Practical 2 a] A star algorithm

Code:

import heapq

def a\_star\_search(graph, heuristic\_graph, start, destination):

open\_list = [(0, start)]

closed\_list = set()

g\_scores = {node: float('inf') for node in graph}

g\_scores[start] = 0

parents = {}

total\_distance = 0

while open\_list:

\_, current\_node = heapq.heappop(open\_list)

closed\_list.add(current\_node)

if current\_node == destination:

path = []

while current\_node in parents:

path.append(current\_node)

total\_distance += graph[parents[current\_node]][current\_node]

current\_node = parents[current\_node]

path.append(start)

path.reverse()

return path, total\_distance

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

if neighbor in closed\_list:

continue

tentative\_g\_score = g\_scores[current\_node] + distance

if tentative\_g\_score < g\_scores[neighbor]:

g\_scores[neighbor] = tentative\_g\_score

f\_score = tentative\_g\_score + heuristic\_graph[neighbor] # f(n) = g(n) + h(n)

heapq.heappush(open\_list, (f\_score, neighbor))

parents[neighbor] = current\_node

return None

if \_\_name\_\_ == "\_\_main\_\_":

Graph = {

"Bandra Kurla Complex": {"Eastern Express Hwy": 42.3},

"Eastern Express Hwy": {"Bandra Kurla Complex": 42.3, "Chembur": 39.8},

"Chembur": {"Eastern Express Hwy": 39.8, "Kurla": 36.5},

"Kurla": {"Chembur": 36.5, "Parel": 33.2},

"Parel": {"Kurla": 33.2, "Charni Road": 28.9},

"Charni Road": {"Parel": 28.9, "Marine Drive": 4.5, "Nariman Point": 3.9, "Chhatrapati Shivaji Terminus": 2.4},

"Western Express Highway": {"Charni Road": 27.6},

"Haji Ali Dargah": {"Charni Road": 22.0},

"Mahalaxmi Temple": {"Charni Road": 20.0},

"Dhobi Ghat": {"Charni Road": 17.1},

"Marine Drive": {"Charni Road": 4.5},

"Nariman Point": {"Charni Road": 3.9},

"Chhatrapati Shivaji Terminus": {"Charni Road": 2.4}

}

heuristic\_graph = {

"Bandra Kurla Complex": 0,

"Eastern Express Hwy": 42.3, # Heuristic value is the direct distance to BKC

"Chembur": 82.1, # Heuristic value is the direct distance to BKC

"Kurla": 118.6, # Heuristic value is the direct distance to BKC

"Parel": 152.1, # Heuristic value is the direct distance to BKC

"Charni Road": 178.5, # Heuristic value is the direct distance to BKC

"Western Express Highway": 27.6,

"Haji Ali Dargah": 22.0,

"Mahalaxmi Temple": 20.0,

"Dhobi Ghat": 17.1,

"Marine Drive": 4.5,

"Nariman Point": 3.9,

"Chhatrapati Shivaji Terminus": 2.4

}

start = "Charni Road"

destination = "Bandra Kurla Complex"

path, total\_distance = a\_star\_search(Graph, heuristic\_graph, start, destination)

if path:

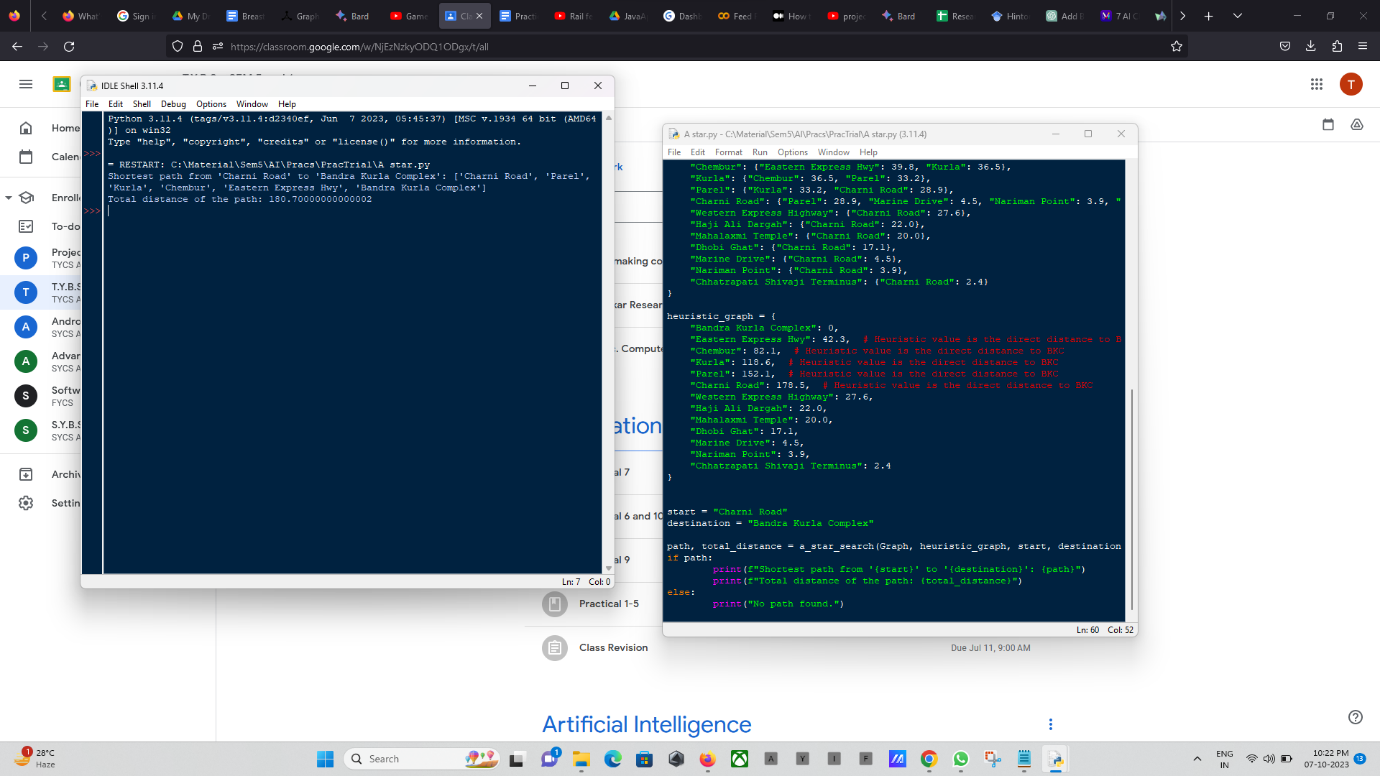
print(f"Shortest path from '{start}' to '{destination}': {path}")

print(f"Total distance of the path: {total\_distance}")

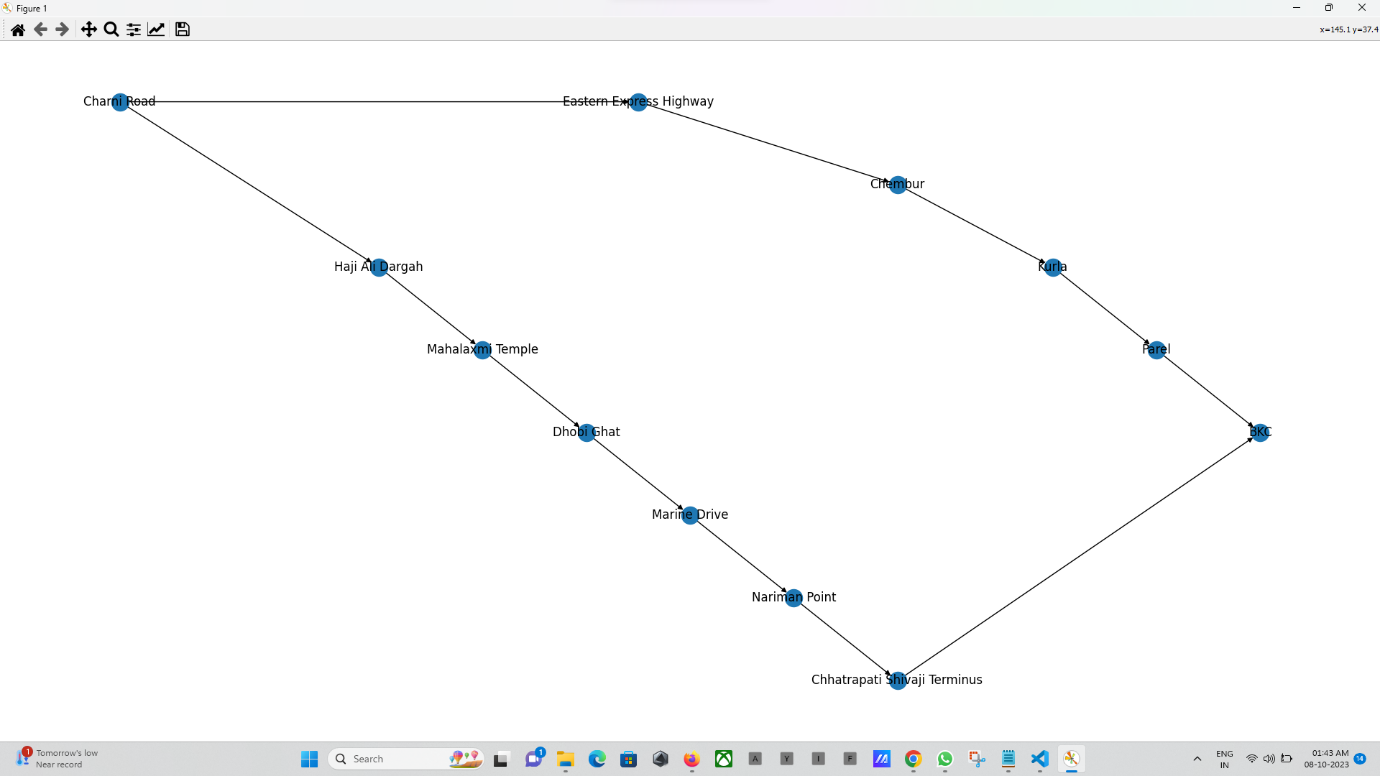
else:

print("No path found.")

Output:



Graph:



b] BFS (Best First Search) Algorithm

Code:

import heapq

def bestfirstsearch\_iterative(graph, heuristic\_graph, start, destination):

visited = set()

priority\_queue = [(heuristic\_graph[start], start, 0)]

total\_distance = 0

while priority\_queue:

\_, current\_node, dist = heapq.heappop(priority\_queue)

visited.add(current\_node)

print(current\_node)

total\_distance += dist

if current\_node == destination:

return total\_distance

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

if neighbor not in visited:

visited.add(neighbor)

heapq.heappush(priority\_queue, (heuristic\_graph[neighbor], neighbor, distance))

return total\_distance

def bestfirstsearch\_recursive(graph, heuristic\_graph, visited, priority\_queue, destination, total\_distance):

if not priority\_queue:

return total\_distance

\_, current\_node, dist = heapq.heappop(priority\_queue)

visited.add(current\_node)

print(current\_node)

total\_distance += dist

if current\_node == destination:

return total\_distance

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

if neighbor not in visited:

visited.add(neighbor)

heapq.heappush(priority\_queue, (heuristic\_graph[neighbor], neighbor, distance))

return bestfirstsearch\_recursive(graph, heuristic\_graph, visited, priority\_queue, destination, total\_distance)

def bestfirstsearch\_traverse(graph, heuristic\_graph, start, destination, traverse):

visited = set()

priority\_queue = [(heuristic\_graph[start], start, 0)]

total\_distance = 0

if traverse == "iterative":

total\_distance = bestfirstsearch\_iterative(graph, heuristic\_graph, start, destination)

elif traverse == "recursive":

total\_distance = bestfirstsearch\_recursive(graph, heuristic\_graph, visited, priority\_queue, destination, total\_distance)

print(f"The final distance from '{start}' to '{destination}' is: {total\_distance}")

if \_\_name\_\_ == "\_\_main\_\_":

Graph = {

"Bandra Kurla Complex": {"Eastern Express Hwy": 42.3},

"Eastern Express Hwy": {"Bandra Kurla Complex": 42.3, "Chembur": 39.8},

"Chembur": {"Eastern Express Hwy": 39.8, "Kurla": 36.5},

"Kurla": {"Chembur": 36.5, "Parel": 33.2},

"Parel": {"Kurla": 33.2, "Charni Road": 28.9},

"Charni Road": {"Parel": 28.9, "Marine Drive": 4.5, "Nariman Point": 3.9, "Chhatrapati Shivaji Terminus": 2.4},

"Western Express Highway": {"Charni Road": 27.6},

"Haji Ali Dargah": {"Charni Road": 22.0},

"Mahalaxmi Temple": {"Charni Road": 20.0},

"Dhobi Ghat": {"Charni Road": 17.1},

"Marine Drive": {"Charni Road": 4.5},

"Nariman Point": {"Charni Road": 3.9},

"Chhatrapati Shivaji Terminus": {"Charni Road": 2.4}

}

heuristic\_graph = {

"Bandra Kurla Complex": 0,

"Eastern Express Hwy": 42.3, # Heuristic value is the direct distance to BKC

"Chembur": 82.1, # Heuristic value is the direct distance to BKC

"Kurla": 118.6, # Heuristic value is the direct distance to BKC

"Parel": 152.1, # Heuristic value is the direct distance to BKC

"Charni Road": 178.5, # Heuristic value is the direct distance to BKC

"Western Express Highway": 27.6,

"Haji Ali Dargah": 22.0,

"Mahalaxmi Temple": 20.0,

"Dhobi Ghat": 17.1,

"Marine Drive": 4.5,

"Nariman Point": 3.9,

"Chhatrapati Shivaji Terminus": 2.4

}

start = "Charni Road"

destination = "Bandra Kurla Complex"

bestfirstsearch\_traverse(Graph, heuristic\_graph, start, destination, traverse="recursive")

Output:

