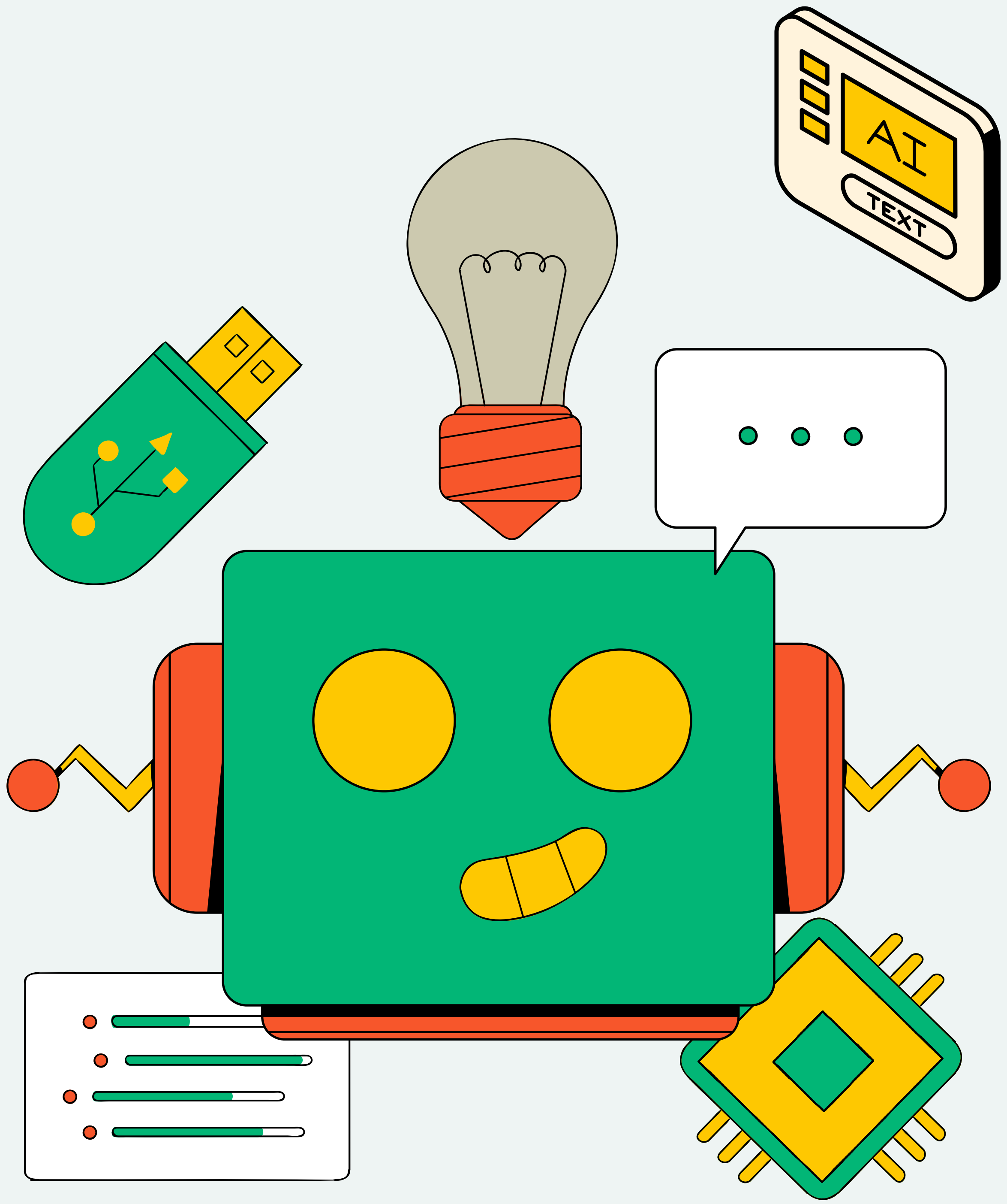
THYNK UNLIMITED

WE LEARN FOR THE FUTURE

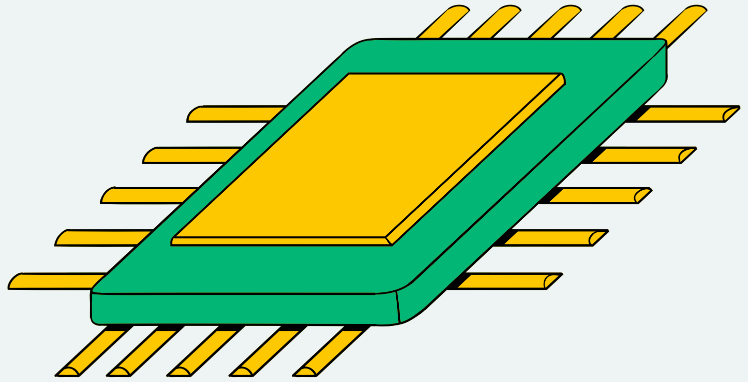


CASE STUDY 1: SOLVING REAL WORLD PROBLEM

PRELIM EXAM

PRESENTED BY:

KARL MALAKI
EUGENE DELA CRUZ
MARC DOLORES



PROBLEM

the chosen real world problem is the optimising delivery routes for logistics company by finding the fastest and shortest route possible

ALGORITHMS USED

- Graph Theory
- Dijkstra Algorithm

1ST ITERATION

problem identification

how can we optimize the delivery routes for logistics company in order to save travel time?

decomposition

sub problems:

- graphing the routes
- finding the shortest path

pattern recognition

using gps apps like waze and google maps

abstraction

relevant: routes, stopovers, distance
irrelevant: package, transportation

2ND ITERATION

problem identification

how can we find the fastest route in optimizing delivery routes?

decomposition

sub problems:

- finding shortest route
- calculating traffic

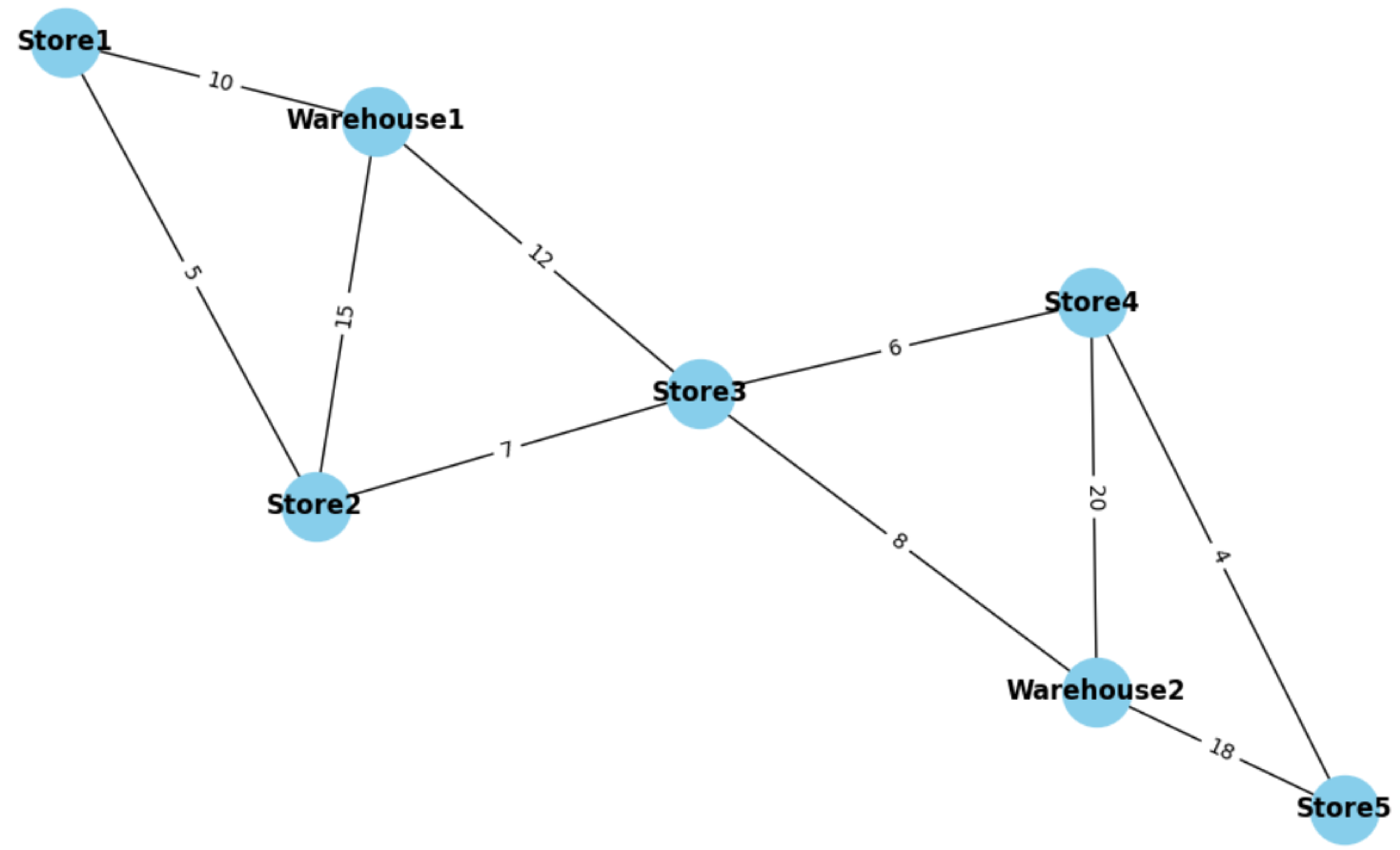
pattern recognition

using gps apps like waze and google maps

abstraction

relevant: routes, stopovers, traffic
irrelevant: transportation, road

GRAPH


Store1
10
Warehouse1
5
15
12
Store4
6
Store3
7
Store2
20
8
4
Warehouse2
18
Store5

CODES:

```
def shortestPath(self, src, dest):
    # Initialize distances and previous nodes
    distances = {node: float('inf') for node in self.adj}
    distances[src] = 0
    previous = {node: None for node in self.adj}

    # Priority queue implemented using a list
    pq = [(0, src)]

    while pq:
        # Pop node with the smallest distance from the priority queue
        current_distance, current_node = min(pq)
        pq.remove((current_distance, current_node))

        # Check if the current node is the destination
        if current_node == dest:
            break

        # Explore neighbors of the current node
        for neighbor, weight in self.adj[current_node]:
            distance = current_distance + weight
            # Update distance and previous node if a shorter path is found
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous[neighbor] = current_node
                pq.append((distance, neighbor))

    # Reconstruct the shortest path from src to dest
    shortest_path = []
    current = dest
    while current is not None:
        shortest_path.insert(0, current.getName())  # Append node name instead of node object
        current = previous[current]

    return shortest_path, distances[dest]
```

TESTING:

```
Warehouse1 ---->
    Store1 (Weight: 10)
    Store2 (Weight: 15)
    Store3 (Weight: 12)
Warehouse2 ---->
    Store3 (Weight: 8)
    Store4 (Weight: 20)
    Store5 (Weight: 18)
Store1 ---->
    Store2 (Weight: 5)
Store2 ---->
    Store3 (Weight: 7)
Store3 ---->
    Store4 (Weight: 6)
Store4 ---->
    Store5 (Weight: 4)
Store5 ---->
Enter source (Warehouse1/Warehouse2): Warehouse1
Enter destination (Store1/Store2/Store3/Store4/Store5): Store5

Shortest path from Warehouse1 to Store5 is ['Warehouse1', 'Store3', 'Store4', 'Store5']
Shortest distance from Warehouse1 to Store5 is 22
```