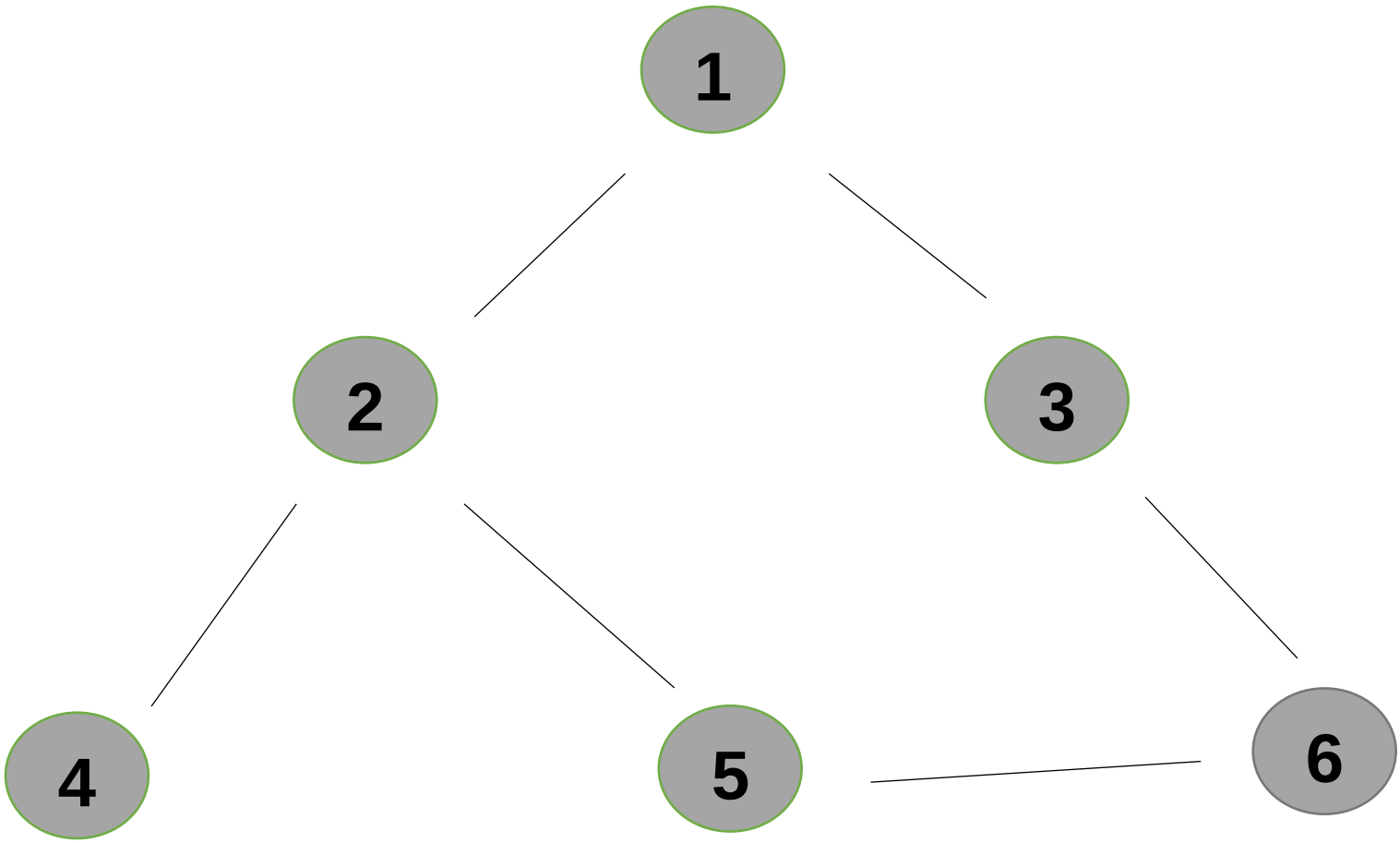
AI EX 4

BFS AND DFS

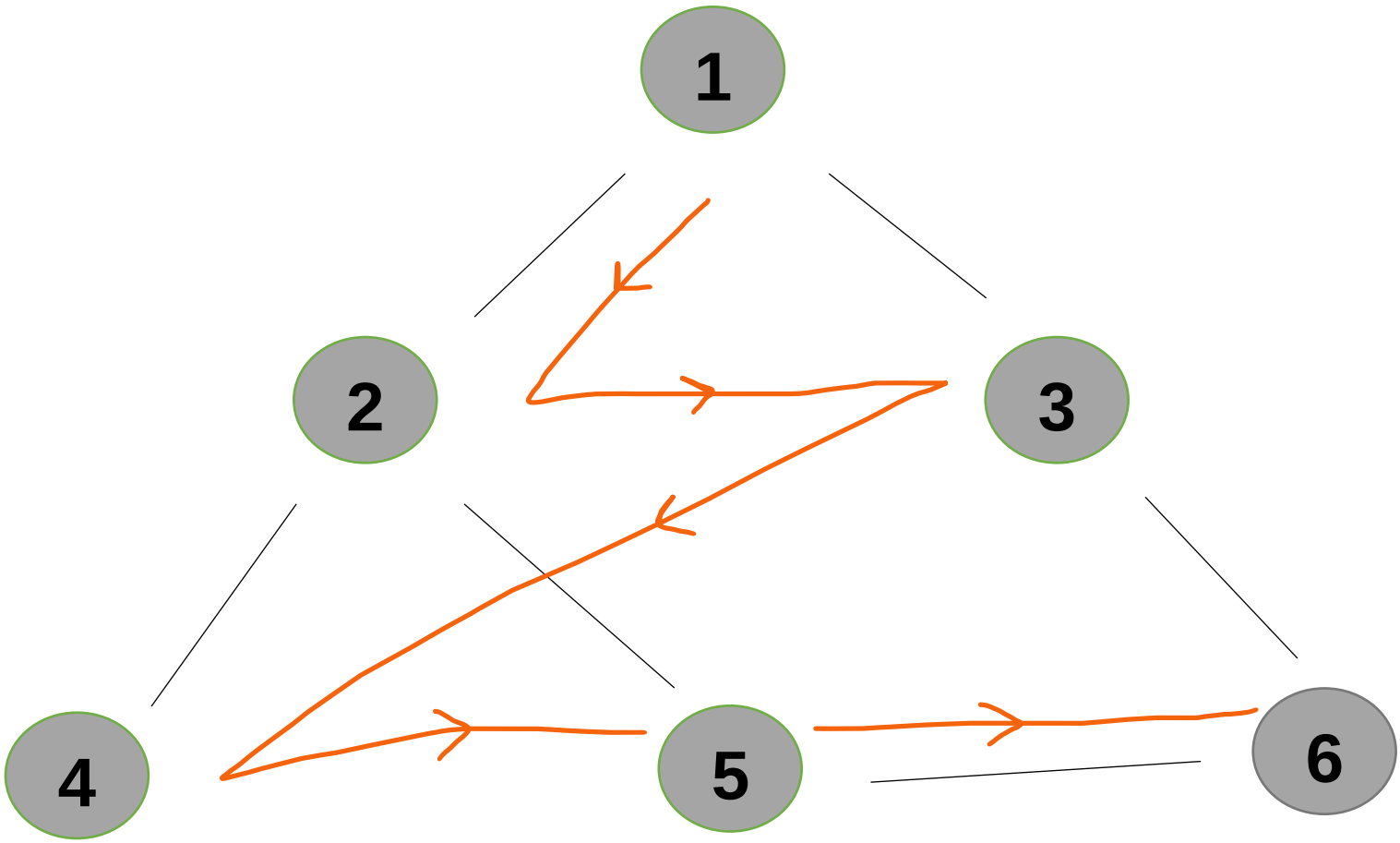
AIM: To create a Graph and implement traversal techniques (BFS & DFS).

GRAPH:



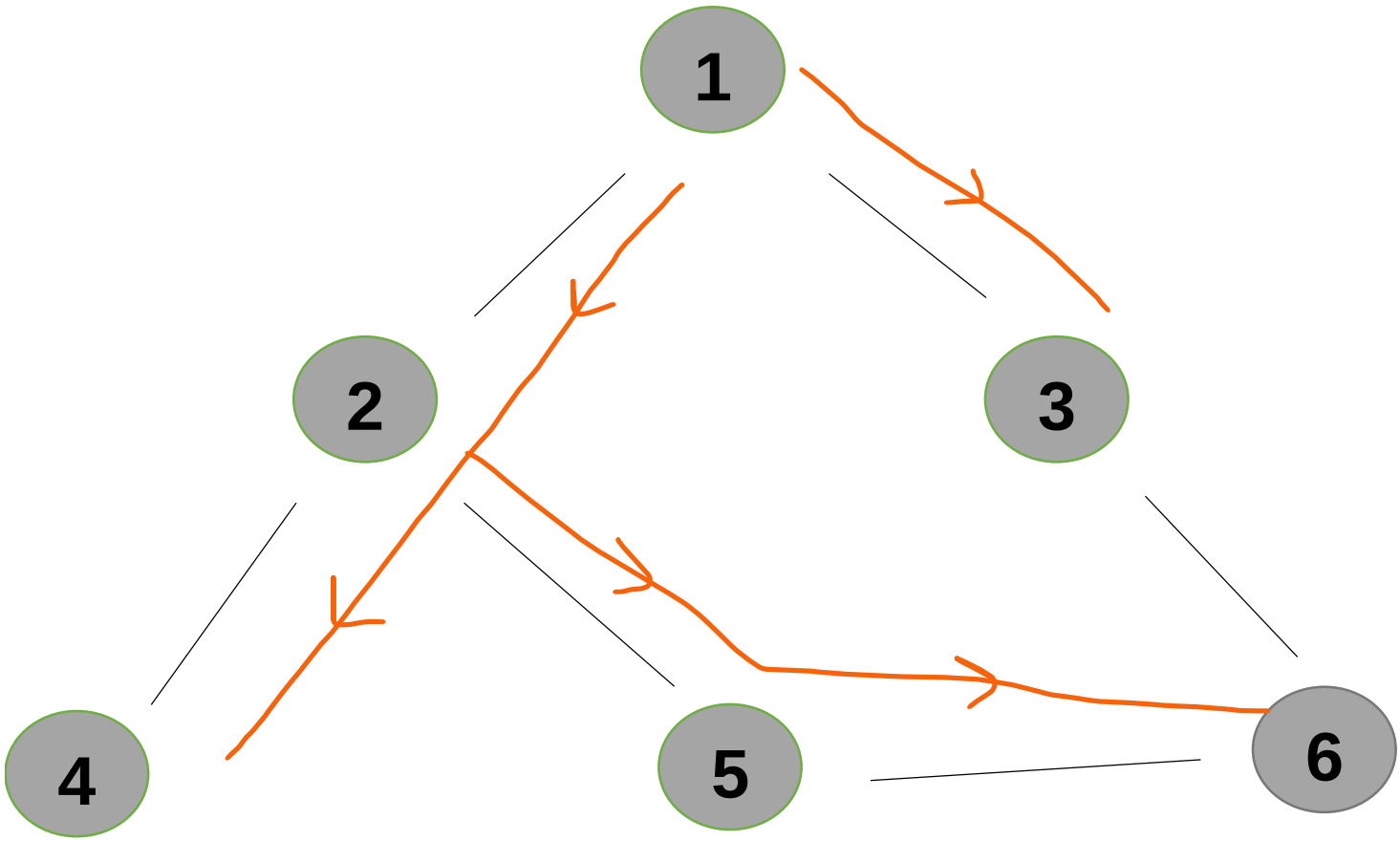
BFS:

- Define a Queue of size total number of vertices in the graph.
- Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
- Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.
- When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
- Repeat steps 3 and 4 until queue becomes empty.
- When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

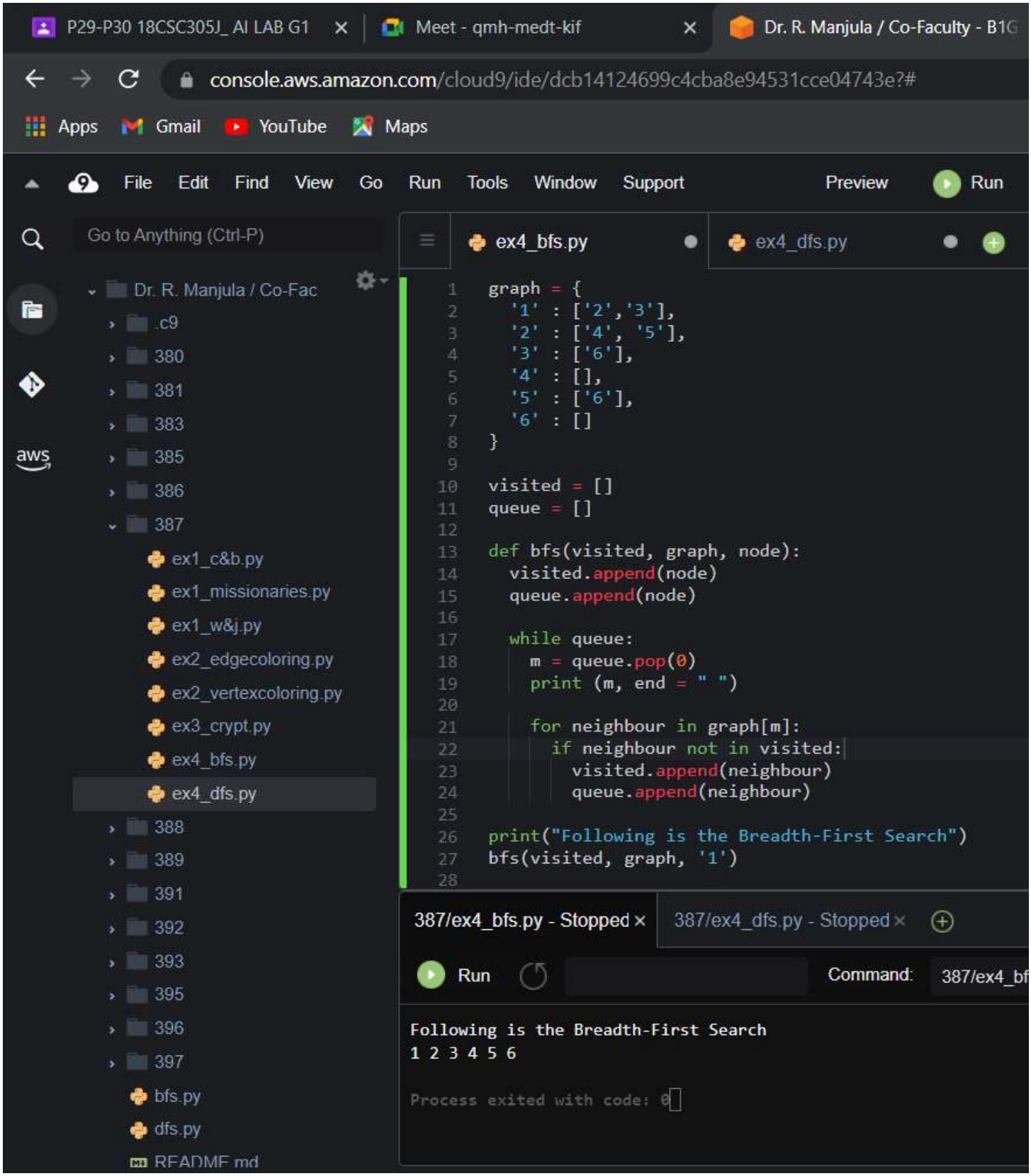


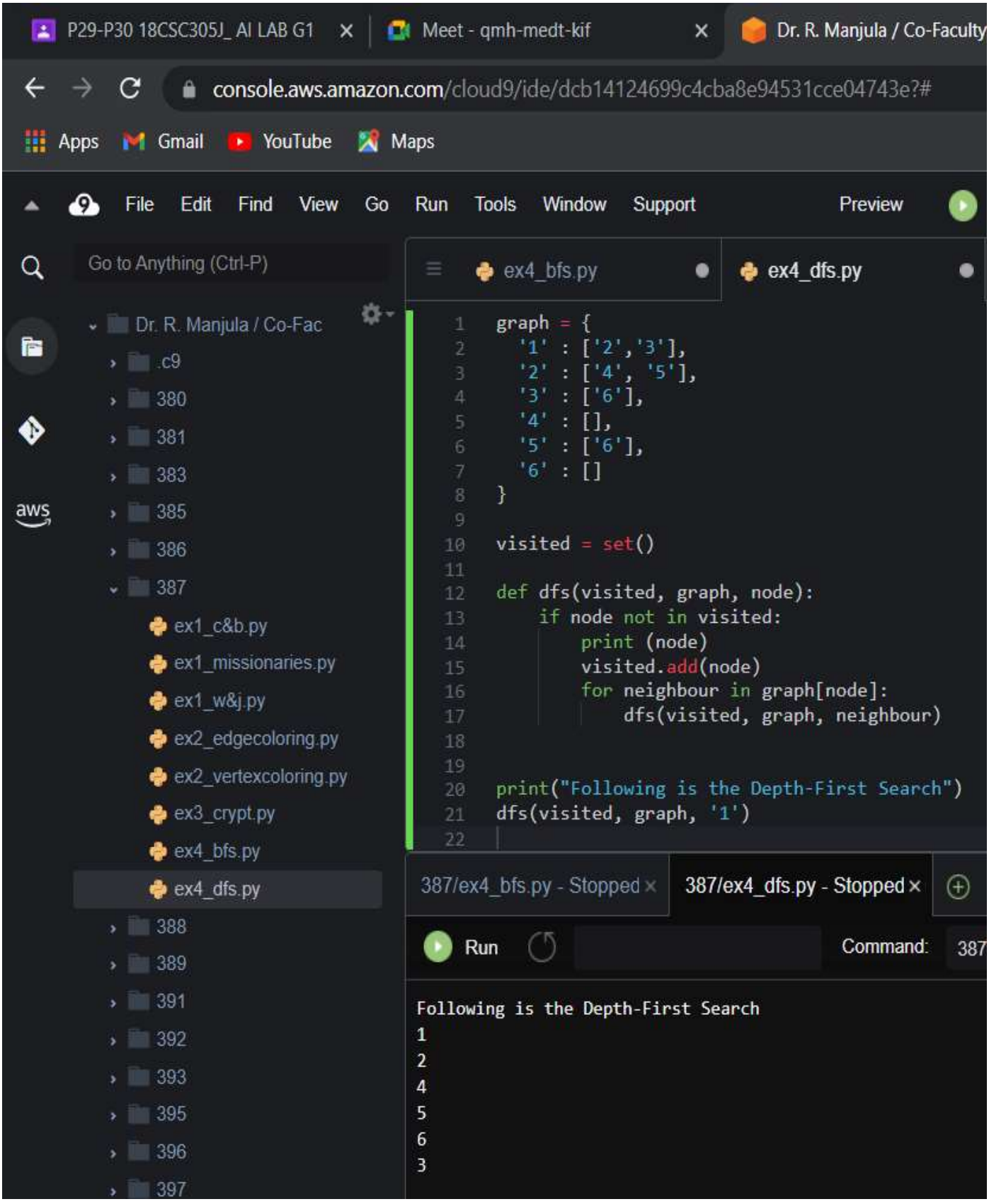
DFS:

- Define a Stack of size total number of vertices in the graph.
- Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
- Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.
- Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
- When there is no new vertex to visit then use back tracking and pop one vertex from the stack.
- Repeat steps 3, 4 and 5 until stack becomes Empty.
- When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph



OUTPUT:


P29-P30 18CSC305J_ AI LAB G1
Meet - qmh-medt-kif
Dr. R. Manjula / Co-Faculty - B1G
console.aws.amazon.com/cloud9/ide/dcb14124699c4cba8e94531cce04743e?#
Apps
Gmail
YouTube
Maps
File Edit Find View Go Run Tools Window Support
Preview
Run
Go to Anything (Ctrl-P)
Dr. R. Manjula / Co-Fac
.c9
380
381
383
385
386
387
ex1_c&b.py
ex1_missionaries.py
ex1_w&j.py
ex2_edgecoloring.py
ex2_vertexcoloring.py
ex3_crypt.py
ex4_bfs.py
ex4_dfs.py
388
389
391
392
393
395
396
397
bfs.py
dfs.py
README.md
ex4_bfs.py
ex4_dfs.py
graph = {
'1' : ['2','3'],
'2' : ['4', '5'],
'3' : ['6'],
'4' : [],
'5' : ['6'],
'6' : []
}
visited = []
queue = []
def bfs(visited, graph, node):
visited.append(node)
queue.append(node)
while queue:
m = queue.pop(0)
print (m, end = " ")
for neighbour in graph[m]:
if neighbour not in visited:
visited.append(neighbour)
queue.append(neighbour)
print("Following is the Breadth-First Search")
bfs(visited, graph, '1')
387/ex4_bfs.py - Stopped
387/ex4_dfs.py - Stopped
Run
Command:
387/ex4_bf
Following is the Breadth-First Search
1 2 3 4 5 6
Process exited with code: 0


P29-P30 18CSC305J_ AI LAB G1
Meet - qmh-medt-kif
Dr. R. Manjula / Co-Faculty
console.aws.amazon.com/cloud9/ide/dcb14124699c4cba8e94531cce04743e?#
Apps
Gmail
YouTube
Maps
File Edit Find View Go Run Tools Window Support Preview
Go to Anything (Ctrl-P)
Dr. R. Manjula / Co-Fac
.c9
380
381
383
385
386
387
ex1_c&b.py
ex1_missionaries.py
ex1_w&j.py
ex2_edgecoloring.py
ex2_vertexcoloring.py
ex3_crypt.py
ex4_bfs.py
ex4_dfs.py
388
389
391
392
393
395
396
397
ex4_bfs.py
ex4_dfs.py
graph = {
'1' : ['2','3'],
'2' : ['4', '5'],
'3' : ['6'],
'4' : [],
'5' : ['6'],
'6' : []
}
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)
print("Following is the Depth-First Search")
dfs(visited, graph, '1')
387/ex4_bfs.py - Stopped
387/ex4_dfs.py - Stopped
Run
Command:
387
Following is the Depth-First Search
1
2
4
5
6
3

CODE:

BFS:

```
graph = {
  '1' : ['2','3'],
  '2' : ['4', '5'],
  '3' : ['6'],
  '4' : [],
  '5' : ['6'],
  '6' : []
}

visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, '1')
```

DFS:

```
graph = {
  '1' : ['2','3'],
  '2' : ['4', '5'],
  '3' : ['6'],
  '4' : [],
  '5' : ['6'],
  '6' : []
}

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)



print("Following is the Depth-First Search")
dfs(visited, graph, '1')
```

RESULT: The BFS and DFS Problem was implemented successfully where the output is displayed on executing the program

RA1911003010387

CHITRALEKHA.CH