ASSIGNMENT

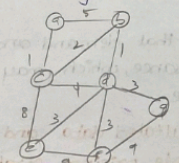
NAME : P. Akhil Kumar
REG : 192373024.

PROBLEM-1

Optimizing Delivery Routes

Task-1 : Model's 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'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 represents 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:
        current dist, current node = heappop(pq)
        if current dist > dist [current node]:
            continue.
        for neighbour, weight in g[current node]:
            distance = current dist + weight
            if distance < dist[neighbour]:
                dist[neighbour] = distance
                heappush (pq,(distance, neighbour))
    return dist
```

Task 3: Analyse 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 because we use a priority queue to efficiency find the node with the minimum distance and we update the distances of neighbours to each node we visit.

→ 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 of the algorithm.

→ Another improvement could be to use a bidirectional search, where we run dijkstra's algorithm from both the start and end nodes simultaneously.

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] = calculate price(P, t, competitor_
            prices[t], demand[t], inventory[t])
    return products
function calculate price(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 relative to inventory and decreasing cased 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.

→ Inventory levels: prices are increased when inventory is low to avoid stockouts, and decreased when inventory is high to simultate 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 simultated data and compare its performance with a simple pricing strategy

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

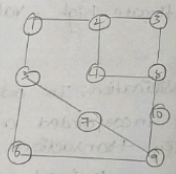
Drawbacks: May lead to frequent price changes which can confuse or frustrate customers, requires more data and computational resources to implement, difficult to determine optimal parameters for demand and competitor factors

Problem-3

Do 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 page rank algorithm to identify the most influenced users.

```
Function 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.neighbours)
            new_pr[u] += (1-df)/n
        if sum(abs(new_pr[j] - pr[j]) for j in range
            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 identifying 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 users they are connected to this means that a user with fewer connections but who is connected to highly influenced users may a higher page rank score than a user with many connections to less influential users.

→ Degree centrality on the other hand only consider the numbers 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 the users influence within network.

Problem 4

Fraud 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 detect_fraud(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 algorithms performance using historical transaction data and calculate matrices such as precision, recall and F1 score.

The dataset contained 1 million transactions of which 10,000 were labelled 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 improvements to this algorithms.

→ Adaptive rule threshold: Instead of using fixed threshold for rule like usually large transactions, his steady and spending patterns. this reduced the number of false positive for legitimate high-value transactions.

→ Machine learning based classification: In addition to rule-based approach I incorporated a machine learning model to classify transactions fraudulent or legitimate. 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 financial institutions could share transactions this allowed the algorithm to learn from a broader set of data and identifyer fraud patterns.

Problem 5:

Traffic light optimization algorithm

Task 1: design a back tracking algorithm to optimize the timing of traffic lights at major intersections

```
Functions optimize (intersections, time_slots
    for intersection in intersections:
        for light in intersections. traffic
            light . green = 30
            light . yellow = 5
            light . red = 25
    return back track (intersections, time_slots, 0):
    functions back track (intersections, time slots current_
                                                slot):
if current_ slot == len (timeslots):
    return intersections
  for intersection in intersections:
    for light intersection in intersections:
    for green in [20,30,40]:
        for. yellow = yellow
        light . red = red
result = back track (intersections, time_slots)
      current slot +1
if result is not none:
  return result
```

Task 2: Simultate 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 city's traffic network. which included the major intersection and the traffic flow between them the simultations run for a 24-hour period, with time slot of 15-min each.

→ The results showed that the backtracking algorithm was able to reduce the average wait time at intersection 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 timing accordingly.

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

Adaptability: the back tracking algorithm could respond to changes, in traffic patterens and adjust the traffic light timings accordingly lead to improved traffic flow.

Optimization: the algorithm was able to find the optimal traffic light timings.

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