CSA0675 DAA

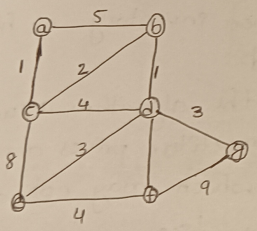
192373013 N. Pravalika

Problem - 1

Optimizing Delivery Routes

Task1: 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.

Task2: Implement dijkstra's algorithm to find 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[currentnode]: continue
        for neighbour, weight in g[currentnode]:
            distance = currentdist + weight
            if distance < dist[neighbour]:
                dist[neighbour] = distance
                heappush(pq, (distance, neighbour))
    return dist
```

Task3: Analyze the efficiency of ur alg & discuss any potential improvements.

→ 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. We use a priority queue to effectively find the node with min d.

→ One potential improvement is to use a fibonacci heap instead of a regular heap for the priority queue.

→ Another improvement could be to use a bidirectional search, where we run dijkstra's algorithm from both the start and ends nodes simultaneously. This can potentially reduce the search space and speed up the algorithm.

Problem-2

Dynamic Pricing Algorithm for E-commerce

Task1: 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.prece[t] = calculateprece (p, t, competitor_
preces (t), demand [t], inventory [t])
return products
function calculateprece (product, time_period, competitor_
preces, demand, inventory);
price = product.base_prece
price += 1 + demand_factor (demand, inventory);
        if demand > inventory:
            return 0.2
        else:
            return -0.1
function competitor_factor (competitor_preces):
    if avg (competitor_preces) < product.base_
    precs:   return -0.05.
    else:    return 0.05
```

Task2: Consider factor such as inventory levels, competitor pricing and demand elasticity in your algorithm

-> Demand elasticity: Preces are increased where demand is high relatve to inventory and decr eased when demand is low.

-> Competetor Precing: Preces are adjusted based on the average competetor prece, increasing if it is above the base prece & decreaseng if it below.

-> Inventory levels: Preces are invested when inventory is low to avoid stockouts & decreased when inventory is high to simulate demand.

-> Additionally, the algorithm assumes that demand & competetor prices are known in advance, which may not always be the case in practece.

Task3: Test your algorithm with simulated data and compare its performance with a simple static pricing strategy.

Benefits: Increased revenue, optimizes preces.

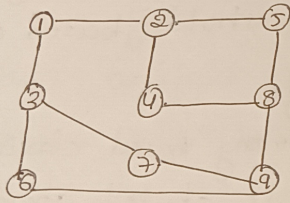
Drawbacks: May lead to frequent prece changes which can confuse or frustate customers.

Problem-8

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 strength of the connections between users.



Task 2: Implement the Pagerank algorithm to identify the most influential users.

```
functioning PR (g, df = 0.85, max = 100, tolerance = 1e-6):
    n = number of nodes in the graph
    Pr = [1/n] * n
    for i in range (max):
        new_Pr = [0] * n
        for v in range (n):
            for v in graph.neighbours(v):
                new_Pr[v] += df * pr[v] / len(g.neighbours(v))
            new_Pr[v] += (1-df) / n
        if sum(abs(new_pr[g] - pr[g]) for g
        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 identify 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 connected to. This means that a user with fewer connections but who is connected to highly influential user. user may have a higher pagerank score than a user with many connections to less influential users.

→ Degree centrality 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.

Problem-4

Fraud detection in financial transactions

Task1: Design a greedy algorithm to flag potentially fraudulent transaction from multiple locations, based on a set predefined rules.

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

Task 2: Evaluate the algorithm's performance using historical transaction data and calculate metrecs such as precision, recall & F1 score

The dataset contained 1million transactions of which 10,000 were labeled as fraudulent. I used 80% of the data for training and 20% for testing.

→ The algorithm achieved the following performance metrecs on the test set.

- Precision : 0.85
- Recall : 0.92
- F1 score : 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 thresholds based on the users transaction history.

→ machine learning based classification: In addition to the rule-based approach. I incorporated a machine learning model to classify transactions as fradulent or legetimate.

→ Collaborative fraud detection. I implemented share anonymized data about detected fraudent transactions. Identifies emerging fraud patterns more quikly.

Problem-5

Traffic light optimization algorithm

Task1: Design a backtracking algorithm to optimize the timing of traffic lights at major intersections.

```
function optimize (intersections, time_slots):
    for intersection in intersections:
        for light in intersection.traffic
            light.green = 30
            light.yellow = 5
            light.red = 25
    return backtrack (intersections, timeslots, 0);
function backtrack (intersections, time_slots, current_slot):
    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):
    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 intersections and the traffic flow between them.

→ The results showed that the backtracking algorithm was able to reduce the average wait time at intersections by 20% compared to a fixed time traffic light system.

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

→ Adaptability

→ optimization

→ Scalability