SINGLE SOURCE

import sys

# function to find shortest distances# Initialize distances to all nodes as infinity.

def find\_shortest\_distances(graph, source):

distances = {node: sys.maxsize for node in graph}

distances[source] = 0

unvisited = set(graph.keys())

while unvisited:

current\_node = min(unvisited, key=lambda node: distances[node])

unvisited.remove(current\_node)

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

if neighbor in unvisited:

new\_distance = distances[current\_node] + weight

if new\_distance < distances[neighbor]:

distances[neighbor] = new\_distance

return distances

# Greedy Best-First Search function to find path

def greedy\_best\_first\_search(graph, heuristics, start, goal):

visited = set()

path = []

current = start

while current != goal:

path.append(current)

visited.add(current)

neighbors = {node: heuristics[node] for node in graph[current] if node not in visited}

if not neighbors:

print("No path found!")

return []

current = min(neighbors, key=neighbors.get)

path.append(goal)

return path

#  Start from source. At each step, select unvisited neighbor with **lowest heuristic value**.Continue until the goal is reached.

# Graph and heuristic input

graph = {}

heuristics = {}

print("In single source we start with source node and considering the next lowest heuristic value at each step to reach destination.")

print("Neighbor with the lowest heuristic value is selected")

n = int(input("Enter the number of edges: "))

for i in range(n):

edge = input("Enter the edge (source destination weight): ").split()

source, destination, weight = edge[0], edge[1], int(edge[2])

if source not in graph:

graph[source] = {}

graph[source][destination] = weight

if destination not in graph:

graph[destination] = {}

for node in graph.keys():

h = int(input(f"Enter the heuristic value for node {node}: "))

heuristics[node] = h

# For each node in the graph, you input a heuristic value (an estimate of the remaining cost to reach the goal).

# Find shortest distances

source\_node = input("\nEnter the source node for distance calculation: ")

distances = find\_shortest\_distances(graph, source\_node)

# Show distances and heuristics

print("\nShortest distances and heuristic values:")

for node in graph.keys():

distance\_display = distances[node] if distances[node] != sys.maxsize else "Infinity"

print(f"Node: {node} | Distance: {distance\_display} | Heuristic: {heuristics[node]}")

# Now find path from source to goal using Greedy Best-First Search

print("\n--- Path Finding Using Greedy Best-First Search ---")

start = input("Enter the source node: ")

goal = input("Enter the goal node: ")

path = greedy\_best\_first\_search(graph, heuristics, start, goal)

if path:

print("\nPath taken:")

print(" -> ".join(path))

#Enter the number of edges: 4

#Enter the edge (source destination weight): A B 15

#Enter the edge (source destination weight): B C 20

#Enter the edge (source destination weight): C D 25

#Enter the edge (source destination weight): D E 12

#Enter the heuristic value for node A: 2

#Enter the heuristic value for node B: 1

##Enter the heuristic value for node C: 0

#Enter the heuristic value for node D: 3

#Enter the heuristic value for node E: 4

#Enter the source node for distance calculation: A

#Shortest distances and heuristic values:

#Node: A | Distance: 0 | Heuristic: 2

#Node: B | Distance: 15 | Heuristic: 1

#Node: C | Distance: 35 | Heuristic: 0

#Node: D | Distance: 60 | Heuristic: 3

#Node: E | Distance: 72 | Heuristic: 4

#--- Path Finding Using GreedySearch ---

#Enter the source node: A

#Enter the goal node: D

#Path taken:

#A -> B -> C -> D

JOB SCHEDULING

def heuristic(profit, deadline):

return profit / deadline if deadline != 0 else profit

profit = []

jobs = []

deadline = []

print("\nJobs with higher profit and lower deadline will have a higher heuristic value.")

print(" A high heuristic value suggests the job is more urgent and more rewarding, so it should be scheduled earlier.")

n = int(input("\nEnter the number of jobs: "))

for i in range(n):

p = int(input(f"Enter the profit of job {i + 1}: "))

profit.append(p)

j = input(f"Enter the name of job {i + 1}: ")

jobs.append(j)

d = int(input(f"Enter the deadline of job {i + 1}: "))

deadline.append(d)

job\_data = []

# Calculate heuristic for each job

for i in range(n):

h = heuristic(profit[i], deadline[i])

job\_data.append((h, profit[i], jobs[i], deadline[i]))

job\_data.sort(key=lambda x: x[0], reverse=True)

max\_deadline = max(deadline)

slot = [0] \* (max\_deadline + 1)

scheduled\_jobs = ['null'] \* (max\_deadline + 1)

total\_profit = 0

# Schedule jobs based on sorted heuristic values

for h, p, name, d in job\_data:

for j in range(min(d, max\_deadline), 0, -1):

if slot[j] == 0:

scheduled\_jobs[j] = name

slot[j] = 1

total\_profit += p

break

print("\nJobs scheduled:", [job for job in scheduled\_jobs[1:] if job != 'null'])

print("Total profit:", total\_profit)

print("\nHeuristic Function Used:")

print("f(job) = profit of the job")