Single-Source Shortest Path Problem

import sys

# Print student details

print("\nName: Prachi Karande")

print("Roll No: TACO22134")

# Function to perform Greedy Search (Dijkstra-like approach without priority queue)

def greedy\_search(graph, source):

# Initialize all distances to infinity (sys.maxsize)

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

distances[source] = 0 # Distance to source is 0

# Set of all unvisited nodes

unvisited = set(graph.keys())

while unvisited:

# Greedily pick the node with the smallest distance

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

unvisited.remove(current\_node)

# Update distances for neighbors of the 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

# Ensure all nodes are included in result (unreachable nodes remain as infinity)

for node in graph:

if node not in distances:

distances[node] = sys.maxsize

return distances

# Main part to input graph and call greedy search

graph = {}

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

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

distances = greedy\_search(graph, source)

# Display shortest distances

print("Shortest distances from source node:", distances)

# Heuristic explanation

print("\nHeuristic Function Used:")

print("f(node) = distances[node] (minimized at each step)")

print("At each step, the algorithm greedily selects the node with the smallest known distance from the source node")

print("and updates the distances of its neighbors.")

Job Scheduling Problem

# Print student details

print("\nName: Prachi Karande")

print("Roll No: TACO22134")

# Input job details: profit, job name, deadline

profit = []

jobs = []

deadline = []

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

for i in range(n):

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

profit.append(p)

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

jobs.append(j)

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

deadline.append(d)

# Combine all job details into a list of tuples

profitNJobs = list(zip(profit, jobs, deadline))

# Sort the jobs based on descending profit (Greedy choice)

profitNJobs = sorted(profitNJobs, key=lambda x: x[0], reverse=True)

# Initialize time slots for jobs and answer array

slot = [0] \* (n + 1) # 0 means slot is free

total\_profit = 0

ans = ['null'] \* (n + 1) # Store selected jobs

# Greedy Job Scheduling

for i in range(n):

job = profitNJobs[i]

# Find a free slot for this job before its deadline

for j in range(job[2], 0, -1):

if slot[j] == 0:

ans[j] = job[1]

total\_profit += job[0]

slot[j] = 1

break

# Final Output

print("\nJobs scheduled:", ans[1:]) # Skip index 0

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

# Heuristic explanation

print("\nHeuristic Function Used:")

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

print("Jobs are scheduled greedily by selecting the job with highest profit first")

print("and placing it in the latest available slot before its deadline.")