G. Lakshmi Mounika
192372110 CSA-0677

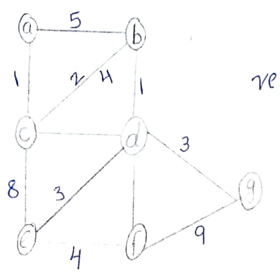
Design Analysis of algorithm

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 weight representing travel times.

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 = [(0,s)]
    while pq:
        currentdist, currentnode, heappop(pq)
        if currentdist > dist [current node] :
            continue
        for neighbour, weight in g [currentnode]
            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 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 & we update the distances of the neighbors 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 operations performance of the algorithm

-> Another improvement could be to use a bidirectional search, where we run dijkstra's algorithm from 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 stategy for a set of Products over a given Period

```
function dp (pr, tp):
    for each pr in p in products:
        for each tpt in tp:
            p. price [t] = calculate price (p.t,
competition - 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, Compe pricing, and demand elasticity in your algorit

-> Demand elasticity: Prices are increased when demand high relative to inventory, and decreased when deman is low.

-> Competitor Pricing: Prices are adjusted based on the av ge Competitor price, increasing if it is above the base p and decreasing if it is below

-> Inventory levels:: Prices are increased when inventory is low to avoid stacknots, 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 Pratice

TASK 3:- Test your algorithm with simulated data and Compare its performance with a simple static pricing Stategy.

Benefits: Increased revenue by adapting to market conditions, optimizes Prices based on demand, inventory, and Competitor Prices, allows for more granular Control over Pricing

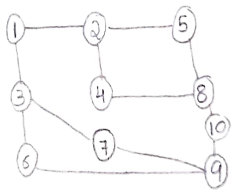
Drawbacks: May be lead to frequent Price changes which can Confuse or frustrate customers, requires more data and Computational resource to implement, diffenlt to determine optimal Parameters for demand and competitor factor

PROBLEM-3

Social network Analysis

TASK 1:- Model the social network as a graph where users are nodes and connection are edges

The social edges networks can be modeled as a directed graph, where each users is represented as a node. and the connections between users are represented as edges. The edges are 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 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[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

-> 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 with fewer connection but who is connected to highly influential users may have a higher PageRank score than a user with many connections to less inflential users

-> Degree centrality on the other hand, only considers the number of connections a user has, without taking into accounts the importance of those connections. while degree certality can be a useful measure in some scenarios it may not be best indication of a user's influence within the network.

PROBLEM-4

Frand detection in financial Transactions:

TASK 1:- Design a greedy algorithm to flag Potentially fradulent 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 check Rules (transactions, rules):
    for each transaction t in transactions:
        if detect fraud (t, rules):
            flag t as Potentially frandulent.
    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 10,000 were labeled as frandnlent 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 hi true positive rate [recall] while maintaing a reasonabl low false Positive rate [precision]

TASK 3: Suggest and implement Potential improvements this algorithm.

-> Adaptive rule threshold: Instead of using fixed th sholds for rule like "unusually large transactions", adjusted the thresholds based on the users transacti history and spending Pattern's. This reduced the nu ber of false Positive for legitimate high-value transactions.

-> Collaboration fraud detection: I implemented a sys where financial institution could share anonymized data about detected fraudulent transactionas. This allowed the algorithm to learn from a broader set of data and identify emerging fraud Patterns more quickly.

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-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 ( intersections, time, solts, current
                                                slots):
    if current_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 simulations was run for an 24-hour period, with time slots of 15 min each

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

-> Adaptability: The backtracking algorithm could respond to changes in traffic Patterns and adjust the traffic light timings accordingly lead to improved traffic flow