

VAIBHAV MISHRA

RA1911033010075

## EXPERIMENT No. 4A - IMPLEMENTATION OF BFS

AIM.

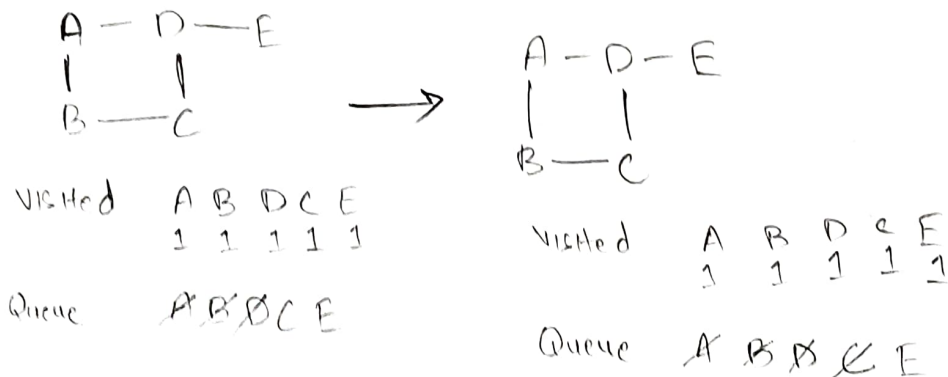
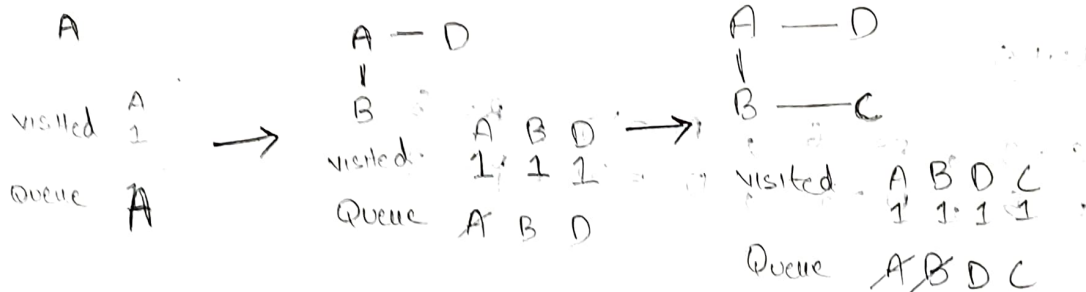
To implement BFS in python

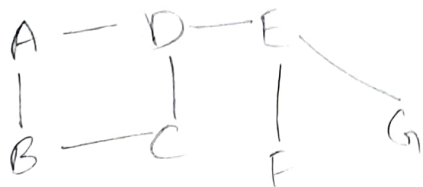
### ALGORITHM

1. Create a queue
2. Mark each new node as visited and put that node into the queue
3. While Queue is non-empty
4. Remove the head of queue
5. Mark and enqueue all (unvisited) neighbors

### ILLUSTRATION.

BFS



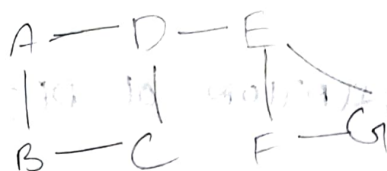


visited A B D C E F G  
1 1 1 1 1 1 1

visited A B D C E F G  
1 1 1 1 1 1 1

Queue A B ~~D~~ ~~C~~ ~~E~~ F G

Queue A B ~~D~~ ~~C~~ ~~E~~ F G

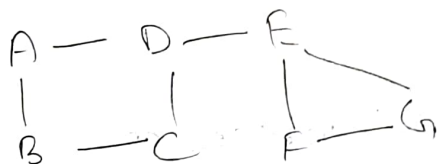


visited A B D C E F G  
1 1 1 1 1 1 1

Queue A B ~~D~~ ~~C~~ ~~E~~ F G

BFS Traversal: A B D C E F G

DFS



A  
visited  
Stack

A  
1  
B  
D



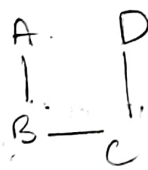
A  
B  
visited  
Stack

A B  
1 1  
C  
D

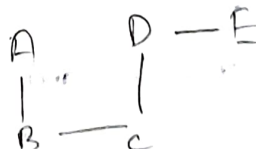


A  
B  
C  
visited  
Stack

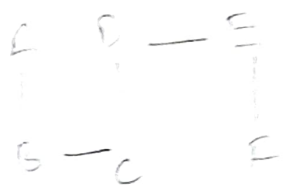
A B C  
1 1 1  
D



visited A B C D  
1 1 1 1  
Stack E



visited A B C D E  
1 1 1 1 1  
Stack F G



visited: A B C D E F  
 1 1 1 1 1 1

Stack  
 [G]



visited: A B C D E F G  
 1 1 1 1 1 1 1

stack  
 [ ]

## EXPERIMENT No. 4b - IMPLEMENTATION OF DFS

### AIM

To implement DFS in python

20/02/2023

### ALGORITHM

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of 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.
4. Add the ones which aren't in the visited list to the top of the stack
5. Keep repeating steps 2 and 3. until, the stack is empty.

# EXPERIMENT No. 4c - IMPLEMENTATION OF UCS

## AIM

To calculate the shortest path with cost for the given problem using UCS. (Uniform Cost Search)

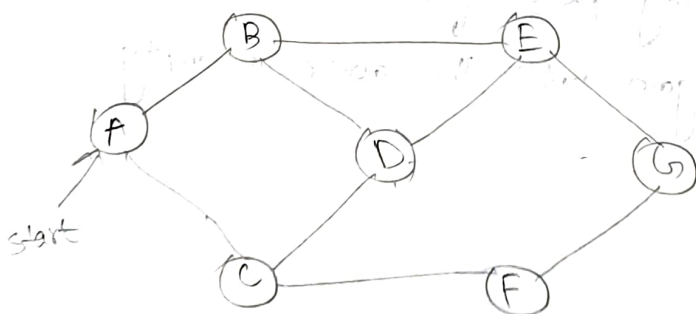
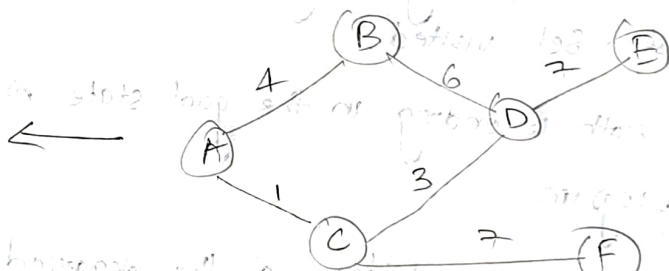
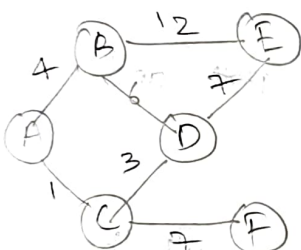
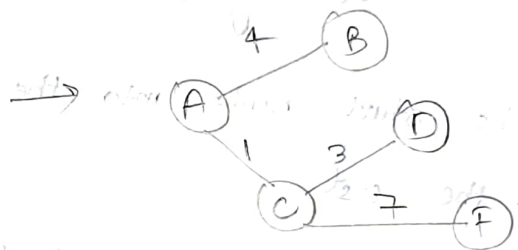
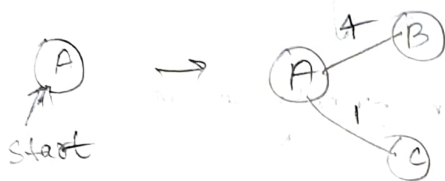
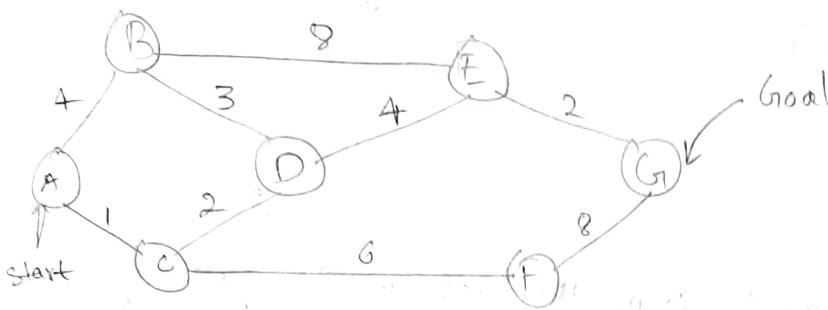
## ALGORITHM

1. Start
2. Get the graph as an input from the user in the form of (node, adjacent, node, weights). Get start and goal node also in the input.
3. Insert the start node into the main priority queue along with the cost.
4. Dequeue the maximum priority element from the priority queue. Here the maximum priority is given to the element that has minimum cost. Set visited.
5. If the path is ending in the goal state print the path and exit the program.
6. Else insert all the children of the dequeued element, with the cumulative costs in the priority queue if not visited.
7. Repeat given step 4 again until the queue is empty.
8. Stop.

## RESULT

BFS, DFS and UCS was successfully implemented in Python.

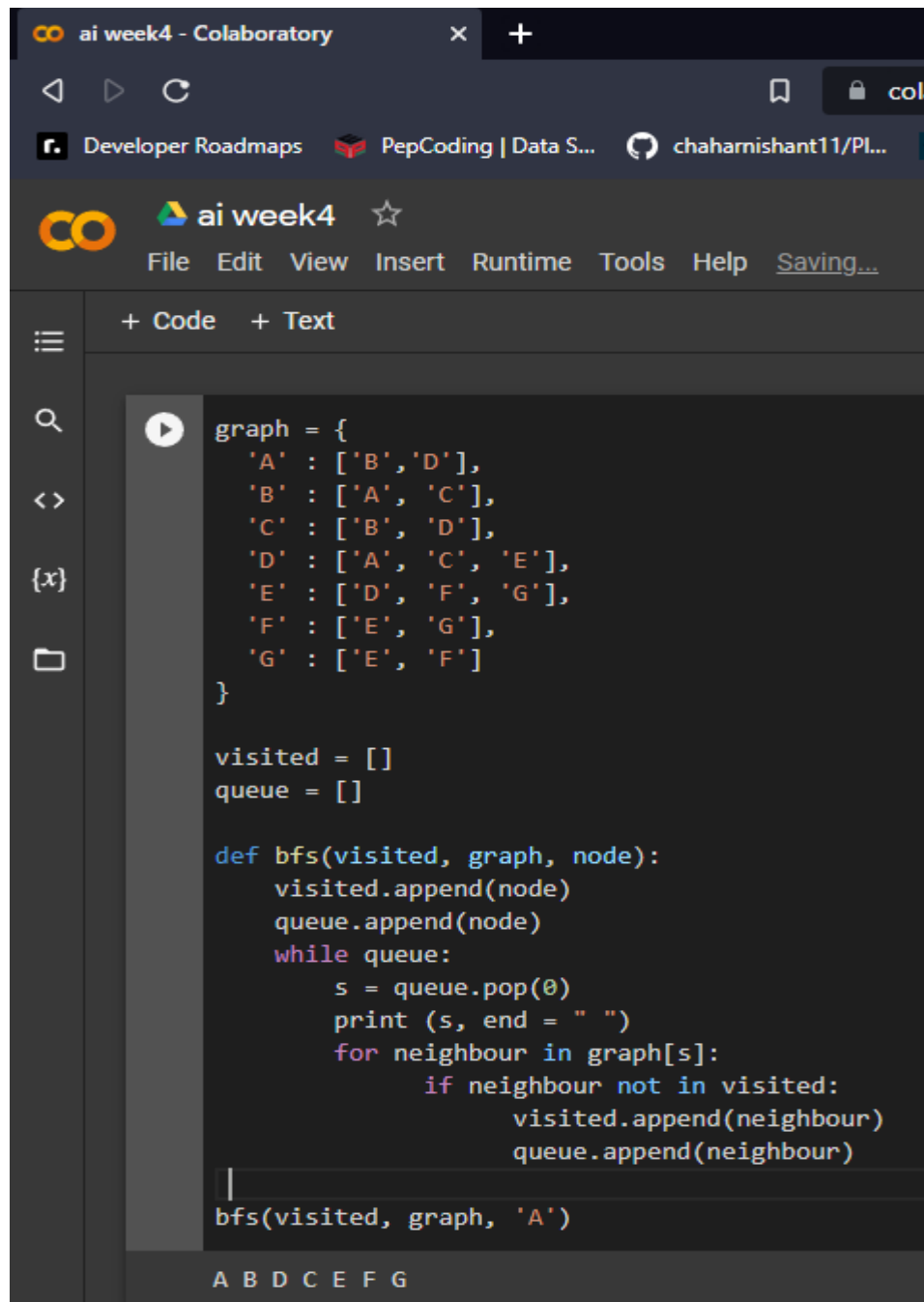
# I-LLUSTRATION



Minimum Cost = 9

Path = [A, C, D, E, G]

## BFS



The image shows a Google Colaboratory notebook interface. The browser tab is titled "ai week4 - Colaboratory". The notebook has a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar, there are tabs for "+ Code" and "+ Text". The code editor contains a Python script for a Breadth-First Search (BFS) algorithm. The graph is defined as follows:

```
graph = {
    'A' : ['B', 'D'],
    'B' : ['A', 'C'],
    'C' : ['B', 'D'],
    'D' : ['A', 'C', 'E'],
    'E' : ['D', 'F', 'G'],
    'F' : ['E', 'G'],
    'G' : ['E', 'F']
}
```

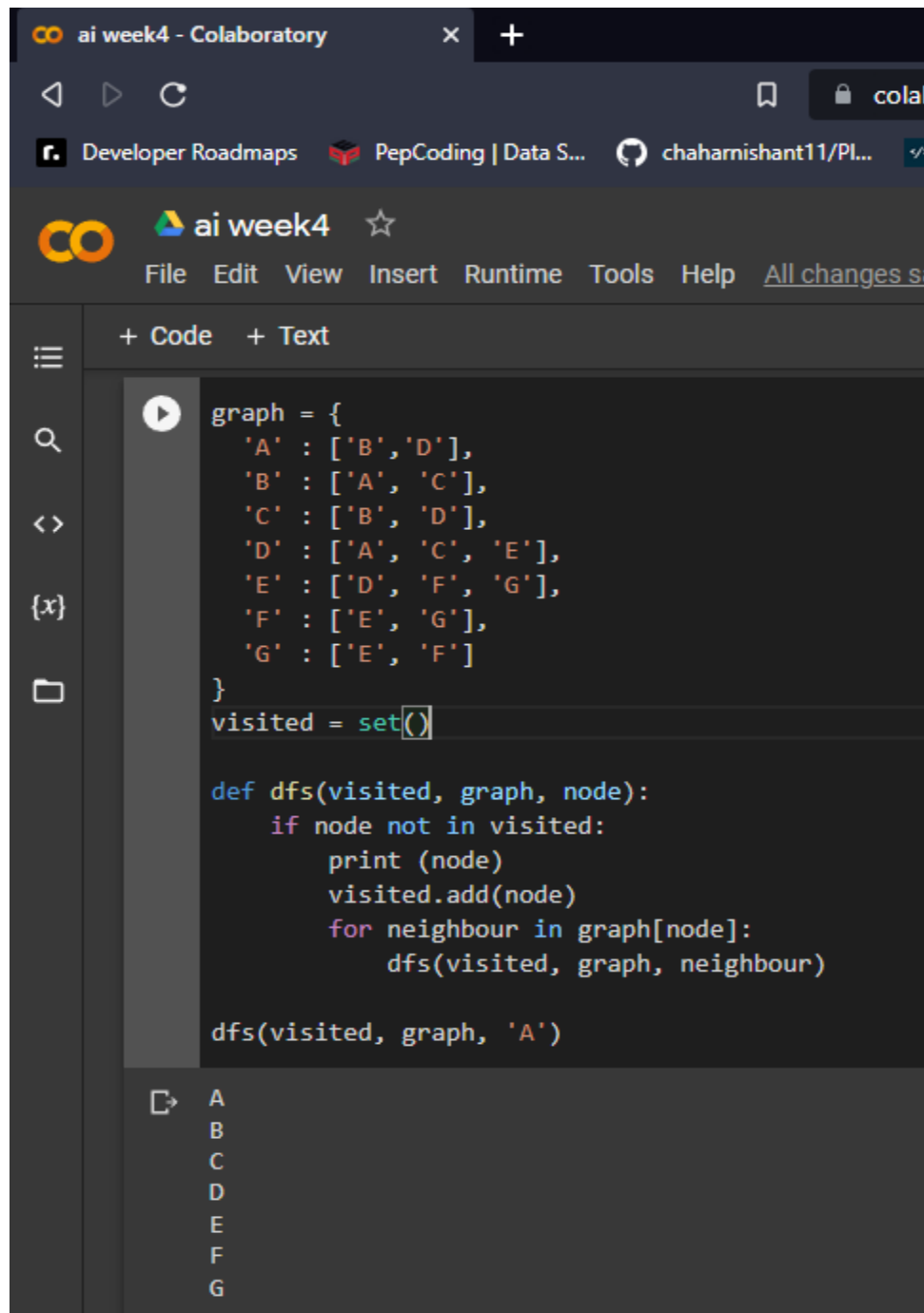
The BFS function is defined as:

```
visited = []
queue = []

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        s = queue.pop(0)
        print (s, end = " ")
        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
    |
bfs(visited, graph, 'A')
```

The output of the BFS algorithm is displayed at the bottom of the notebook: A B D C E F G.

## DFS



The screenshot shows a Google Colaboratory notebook titled "ai week4 - Colaboratory". The notebook contains a Python script for a Depth-First Search (DFS) algorithm on an undirected graph. The graph is defined as a dictionary where keys are nodes and values are lists of adjacent nodes. The script initializes a set to track visited nodes, defines a recursive DFS function, and then calls the function starting from node 'A'. The output of the script is a list of nodes: A, B, C, D, E, F, G.

```
graph = {
    'A' : ['B', 'D'],
    'B' : ['A', 'C'],
    'C' : ['B', 'D'],
    'D' : ['A', 'C', 'E'],
    'E' : ['D', 'F', 'G'],
    'F' : ['E', 'G'],
    'G' : ['E', 'F']
}

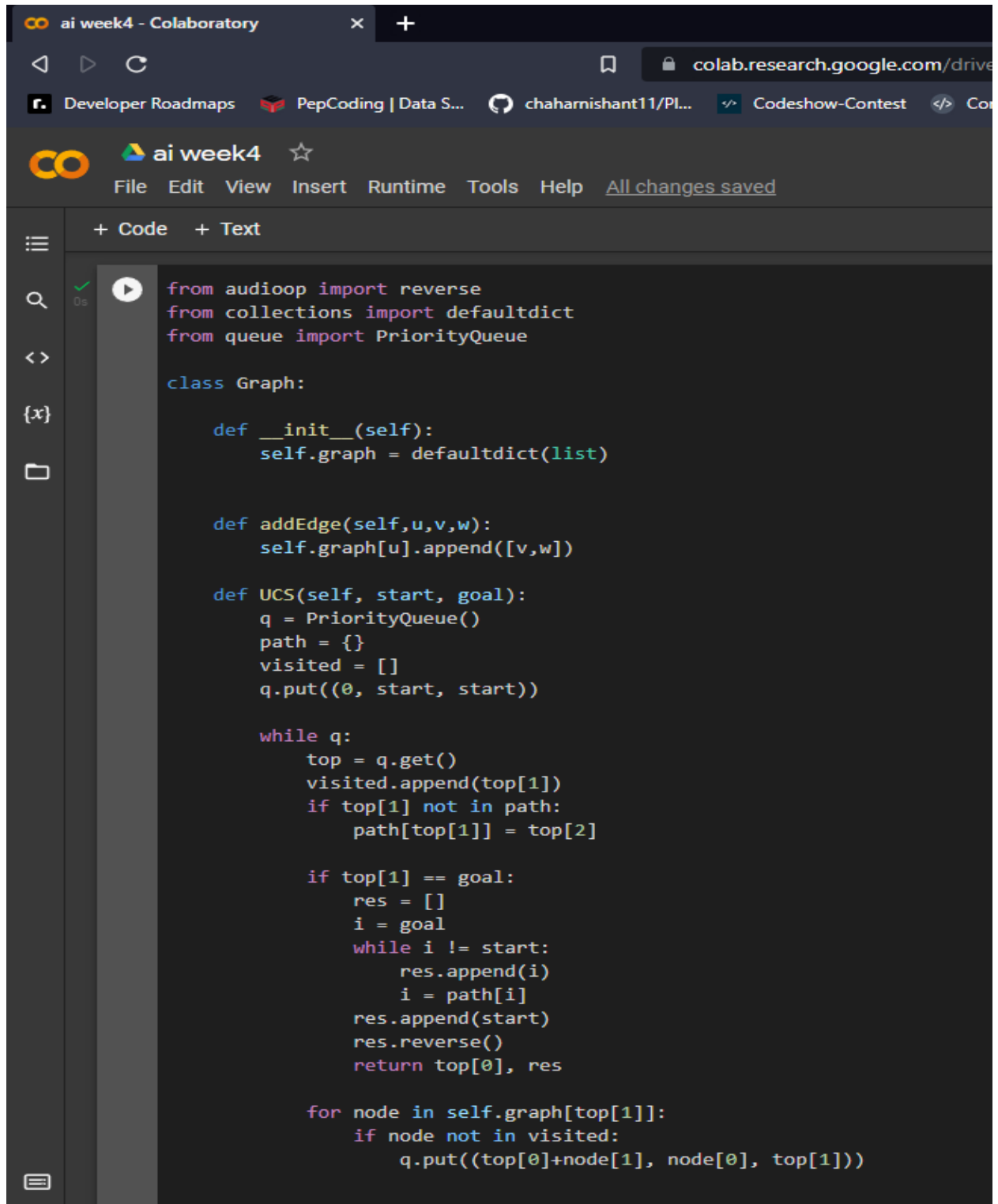
visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

dfs(visited, graph, 'A')
```

A  
B  
C  
D  
E  
F  
G

## UCS



```
from audioop import reverse
from collections import defaultdict
from queue import PriorityQueue

class Graph:

    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self,u,v,w):
        self.graph[u].append([v,w])

    def UCS(self, start, goal):
        q = PriorityQueue()
        path = {}
        visited = []
        q.put((0, start, start))

        while q:
            top = q.get()
            visited.append(top[1])
            if top[1] not in path:
                path[top[1]] = top[2]

            if top[1] == goal:
                res = []
                i = goal
                while i != start:
                    res.append(i)
                    i = path[i]
                res.append(start)
                res.reverse()
                return top[0], res

            for node in self.graph[top[1]]:
                if node not in visited:
                    q.put((top[0]+node[1], node[0], top[1]))
```



```
g = Graph()
g.addEdge('A', 'B', 4)
g.addEdge('A', 'C', 1)
g.addEdge('B', 'D', 3)
g.addEdge('B', 'E', 8)
g.addEdge('C', 'D', 2)
g.addEdge('C', 'F', 6)
g.addEdge('D', 'E', 4)
g.addEdge('E', 'G', 2)
g.addEdge('F', 'G', 8)

print ("Uniform Cost Search: ")
start = 'A'
end = 'G'
minCost, path = g.UCS(start, end)
print("Minimum cost from %s to %s => " % (start,end), minCost)
print("Path of traversal => ", path)
```

```
Uniform Cost Search:
Minimum cost from A to G => 9
Path of traversal => ['A', 'C', 'D', 'E', 'G']
```