DAA - Assignment

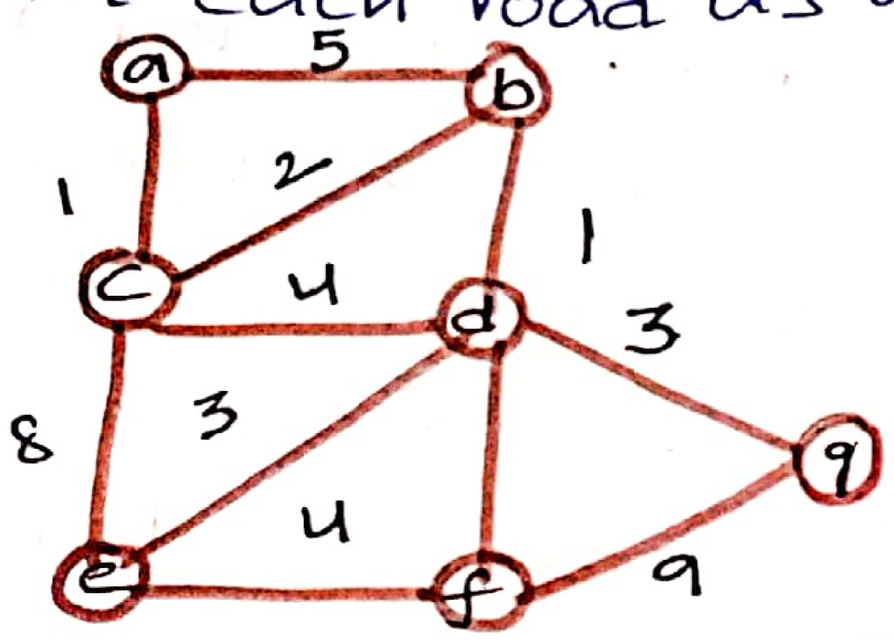
Name:- V. Pallavi
Course Code:- CSA0677
192372113

Problem-1

Optimizing Delivery Routes

TASK 1: Model the city's road network as a Graph where intersections are nodes and roads are edges with weights repesenting 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 edges can represent the travel time between intersections.

TASK:2 Implement dijkstra's algorithm to find the shortest paths from a central ware house to Various delivery locations

```
function dijkstra(g,s):
    dist={node: float('inf') for node in g
    }
    dist[s]=0
    pq=[(0,s)]
    while pq:
        Current dist, Currentnode=heappop(pq)
        if Currentdist>dist [currentnode]:
            Continue.
        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 queue to efficiency find the node with the minimum distance, and we update the distances of the neighbours for each node we visit.

→ Once potential improvement is to use la 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 of the algorithm.

→ Another improvement Could be to use a bi-directional 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 tpt in tp;
            P. price[t] = CalculatepriceCP, t.
    Competition-prices[t], demand [t], inventory[t])
 return products.
function Calculateprice(product time period,
    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 Competition-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 reldative to inventory. and decreased when demand is low.

→ Competitior pricing: prices 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 known in advance, which may not always be the case in practice.

TASK3: 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

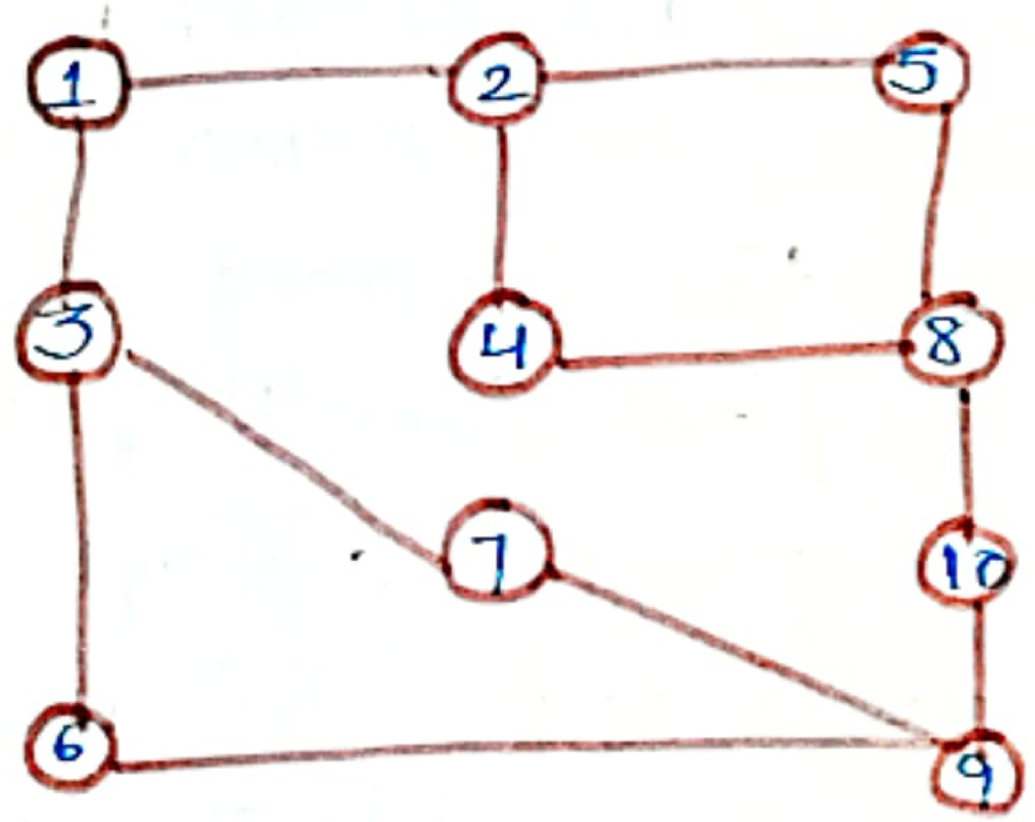
Drawbacks: may lead to frequent price change. Which Can Confuse or frustated Customers, requires more data and Computational resources to implement difficult to determine Optimal Parameters.

PROBLEM-3

Social network Analysis

TASK 1:- Model the social 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 represented as edges. The edges can be weight to represent the strength of the connections between users.



TASK 2: Implement the paage rank algorithm to identify the most influential users.

```
functiong PR(g, df=0.85, mi=100, tolerance=le-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.neighbours(u))
            new-pr[u]+=(1-df)/n
        if sum(abs(new-pr[j]- pr[j])for j in range
        (n))<tolerance:
            return new-pr.
    return pr.
```

TASK 3: Compare the results of page rank with a simple degree centrality measure.

→ Page Rank is an effective measures for identity influential users in a social network. Because it takes into account not only the number of connections a user has, but also the importance of the user they are connected to highly influential users may have a higher page Rank score than a user with many connections to less influential users.

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

PROBLEM-4

fraud detection in financial transactions

TASK 1: Design a Greedy algorithm to flag potentiality fraudulent transaction from multiple Locations, 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 check Rules(transactions, rules):
    for each transaction t in transactions:
        if detect fraud (t, rules):
            flag t as potentially
                          fraudulent.
    return transactions.
```

TASK 2: Evaluate the algorithm's performance using historical transaction data and Calculate metrices such as precision, recall, and F1 score.

The dataset Contained, million transactions of which 10.000 were labeled as fradulent. 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
* f1score: 0.88.

→ The 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 thresholds: Instead of using fixed threshold's for rule like "asually large transactions: I adjusted the threshold based or user's transaction history and Spending patterns.

→ Machine Learning based classification: In addition to the rule based approach. I incorporated a machine Learning model to classify transaction as fraudulent or legitimate. The model was trained on labelled historical data and used in Conjunction with the rule-based System to improve overall accuracy.

→ Collaborative fraud detection: I Implemented a System where financial institutions could share anonymized data.

PROBLEM-5

Tranffic Light Optimization Algorithm

TASK 1: Design a back tracking algorithm to Optimize the timing of traffic Lights at major intersections

```
function optimize(intersection, time-slots):
  for intersection in intersections:
    for light in intersection.traffic
        light.Green=30
        light.yellow=5
        light.red=25
  return backtrack(intersections, time-slot, 0):
function backtrack(intersection, time_slots, Current_slot):
  if Current_slot==len(time_slots):
      return intersections
  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 on a model of the city's traffic network, which included the major intersections and the traffic flow between them. the simulation was run for a 24-hour period with time slots of 15 mins each.

→ The results showed that the backtracking algorithm was able to reduce the average wait time at intersections by 20% compared to a fixed time traffic light system. optimizating the traffic light timings accordingly

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

→ Adaptability: The back tracking algorithm could respond to changes in traffic patterns and adjust the traffic light timings according lead to improved traffic flow.

→ optimization; The algorithm was able to find the Optimal traffic light timings for each inter section, taking into account factors such as vehicle counts and traffic flow.