***Problem 1: Optimizing Delivery Routes (Case Study)***

**Scenario:**

You are working for a logistics company that wants to optimize its delivery routes to minimize fuel consumption and delivery time. The company operates in a city with a

complex road network.

**Tasks:**

1. Model the city's road network as a graph where intersections are nodes and roads

are edges with weights representing travel time.

2. Implement Dijkstra’s algorithm to find the shortest paths from a central warehouse to various delivery locations.

3. Analyze the efficiency of your algorithm and discuss any potential improvements or

alternative algorithms that could be used.

**Deliverables:**

● Graph model of the city's road network.

● Pseudo code and implementation of Dijkstra’s algorithm.

●Analysis of the algorithm’s efficiency and potential improvements.

**Reasoning:** Explain why Dijkstra’s algorithm is suitable

for this problem. Discuss any

assumptions made (e.g., non-negative weights) and how different road conditions (e.g.,

traffic, road closures) could affect your solution.

**PSEUDO CODE:**

function Dijkstra(Graph, source):

dist[source] ← 0 // Initial distance from source to source is 0

for each vertex v in Graph:

if v ≠ source

dist[v] ← ∞ // Initial distance from source to all other vertices is infinity

add v to Q // All nodes initially in Q

while Q is not empty: // While Q is not empty

u ← vertex in Q with min dist[u] // Get the vertex u with the smallest distance

remove u from Q

for each neighbor v of u: // For each neighbor v of u

alt ← dist[u] + length(u, v)

if alt < dist[v]: // If there's a shorter path to v through u

dist[v] ← alt // Update the shortest distance to v

prev[v] ← u // Update the previous vertex of v

return dist[], prev[] // Return the shortest distance and the previous vertex arrays

**GRAPH:**

4

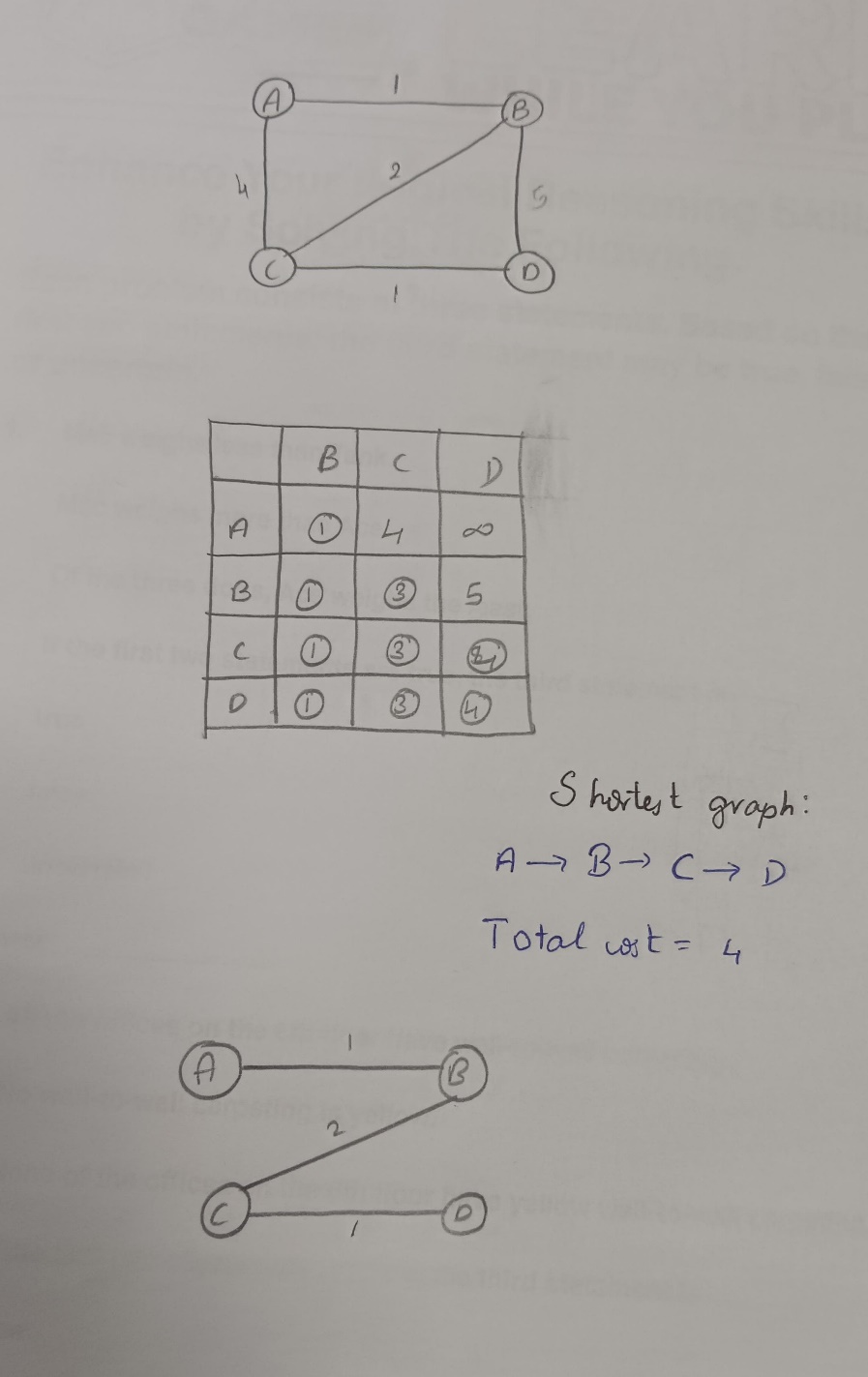
1

1

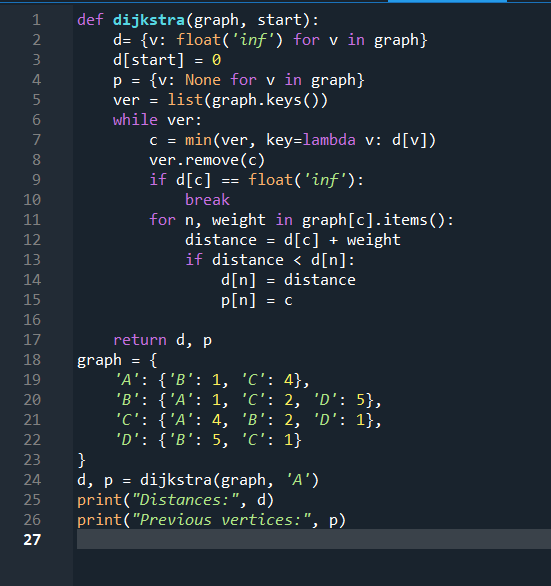
5

2

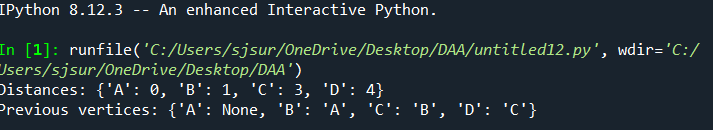
**ANALYTICAL:**



**CODE:**



**OUTPUT:**



***Problem 2: Dynamic Pricing Algorithm for E-commerce Scenario***

**Scenario:**

An e-commerce company wants to implement a dynamic pricing algorithm to adjust the prices of products in real-time based on demand and competitor prices.

**Tasks:**

1. Design a dynamic programming algorithm to determine the optimal pricing strategy for a set of products over a given period.

2. Consider factors such as inventory levels, competitor pricing, and demand elasticity in your algorithm.

3. Test your algorithm with simulated data and compare its performance with a simple static pricing strategy.

**Deliverables:**

● Pseudocode and implementation of the dynamic pricing algorithm.

● Simulation results comparing dynamic and static pricing strategies.

● Analysis of the benefits and drawbacks of dynamic pricing.

**Reasoning:** Justify the use of dynamic programming for this problem. Explain how you incorporated different factors into your algorithm and discuss any challenges faced during implementation.

**PSUEDO CODE:**

function dynamic\_pricing(T, P, I, E, C):

n = length of P

initialize dp and ps as 2D arrays of zeros with dimensions [n][T]

for t from 0 to T-1:

for i from 0 to n-1:

set mp (maximum profit) to negative infinity

set op (optimal price) to 0

for price from C[i] - 10 to C[i] + 10:

if I[i] <= 0:

break

calculate demand as max(E[i] \* (price - C[i]), 0)

calculate profit as price \* demand

if t > 0:

profit = profit + dp[t-1][i]

if profit > mp:

mp = profit

op = price

dp[t][i] = mp

ps[t][i] = op

update I[i] to max(E[i] \* (op - C[i]), 0)

return ps, dp[T-1]

T = 3

P = ['P1', 'P2']

I = [100, 100]

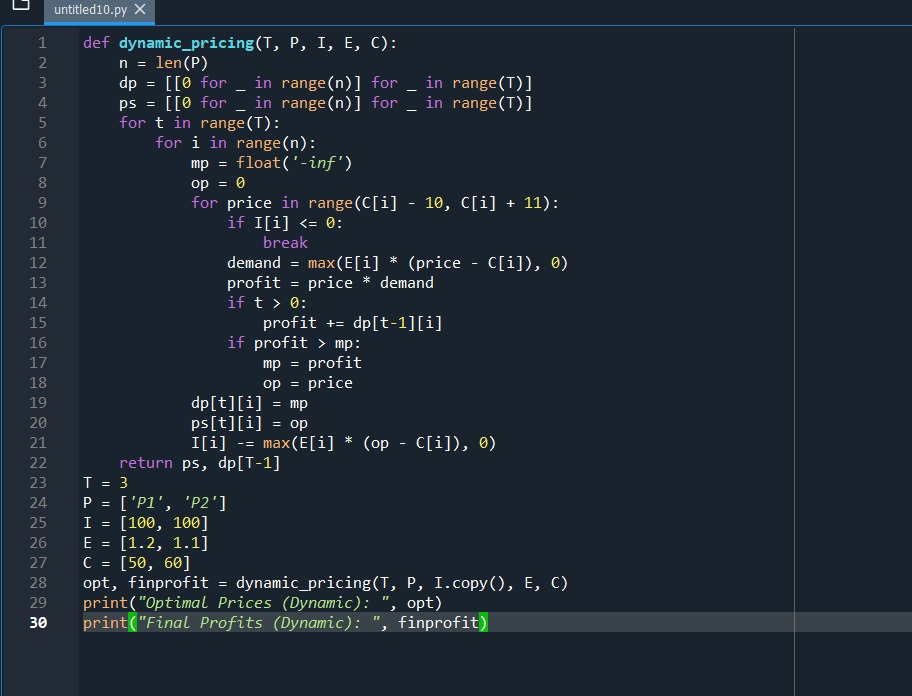
E = [1.2, 1.1]

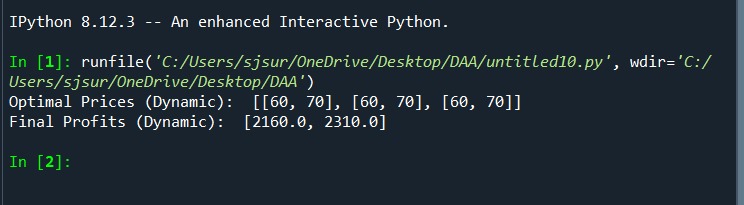
C = [50, 60]

opt, finprofit = dynamic\_pricing(T, P, copy of I, E, C)

print "Optimal Prices (Dynamic):", opt

print "Final Profits (Dynamic):", finprofit





***Problem 3: Social Network Analysis (Case Study) Scenario***

**Scenario:**

A social media company wants to identify influential users within its network to target for marketing campaigns.

**Tasks:**

1. Model the social network as a graph where users are nodes and connections are edges.

2. Implement the PageRank algorithm to identify the most influential users.

3. Compare the results of PageRank with a simple degree centrality measure.

**Deliverables:**

● Graph model of the social network.

● Pseudocode and implementation of the PageRank algorithm.

● Comparison of PageRank and degree centrality results.

**Reasoning:**

Discuss why PageRank is an effective measure for identifying influential users. Explain the differences between PageRank and degree centrality and why one might be preferred over the other in different scenarios.

**PSUEDO CODE:**

function pagerank(graph, damping\_factor=0.85, max\_iterations=100, tol=1.0e-6):

num\_nodes = length of graph

ranks = dictionary where each node is initialized to 1 / num\_nodes

new\_ranks = copy of ranks

for each iteration from 0 to max\_iterations:

for each node in graph:

rank\_sum = 0

for each neighbor in graph:

if node is in graph[neighbor]:

rank\_sum += ranks[neighbor] / length of graph[neighbor]

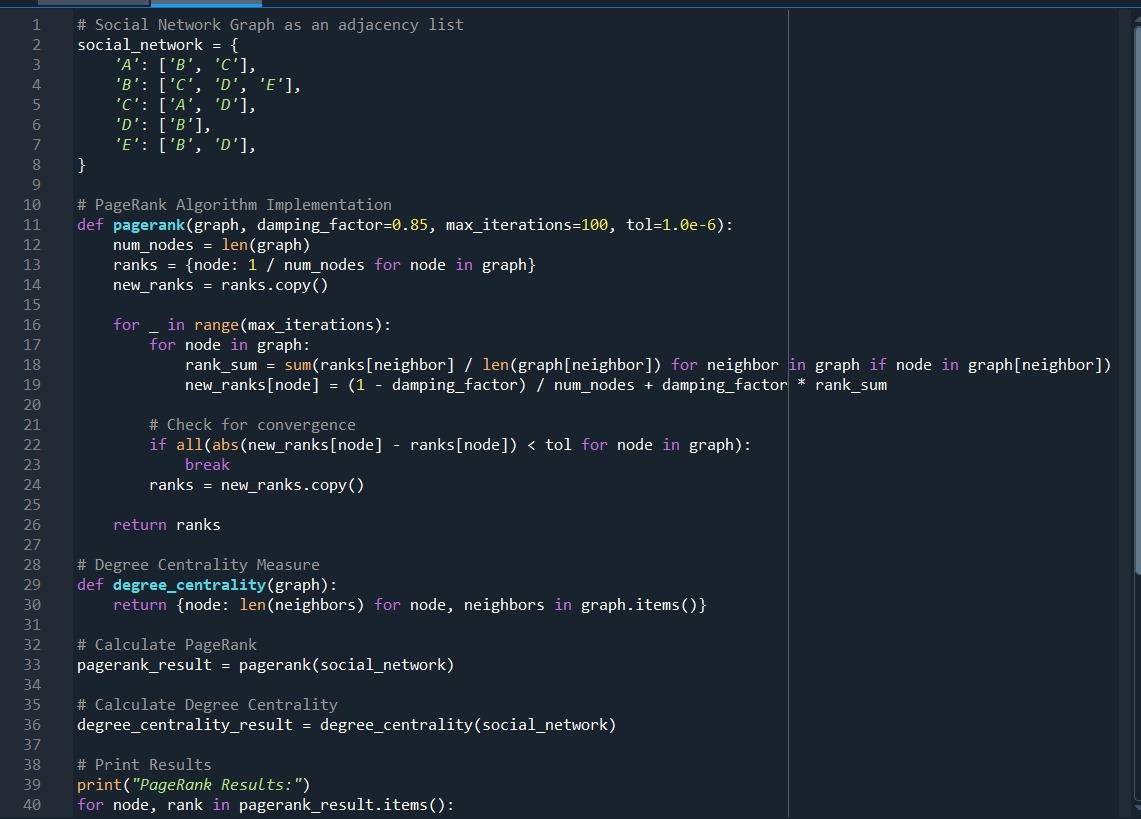
new\_ranks[node] = (1 - damping\_factor) / num\_nodes + damping\_factor \* rank\_sum

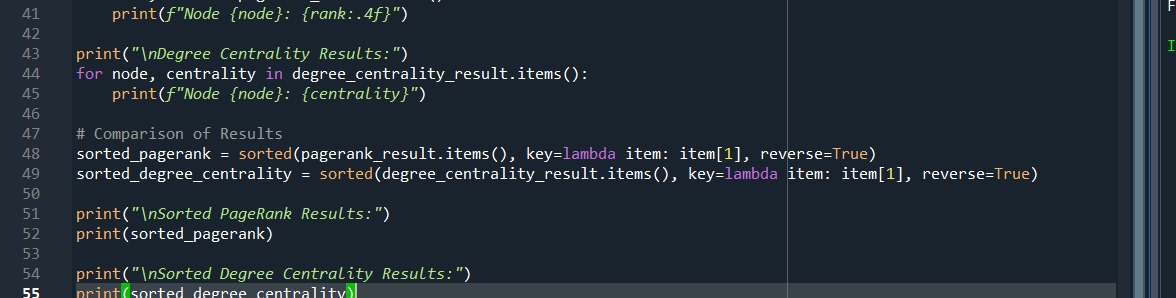
if all differences between new\_ranks and ranks are less than tol:

break

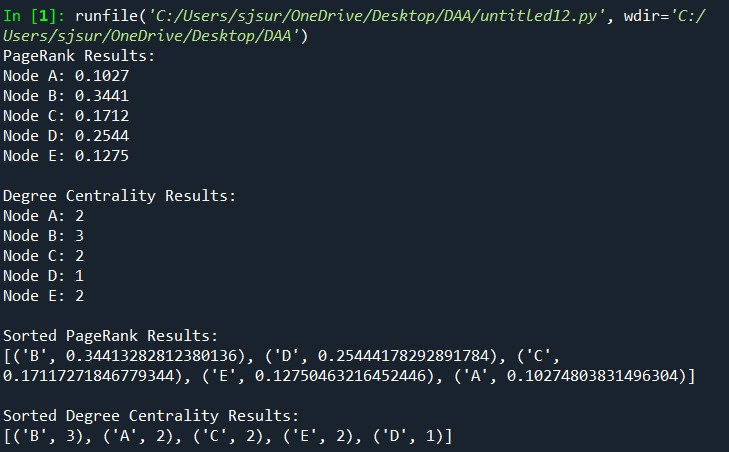
ranks = copy of new\_ranks

return ranks





**OUTPUT:**



***Problem 4: Fraud Detection in Financial Transactions***

**Scenario:** A financial institution wants to develop an algorithm to detect fraudulent

transactions in real-time.

**Tasks:**

1. Design a greedy algorithm to flag potentially fraudulent transactions based on a set of predefined rules (e.g., unusually large transactions, transactions from multiple

locations in a short time).

2. Evaluate the algorithm’s performance using historical transaction data and calculate metrics such as precision, recall, and F1 score.

3. Suggest and implement potential improvements to the algorithm.

**Deliverables:**

● Pseudo code and implementation of the fraud detection algorithm.

● Performance evaluation using historical data.

● Suggestions and implementation of improvements.

**Reasoning:** Explain why a greedy algorithm is suitable

for real-time fraud detection. Discuss

the trade-offs between speed and accuracy and how your algorithm addresses them

**PSEUDO CODE:**

input: transaction

output: fraud\_flag (true/false)

initialize fraud\_flag to false

if transaction amount > predefined large amount threshold:

set fraud\_flag to true

if transaction location is in list of flagged locations:

set fraud\_flag to true

if transaction occurs within short time from another transaction at a different location:

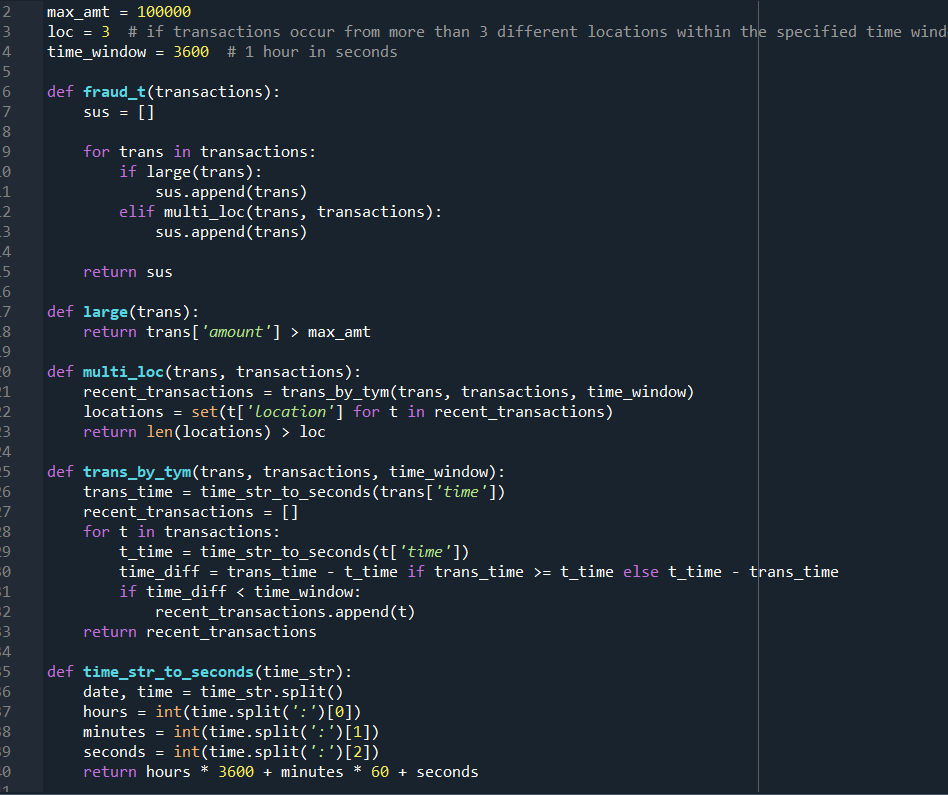
set fraud\_flag to true

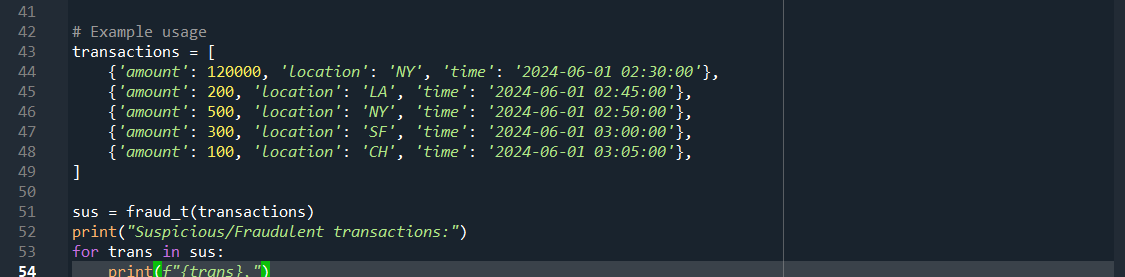
if transaction frequency exceeds threshold within a given timeframe:

set fraud\_flag to true

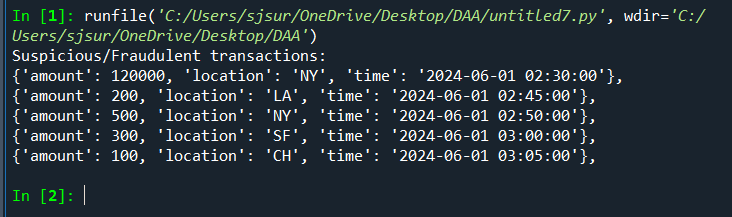
return fraud\_flag

**CODE:**





**OUTPUT:**



***Problem 5: Real-Time Traffic Management System***

**Scenario:**

A city’s traffic management department wants to develop a system to manage traffic lights in real-time to reduce congestion.

**Tasks:**

1. Design a backtracking algorithm to optimize the timing of traffic lights at major

intersections.

2. Simulate the algorithm on a model of the city's traffic network and measure its impact

on traffic flow.

3. Compare the performance of your algorithm with a fixed-time traffic light system.

**Deliverables:**

● Pseudo code and implementation of the traffic light optimization algorithm.

● Simulation results and performance analysis.

● Comparison with a fixed-time traffic light system.

**Reasoning:**

Justify the use of backtracking for this problem. Discuss the complexities

involved in real-time traffic management and how your algorithm addresses them.

**PSUEDO CODE:**

function optimize\_traffic\_lights(intersections, max\_duration, min\_duration):

best\_config ← None

best\_traffic\_flow ← ∞

function backtrack(current\_intersection, current\_config):

if current\_intersection == len(intersections):

traffic\_flow = simulate\_traffic\_flow(current\_config)

if traffic\_flow < best\_traffic\_flow:

best\_traffic\_flow = traffic\_flow

best\_config = current\_config.copy()

return

for duration in range(min\_duration, max\_duration + 1):

current\_config[current\_intersection] = duration

backtrack(current\_intersection + 1, current\_config)

backtrack(0, [0] \* len(intersections))

return best\_config

function simulate\_traffic\_flow(config):

// Simulate traffic flow based on the current configuration

// Return a metric representing traffic congestion (e.g., average delay time, total waiting time)

traffic\_flow = 0

for intersection, duration in enumerate(config):

traffic\_flow += calculate\_traffic\_flow\_at\_intersection(intersection, duration)

return traffic\_flow

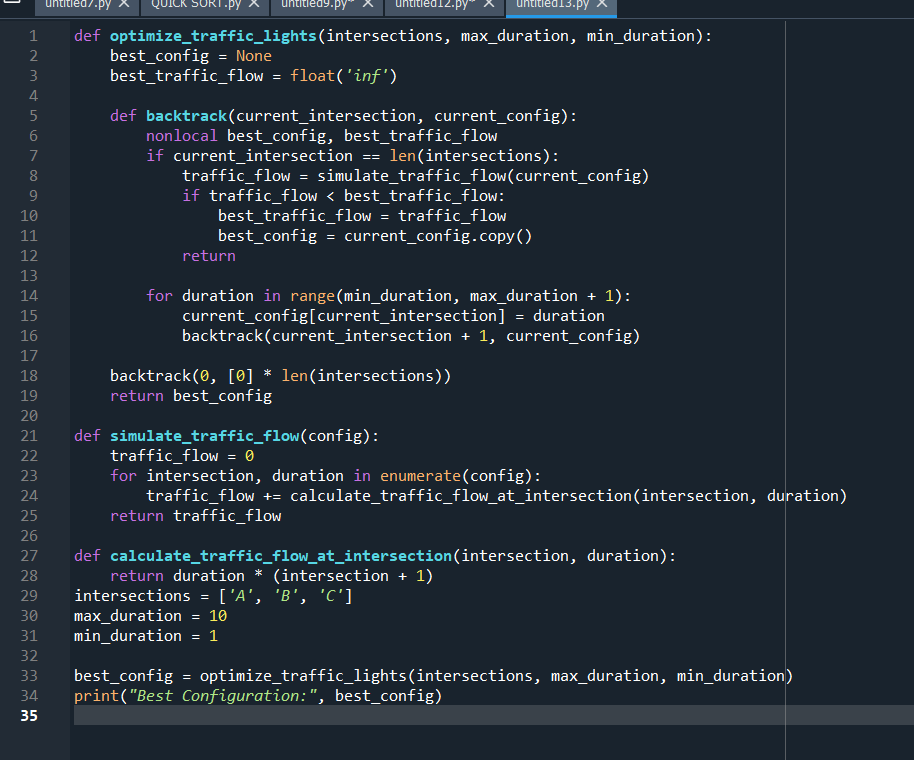
function calculate\_traffic\_flow\_at\_intersection(intersection, duration):

// Calculate the traffic flow for a specific intersection with a given green light duration

// This function uses traffic models and simulation data

return simulated\_traffic\_flow\_value

**CODE:**



**OUTPUT:**

