class Node:

def \_\_init\_\_(self, name, parent=None):

self.name = name

self.parent = parent

self.g = float('inf')

self.h = 0

self.f = float('inf')

def \_\_lt\_\_(self, other):

return self.f < other.f

def reconstruct\_path(node):

path = []

while node:

path.append(node.name)

node = node.parent

return path[::-1]

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

open\_list = [start]

closed\_list = set()

start.g = 0

start.f = start.g + heuristic[start.name]

while open\_list:

current\_node = min(open\_list, key=lambda node: node.f)

open\_list.remove(current\_node)

closed\_list.add(current\_node.name)

if current\_node.name == goal.name:

return reconstruct\_path(current\_node)

for neighbor\_name, cost in graph[current\_node.name]:

if neighbor\_name in closed\_list:

continue

neighbor = Node(neighbor\_name, current\_node)

tentative\_g = current\_node.g + cost

if tentative\_g < neighbor.g:

neighbor.g = tentative\_g

neighbor.h = heuristic[neighbor.name]

neighbor.f = neighbor.g + neighbor.h

if neighbor not in open\_list:

open\_list.append(neighbor)

return None

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

graph = {

'A': [('B', 1), ('C', 4)],

'B': [('A', 1), ('C', 2), ('D', 5)],

'C': [('A', 4), ('B', 2), ('D', 1)],

'D': [('B', 5), ('C', 1)]

}

heuristic = {

'A': 7,

'B': 6,

'C': 2,

'D': 0

}

start\_node = Node('A')

goal\_node = Node('D')

path = a\_star(start\_node, goal\_node, graph, heuristic)

if path:

print("Path found:", path)

else:

print("No path found.")