In mathematics, a graph can be defined as a pictorial representation or a diagram that represents data or values in an organized manner. The points on the graph often represent the relationship between two or more things.

Mathematically speaking, a graph G comprises a set of vertices and a set E of edges. Each edge in E is pair (a,b) of vertices in V. If (a,b) is an edge in E, we connect a and b in the graph drawing of G.

When speaking about size, the size of G is the number n of Vertices in V. The order of G is the number L of edges in E.

The density of G is the ration of edges in G to the maximum possible number of edges. The density function is given:

The degree of vertex a is the number of vertices to which a is linked by an edge. The degree sequence for a graph is the vector(d1,d2,…,dn)

A subgraph og G is the subset W of the vertex set V together with all of the edges that connect pairs of vertices in W.

A path from vertex a to vertex b is an ordered sequence a = v0, v1, …, vm =b

Of distinct vertices in which each adjacent pair (vj-1, vj) is linked by an edge. The length of the path is m.

A cycle is an ordered sequence a=v0, v1, …, vm=a of vertices in which each adjacent pair(vj-1, vj) of vertices is linked by an edge, and v0, v1, …, vm-1 are distinct. The length of the cycle is m.

A bipartite graph is a vertex set that can be partitioned into 2 subsets, and there are no edges linking vertices in the same set. A complete bipartite graph is a graph in which all possible edges are present.

The connectivity k(G) of a connected graph G is the minimum number of vertices that need to be removed to disconnect the graph or make it empty. A graph with more than one component has connectivity 0.

A directed graph F comprises a set V of vertices and a set R of arcs. Each arc in E is an ordered pair (a,b) of vertices in V. If (a,b) is an arc in E, we draw an arc from a to b in the graph drawing of G.

The indegree of vertex a is the number of vertices to which a is linked by an arc. The outdegree of vertex a is the number of vertices linked to a by an arc.

A path from vertex a to vertex b is an ordered sequence a=v0, v1, …, vm =b of distinct vertices in which either (vj-1, vj) is linked by an arc. The length of the semipath is m.

Two very well-known algorithms found in the literature are Depth First Search and Breadth first Search.

Well, starting with the former, a standard DFS implementation puts each vertex of the graph into one of the two categories:

1. Visited.
2. Not Visited

The purpose of the algorithm is mark each vertex as visited while avoiding cycles. The DFS algorithms works as follows:

1. Start by putting any one of the graph’s vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex’s adjacent nodes. Add the ones which are not in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

An example implemented in python is the following one:

# DFS algorithm in Python

# DFS algorithm

def dfs(graph, start, visited=None):

if visited is None:

visited = set()

visited.add(start)

print(start)

for next in graph[start] - visited:

dfs(graph, next, visited)

return visited

graph = {'0': set(['1', '2']),

'1': set(['0', '3', '4']),

'2': set(['0']),

'3': set(['1']),

'4': set(['2', '3'])}

dfs(graph, '0')

A standard BFS implementation puts each vertex of the graph onto one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles. The algorithm works as follows:

1. Start by putting any one of the graph’s vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex’s adjacent nodes. Add the ones which are not in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

An example of BFS in python is the following one:

# BFS algorithm in Python

import collections

# BFS algorithm

def bfs(graph, root):

visited, queue = set(), collections.deque([root])

visited.add(root)

while queue:

# Dequeue a vertex from queue

vertex = queue.popleft()

print(str(vertex) + " ", end="")

# If not visited, mark it as visited, and

# enqueue it

for neighbour in graph[vertex]:

if neighbour not in visited:

visited.add(neighbour)

queue.append(neighbour)

if \_\_name\_\_ == '\_\_main\_\_':

graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}

print("Following is Breadth First Traversal: ")

bfs(graph, 0)

Directed Cyclic Graph.

So, let us construct the path finding. Considering a G that consists of V number of vertices and E number of edges, we assume that V is the number of coins in the uniswap analysis and E is the number of trading pairs.

Moving on the example in uniswap analysis regarding the path finding the implementation of DFS is the following one in python as found in the GitHub repository:

Defining a function in python which takes seven arguments as an input. Therefore, it iterates the number of trading pairs in the graph as a length in for loop. For every iteration a path is inserted into the newPath variable. At the variable pair we iteratively set the trading pairs each time. Therefore we examine four different cases in order to finally return the appropriate the circle path within the trading pairs. So, starting with the first one, we examine the case where the trading pair of the the first token and address found in the graph’s vertex and the second ones are not equal meaning not found in the graph’s circle therefore we continue to the algorithm. Furthermore, if the trading par of the reserve amount of money divided by the power of 10 of the trading pair found in the first token is less than one or the trading pair of the second reserve divided by the trading pair divided by the pair ot the second token is less than one then we continue to our algorithm. Moving to the third statement, in case the token of the trading address is equal to the pair of the graph’s path of the first address’ token therefore we insert into a temporary variable the second trading pair of the address’ token. If not, we insert the first one.

If the address’ temporary variable if equa to the tokenOut of the trading address as given as an argument and the length of the path is more than 2 in the graph therefore we set a variable c the routes of the current trading pairs and the newpath so far. In addition, we append the corresponding circle to the circles graph. If the aforementioned method is invalid and the maximum Hops given as an argument is more than 1 and the length of the trading pairs is more than 1 therefore we set into a new variable the number of rows found in the pairs path every time they appear summed by the next, following number of rows appearing in the circle.

Therefore, we call recursively the function. In programming language, when mentioning that aa function is called recursively that means that it is called into the function with the same name but with different arguments so that new values are inserted. Apparently, the goal over here is to construct a trading pair and find the best path of the trading pairs into the graph. Last but not least, the final graph is returned.

Regarding the optimal input amount we set the parameters of R0 and R1 as reserve of the trading pairs and Δα and Δb as the parameters of the amount of A to trade Δb αμοθντ οφ Β ασσθμινγ τηατ τηε φεε ις 1-ρ. Feeling the following equation it gives us a constant on both of the sides as follows.

In the trading pair of Δα and Δβ multiplied by r fee we add the first Reserve A of the trading pair. Therefore, we multiple that quantity by subtracting the Δβ from the reserve for B which is R1. That constantly all the time remains the same.

Regarding the example of A to B to C to A assuming that D is returning to A, in order to get the maximum optimal representation profit amongst the trading pairs we find the derivative between the subtract of Δα and Δα derivative subject to the following reserves of the trading pairs are greater than 0 and the amount of N for each amount is greater than zero. Therefore, iteratively, we get constant marker function providing the reserves the amount of N for trading N+1 regarding the trading fees.

Considering the case that we would like to trade A to C, we may use B as a bridge into the equations. That is simple called virtual trading pool for A and C. We solve the equations of 4 and 6 accordingly as found in the literature. The goal is to keep reserves zero, one and two into the equation so that we set them as E0 and E1 thereafter. The goal is to find the amount of trade C which is the final one proportional to the initial amount of trade which is a into the equation considering the cases that the reserves are kept as they are into the equation.

In addition, in order to find the optimal path from A to A as mentioned in the documentation clearly , the Δα prime is found through E0 and E1 accordingly based on the general approved equation no. 6. Therefore, f is the profit calculated its derivative which gives us the final optimal input amount.