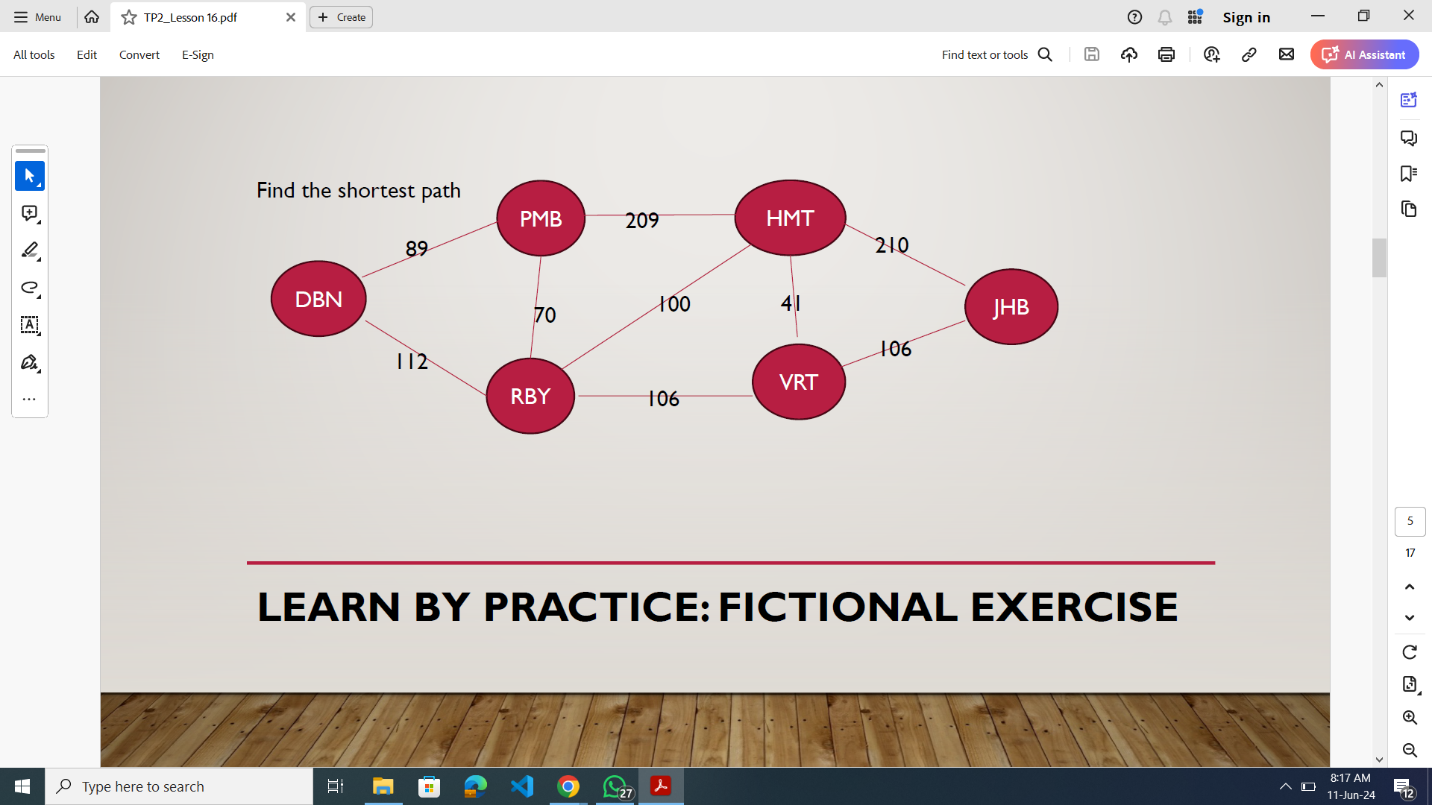
**Technical programming 2**

21938983 Cenga AE

22120426 Gumede AO

21922079 Mgenge Sf



• Firstly we identified Johannesburg(JHB) and Durban(DBN).

• Then we used the Dijkstra’s algorithm to find the shortest path between the two(JHB & DBN)

import heapq

def dijkstra(graph, start, end):

queue = [(0, start)]

distances = {node: float('infinity') for node in graph}

distances[start] = 0

previous\_nodes = {node: None for node in graph}

while queue:

current\_distance, current\_node = heapq.heappop(queue)

if current\_distance > distances[current\_node]:

continue

for neighbor, weight in graph[current\_node].items():

distance = current\_distance + weight

if distance < distances[neighbor]:

distances[neighbor] = distance

previous\_nodes[neighbor] = current\_node

heapq.heappush(queue, (distance, neighbor))

path = []

current\_node = end

while current\_node is not None:

path.append(current\_node)

current\_node = previous\_nodes[current\_node]

path = path[::-1]

if distances[end] == float('infinity'):

return None, float('infinity')

return path, distances[end]

# Graph representation

graph = {

'DBN': {'PMB': 89, 'RBY': 112},

'PMB': {'RBY': 70, 'HMT': 209},

'RBY': {'PMB': 70, 'HMT': 100, 'VRT': 106},

'HMT': {'JHB': 210, 'VRT': 41},

'VRT': {'JHB': 106},

'JHB': {}

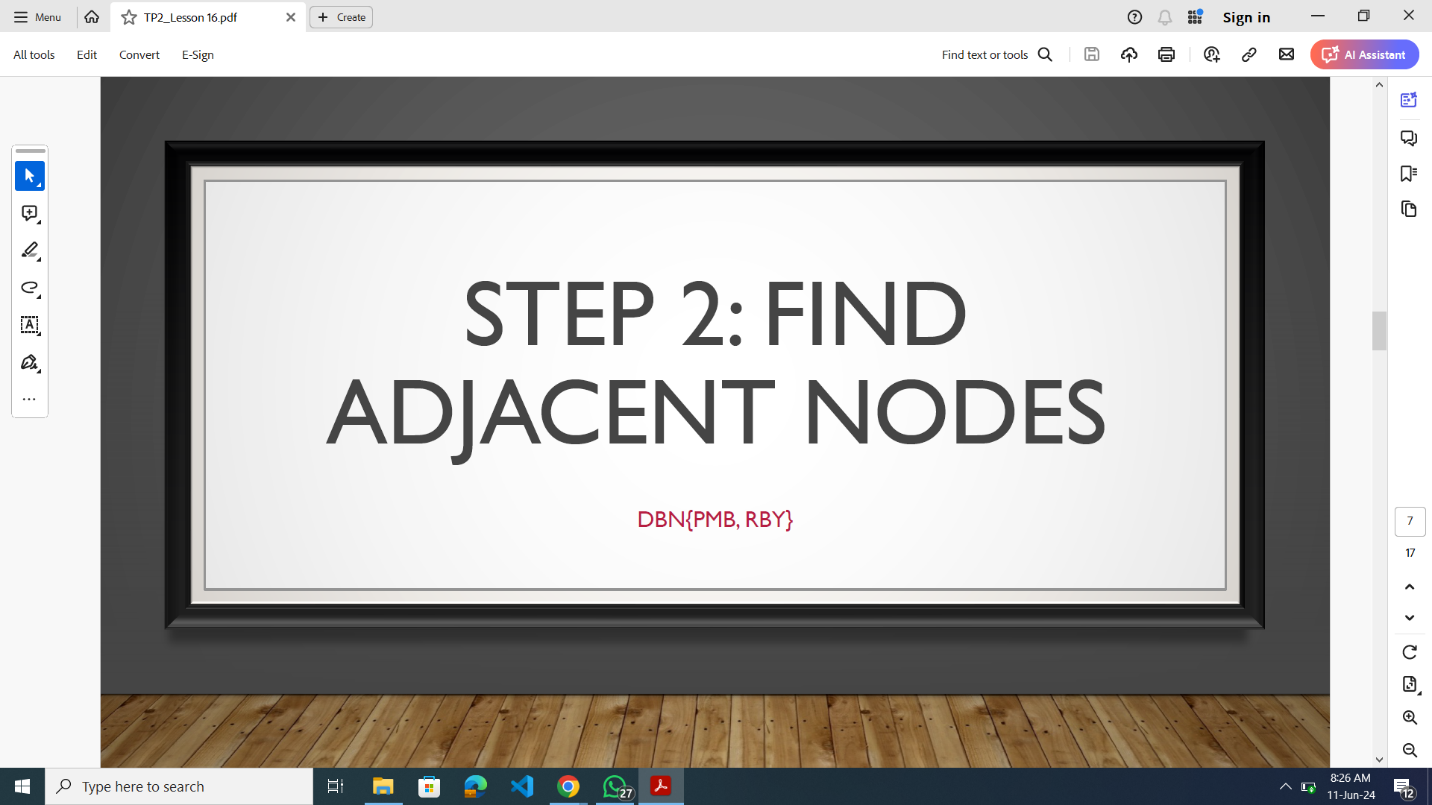
}

# Here we are finding the shortest path from DBN to JHB

path, distance = dijkstra(graph, 'DBN', 'JHB')

print(f"Shortest path: {path}")

print(f"Distance: {distance}")



• Secondly we wanted to find the adjacent nodes for the given node in the grapgh, the nodes were DBN,PMB, and RBY.

def find\_adjacent\_nodes(graph, node):

if node in graph:

adjacent\_nodes = graph[node].keys()

return adjacent\_nodes

else:

return None

# Graph representation

graph = {

'DBN': {'PMB': 89, 'RBY': 112},

'PMB': {'RBY': 70, 'HMT': 209},

'RBY': {'PMB': 70, 'HMT': 100, 'VRT': 106},

'HMT': {'JHB': 210, 'VRT': 41},

'VRT': {'JHB': 106},

'JHB': {}

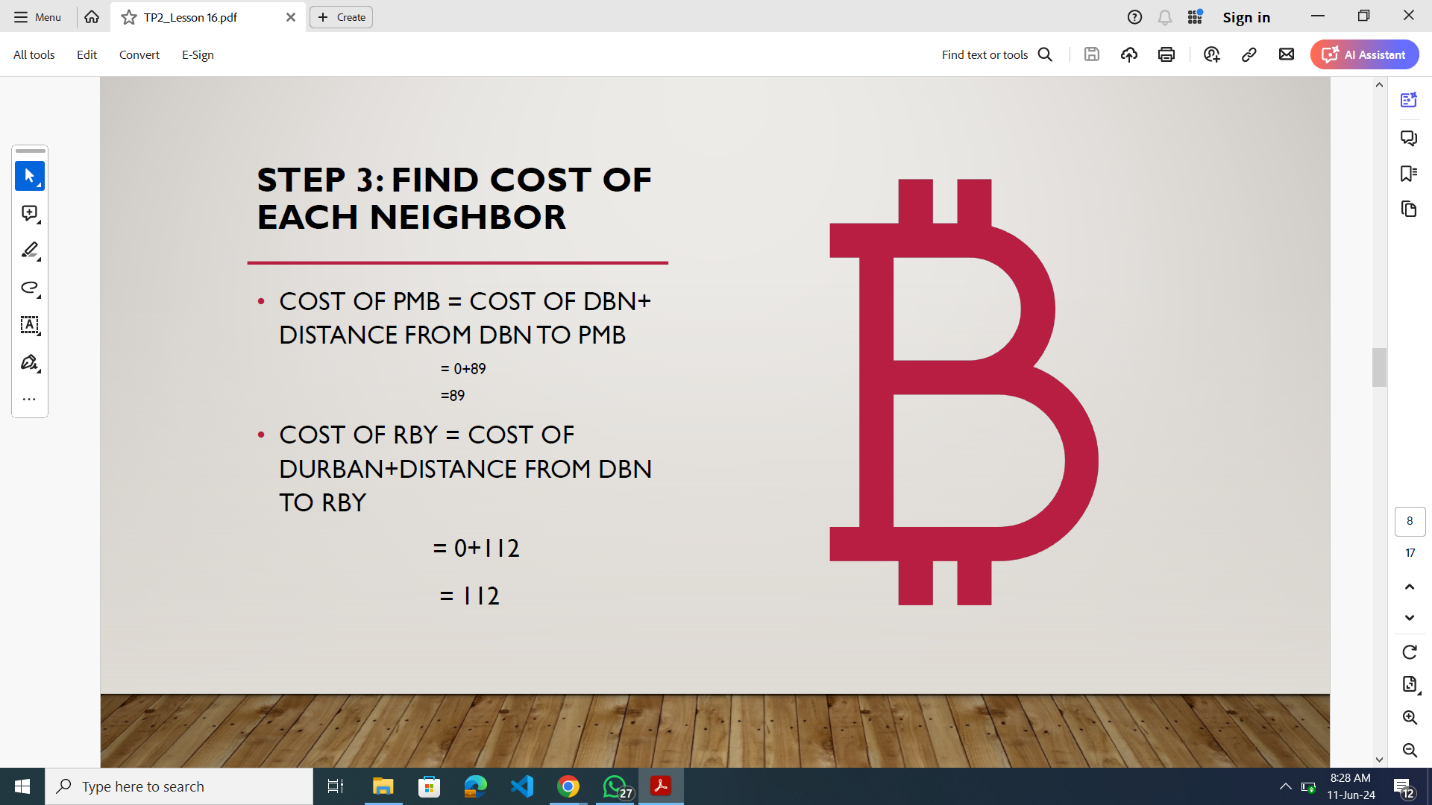
}

# Here we are finding adjacent nodes for 'DBN'

adjacent\_nodes = find\_adjacent\_nodes(graph, 'DBN')

print(f"Adjacent nodes to DBN: {list(adjacent\_nodes)}")

# The answer for this Sir should be : Adjacent nodes to DBN: [‘PMB’ , ‘RBY’]



• Here we were calculating the cost of reaching each neighbor from the starting node ('DBN').

• We have been given in the above image that the initial “cost for ‘DBN’ is 0.

import heapq

def calculate\_costs(graph, start):

distances = {node: float('infinity') for node in graph}

distances[start] = 0

for neighbor, weight in graph[start].items():

cost = distances[start] + weight

print(f"Cost to reach {neighbor} from {start}: {cost}")

distances[neighbor] = cost

# Graph representation

graph = {

'DBN': {'PMB': 89, 'RBY': 112},

'PMB': {'RBY': 70, 'HMT': 209},

'RBY': {'PMB': 70, 'HMT': 100, 'VRT': 106},

'HMT': {'JHB': 210, 'VRT': 41},

'VRT': {'JHB': 106},

'JHB': {}

}

# Here we are calculating costs for neighbors of 'DBN'

calculate\_costs(graph, 'DBN')