Online Food Delivery System

BTech/II Year CSE/IV Semester

19CSE212/Data Structures and Algorithms

Case Study Report

Roll.No	Name
CB.EN.U4CSE21253	Samarth P
CB.EN.U4CSE21236	Ajay Kumar

Department of Computer Science and Engineering

Amrita School of Computing, Coimbatore

Amrita Vishwa Vidyapeetham

2022 -2023 Even Semester

Table of Contents

Chapter 1 Introduction :

In the fields of computer science and data management, effective data organisation and manipulation are essential for resolving challenging issues. Each of the traditional data structures, including arrays, linked lists, trees, and graphs, has advantages and disadvantages. Hybrid data structures, however, provide an effective solution in some situations where a single data structure may not be sufficient.

In order to take advantage of each data structure's unique advantages and meet specific problem needs, hybrid data structures integrate numerous data structures. Hybrid data structures can offer increased efficiency, optimised resource utilisation, and higher performance by combining the benefits of various data structures. By bridging the gaps between diverse data structures and maximising their combined advantages, they offer a flexible method of problem-solving.

1.1 Objective :

The objective of our project is to design and implement a hybrid data structure that combines the strengths of a graph and a queue. This hybrid structure will be utilized to create an efficient online food delivering system.

The key objectives of our project include:

Effective Route Planning: We intend to build algorithms for route planning by utilising the graph component of the hybrid data structure. These algorithms will allow us to determine the best delivery routes while taking into account variables like distance, and consumer locations.

Order Management: Incoming food orders will be managed using the hybrid structure's queue component. Orders will be given priority using algorithms we'll build, resulting in prompt delivery and effective order processing.

Practical Applications: Our project aims to demonstrate the practical applications of the hybrid data structure in an online food delivering system. We will demonstrate how the structure can increase the system's overall effectiveness and performance, resulting in higher client satisfaction and business expansion.

Analysis of Time and Space Complexity: We will examine the algorithms used in the system and the hybrid data structure's time and space complexity. This analysis will assist us in understanding the structure's effectiveness, locating any potential bottlenecks, and, if necessary, optimising the algorithms.

1.2 Significance of Hybrid Data Structures :

Hybrid data structures offer a powerful solution for solving complex problems efficiently. By combining the strengths of multiple data structures, they provide a versatile approach to problem-solving. In the context of an online food delivering system, hybrid data structures enable us to address challenges such as route planning and order management, optimizing resource utilization and improving overall system performance.

In conclusion, our project focuses on the design and implementation of a hybrid data structure for an online food delivering system. By leveraging the strengths of a graph and a queue, we aim to create an efficient and robust system that optimises route planning and order management. The practical applications of the hybrid data structure in the context of online food delivery highlight its significance in solving complex problems efficiently.

1.3 Github repository link :

https://github.com/Ajay0612/Online_Food_Order_Delivery

Chapter 2 Implementation:

The implementation of the hybrid data structure in our online food delivering system involves the integration and interplay of two constituent data structures: a graph and a queue. The queue controls the incoming food orders for effective order processing, while the graph is in charge of route planning and determining the best delivery routes.

2.1 Graph Implementation :

The graph component of the hybrid data structure is implemented using an adjacency list representation. Each vertex in the graph represents a location, and edges represent the distances between locations. We store the graph as a dictionary, where the keys are the vertices (locations) and the values are lists of neighbouring vertices along with their respective weights (distances).

The graph implementation provides methods for adding edges, retrieving neighbours of a vertex, and calculating the shortest path from a source vertex using Dijkstra's algorithm. The shortest path calculation is essential for determining the optimal delivery path from the food shop to the customer's location.

2.2 Queue Implementation :

The queue component of the hybrid data structure is implemented using a list. The queue maintains a FIFO (First-In-First-Out) order to process food orders in the order they are received. New orders are enqueued at the end of the list, and orders are dequeued from the front of the list.

The queue implementation provides methods for enqueueing and dequeuing orders, as well as checking if the queue is empty. This allows us to efficiently manage the incoming food orders and ensure timely delivery.

2.3 Integration and Interplay :

The integration of the graph and the queue within the hybrid data structure is crucial for the online food delivering system's overall functionality. When a new food order is placed, it is enqueued in the delivery queue. The order's

corresponding customer location is determined, and the graph's shortest path algorithm is invoked to calculate the optimal delivery path from the food shop to the customer's location.

The graph component utilizes the adjacency list representation to determine the neighbouring locations of each vertex and their respective distances. The shortest path algorithm calculates the distances from the source (food shop) to all other locations, allowing us to identify the customer's location and the optimal delivery path.

The interplay between the graph and the queue enables efficient order processing and delivery. The delivery queue ensures that orders are processed in the order they are received, while the graph facilitates route planning for optimal delivery paths.

2.4 Design Choices and Trade-offs :

During the implementation phase, several design choices and trade-offs were made to ensure the efficiency and functionality of the hybrid data structure:

Choice of Data Structures : The graph was implemented using an adjacency list representation, which offers efficient retrieval of neighbours and is suitable for sparse graphs. This choice was made to optimize memory usage and improve the efficiency of graph-related operations. Similarly, a list-based queue was chosen for the delivery queue due to its simplicity and efficient enqueue and dequeue operations.

Space Complexity vs. Time Complexity : The choice of using an adjacency list for the graph representation was driven by the trade-off between space complexity and time complexity. While an adjacency matrix provides constant-time access to edge weights, it consumes more memory. The adjacency list representation, on the other hand, requires traversing the list to retrieve edge weights but saves memory by only storing non-zero weights.

Order Processing vs. Route Planning: The interplay between the queue and the graph involves a trade-off between order processing efficiency and route planning accuracy. By prioritizing order processing, we ensure timely delivery and customer satisfaction. However, the graph-based route planning might introduce slight delays in determining the optimal delivery path, especially for larger graphs with complex routes.

Trade-off between Flexibility and Efficiency: One design choice is to strike a balance between the flexibility of the data structure and its efficiency. For example, while a hybrid data structure combining a graph and a queue offers efficient order processing and route planning, it may limit the flexibility to incorporate additional features or modifications in the future.

Chapter 3 Practical Applications

The hybrid data structure combining a graph and a queue offers a versatile solution for various practical applications. Some notable examples include:

Package Delivery Routing : The hybrid data structure can be applied to optimize package delivery routing systems. By representing the road network as a graph and using the queue to store delivery orders, the system can efficiently calculate the shortest paths between locations and dispatch delivery vehicles accordingly. This combination enables effective route planning, minimizing travel distances and improving overall delivery efficiency.

Emergency Service Dispatch : The hybrid data structure can be employed in emergency service systems, such as ambulance dispatch or fire department operations. By utilizing the graph to represent the geographical layout and the queue to store emergency calls, the system can efficiently locate and dispatch the nearest available emergency units. This combination enables swift and effective emergency response, potentially saving lives and minimizing damages.

Ride-Sharing and Taxi Dispatch: Ride-sharing platforms and taxi dispatch systems can leverage the hybrid data structure to optimize trip planning and driver allocation. By using the graph to model the road network and the queue to manage ride requests, the system can efficiently calculate optimal routes, match riders with available drivers, and minimize wait times.

In these practical applications, the combination of a graph and a queue in the hybrid data structure enables efficient operations. The graph allows for modeling and analyzing complex relationships between entities, while the queue facilitates order management and prioritization. The integration of these data structures empowers the systems to make informed decisions, optimize resource allocation, and provide effective solutions to complex logistical problems.

By utilizing the hybrid data structure, these applications can achieve improved efficiency, reduced costs, and enhanced customer satisfaction. The flexibility and adaptability of the hybrid data structure make it a valuable tool in solving real-world problems that involve complex data relationships and efficient order processing.

Chapter 4 Performance Analysis

4.1 Time Complexity:

‘order_food()’: The time complexity of this operation is **O(1)** as it involves appending the order to the list and enqueueing the order ID in the delivery queue, both of which are constant time operations.

‘view_order()’: The time complexity of this operation is **O(1)** as it involves accessing the order from the list using the provided order ID.

‘**change_order()**’: The time complexity of this operation is **O(1)** as it involves accessing the order from the list using the provided order ID.

‘**cancel_order()**’: The time complexity of this operation is **O(N)** in the worst case scenario, where N is the number of orders. This is because it involves searching for the order ID in the list and removing it, which may require shifting elements in the list.

‘**deliver_orders()**’: The time complexity of this operation depends on the number of orders and the complexity of calculating the shortest path for each order. Assuming the hybrid graph queue uses BFS to calculate the shortest path, the time complexity is approximately **O(N * (V + E))**, where N is the number of orders, V is the number of locations, and E is the number of edges in the graph.

4.2 Space Complexity:

‘orders’ : The space complexity of the order’s list is O(N), where N is the number of orders. This is because the list stores information about each order, including the customer name, address, and food order.

‘delivery_queue’ queue : The space complexity of the delivery_queue queue is also O(N), where N is the number of orders. In the worst case, all orders are enqueued in the delivery_queue. Each order ID occupies constant space, and the queue itself requires additional space to maintain its internal structure.

‘graph’ dictionary: The space complexity of the graph dictionary depends on the number of locations and the number of edges in the graph representation. Let V be the number of locations and E be the number of edges. The space complexity of the graph dictionary is O(V + E). Each location is represented as

a key in the dictionary, and its corresponding value is a list of neighbours with their respective weights.

Overall, the space complexity of the hybrid data structure is **O(N + V + E)**, where N is the number of orders, V is the number of locations, and E is the number of edges in the graph.

The space complexity includes the memory utilization for storing the order details, the delivery queue, and the graph representation. It does not include the space required for other variables and temporary storage during the execution of operations.

4.3 Performance Comparison :

The hybrid data structure used in the online food delivery system combines the functionality of a list, a queue, and a graph data structure. This combination allows for efficient storage and manipulation of orders, efficient delivery queue management, and shortest path calculations using Dijkstra's algorithm.

Comparing it with individual data structures:

- Using a separate list to store orders would have similar time complexity for most operations but may require additional memory overhead for managing the delivery queue.
- Using a separate queue for the delivery queue would have similar time complexity for enqueue and dequeue operations, but may have limitations in terms of efficiently calculating the shortest path for each order.
- Using a separate graph data structure would allow efficient shortest path calculations but may require additional data structures for managing orders and the delivery queue.

The hybrid data structure combines the advantages of these individual data structures, resulting in a more efficient implementation for the online food delivery system. It optimizes both time complexity and space complexity by leveraging the strengths of each constituent data structure.

Chapter 5 Discussion

5.1 Practicality and Effectiveness:

The implemented hybrid data structure in the online food delivery system proves to be highly practical and effective in real-world scenarios. By combining the graph and queue data structures, it offers efficient management of orders and enables optimized delivery path calculations. This hybrid approach allows for seamless food delivery operations, ensuring timely and accurate deliveries to customers.

The practicality of the hybrid data structure lies in its ability to handle large-scale datasets with varying numbers of orders and locations. It efficiently processes operations such as placing orders, viewing order details, modifying orders, canceling orders, and delivering orders. The integration of the graph data structure allows for the calculation of shortest paths, ensuring optimal delivery routes and minimizing travel distances.

5.2 Limitations, Challenges, Future Improvements:

While the implemented hybrid data structure demonstrates practicality and effectiveness, there are certain limitations and challenges to consider:

Scalability: Although the hybrid data structure performs well with large-scale datasets, there may be scalability challenges when dealing with extremely large volumes of orders and locations. As the dataset grows significantly, the time complexity of calculating shortest paths may increase, affecting overall performance. Mitigating this limitation could involve exploring more advanced graph algorithms or distributed computing techniques.

Dynamic Updates: The hybrid data structure may face challenges when handling frequent updates or changes in the underlying graph. For example, if the delivery network undergoes modifications or additions, updating the graph structure in real-time can be complex. Addressing this challenge would involve designing efficient algorithms for dynamic updates to maintain consistency and minimize disruptions.

Optimal Path Calculation: While the hybrid data structure provides efficient delivery path calculations, there is room for further optimization. Future improvements could involve incorporating advanced routing algorithms, considering real-time traffic information, or considering additional factors such as road conditions or delivery time windows. This would enhance the accuracy and efficiency of the delivery process.

Robustness and Fault Tolerance: Ensuring the hybrid data structure's robustness and fault tolerance is crucial. Future enhancements could involve incorporating fault-tolerant mechanisms to handle system failures, network disruptions, or data inconsistencies. This would enhance the reliability and resilience of the online food delivery system.

The implemented hybrid data structure proves to be practical and effective in real-world scenarios. It offers efficient order management, optimized delivery path calculations, and overall improved performance. While there are limitations and challenges, addressing scalability issues, dynamic updates, optimizing path calculations, and enhancing system robustness can pave the way for future improvements and ensure the continued success of the hybrid data structure in online food delivery and similar applications.

Chapter 6 Conclusion

The project aimed to design and implement a hybrid data structure for an online food delivery system, combining graph and queue data structures to efficiently manage orders and optimize delivery routes. Throughout the project, several key findings and outcomes emerged, demonstrating the practical applications, performance analysis, and efficiency of the hybrid data structure.

Firstly, the hybrid data structure proved to be highly practical in the context of online food delivery. By integrating the graph data structure, it facilitated the calculation of shortest paths, enabling optimal delivery routes and minimizing travel distances. The queue data structure efficiently managed the order queue, ensuring timely processing and delivery.

The performance analysis of the hybrid data structure revealed favourable time and space complexity for the supported operations. Key operations such as order placement, modification, and cancellation achieved constant time complexity, providing quick response times. The delivery process, involving path calculations, showed a time complexity of O(N * (V + E)), where N is the number of orders, V is the number of locations, and E is the number of edges in the graph.

In terms of space complexity, the hybrid data structure utilized memory resources effectively, with a space complexity of O(V + N + E). This optimized space utilization minimized overhead and memory consumption.

Insights gained from the implementation and evaluation of the hybrid data structure include the importance of carefully selecting and combining data structures based on the problem domain. The project highlighted the significance of leveraging graph algorithms, such as dijkstra's algorithm for shortest path calculations, to optimize delivery routes. It also emphasized the trade-offs and challenges in terms of scalability, dynamic updates, and path optimization, providing opportunities for future enhancements.

In conclusion, the project successfully implemented a hybrid data structure for an online food delivery system, showcasing its practicality, performance, and efficiency. The findings and insights obtained contribute to the broader understanding of hybrid data structures and their application in solving complex problems. The project serves as a foundation for further research and improvements in the domain of online food delivery systems and similar applications.

Chapter 7 References

Books:

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. , Introduction to Algorithms.

[2] Narsimha Karumanchi, Data Structures and Algorithms Made Easy

Websites:

[1] N.E.Goller, "Hybrid Data Structures Defined by Indirection" [Online] Available URL: https://academic.oup.com/comjnl/article/28/1/44/468048

[2] "Big O Analysis and Hybrid Data Structures" [Online]. Available: URL: https://docs.onenetwork.com/NeoHelp/devnet/big-o-analysis-and-hybrid-data-structures-79141067.html

[3] Dario Radecic, "Data Structures and Algorithms With Python" *IEEE Explore*. [Online] Available URL: https://towardsdatascience.com/data-structures-and-algorithms-with-python-learn-stacks-queues-and-deques-in-10-minutes-e7c6a2a1c5d5

[4] JavaTpoint, "Data Structures and Algorithms in Python" . [Online] Available URL: https://www.javatpoint.com/data-structures-and-algorithms-in-python-set-1