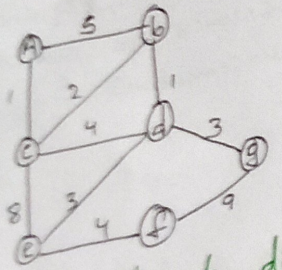
Problem-1

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'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 warehouse to various delivery locations.

```
function dijkstra(g,s):
    dist = {node:float('inf') for node in g}
    dist[s] = 0
    pq = [s]
    while pq:
        currentdist, currentnode = heappop(pq)
        if currentdist > dist[currentnode]:
            Go
```

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 efficiently find the node with the minimum distance, and we update the distances of the neighbours for 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 nodes simultaneously. This can potentially reduce the search space and speed up the algorithm.

Problem-2

Dynamic Pricing Algorithm for E-Commerence.

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] = Calcul
```

Competi

```
function CalculatePrice(Product, TimePeriod, Competition,
        Prices, demand, inventory):
Price = Product.basePrice
Price += 1 + demand.factor(demand, inventory):
  if demand > inventory:
      return 0.2

  else:
    return 0.1
function CompetitorFactor(CompetitorPrices):
  if any(CompetitorPrices) 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 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.

Task 3:- Test your algorithm with simulates 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.

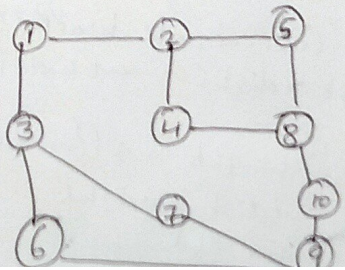
Drawbacks: May lead to frequent Price changes which can confuse or frustrate customers, requires more data and computational resource to implement difficult to determine optimal parameters for demand and Competitor factor

Problem 3

Social network Analysis.

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

The social network can be modeled as a directed graph. Where each user is represented as a node and the connections between user are represented as the 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 influential 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(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 Pagerank with a simple degree Centrality measure.

→ PageRank 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 with fewer connections but who is connections. but who is connected to highly influential users may have a higher PageRank score than a user with many connections to less influential users.

Problem 1:

Fraud detection in financial transactions

Task 1: Design a greedy algorithm to flag fraudulent transactions based on a set of predefined rules.

```
function detectFraud(transactions, rules):
    for each rule in rules:
        if rule.Check(transaction):
            return true
    return false
function CheckRule(transaction, rules):
    for each transaction t in transactions:
        if elected fraud (t, rules):
            flag as Potentially
                fraudulent
    return transactions
```

Task 2:

Evaluate the algorithm's performance using historical transaction data and calculate metrics and F1 Scale.

→ The algorithm delivered the following performance metrics on the test set

- Precision: 0.85
- Recall = 0.92
- F1Scale = 0.88

→ These result indicate that the algorithm has a high true positive rate while maintaing a reasonably low false positive rate.

Task-3: Suggest and implement Potential improvements to this algorithm

-> Adaptive rule thresholds instead of using fixed thresholds. For rule like unusually large transactions I adjusted the thresholds based on the users transaction history and spending patterns. This adjusted the reduced high-value fractions.

-> Machine leaning based classification: In addition to the rule based approach, I incorporated a machine learning model to classify transcations.

-> Collabrative fraud detection: Implemented a system where financial institutions could share anonymized data about detected fraudulent transaction this allowed the algorithm to lean from a broader set of data.

Problem : 5

Traffic light optimization algorithm.

Task 1: Design a backtracking algorithm to optimize the timining of traffic lights at major intersections.

```
function optimize (intersection, time-slots):
    for intersection in intersections:
        for light in intenesection, traffic.
            light. green = 30
            light. yellow : 5.
            light. red = 25
return back track (intersections, time-Slots, 0).
function backtrack (intersections, time, slots,
                    current_slot):
    if current_Slot == len (time-slots)
        return intersections.
    for intersection in intersections:
        for green in [20, 30, 40]:
            for yellow in [3,5,7]:
            for red in [20, 25, 30]:
                light. green = green.
                light-red : red.
result = backtrack (intersection time slots,
                    if.
result is not none: current_slot +1)
        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 intersection and the traffic flow between them the Simulation was run for on 24-hours Period.

→ The results showed that the backtraining 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 Changes in traffic Pattern the day.

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

→ Adaptability the backtracing algorithm Could respond to Changes in traffic Pattern and adjust the timings lead to improved.

→ Scalability:- The backtracking approach can be easly extended handle a langer number of time Slots. making for traffic networks.