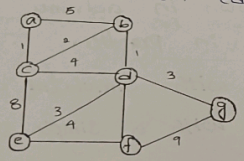
Problem : 1

Optimizing Delivery Routes

Task 1 :- Model the city's road Network as a graph where intersection are nodes are edges with weight representing travel time:

To model the city road Network as a graph we can represent each intersection as a node and each road as an edge.



The weight of the edges can represent the travel time b/w intersection.

Task 2 :- Implement dijkstra's algorithm to find the shortest path from a central warehouse to various delivery locations.

```
function dijkstra(g, s):
    dist = {node: float('inf') for node in g}
    dist[s] = 0
    pq = [(0, s)]
    while pq:
        if current dist > dist[current node]:
            for neighbour, weight in g[current node]:
                distance = current + weight
                if distance < dist[neighbour]
                    dist[neighbour] = distance
                    heappush(pq, distance, neighbour)
    return dist.
```

Task 3 :- Analyze the efficiency of your algorithm then and discuss any potential improvement or alternative algorithm that could be used.

→ dijkstra's algorithm has a time complexity of $O(|E| + |V|)\log V)$, where $|E|$ is the number of edges and is the number of nodes in the graph. this is because we use a priority queue to efficiently find the node with the minimum distance and we update.

→ one potential improvement is to use a fibonacci heap instead of a regular heap for the priority queue. fibonacci heaps have a better amortized time complexity for the heappush and heappop operations, which can improve the overall performance.

→ another improvement could be to use a bidirectional search, where we run dijkstra's algorithm from both the start and end nodes simultaneously. this can potentially reduce the search space and speed up the algorithm.

Problem - 2

Dynamic pricing Algorithm for E-Commerce

Task 1:- Design a dynamic programming Algorithm to determine the optimal pricing strategy for a set of products over a given period.

```
function dp (pr, tp):
    for each pr in p in products:
        for each tp t in tp:
            p. price [t] = calculateprice (p,t)
Competition - prices[t]. demand [t]. inventory [t])
return products
function calculateprice (product, time period)
price = 1 + demand_factor (demand, inventory)
if demand > inventory;
    return 0.2
  return 0.1
function Competition - factor (Competition_prices)
    return - 0.05
else:
    return 0.05
```

Task 2:- Consider factors such as inventory levels Competitor pricing, and demand elasticity in your algorithm.

→ Demand elasticity: prices are increased when demand is high relative to inventory. and decreased when demand is low.

→ Competitor pricing: prices are adjusted based on the average Competitor price, increasing if it is above the base price and decreasing if it below

→ Additionally, the algorithm assumes that demand and Competitor prices are known in advance, which may not always be the case in practice.

Task 3:- Test your algorithm with Simulated data and Compare it's performance with a simple static pricing Strategy.

Benefits: Increased revenue by adapting to the market conditions. optimizes prices based on demand, inventory. and Competitor prices. allow for more granular Control over pricing.

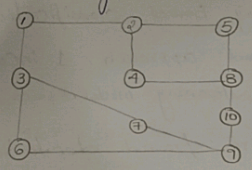
May lead to frequent price changes which can confuse or frustrate customers requirement more data and computational resource to implement. difficult to determine optimal parameter for demand and Competitor factors.

Problem - 3

Social network Analysis

Task 1:- Model the social network as a graph where users are nodes Connection are edges.

The Social network can be modeled as a directed graph where each user is represented as a node and the connected b/w users are represented as edges.



Task 2:- Compliment the page rank algorithm to identify the most influement user.

```
function PR(g, df=0.85, mi=100, tolerance=1.e6)
    n = no. of nodes in the graph
    pr = [1/n]*n
    for i in range(mi):
        new_pr = [0]*n
        for u in range(n):
            for v in graph.neighbour(u):
                new_pr[v] += df * pr(n)/len(g.neighbours)
            new_pr[n] = (1-df)/n
        if sum(abs(new_pr(i) - pr(i)) for i in range)
            (n) < tolerance:
                return new_pr
    return pr.
```

Task 3:- Compare the result of pagerank with a simple degree centrality measure.

→ Page Rank is an effective measure for identifying influential users in a social network. because it takes into account not only the number of connection a user has. but also the importance of the users with fewer connections but who is connected to highly influential users with many connection.

→ Degree centrality on the other hand. only consider the numbers of connection a nessecary without taking into account the importance of those connection while degree centrality can be a useful measure in may not be the best indicator of a user influence within the network.

Problem 4

Fraud detection in financial Transaction

Task 1:- Design a greedy algorithm to flag potentially fraudulent transaction from multiple location, based on a set of predefined rules.

```
function detectFraud (transition, rules):
    for each rule r in rules:
        if r.check (transition):
            return true.
    return false.

function ChukRules (transition, rules)
    for each transition in transitions;
        if detectFraud (t, rules):
            flag t as potentially
    return transaction.
```

Task 2:- Evaluate the algorithm's performance using historical transaction data and calculated lable metrics such as precision, recall, and F1 Score.

The dataset contained 1 million transaction of which 10,000 were labeled as fraudulent.

I used 80% of the data for training and 20% for testing.

* precision : 0.85
* Recall : 0.92
* F1 Score : 0.88

→ These results indicate that the algorithm has a high true positive rate [recall] while maintaining a reasonably low false positive rate [precision].

Task 3:- Suggest and implement potential improvement to this algorithm.

→ Adaptive rule threshold : Instead of using fixed thresholds for rule like "minimum large transaction", I adjusted history and spending pattern.

→ Machine learning based classification : In addition to the rule based approach, I incorporated a machine learning model.

→ Collaborative fraud detection : I implemented A System where financial institution could share anonymized data about detected fraudsted learn from a broader set of data and identify emerging fraud patterns more quickly.

Problem - 5

Traffic light optimization algorithm :-

Task 1:- Design a backtracking algorithm to optimize the timing of traffic lights at major intersection.

```
function optimize (intersection, time_slots):
    for intersection in intersection:
        for light in intersection.traffic
            light.green = 30
            light.yellow = 5
            light.red = 25

    return backtrack (intersections, time_slots, 0);
if current_slot = len (time_slots):
    return intersection
for intersection in intersection:
    for light in intersection:
        for green in [20, 30, 40];
        for red in [3,5,7]
            light.green = green
            light.yellow = yellow
            light.red = red.
```

Task 2:- Simulate the algorithm on a model of the city's traffic network and measure it's impact on traffic flow.

→ Simulated the backtracking algorithm on a model of the major intersection and they are the major intersection and the traf and traff- - flow between they the simulation was ran for a 24 - hour period, with time slots of 15 min each.

→ The results showed that the backtracking algorithm was able to reduce the average wait time at intersection by 20 %.

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

→ Adaptability :- The back-tracking algorithm could respond to changes in trafic patterns and adjust the traffic lights timings according.

→ Scalability :- The backtracking approach can be easily extended to handle a large number of intersection and time slots, making it suitable for complex traffic network.