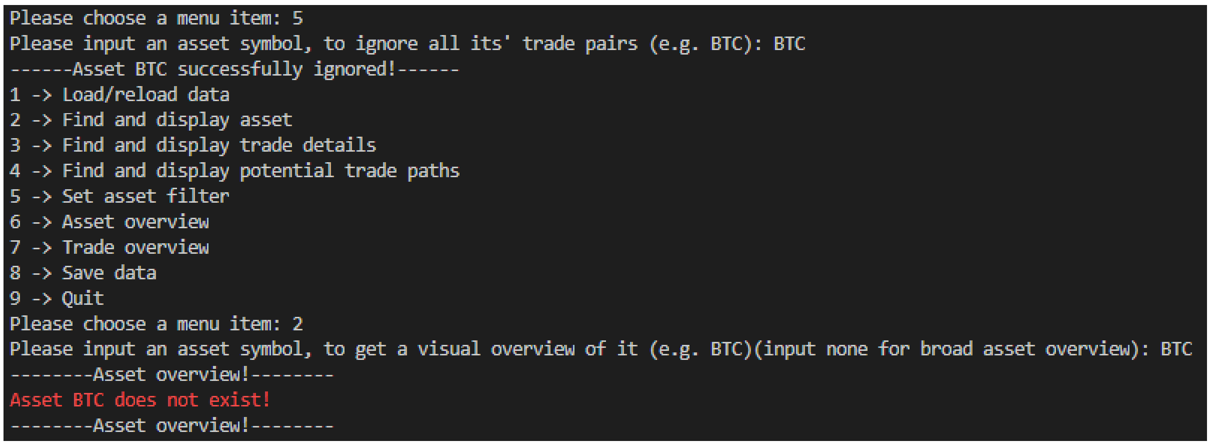
```
Please choose a menu item: 3
Please input a trade symbol to find its' details (e.g. ETHBTC): EEEEE
------Trade details found!------
Trade EEEEE does note exist!
------Trade details found!------
```

Option 4:



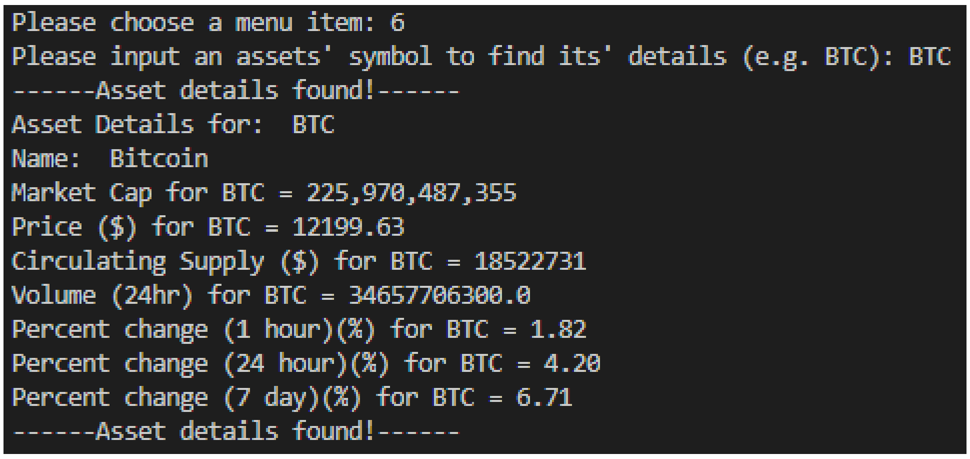
```
Please choose a menu item: 4
Please input a base asset and quote asset to display direct and indirect paths between the two:
Base asset (e.g. BTC): BTC
Quote asset (e.g. ETH): ETH
BTC -> BUSD -> DAI -> BNB -> ETH
BTC -> BUSD -> DAI -> BNB -> PAX -> ETH
BTC -> BUSD -> DAI -> BNB -> PAX -> TUSD -> ETH
BTC -> BUSD -> DAI -> BNB -> TUSD -> ETH
BTC -> BUSD -> DAI -> BNB -> USDC -> PAX -> ETH
BTC -> BUSD -> DAI -> BNB -> USDC -> PAX -> TUSD -> ETH
BTC -> BUSD -> DAI -> BNB -> USDC -> TUSD -> ETH
BTC -> BUSD -> USDT -> DAI -> BNB -> ETH
BTC -> BUSD -> USDT -> DAI -> BNB -> PAX -> ETH
BTC -> BUSD -> USDT -> DAI -> BNB -> PAX -> TUSD -> ETH
BTC -> BUSD -> USDT -> DAI -> BNB -> TUSD -> ETH
BTC -> BUSD -> USDT -> DAI -> BNB -> USDC -> PAX -> ETH
BTC -> BUSD -> USDT -> DAI -> BNB -> USDC -> PAX -> TUSD -> ETH
BTC -> BUSD -> USDT -> DAI -> BNB -> USDC -> TUSD -> ETH
BTC -> DAI -> BNB -> ETH
```

Option 5:



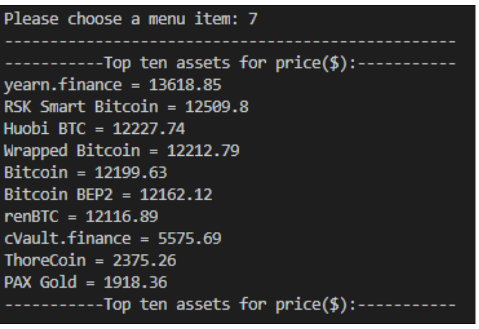
```
Please choose a menu item: 5
Please input an asset symbol, to ignore all its' trade pairs (e.g. BTC): BTC
------Asset BTC successfully ignored!------
1 -> Load/reload data
2 -> Find and display asset
3 -> Find and display trade details
4 -> Find and display potential trade paths
5 -> Set asset filter
6 -> Asset overview
7 -> Trade overview
8 -> Save data
9 -> Quit
Please choose a menu item: 2
Please input an asset symbol, to get a visual overview of it (e.g. BTC)(input none for broad asset overview): BTC
--------Asset overview!--------
Asset BTC does not exist!
--------Asset overview!--------
```

Option 6:

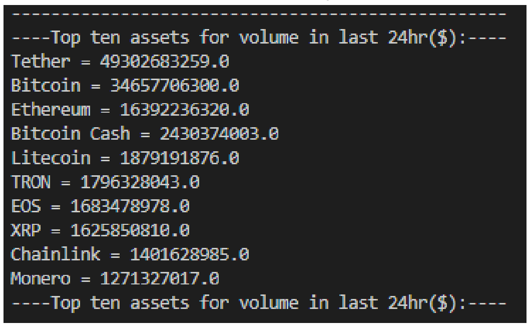


```
Please choose a menu item: 6
Please input an assets' symbol to find its' details (e.g. BTC): BTC
------Asset details found!------
Asset Details for:  BTC
Name:  Bitcoin
Market Cap for BTC = 225,970,487,355
Price ($) for BTC = 12199.63
Circulating Supply ($) for BTC = 18522731
Volume (24hr) for BTC = 34657706300.0
Percent change (1 hour)(%) for BTC = 1.82
Percent change (24 hour)(%) for BTC = 4.20
Percent change (7 day)(%) for BTC = 6.71
------Asset details found!------
```

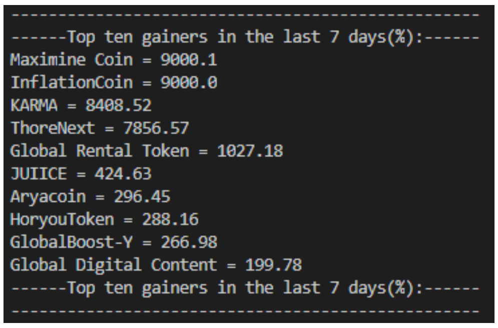
Option 7:



```
Please choose a menu item: 7
------------------------------------------------
----------Top ten assets for price($):----------
yearn.finance = 13618.85
RSK Smart Bitcoin = 12509.8
Huobi BTC = 12227.74
Wrapped Bitcoin = 12212.79
Bitcoin = 12199.63
Bitcoin BEP2 = 12162.12
renBTC = 12116.89
cVault.finance = 5575.69
ThoreCoin = 2375.26
PAX Gold = 1918.36
----------Top ten assets for price($):----------
```

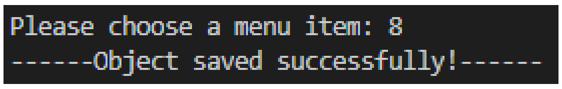


```
--------------------------------------------------
----Top ten assets for volume in last 24hr($):----
Tether = 49302683259.0
Bitcoin = 34657706300.0
Ethereum = 16392236320.0
Bitcoin Cash = 2430374003.0
Litecoin = 1879191876.0
TRON = 1796328043.0
EOS = 1683478978.0
XRP = 1625850810.0
Chainlink = 1401628985.0
Monero = 1271327017.0
----Top ten assets for volume in last 24hr($):----
```



```
------------------------------------------------
------Top ten gainers in the last 7 days(%):------
Maximine Coin = 9000.1
InflationCoin = 9000.0
KARMA = 8408.52
ThoreNext = 7856.57
Global Rental Token = 1027.18
JUIICE = 424.63
Aryacoin = 296.45
HoryouToken = 288.16
GlobalBoost-Y = 266.98
Global Digital Content = 199.78
------Top ten gainers in the last 7 days(%):------
------------------------------------------------
```

Option 8:



```
Please choose a menu item: 8
------Object saved successfully!------
```

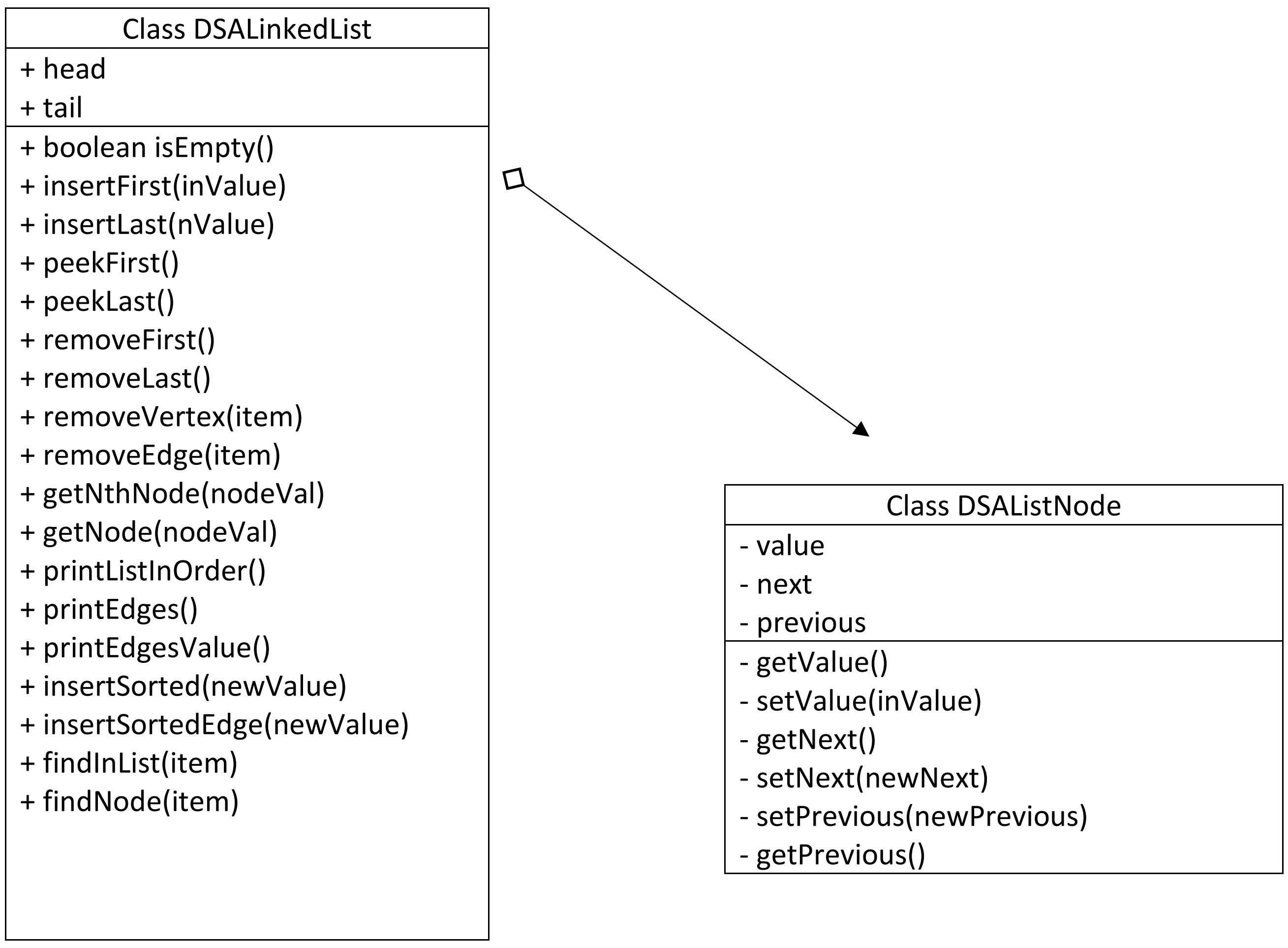
Future Work

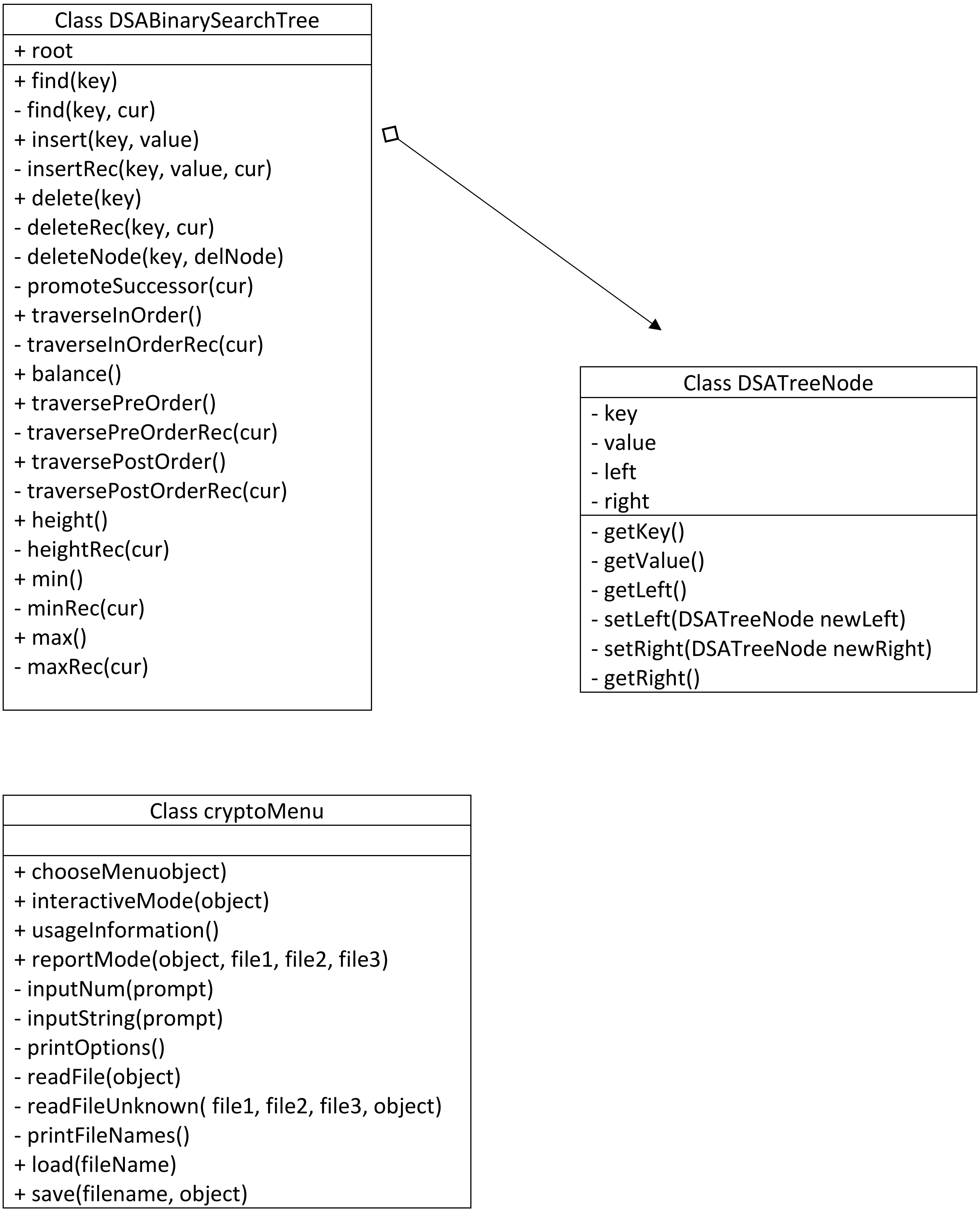
A suggested enhancement to this program would be a way to make it run faster. As seen in the justification section below, the expected run time for this program (to complete the test harness), is 4 minutes and 27 seconds. This is quite long, and a way to make this shorter may be to simplify some of the code. Another improvement is to have a way for the user to find, not only the paths between two assets, but the shortest, as well as longest paths. The user could also be told how many paths are present between those two assets.

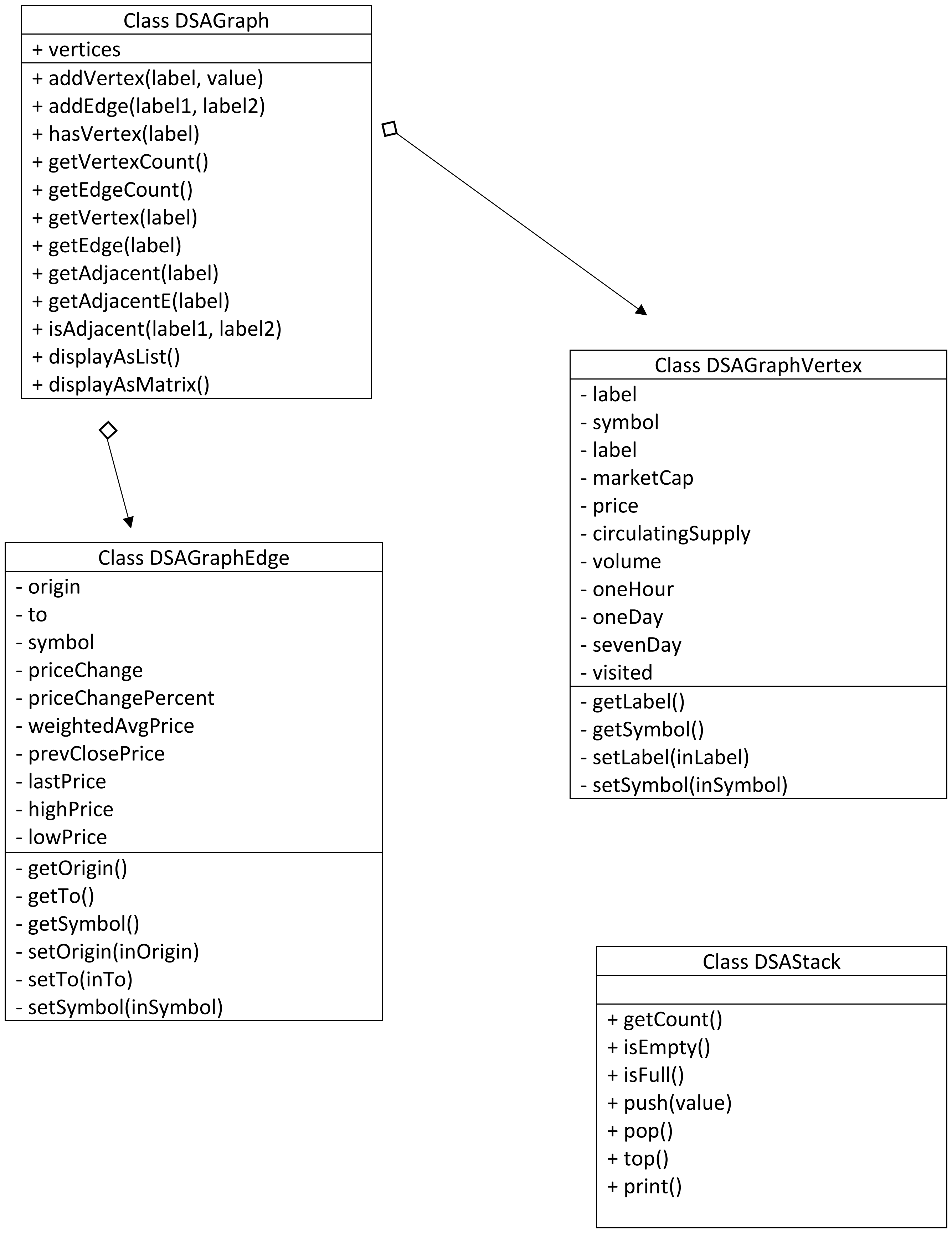
Traceability Matrix

		Requirements	Design/Code	Test
1	**Driver/Menu & Modes**	1.1 Systems displays usage if called without arguments.	graphMenu.chooseMenu()	graphMenuTestHarness()
		1.2 System displays report mode with '-r' argument.	graphMenu.chooseMenu()	graphMenuTestHarness()
		1.3 System displays interactive menu with '-I' argument.	graphMenu.chooseMenu()	graphMenuTestHarness()
		1.4 In interactive mode, user enters command and system responds.	graphMenu.interactiveMode()	graphMenuTestHarness()
		1.5 Menu loops if wrong user input.	graphMenu.interactiveMode()	graphMenuTestHarness()
		1.6 System displays error message for wrong user sub input.	graphMenu.interactiveMode()	graphMenuTestHarness()
2	**FileIO**	2.1 System uses global variables of filenames.	graphMenu._readFile()	fileIOTestHarness()
		2.2 System reads files and creates the corresponding graph.	graphMenu._readFile()	fileIOTestHarness()
		2.3 Save object to filename specified by global variable.	graphMenu.save()	fileIOTestHarness()
		2.4 Load object from filename specified by global variable.	graphMenu.load()	fileIOTestHarness()
3	**Display Asset**	3.1 Find asset and create graph of that asset	graphLinkedList.visualiseAsset()	assetAndTradeTestHarness()
4	**Trade Details**	4.1 Find trade and get corresponding trade details	graphLinkedList.getTradeDetails()	assetAndTradeTestHarness()
5	**Trade Paths**	5.1 Get adjacency list for every asset	graphLinkedList.getEdgeReverse()	assetAndTradeTestHarness()
		5.2 Use adjacency list to find all paths from a to b	graphLinkedList.printAllPaths()	assetAndTradeTestHarness()
6	**Asset Filter**	6.1 Find vertex	linkedList.removeVertex()	assetAndTradeTestHarness()
		6.2 Find all edges going to and from user inputted asset	linkedList.removeEdge()	assetAndTradeTestHarness()
		6.3 Remove all edges and Vertex for given asset	graphLinkedList.addAssetFilter()	assetAndTradeTestHarness()
7	**Asset Overview**	7.1 Find asset from user input and get corresponding trade details	graphLinkedList.getAssetDetails()	assetAndTradeTestHarness()
8	**Trade Overview**	8.1 Add all assets to binary search tree	graphLinkedList.getTopTen()	assetAndTradeTestHarness()
		8.2 Print top 10 values	graphLinkedList.getTopTen()	assetAndTradeTestHarness()
9	**Linked List**	9.1 Fully functioning linked list	linkedList()	linkedListTestHarness()
10	**Binary Search Tree**	10.1 Fully functioning Binary Search Tree	binarySearchTree()	bstTestHarness()
11	**Stack**	11.1 Fully functioning stack	DSAStack()	DSAStackTestHarness()

Class diagrams




Class DSABinarySearchTree
+ root
+ find(key)
- find(key, cur)
+ insert(key, value)
- insertRec(key, value, cur)
+ delete(key)
- deleteRec(key, cur)
- deleteNode(key, delNode)
- promoteSuccessor(cur)
+ traverseInOrder()
- traverseInOrderRec(cur)
+ balance()
+ traversePreOrder()
- traversePreOrderRec(cur)
+ traversePostOrder()
- traversePostOrderRec(cur)
+ height()
- heightRec(cur)
+ min()
- minRec(cur)
+ max()
- maxRec(cur)
Class DSATreeNode
- key
- value
- left
- right
- getKey()
- getValue()
- getLeft()
- setLeft(DSATreeNode newLeft)
- setRight(DSATreeNode newRight)
- getRight()
Class cryptoMenu
+ chooseMenuobject)
+ interactiveMode(object)
+ usageInformation()
+ reportMode(object, file1, file2, file3)
- inputNum(prompt)
- inputString(prompt)
- printOptions()
- readFile(object)
- readFileUnknown(file1, file2, file3, object)
- printFileNames()
+ load(fileName)
+ save(filename, object)


Class DSAGraph
+ vertices
+ addVertex(label, value)
+ addEdge(label1, label2)
+ hasVertex(label)
+ getVertexCount()
+ getEdgeCount()
+ getVertex(label)
+ getEdge(label)
+ getAdjacent(label)
+ getAdjacentE(label)
+ isAdjacent(label1, label2)
+ displayAsList()
+ displayAsMatrix()
Class DSAGraphVertex
- label
- symbol
- label
- marketCap
- price
- circulatingSupply
- volume
- oneHour
- oneDay
- sevenDay
- visited
- getLabel()
- getSymbol()
- setLabel(inLabel)
- setSymbol(inSymbol)
Class DSAGraphEdge
- origin
- to
- symbol
- priceChange
- priceChangePercent
- weightedAvgPrice
- prevClosePrice
- lastPrice
- highPrice
- lowPrice
- getOrigin()
- getTo()
- getSymbol()
- setOrigin(inOrigin)
- setTo(inTo)
- setSymbol(inSymbol)
Class DSAStack
+ getCount()
+ isEmpty()
+ isFull()
+ push(value)
+ pop()
+ top()
+ print()

Class descriptions

cryptoMenu

This class handles all of the user input for the program. It prompts the user and then calls functions from other classes to do certain tasks. The reason this class exists is for an easy and understandable interface in case the options need to be edited in the future.

DSAGraph

Does all of the tasks that involve creating and manipulating a graph.

DSAGraphEdge

Contains all of the data for the edges. Allow us to filter and find specific data, as well as find trade paths from a to b.

DSAGraphVertex

Contains all the data for the vertices in the graph. Allows us to store abstract data like price and circulating supply. This class and DSAGraphEdge were created as we need to store lots of data inside both the trades and the assets themselves. These allow us to store a variety of custom data.

DSABinarySearchTree

Houses the functions for inserting and manipulating data inside of the tree. Also creates the tree based on user input. This class was chosen to be included as it is a good way to sort values. Trees are relatively fast at sorting and accessing values. For accessing data, the best case is O(1), while worst and average cases are O(log N). For inserting sorted data into the tree, the best worst and average cases are all O(log N). In a tree with a lot of elements, this can take a long time, but for out program with roughly 1000 elements, we will only have 10 or 11 levels to our tree

DSATreeNode

Contains the values stored in each tree 'node' as well as the left and right value of each node, allowing us to sort and traverse the tree quickly.

DSALinkedList

Allows us to create and manipulate data inside of a linked list. Lets us access each element, find an element upon request, and delete elements.

DSAListNode

Contains the value for each 'node' in the linked list. Also contains a pointer to the next and previous nodes in the list, making it easy for us to find it.

DSAStack

Creates a stack that we can push and pop elements onto and from. This method was chosen due to a stack's LIFO property. This means it is always O(1) for accessing data, as we only access the last data point at any given time

Justification

The reason for using a graph to store all of the data for this program, is because a graph is a data structure that allows and represents connections between items. The task for this assignment was to find paths between trades, linking them to each other, which is why this data structure was chosen.
The data for the assets was originally supplied in csv format, but upon testing, it was found that it took 4x as long to import the data from the csv file into the graph. To solve this issue, the data was instead put into and converted to JSON format.

Included in the allTestHarness.py is a timer, that will time how long it takes for the program to execute. This can be seen as the expected performance of the program. From testing, this value is 4 minutes and 27 seconds. In order to replicate this test, run the allTestHarness.py file, and follow the prompts.

References

[1]
"CS240: Data Structures & Algorithms I". 2020. *Cpp.Edu*. https://www.cpp.edu/~ftang/courses/CS240/lectures/adt.htm#:~:text=An%20ADT%20is%20a%20mathematical,of%20many%20different%20data%20structures.

[2]
"JSON". 2020. *Json.Org*. https://www.json.org/json-en.html.