 

**Cairo University**

**Faculty of Engineering**

**Credit Hours System**

Phase 1:Restaurant Management

Data Structure Implementations



Team Number: 1

Team members:

**eSRAA HAMED**

**FATMAH ISSAM**

**MARIAM ESSAM**

**MENNA KAMAL**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**1)Data Structure of Orders**

* ***Normal and Vegan Waiting Orders***

Note that in general, pointers are used in DS not the orders themselves to avoid any errors resulting from the arrangement of orders.

Restaurant management needs to be extremely organized so there are several conditions the restaurant is required to meet.

Since the restaurant requires us to put each type of order with a certain sequence (VIP then Vegan then Normal), an ordinary queue of linked nodes was chosen for the normal and vegan orders. A queue allows us to implement a “first in first out”(FIFO) scenario. This concept of queue is optimized as it is needed to have the first vegan order to arrive be served first and finished first in a vegan order list. The same is true for normal waiting orders.

Another factor that caused us to choose queue for these types of orders over other data structures is the complexity of the operations of queue. Note that the restaurant has 3 stages the orders go through: Wait, Serve, and Finish. Because of these stages, the waiting orders would need to use the delete operation or dequeue in the case of queue. The worst case time complexity of an deletion in a queue is O(1), which is a nearly ideal solution to avoid the negative effects of high complexities that would later lead to lack of clarity.

* ***Waiting VIP Orders***

Usually, in any restaurant, a VIP(very important person) is supposed to treated in a very special manner depending on several factors the restaurant management system would determine. Because of that, the best type of data structure to be used in implementation of VIP waiting orders is a special kind of queue, which is a priority queue. In a priority queue of linked nodes, each element or order would have a priority. A priority queue would have the VIP with the highest priority first be served before a VIP with a lower priority. To determine the priority of each (waiting)VIP, a priority equation is formulated according to the order’s money, order’s size, and arrival time of the VIP.

* ***Inservice Orders***

As mentioned before, the orders go through 3 different stages, so for an order to go through in-service stage, it should be inserted easily. Although

the sequence of the orders doesn’t matter in “in-service” as much as in “waiting”, the complexity of queue insertions and deletions is very appealing, which is why we chose queue for in-service orders.

* ***Finished Orders***

Since finished orders are meant to be deleted at the end, a deletion operation with a O(1) complexity is required like dequeue. Thus, finished orders are put into a queue.

**2)Data Structures of Cooks**

Cooks are different from orders. They don’t need to be in a queue because that would limit what we can do with cooks from assignment of orders to finding and checking their availability. Hence, we chose linked list for the implementation of cooks. Linked list allows us to freely create and use various operations such as searching, sorting, traversing etc. Furthermore, linked lists are dynamic, flexible, and can expand and contract its size unlike arrays that should have fixed size. Memory utilization in linked list is also more efficient than in an array. If executed properly, linked list operations like search and remove could provide a complexity of O(1) in the best case and O(n) in the worst case.

**3)Data Structures of Events**

Events like arrival, cancellation, and promotion are to be inputted through user. When an input comes in, it needs to go to a place where it could be stored while its processing takes place. This place should be a queue. The **queue** will store a series of **notifications or requests(events)** in first-in, first-out order. Sending a notification **enqueues the request and returns**. The request processor then **processes the events from the queue** at a later time or given condition. Events need operations like enqueue, dequeue, and peek-front, which all have a complexity of O(1).