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

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

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

PROBLEM-5:

Traffic light optimization Algorithm

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

```
function optimization (intersections, time-slots):
  for intersecting in intersections:
for light in intersection.traffic:
      light.green = 30
      light.yellow = 5
      light.red = 25
return backtrack (intersections, time-slot, 0):
function backtrack (intersections, time slot, current-slot):
  if current-slot == len(time-slot):
     return intersections
  for interactions in intersections:
  for light in interaction.traffics:
    for green in [20, 30, 40);
    for yellow in [20, 25, 30]:
       light.green = green
       light.yellow = yellow
       light.red = red
result = back track (intersections, time, slot, if result is not sense,
current, slot +1)
  return result
```

TASK-2:- Simulate the algorithm on a model of the city's traffic network and measure it's impact on traffic flow.

-> Simulated the back tracking algorithm on model of the city's traffic network, which included the major intersections and the traffic included the simulation was run for a 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 light system. The algorithm was able to adapt to changes in traffic patterns throughout the day optimizing the traffic light timings accordingly.

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

PROBLEM 4 :-

Fraud detection in financial Transactions :

TASK 1:- Design a greedy Algorithm to flag potentially fraudulent transaction from predefined rules.

```
functions detected fraud (transaction, rules) :
 for each rule r in rules :
if r. check (transaction) :
   return true
return false
function . Check Rules (transaction, 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 calculate metrics such as precision, recall and F1 score.

The dataset contained, million transactions, of which 10,000 were labelled as fraudelent. 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
- F1score : 0.88

→ These results indicate that the algorithms has a high true positive rate [recall] while maintaing a reasonably low false positive rate [precision].

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

→ Adaptive rule threshold's : Instead of using fixed thresholds for rule like "unusually large transactions". I adjusted the thresholds based on the user's transacted history and spending pattern's. This reduced the number of false positive for legimate high-value transactions.

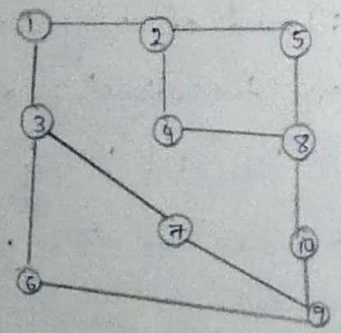
→ Collaborative fraud detections. I implemented band where financial institutions could share financial institutions could share anonymized data detected fraudulent transactions. This allowed the algorithms to learn from a boader.

PROBLEM-3 :

Social network Analysis :

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

The social network can be modeled as a directed graph, where each user is represented as a node. and the connections between uses are represented as edges. The edges can be weighted to represent the strength of the connections between them.



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

```
functioning 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 - neighbour (u):
      new - pr[v] = df [n] / len (g. neighbour (u))
       new _ pr [n] = (1-df )/n
      if sum ( abs (new _ pr[i] - pr(j]) for j in range
            (n)< tolerance :
        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 identyfying influential users in a Social network. because it takes into account not only the number of connections a users they are connected to. This means than a users with fewer connections users may have a higher page Rank Score than a user with many connections to less influential users.

-> Degree Centrality on the other hand, only Consider the number of Connections a user has without taking into account the importance of those Connections while degree centrality can be a useful measures in Some scenarios, 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, competition - prices (t), demand (t),
inventory (t)]
 return products
function calculate price (product, time, period, compelitor, prices, demand):
  price = product.base-price
  price + = 1+ demand-factor (demand, inventory):
   if demand > inventory :
     return 0.2
  else:
  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 elastricity : 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 stackonts, 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.

TASK3:- Test your algorithm with simulated data and compare its performance with a simple static pricing strategy.

Benefits: Increased revenue by adapting to marked conditions, optimizes prices based on demand inventory, and competitor prices, allows for more granualar control over pricing.

Analysis the efficiency and potential improvements :

Dijisktra's Algorithm has a time complexity of $O(|V|+|E|)\log|V|$ using a binary heap (or) $O(|V|+|E|)\log E)$ using a Fibonacci heap, where $|V|$ is the number of vertices (intersections) and $|E|$ is the number of edges (roads). The space complexity is $O(|V|)$ for storing the distance and previous nodes.

Analyzing the Efficiency and potential improvements :-

1. Use a more efficient data structure. Instead of a binary tree using a Fibonacci heap can improve the time complexity to $O(|V|+|E|)$ $\log|V|$.

2. Implement bidirectional Search :

By Searching from both are source and destination simultaneously the search space can be reduced potentially reducing to faster results.

3. Utilize real-time traffic data :

Incorporating live traffic data time into the edge weighs can provide more accurate and up-to data route recommendation.

Assumptions and Considerations :-

1. Non-negative weights :-

Dijkstra's Algorithm assume that all edge weighs (travel times) are non-negative. If negative weights are present, the algorithm may not find the correct shortest paths.

2. Static road network :-

The current implementation assume a static road network. In reality road conditions can change due to factors such as traffic. road closures (or) constructions.

Implementing * Djistra's * Algorithm :-

Dijkstra's Algorithm is suitable for this problem because it finds the shortest path between a source node and all other nodes in a weighted graph with non-negative edge weighs. Here's the pseudocode for Dijkstra's algorithm.

pseudo code :-

```
functions Dijkstra's (Graph, Source):
 for each vertex v in Graph
     dist [v] = Infinity
     prev [v] = Undefined

     dist [source] = 0
 Q = The set of all nodes in Graph while Q is not empty:
 u: = Vertex in with smallest dist [u]
    remove u from Q
 for each neighbour v of u still in Q:
    alt: = dist [u] + weight (u+v)
  if alt < dist [v]:
    prev [v] : = u
 return dist, prev
 import hap q
def  dijkstra (graph, source);
  dist = {node : float ('inf') for node in graph }
    dist [source] = 0
  prev = {node: None for node in graph}
  heap = [( 0, source)]
  while heap;
 Current_dist , current_node = heap q.heap pop (heap)
 if current_dist > dist [current_node]:
    Continue
for neighbour weight in graph [current_node].items ():
  distance = current_dist + weight
  if distance < alt [neighbour]
    prev [neighbour] = current.node
  heap.q . heap push (heap, (distance , neighbour))
   return dist, prev
```

Name :- A.Akash
Reg No :- 192373015
Code :- CSA0676

ASSIGNMENT

PROBLEM 1 :-

OPTIMIZING DELIVERY ROUTES :-

Modelling the city's Road network as a graph :-

To model the city's road network as a graph we can represent each intereseсtion as a made and each node road connecting two intersections as an edge the weighs of edges can represent the travel time between the Connected intersections..

Here's an example of how the graph could be represented.

Graph:-

{

'A' = {'B': 5, 'C' : 1}

'B = {'A': 5, 'C': 2, 'D' : 1},

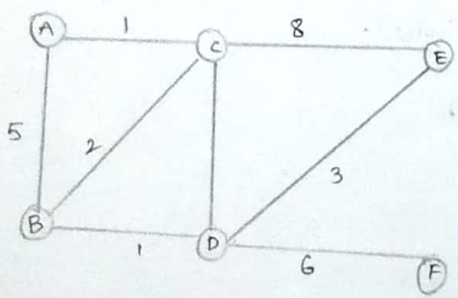
'C' = {'A' :1, 'B' :1, 'D' : 4, 'E' : 8},

'D = {'B' :1, 'C': 4, 'E': 3, 'F' :6 },

'E' = {'C': 8, 'D': 3},

'F : {'D = 6}

The above example graph should be represented as



In this example, the graph is represented as a dictionary. Where each key represents a node and is value is another dictionary containing the neighbouring nodes and their corresponding weighs.