**4.1 Queue**

|  |  |  |  |
| --- | --- | --- | --- |
| Easy | | | |
| 1. Reverse First k Elements of Queue |  | 1. Level with maximum number of nodes |  |
| 1. Implement a Queue using an Array |  | 1. Breadth First Search or BFS for a Graph |  |
| 1. Print all elements of a queue in a new line |  | 1. Find Minimum Depth of a Binary Tree |  |
| Medium | | | |
| 1. Implement a Deque |  | 1. Detect cycle in an undirected graph using BFS |  |
| 1. Implement a Circular Queue |  | 1. Find next right node of a given key |  |
| 1. Check if a queue can be sorted into another queue using a stack |  | 1. Minimum steps to reach target by a Knight |  |
| 1. Implement Stack using Queues |  | 1. Islands in a graph using BFS |  |
| 1. Implement Stack using Two Queues |  | 1. Flood Fill Algorithm |  |
| 1. Implement Queue using Two Stacks |  | 1. Minimum steps to reach target by a Knight |  |
| 1. Design a Queue data structure to get minimum or maximum in O(1) time |  | 1. First negative integer in every window of size k |  |
| 1. Check whether a given graph is Bipartite or not |  | 1. Level order traversal in spiral form |  |
| 1. Print Right View of a Binary Tree |  | 1. Minimum time required to rot all oranges |  |
| 1. An Interesting Method to Generate Binary Numbers from 1 to n |  | 1. Queue based approach or first non-repeating character in a stream |  |
| 1. Implement a Queue using a Stack |  | 1. Shortest distance in a maze |  |
| 1. Reverse a queue using recursion |  | 1. Geek in a Maze |  |
| 1. Implement Priority Queue using Linked List |  | 1. Find shortest safe route in a path with landmines |  |
| 1. Implement Queue using Deque |  | 1. Find the first circular tour that visits all petrol pumps |  |
| 1. Flatten a multilevel linked list |  | 1. Connect Nodes at Same Level |  |
| Hard | | | |
| 1. Find the first non-repeating character from a stream of characters |  | 1. Maximum cost path from source node to destination |  |
| 1. Maximum of all subarrays of size k using a queue |  | 1. Trapping Rain Water |  |
| 1. Implement LRU Cache using Queue |  | 1. Maximum cost path from source node to destination |  |
| 1. Design a Queue data structure to get the maximum or minimum of sliding window |  | 1. Snake and Ladder Problem |  |
| 1. Find if there is a path between two vertices in a directed graph |  | 1. Minimum Cost Path in a directed graph via given set of intermediate nodes |  |
| 1. Design a Data Structure for LRU Cache |  | 1. Turn a Queue into a Priority Queue |  |
| 1. Trapping Rain Water |  | 1. Interchange elements of Stack and Queue without changing order |  |

|  |
| --- |
| Link: <https://www.geeksforgeeks.org/top-50-problems-on-queue-data-structure-asked-in-sde-interviews/> |

**Queue problem given by lab teacher.**

**Class Example :**

**Question 1 - Queue Implementation with Array:**

You are required to implement a queue data structure using an array in C. The queue should

have the following functionalities:

a) void enqueue(int item): This function should enqueue an integer item into the queue.

b) int dequeue(): This function should remove and return the front item from the queue.

c) int front(): This function should return the front item from the queue without removing it.

d) int isEmpty(): This function should return 1 if the queue is empty, and 0 otherwise.

Write the C code for the queue implementation with an array and demonstrate its usage by

enqueuing a few elements, dequeuing elements, and checking whether the queue is empty.

**Question 2 - Queue Implementation with Linked List:**

You are required to implement a queue data structure using a singly linked list in C. The queue

should have the following functionalities:

a) void enqueue(int item): This function should enqueue an integer item into the queue.

b) int dequeue(): This function should remove and return the front item from the queue.

c) int front(): This function should return the front item from the queue without removing it.

d) int isEmpty(): This function should return 1 if the queue is empty, and 0 otherwise.

Write the C code for the queue implementation with a linked list and demonstrate its usage by

enqueuing a few elements, dequeuing elements, and checking whether the queue is empty.

Please ensure that your implementations handle edge cases appropriately and provide the

correct output for different scenarios.

**Question 3 - Circular Queue Implementation with Array:**

You are required to implement a circular queue data structure using an array in C. The circular

queue should have the following functionalities:

a) void enqueue(int item): This function should enqueue an integer item into the circular queue.

b) int dequeue(): This function should remove and return the front item from the circular queue.

c) int front(): This function should return the front item from the circular queue without removing

it.

d) int isEmpty(): This function should return 1 if the circular queue is empty, and 0 otherwise.

e) int isFull(): This function should return 1 if the circular queue is full, and 0 otherwise.

Write the C code for the circular queue implementation with an array and demonstrate its usage

by enqueuing a few elements, dequeuing elements, checking whether the queue is empty, and

verifying if the queue is full.

Ensure that your implementation handles the circular nature of the queue correctly and provides

the correct output for various scenarios.

**Question 4 - Implement a Queue using Stacks:**

You are tasked with designing a queue data structure using two stacks. Each stack should have

push, pop, and isEmpty functions. The queue should support the following operations:

enqueue(int x): Add element x to the back of the queue.

dequeue(): Remove and retrieve the element from the front of the queue.

Dequeue constant : https://www.geeksforgeeks.org/queue-using-stacks/

Enqueue constant : Code uploaded

**Practice problem from different section**

1. Implement a stack using queue.
2. Reverse the elements of a queue.

**Input**: q[ ] = {10, 20, 30, 40, 50, 60}

**Output**: q[ ] = {60, 50, 40, 30, 20, 10}

**3.** Sort the elements of a queue without using any extra space.

**Input**: q[ ] = {20, 40, 10, 60, 50, 30}

**Output**: q[ ] = {10, 20, 30, 40, 50, 60}

**4.** Given a queue q[ ] and an integer K, remove the integer K from the queue. If

multiple same elements exist, remove the first one.

**Input**: q[ ] = {10, 20, 30, 40, 50, 60}, K = 30

**Output**: {10, 20, 40, 50, 60}

**Input**: q[] = {1, 2, 3, 3}, K = 3

**Output**: {1, 2, 3}

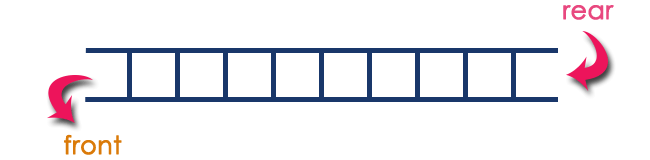
**4.1 Queue in Data Structure | Introduction to Queue**

|  |  |
| --- | --- |
| * It is a linear data structure. * It is a abstract data type * Queue can be implemented by **array, linklist, stack**. * This data algorithm follows **FIFO** (First In First Out) principle or **LILO** | **Rule:**   1. Insertion (will be performed from one end, and the end known as **rear** or **tail**)   insertion from **rear**   1. Deletion (will be performed from another end, and the end known as **head** or **front**)   deletion from **front**  Insertion = enqueue()  Deletion = dequeue() |
| **Logical representation of Queue:**  A whiteboard with writing on it  Description automatically generatedwhatever between front and rear is queue element and except that everything is garbage.  A whiteboard with black text  Description automatically generated  if front = rear (there is one element in the queue) | |
| **Empty queue condition:**   1. front = rear = -1 2. front > rear | **Queue Operation:**   1. enqueue() 2. dequeue() 3. front() / peek() 4. isFull() 5. isEmpty()   these operation time complexity is O(1) |
| **Application of queue:**   1. single shear resource (printer) 2. customer case call 3. processor (sharable resource) |  |

4.1 = done

**Queue Data Structure**

A **Queue Data Structure**is a fundamental concept in computer science used for storing and managing data in a specific order. It follows the principle of “**First in, First out**” **(FIFO)**, where the first element added to the queue is the first one to be removed. Queues are commonly used in various algorithms and applications for their simplicity and efficiency in managing data flow.

A screenshot of a computer

Description automatically generated

*Queue Data Structure*

**What is Queue in Data Structures?**

A queue is a linear data structure that follows the**First-In-First-Out (FIFO)** principle. It operates like a line where elements are added at one end (**rear**) and removed from the other end (**front**).

In a queue data structure, the insertion operation is performed using a function called "**enQueue()**" and deletion operation is performed using a function called "**deQueue()**".

Queue data structure can be defined as follows...

**Queue data structure is a linear data structure in which the operations are performed based on FIFO principle.**

A queue data structure can also be defined as

**"Queue data structure is a collection of similar data items in which insertion and deletion operations are performed based on FIFO principle".**

**Basic Operations of Queue Data Structure**

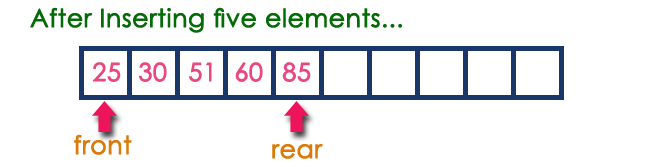
* **Enqueue (Insert)**: Adds an element to the rear of the queue. [ enQueue(value) ]
* **Dequeue (Delete)**: Removes and returns the element from the front of the queue. [ deQueue() ]
* **Peek**: Returns the element at the front of the queue without removing it.
* **isEmpty**: Checks if the queue is empty.
* **isFull**: Checks if the queue is full.
* **display()** - (To display the elements of the queue)

**Applications of Queue**

* Task scheduling in operating systems
* Data transfer in network communication
* Simulation of real-world systems (e.g., waiting lines)
* Priority queues for event processing queues for event processing

**Example**

Queue after inserting 25, 30, 51, 60 and 85.



Queue data structure can be implemented in two ways. They are as follows...

1. **Using Array**
2. **Using Linked List**
3. **Using Stack**

When a queue is implemented using an array, that queue can organize an only limited number of elements. When a queue is implemented using a linked list, that queue can organize an unlimited number of elements.

**4.2 Implementation of Queue using Arrays**

enqueue() and dequeue() operation must have the time complexity of O(1)

A screenshot of a computer

Description automatically generated

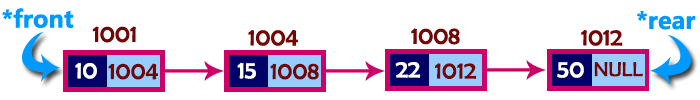
A screenshot of a computer

Description automatically generated

4.2 = DONE

**4.3 Queue Implementation using Linked List in C**

enqueue() and dequeue() operation must have the time complexity of O(1)



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

4.3 = DONE

**4.4 Circular Queue in Data Structure | Circular Queue using Arrays in C**