**A\* Algorithm**

import heapq

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list)

def add\_edge(self, u, v, w):

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

def astar(self, start, goal):

open\_list = [(0, start)]

heapq.heapify(open\_list)

came\_from = {}

g\_score = {node: float('inf') for node in self.graph}

g\_score[start] = 0

f\_score = {node: float('inf') for node in self.graph}

f\_score[start] = self.heuristic(start, goal)

while open\_list:

current = heapq.heappop(open\_list)[1]

if current == goal:

path = self.reconstruct\_path(came\_from, goal)

return path

for neighbor, weight in self.graph[current]:

tentative\_g\_score = g\_score[current] + weight

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

came\_from[neighbor] = current

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

f\_score[neighbor] = tentative\_g\_score + self.heuristic(neighbor, goal)

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

return None

def heuristic(self, node, goal):

# For simplicity, we use Manhattan distance as heuristic

x1, y1 = node

x2, y2 = goal

return abs(x1 - x2) + abs(y1 - y2)

def reconstruct\_path(self, came\_from, current):

total\_path = [current]

while current in came\_from:

current = came\_from[current]

total\_path.append(current)

return total\_path[::-1]

def main():

g = Graph()

while True:

print("\n1. Add Edge")

print("2. Find Shortest Path using A\* Algorithm")

print("3. Quit")

choice = input("Enter your choice: ")

if choice == '1':

u = tuple(map(int, input("Enter source vertex (x, y): ").split()))

v = tuple(map(int, input("Enter destination vertex (x, y): ").split()))

w = int(input("Enter weight for the edge: "))

g.add\_edge(u, v, w)

elif choice == '2':

start = tuple(map(int, input("Enter starting vertex (x, y): ").split()))

goal = tuple(map(int, input("Enter goal vertex (x, y): ").split()))

path = g.astar(start, goal)

if path:

print("Shortest path:", path)

else:

print("No path found!")

elif choice == '3':

print("Exiting...")

break

else:

print("Invalid choice! Please choose a valid option.")

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

main()

Output :-

1. Add Edge

2. Find Shortest Path using A\* Algorithm

3. Quit

Enter your choice: 1

Enter source vertex (x, y): 1

Enter destination vertex (x, y): 2

Enter weight for the edge: 2

1. Add Edge

2. Find Shortest Path using A\* Algorithm

3. Quit

Enter your choice: 1

Enter source vertex (x, y): 2

Enter destination vertex (x, y): 3

Enter weight for the edge: 2

1. Add Edge

2. Find Shortest Path using A\* Algorithm

3. Quit

Enter your choice: 1

Enter source vertex (x, y): 3

Enter destination vertex (x, y): 1

Enter weight for the edge: 3

1. Add Edge

2. Find Shortest Path using A\* Algorithm

3. Quit

Enter your choice:

22

Enter starting vertex (x, y): 1

Enter goal vertex (x, y): 3

Shortest path : 1 2 3