PROBLEM - 5

Traffic light optimization Algorithm.

TASK 1 :- Design a backtracking algorithm to optimize the timing of traffic at major intersections.

```
function optimize (intersections, time - slots):
    for intersection in intersections:
        for light in intersection traffic
            light.green = 30
            light.yellow = 5
            light.red = 25
    return backtrack (intersections, time slots, 0): function.
backtrack (intersections, time - slots, current - slot):
    if current - in 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 (intersections 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 an 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 intersections by 20% compared to a fixed time traffic light system the algorithm was also able to adapt to changes in traffic pattern throughout the day, optimizing the traffic light timings accordingly.

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

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

→ Optimization :- The algorithm was able to find the optimal traffic light timings for each intersection taking into account factors such as vehicle decking counts and traffic flow.

→ Scalability :- The back tracking approach can be easily extended to handle a larger number of intersection and time slots making networks.

PROBLEM 4

fraud detection in financial tran

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 false

function checkRules(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 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 positive rate (recall) while maintaining a reasonably low false positive rate (precision).

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

→ Adaptive rule thresholds: Instead of using fixed thresholds for rules like "unusually large transactions" I adjusted the threshold based on transactions' I adjusted 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 transactions as fandament or legitimate. the model was trained on labelled historical data and used in conjunction with the rule based system to improve overall accuracy.

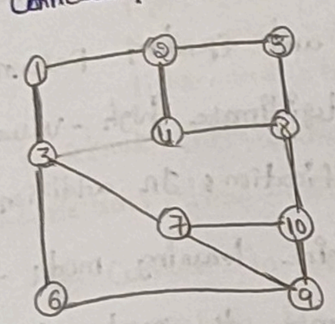
→ Collaborative fraud detection: I implemented a system where financial institutions could share anonymized data about detected fraudent transactions, this allowed the algorithm to learn from a broader set of data and identify emerging fraud patterns more quickly.

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 weighted to represent the strength of the connections between users.



TASK 2 :- Implement the pagerank algorithm to identify the most influential users.

```
function PR (g, df = 0.85, mi = 100, tolerance = (0-6)):
    n = number of nodes in the graph
    pr = (1/n)*n
    for i in range (mi):
        new - pr = (0)*n
        for n in range (n):
for v in graph.neighbours (n):
        new - pr(v) += df * pr(n) / len (g.neighbors (n))
    new - pr(n) += (1-df) / n
if sum (abs (new - pr(3) - pr(3) for ; in range)
        < tolerance:
    return new - pr
return pr
```

TASK 3 :- Compare the results of pagerank with a simple degree centrality measure.

→ Page Rank is an effective measure for identifying influential users social network because it takes into account not only the number of connections a user has, but also the ber of connections a user has connected to this means that a user with fewer influential users may have a higher page rank score than a user with many connections to ly 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.

Week-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 tp t in tp: .
P. price [t] = Calculateprice (P, t)
Competion - 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 (competition - price):
    if avg (competitor - prices) < product base - price:
        return - 0.05
else:    return 0.05.
```

TASK 2:- Consider factors such as Inventory levels, competitor pr demand elasticity in your algorithm.

→ Demand elasticity: prices are increased when demand is high relati to Inventory, and decreased when demand is low.

→ Competitor pricing: prices are adjusted based on the average compet -If on price increasing 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 stimulate demand.

→ 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 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 resource to implement different to determine optimal parameters for demand and competitor factors.

CSA0676

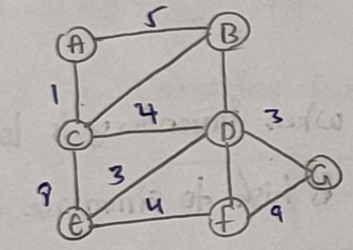
192325043
Gangjaty. Nithin Kumar

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 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 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 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:
        for neighbour weight in g[current node]:
            distance = current dist + weight
            if distance < dist[neighbour]:
                dist[neighbor] = distance
                heappush(pq,(distance, neighbors))
    return dist
```

TASK 3:- Analyze the efficiency of your 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 efficiently find the node with the minimum distance and we update the distance of the neighbors for each node are utilit.

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

→ Anthoer improvement could be to use a bidirectional search where we