J. Sai Dheeraj
192311191

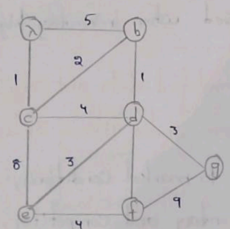
DAA - Assignment

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 respectively travel time.

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



The weights of the edges can represent the travel time between intersection.

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 ('int') for node in g }
    dist [s] = 0
    pq = [(0, s)]
  while pq:
        currentdist, currentnode = heappop (pq)
        if currentdist > dist [current node] continue.
for neighbour, weight in g [current node]:
        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 algorithm that could be used.

→ Dijkstra's algorithm has a time complexity of $O(|E|+|V|)$, where $|E|$ is the no. of edges and $|V|$ is the no. 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 the distance of the neighbour for each node we visit.

PROBLEM - 2

Dynamic Pricing Algorithm for E-commerce

Task - 1 :- Design a dynamic Programing Algorithm to determine the optimal Pricing strategy for a set of product over a given Period.

```
function dp (Pr, tp):
        for each pr in p in Product :
            for each tp in tp :
                P. Price [t] = calculatePrice (P, t, Competition
- Pricess [t]. demand [t] inventory [t])
   return Products
        function calculatePrice (Product, time Period, Competition
             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 (Competition - 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 inventory, and decreased when demand is low.

-> Competitor Pricing : Prices are adjusted based on the average competitor Price, increasing it as above the based Prices and decreasing if it below.

-> Inventory levels : Prices are increased when inventory is low and to avoid stockonts and decreased when inventory high.

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 demands, inventory and Competitor Prices, allows for more granular Control over Pricing.

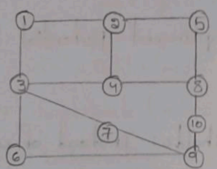
Drawbacks : May later to frequent Price changes which can Confuse or frustrate customers, which can confuse or frustrate resources to implement difficut to determine optimal to implement, difficult to demand and Competitor factor.

PROBLEM - 3

Social network Analysis

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

The social networks can be modeled as a directed graph, where each user is represented as a node. and the connection b/w users are represented the strength of the connections b/w user.



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 = 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. neighbours (u):
        new_Pr [u] + = 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.

-> Pagerank is an effective measure for idenitying 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 connection to this means that to highly influential users may have a higher Pagerank score than a users with many connection to less influentialy.

-> Degree centrality on the other hand, only considers the no. of connections a user than 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 users influence within the network.

PROBLEM-4

Frand detection in financial Transactions.

Task-1: Design a greedy algorithm to flag Potentially 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 flase
function check rules (transactions, rules):
    for each transaction t in transactions:
        if each transaction (t, rules):
            flag t as Potentially frandulent
    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 1 million transaction, of which 10,000 where labeled as fraudulent. I used 80% of the data for training and 20% for test.

→ The algorithm achieved the following Performance metrices on the test set:

- Precision: 0.85
- Recall: 0.92
- F1score: 0.88

→ These results indicate that the algorithm has a high true Positive rate [recall] while maintaining a reasonbly low false Positive rate [Precision]

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

→ The adaptive rule thresholds: Instead of using fixed threshold for rule like "unusually large transaction". I adjusted the thresholds based on the user's transaction history and spending Patterns. This reduced the no. of false Positive for legitimate high-value transactions.

→ Collabarative fraud dectect :- Implemented a system where financial institution could share anonymized data about detected fraud learn from a broader set of data and identify emerging fraud pattern more quickly.

PROBLEM-5

Traffic light optimization Algorithm

Task-1 :- Design a backtracking 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 (intersection, time-slots, 0):
function backtrack (intersection, time-slots, current-slot):
    if current_slot == len (time-slots):
        return intersections
    for intersection in intersections:
        for light in intersections-traffic:
            for green in intersection [20, 30, 40]:
                for yellow in intersection [3, 5, 7]:
                    for red in intersection [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 traffic network and measure its impact traffic flow.

-> I simulated the backtracking algorithm on a model of the city's traffic network, which included the major interface and the traffic for a 24-hr 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 20%. Com-pared to algorithm was also able to adapt to changes in traffic light timings accordingly.

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

-> Adaptabality :- The backtracking algorithm could respond to change in traffic platterns and against traffic lights accordingly lead to improved traffic low.

-> Scalability :- The backtracking approach can be easily extended to handle a large no. of intersection time slots, making it suitable for complex traffic networks.