



CS 460 Final Project

Abdulkhaliq Mohamed




Problem - Task A

- Supermarket layouts can often be inefficient, leading to increased shopping times and customer crowding
- When sections like Dairy, Produce, Bakery, and others are not optimally placed in the store, customers may need to travel longer distances or navigate through crowded aisles to get their items
- This inefficiency not only frustrates shoppers but also affects the overall flow within the store, potentially reducing customer satisfaction and sales
- Therefore, there is a need to optimize the layout of supermarkets to minimize the time customers spend collecting items and to enhance their shopping experience



Solution

- To address this problem, we can apply Dijkstra's algorithm, a graph-based algorithm for finding the shortest paths between nodes, to the supermarket layout
- By modeling each section of the supermarket as a node and the pathways between them as edges with weights representing distance or travel time, Dijkstra's algorithm can determine the most efficient routes through the store
- This approach helps in designing store layouts that minimize crowding and enhance the shopping experience by reducing the overall travel time for customers



Python Implementation

```
import heapq

def dijkstra(graph, start):
    #dictionary will store the shortest path to each node
    shortest_paths = {node: float('inf') for node in graph}
    shortest_paths[start] = 0
    #priority queue to store nodes to be explored and initialized with the start node
    priority_queue = [(0, start)]

    while priority_queue:
        #gets the node with the smallest distance
        current_distance, current_node = heapq.heappop(priority_queue)

        #nodes can be pushed multiple times with different distances
        #only process a vertex the first time we remove it from the priority queue
        if current_distance > shortest_paths[current_node]:
            continue

        #explores each neighbor of the current node
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight

            #only consider this new path if it's better
            if distance < shortest_paths[neighbor]:
                shortest_paths[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return shortest_paths
```



Code Explanation

- We use Python's `heapq` module to manage the priority queue for Dijkstra's algorithm
- You start at the entrance of the supermarket and calculate the shortest path to all other sections, updating the path lengths and predecessors as you traverse through the graph. The result is a dictionary where the keys are the sections of the store and the values are the minimum distances from the entrance to each section
- This can help in determining the optimal path for a customer's shopping route or an employee's restocking route, which ultimately optimizes the internal flow of the supermarket
- Time complexity: $O((V + E) \log V)$ where E is the number of edges and V is the number of vertices
- Space complexity: $O(V + E)$ where E is the number of edges and V is the number of vertices
- Pretty much the same complexity as Dijkstra's algorithm

Output and Explanation

- The following is a sample code that we can input into our supermarket algorithm to get output
- After running the program, this is the output:

Shortest paths from Entrance: {'Entrance': 0, 'Produce': 5, 'Bakery': 2, 'Dairy': 5, 'Meat': 8, 'Checkout': 6}

- In this example, the shortest path from the entrance to the checkout is 6 units (units could be travel time, such as minutes, or distance, such as meters/feet)
- This data helps in designing the layout to minimize overall customer travel time
- Enhanced shopping experience with efficient routing

```
#example supermarket layout as a graph
supermarket_graph = {
    'Entrance': {'Produce': 5, 'Bakery': 2},
    'Produce': {'Dairy': 2, 'Meat': 3},
    'Bakery': {'Dairy': 3},
    'Dairy': {'Checkout': 1},
    'Meat': {'Checkout': 5},
    'Checkout': {}
}
```

```
#starting at the entrance, find the shortest path to all other sections
shortest_paths_from_entrance = dijkstra(supermarket_graph, 'Entrance')
print("Shortest paths from Entrance:", shortest_paths_from_entrance)
```