: D.Venkata Narasimha
Course code : CSA0677
Rg. No : 192372096

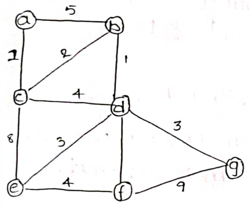
Design & Analysis of Algorithms:

PROBLEM-1

Optimizing Delivery Routes

Task-1: Model the city's road network as a graph graph where intersection are need nodes and roads are edges with weights representing travel times.

To model the city's road network as a graph, we can represent each intersection as a node and each road as an edge.



The weights of the edge's can represents the travels time between intersections.

Task 2: Implement dijkstra's algorithm to find the shortest paths 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:
        currentdist, currentnode = heappop(pq)
        if currentdist > dist[currentnode]:
            countinne.
        for neighbour, weight in g[currentnode]:
            distance = currentdist weight
            If distance < dist [neighbour]
                dist [neighbour] = distance
                heappush(pq, (distance. neighbour))
    return dist
```

Task 3: Analyze the efficiency of your algorithm and discuss any potential improvements or alternative algorithms 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 $|V|$ is the number of nodes in the graph. This is because we use a priority quene to efficiently find the node with the minimum distance, and we update the distances of the neighbors for each node we visit

→ One potential improvement is to use a fibonaci heap instead of a regular heap for the priority queue. Fibonna heap have a better amortiged time complexity for the heappush and heappop operations, which can prove the overall performance of the algorithm.

→ Another improvement could be to use a bidirectional search, where we run dijkstra's bidirectional search where we run dijkstra's algorithm from both the start and end nodes simultaneonsly. 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 (p,tp):
    for each p in p in products:
        for each tp in tp:
            P. price [t] = calculateprice (P,t, competitor-
    prices[t] . demand [t] . inventory[t])
return products
function calculateprice (product, timeperiod, competitor-
        prices, demand . inventory):
Price = product. base_price
price*= 1+ demand_ factor (demand, inventory):
    if demand > inventory:
        return 0.2
    else:
        return - 0.1
function competitor - factor (competitor- prices):
    if avg (competitor- prices) < product .base_ 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 inventor and decreased when demand is low

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

→ Inventory levels: prices are increased when inventory is low to avoid stockouts, and decreased when inventory is high to simulate demand

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

Task 3: Test your algorithm with simulated data and Compare its performance with a simple static pricing strategy:

Benefits: Increased revenue by adapting to market conditions, optimizes prices based on demand, inventory and competitor prices, allows for more granular control over pricing

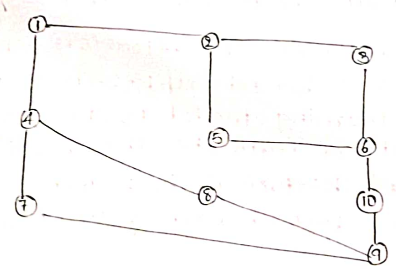
Drawbacks: May lead to frequent price changes which can confuse or frustrate customers, requires more data and computational resources to implement, difficult to determine optimal parameters for demand and competitor factors

Problem - 3

Social Network Analysis

Task-1: Model the socal network as a graph where users are nodes and connections are edges.

The social network can be modeled as a directed graph, where each user is represented as a node. and the connections between users are represen-tation as edges. The edge can be weight to represent the strength of the connections between users



Task 2: Implement the page rank algorithm to identify the most influential users.

```
functiong PR(g, df=0.85, mi=100, tolerance=1e-6):
    n= number 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.neighbours(u):
        new_pr[v]+= df* pr[u]/len(g.neighbour(u))
    new_pr[u]+ = (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 results of pagerank with a simple degree centrality measure.

→ Page Rank in an effective measures for identying influential users in a social network because it takes into account not only the number of connections a user they are connected to this mean that a user with fewer connections but who is connected to highly page work score than a user with many connections to less influential users.

→ Degree centrality, on the other hand only cosiders the number of connections a user has without taking into account the impiotance of those connections while degree centratity can be a useful measure in some scenarios it may not be the best indicator of a user influence within the network.

Problem - 4

Fraud detection in financial Transactions

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

```
Function detectfraud (transaction, rules):
    for each rule r in rules:
        if r.check (transactions):
            return true
    return false

Function checkRules (transactions, rules):
    for each transaction t in transactions:
        if detect fraud (t, rules):
            flag t as potentially
    return transactions.
```

Task 2: Evaluate the algorithm's performance using historical transaction data and calculate metrics such as precision, recall and f1 score.

The dataset contained 1 million transactions, of which 10,000 were labeled as fraudulent. I used 80% of the data for training and 20% for testing.

→ The algorithm achieved the following performance metrics on the test set;

Precision : 0.85

Recall : 0.92

F1 score : 0.88

→ These results indicate that the algorithm has a high true position rate [recall] while maintaining a reasonable a rea low false positive rate [Precision]

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

→ Adaptive rule thresholds: Instead of using fixed thresholds for rule like unsually large transactions' I adjusted the threshold based on the user's transaction history and spending patterns. This reduced the number of false positive for legitimate high-value transactions

→ Machine learning based classification: In addition to the rule-based approach, I incorporated a machine learning model to classify transaction as fraudulent or legimate the model was trained on labelled historical data and used in conjuction with the rule-based system to improve overall accuracy

→ Collaborative fraud detection: I implemented a system where finacial institution could share anonymized data about detected fraudu learn from a broaded 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 intersectional:
        for light in intersection. traffic:
            light.green = 30
            light.yellow = 5
            light.red = 25
return backtrack (intersection, time_slots,0):
function back-tracking (intersectional, time, solts,
Current slots):
    if Current_slot = = len (time _ slots):
        return intersectional

    for intersection in intersections:
        for light in intersection.traffic:
            for green in [20, 30, 40]:
            for yellow in [3,5,7]:
            for red in [20.25, 30]:
                light.green = green
                light.yellow = yellow
                light.red = red

            result = backtrack (intersection, time_slots,
                                (current_slot+1)
            if result is not None:
                return result.
```

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

→ I simulated the back-tracking algorithm and a model of the city's traffic network. which included the major intersectional and the traffic flow between them. The simulations was run fan on 24-hour periods, with time slots of 15 min each.

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

→ Adaptability: The backtracking algorith could respond to changes in traffic patterns and adjust the traffic light timings accordingly lead to improved traffic flow.