Assignment

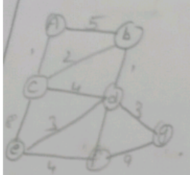
Course Code :- CSA 0677
Reg.no :- 192365028
Name :- G. Charan bhargav

Problem - 1 :-

Optimizing delivery routes :-

Task - 1 model the city's road networks 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 edge Can represent the travel time between intersections.

Task - 2 :- Implement diikstar's algoritham to find the shortest paths a Central warehouse to various delivery locations.

```
functions diikstra (s.s):
    dist = {node : float ('int') for node ing}
    dist [s] = 0
    P2 = [(0,s)]
    while P2 :
  Current dist, Current node = heapPop(Pa)
  if Currentdist → dist [current node]:
  Continue
for neighbour weight ing[current node]:
distance = Current dist + weight
if distance = dist [neighbour]:
    dist [neighbour] = distance
    heappush (Pa (distance , neighbour))
  return dist.
```

Task 3 :- Analyze the efficancy of your algoritham and discuss any Potential improvements (or) alternative algorithams that Could be used.

⇒ diikstar's algoritham 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 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 overall performance of the algoritham.

Problem - 2 :-

Dynamic Pricing algoritham for E-Commerce.

Task 1 :- Design a dynamic programming algoritham to determine the optimal pricing strategy for a set of products over a given p

```
function dp (pr, tP):
    for each Pr in P in products :
        for each tp t in tp :
```

```
if sum(abs(new-Pr(i) - Pr(i)) for i in range(n)) < tol
    return new-Pr
```

```
P.Price(t) = Calculate Price (P.t)
Competiton - Prices(t) demand (t), inventory(t)
return products
function Calculate Price (Product, time-Period)
Price = Product.base-Price
Price *= 1+ demand_factor(demand, inventory):
    return 0.2
else
    return 0.1
function Competation_factor(competitor Prices):
    return -0.05
else:
    return 0.05
```

Task-2:-

Consider factor such as inventory levels Competitor Pricing, and demand elasticity in your algoritham.

→ Demand elastricity:

Prices are increased when demand is high relative to inventory and decreased demand is low.

→ Competitory Pricing:-

Prices are adjusted based on 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 stimulate demand.

→ Additionally the algoritham assumes that demand and Compititor prices.

Task-3:-

Test your algoritham with simulated data and Compare its performance with a simple static pricing.

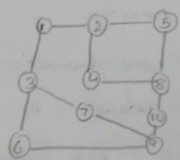
→ Benfits:

Increased revenue by adopting to market Conditions, optimize Prices based on demand, inventory and Competitor prices, allows for more grannuar Control over pricing.

Problem - 3:

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

The social networks 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 hage rank algoritham to identify the most influential users.

```
functiong PR(S) df = 0.85, mi = 100, tolerance = le - 6
        n = no. of nodes in the graph.
        Pr = (1/n) * n
  for i in range (mi):
        new - Pr = [0] * n
  for i in range (n):
for v in graph . neighbours (u):
 new - Pr (v) = df * Pr (u) / len (S. neighbous (u))
if sum( abs(new - Pr(i) - Pr(i)) for i in range (n) < tol
     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 identify influential users in a social network. because it takes into account not only the number of Connections a user has. also the importance of the users they connected to this means that a user with fewer Connections but who is connected to highly influential users.

→ Degree Centrality on the other hand only Considers the number of Connections a user has without taking into account the important of those Connections while degree Centrality Can be a harmful measure some scenarious, it may not be the best in of a user's influence within the network.

Problem 4:-

Trend detection in financial transactions

Task 1:- Design a greedy algorithm to flag Potentially fraudlent transaction from multiplication based on a set of Predefined rules.

```
function detectfraud (transation, rules):
    for each ruler in rules:
        if r.check (transection):
            return true
    return false

function check rules (transaction, rules):
    for each transaction t in transactions:
        if detect fraud (t, rules):
            flag t as potentially fradulent
    return transactions.
```

Task 2:- Evaluate the algorithams performance using historical transaction data and Calculate amount and score.

The dataset Contained 1 million transactions, of which 10,000 were labeled as frandulent used 80%. of the ..ta for training and 20%. for testing.

The algoritham achieved the following performace ..ics on the test set:-

* Precision :- 0.85
* Recall :- 0.92
* F1score :- 0.88

→ The results indicate that the algoritham has a high true Positive rate (recall) while maintaining a resonably low false Positive rate.

Task-3:- Suggest & implement potential improvements to this algoritham.

→ Adaptive rule thresholds : instead of using fixed thres.. for rule like 'unnsually' large transactions ; adjusted the thresholds based on the users transaction histo.. and spending Patterns. this reduced the no. false positive for legitimate high value t..

→ Collaborative fraud detection : implemented a sys.. where financial institutions Could share anony.. data and identify.

Problem 5.

Traffic light optimization algoritham

Task 1: Design a backtracking algoritham to optimize the timing of traffic light at major.

```
function optimize (intersections, time-slots):
    for intersection in intersection:
        for light in intersection traffic
            light . green = 30
            light . yellow = 5
            light . red = 25
    return backtrack (intersection time-slots 0]
function backtrack (intersection, time slots current-slots):
  if   current-slot = = len (time-slots):
       return intersection
  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):
result = backtrack (intersections, time slots)
```

Task 2: Simulate the algoritham on a model of the city's traffic network and measure.

→ simulated the back tracking algoritham 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 a 24-hour period with time slots of 15 min each.

→ The results showed that the backtracking algoritham was able to reduce the average wait time at intersections by 200 %. Compared to a fixed time traffic light system. The algoritham was through the day optimizing the traffic light timin accordingly.