from queue import PriorityQueue

class Graph:

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

self.graph = {}

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

if u not in self.graph:

self.graph[u] = []

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

def a\_star\_search(graph, start, goal):

priority\_queue = PriorityQueue()

priority\_queue.put((0, start))

visited = set()

while not priority\_queue.empty():

cost, current\_node = priority\_queue.get()

if current\_node in visited:

continue

print(current\_node, end=' ')

if current\_node == goal:

print("\nGoal Reached!")

return

visited.add(current\_node)

for neighbor, edge\_cost in graph.graph.get(current\_node, []):

if neighbor not in visited:

priority\_queue.put((edge\_cost + heuristic(neighbor, goal), neighbor))

print("\nGoal not reached.")

def heuristic(node, goal):

return 0

g = Graph()

# Input graph edges

while True:

u = input("Enter start node of edge (or 'done' to finish): ")

if u.lower() == 'done':

break

v = input("Enter end node of edge: ")

cost = int(input("Enter cost of edge: "))

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

start\_node = input("Enter the starting node for A\* search: ")

goal\_node = input("Enter the goal node for A\* search: ")

print("A\* Search:")

a\_star\_search(g, start\_node, goal\_node)

Output:

Enter start node of edge (or 'done' to finish): A

Enter end node of edge: B

Enter cost of edge: 1

Enter start node of edge (or 'done' to finish): A

Enter end node of edge: C

Enter cost of edge: 4

Enter start node of edge (or 'done' to finish): B

Enter end node of edge: D

Enter cost of edge: 2

Enter start node of edge (or 'done' to finish): C

Enter end node of edge: D

Enter cost of edge: 5

Enter start node of edge (or 'done' to finish): D

Enter end node of edge: E

Enter cost of edge: 3

Enter start node of edge (or 'done' to finish): done

Enter the starting node for A\* search: A

Enter the goal node for A\* search: E

A\* Search:

A B D E

Goal Reached!