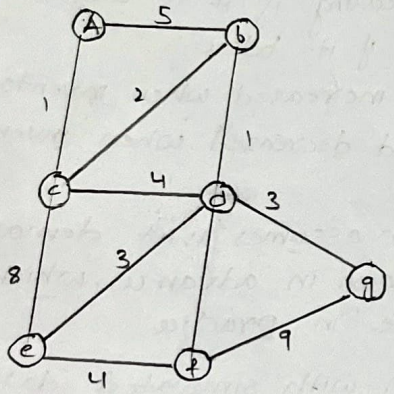
PROBLEM-1

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 dijikstra's algorithm to find the shortest paths from a central warehouse to various delivery locations

```
function dijikstra (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 alternatives algorithms that could be used.

→ dijikstra 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 use a bidirectional search, where we run dijikstra'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 tp t in tp:
            P.price[t] = calculateprice(p, t,
competitor-prices[t], demand[t], inventory[t])
return products
function calculateprice(product, time_period)
    competitor_price, 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 any(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 if it is above the base price and decreasing if it below.

→ Inventory levels: prices are increased when inventory is low to avoid stockants 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

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 control.

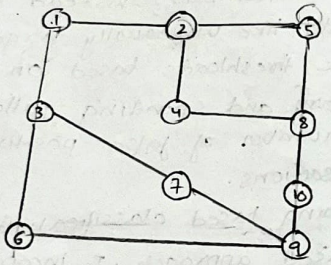
Drawbacks: May lead to frequent prices changes which can confuse or frustate to customers, requires more data and computational resource to implement, difficult to determine optimal parameters for demand and competitor factors

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 strenght of the connections between users



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

```
functioning PR(g, df=0.85, mi=100, tolerance=1e-5):
    n=numbers 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(u):
            new_pr[v] += df * pr[n] /len (g.neighbours(n))
    new_pr[n] += (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 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 influential users may have a higher pagerank score than a user with many connections to less influential users.

→ Degree centralized, 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 users influence within the network.

PROBLEM-5

PROBLEM 4

Fraud detection in financial transactions

TASK 1:- Design a greedy algorithm to flag potentially fradulent transactions 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 transactions t in transactions :
        if detectFraud (t, rules):
            flag t as potentially fraudulent
    return transactions
```

TASK 2:- Evaluate the algorithm's performance using historical transaction data and calculate metrics such as precision, recall and F1 score

The dataset contained 1 million transactions, of which 10000 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:

2:- Simulate the algorithm on a model often

- 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 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 rule like unusually large transactions I adjusted the threshloads based on the user's transaction history and spending patterns. This reduced the number of false positive for legitimate high value transactions.

→ Machine learning based classifications: In addition to the rule based approach, I incoporated a machine learning model to classify transactions as fraudulent.

→ collabrative fraud detection:- Implement A system where financial iteration counted share anomized data about detected fraction

PROBLEM-5

Traffic light optimization algorithm

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

```
function optimize (intersection, time, solt, slots):
    for intersections in intersections
        for light in interaction.traffic
            light.green = to 300
            light.yellow=5
            light.red=25
    return backtrack (intreactions, time slots)
function backtrack (intersection, time, slots, correct_slot)
    if current-slot == len (time-slot)
        return intervention
    for light intervention_traffics:
        for green in [20,30,40]:
        for yellow in [3,5,4]
            for red in [20,25,30]:
        light.green = green
        light.red = red
        light yellow = yellow
result = backtrack (intunection, time_slots)
    if (result is not None : current slot + 1)
        if result is not non current : return result
```

TASK 2 :- simulate the algorithm on a model often city traffic network and measure its impact on traffic flow:

* I simulated the back tracking algorithm on a model of the city traffic network which included the major intersections and the traffic in flow between item, the simulation wal run for 24-hour period (with the time of 15 min each
* The result showed that the back tracking algorithm was able to reduce the average. wait at intersection by 20 /. compared to a fixed time traffic light system.

TASK-3 :- Compare the performance of your algorithm with a fixed time weight

* Adaptadly :- the backtracking algorithm could respond to chaus in traffic patterns and adjust the traffic light
* optimization : the algorithm was able to find the optimal traffic light timings for each intersection, taking into account factors such as vehicle count and traffic flow.