**LAR Liyanage**

**27336**

**Tutorial 04**

1. What is a circular queue?

* A circular queue, also known as a circular buffer or ring buffer, is a data structure that behaves like a queue or a buffer, but with the added feature that the data elements are stored in a circular fashion. This means that when a new element is added to the queue, it is placed at the end of the queue, but if the end of the queue is reached, the new element is added to the beginning of the queue, effectively overwriting the oldest element in the queue. Similarly, when an element is removed from the queue, the oldest element in the queue is removed, but if the beginning of the queue is reached, the removal continues from the end of the queue.

1. What are the characteristics of the circular queue?

* Circular Arrangement :- The circular queue is arranged in a circular manner, where the last element is connected to the first element . This circular arrangement allows for efficient enqueueing and dequeuing operations.
* Fixed Size :- Circular queues have a fixed size, which is determined during their initialization. Once the queue reaches its maximum capacity, it is considered full and further enqueue operations are not allowed until there is space available through dequeuing.
* Front and Rear Pointers :- A circular queue maintains two pointers - the front pointer points to the front of the queue, and the rear pointer points to the rear of the queue. These pointers keep track of the positions for enqueueing and dequeuing operations.
* Enqueue Operation :- In a circular queue, new elements are added at the rear end of the queue. When an element is enqueued, it is placed in the position pointed to by the rear pointer, and the rear pointer is incremented to the next position, wrapping around to the beginning if necessary.
* Dequeue Operation :- Elements are removed from the front of the circular queue. When an element is dequeued, it is removed from the position pointed to by the front pointer, and the front pointer is incremented to the next position, wrapping around to the beginning if necessary.
* Efficient Utilization of Memory :- Circular queues efficiently utilize memory by reusing space when elements are dequeued. Unlike a traditional linear queue, the circular queue does not require shifting elements to fill empty spaces, resulting in constant-time complexity for enqueue and dequeue operations.
* Time Complexity :- The enqueue and dequeue operations in a circular queue have a time complexity of O(1) (constant time), as the positions for insertion and removal can be calculated based on the current front and rear pointers.
* Continuous Storage :- A circular queue is often implemented using an array or a linked list with the rear and front pointers appropriately maintained. In an array-based implementation, the elements are stored in a contiguous manner.

1. Give applications of the circular queue.

* Buffer management :- Circular queues are often used to implement buffers in operating systems and embedded systems. In these systems, data is continuously produced and consumed, and a circular queue provides an efficient way to manage the buffer space. The circular nature of the queue allows for seamless wrapping around and continuous utilization of the available buffer.
* Cache management :- Circular queues can be used to implement cache algorithms like the Least Recently Used (LRU) or First In First Out (FIFO) replacement policies. The circular nature of the queue allows for efficient management of cache entries, ensuring that the most recently accessed or least recently used items are appropriately placed in the cache.
* Job scheduling :- Circular queues are used in job scheduling algorithms to manage a queue of processes or tasks. When a process finishes execution, it can be moved to the end of the queue, ensuring fair scheduling and resource allocation. The circular nature of the queue allows for efficient management of the job queue without the need for resizing or shifting elements.
* Printer spooling :- Circular queues are used in spooling systems, where multiple print jobs are queued for processing. The circular nature of the queue ensures that the printer can continuously process incoming print jobs without requiring costly resizing operations.
* Simulation and event-driven systems :- Circular queues are used in simulation systems or event-driven systems to manage events or processes that occur over time. The circular nature of the queue allows for efficient handling of time-based events and ensures that the queue can wrap around and handle future events seamlessly.

1. What is the algorithm of the circular queue?

* Initialization :- Create an empty circular queue by initializing the front and rear pointers to -1.
* Enqueue (Insertion) :- To add an element to the circular queue:
* Dequeue (Deletion) :- To remove an element from the circular queue:
* Is Full :- This operation checks if the circular queue is full or not.
* is Empty :- This operation checks if the circular queue is empty or not.
* Get Front :- This operation returns the front element of the circular queue without removing it.
* Get Size :- This operation returns the current size of the circular queue.

1. Write a simple program of circular.

import math

def calculate\_circumference(radius):

circumference = 2 \* math.pi \* radius

return circumference

def calculate\_area(radius):

area = math.pi \* radius \*\* 2

return area

radius = float(input("Enter the radius of the circle: "))

circumference = calculate\_circumference(radius)

print(f"The circumference of the circle is: {circumference}")

area = calculate\_area(radius)

print(f"The area of the circle is: {area}")

1. Compare and contrast linear queues and circular queues.

* Structure.
* Linear Queue :- In a linear queue, elements are stored in a linear manner, with two pointers indicating the front and rear of the queue.
* Circular Queue :- In a circular queue, elements are stored in a circular manner, with two pointers indicating the front and rear of the queue. The rear pointer wraps around to the beginning of the queue when it reaches the end.
* Memory Utilization.
* Linear Queue :- In a linear queue, the rear pointer keeps moving towards the end of the queue. If elements are dequeued from the front, the space at the front of the queue becomes unusable, leading to inefficient memory utilization.
* Circular Queue :- In a circular queue, the rear pointer wraps around to the beginning of the queue, allowing for efficient memory utilization. The space at the front of the queue can be reused when elements are dequeued, maximizing memory usage.
* Implementation Complexity.
* Linear Queue :- Implementing a linear queue is relatively simpler compared to a circular queue since it does not involve handling the wraparound of pointers.
* Circular Queue :- Implementing a circular queue requires additional logic to handle the wraparound of pointers. The circular movement of the rear pointer must be carefully managed to ensure correct enqueue and dequeue operations.
* Performance.
* Linear Queue :- In terms of performance, linear queues are efficient for enqueue and dequeue operations as long as the queue does not become full. However, if the queue becomes full, further enqueues may result in overflow errors.
* Circular Queue :- Circular queues offer better performance compared to linear queues when handling large amounts of data. They efficiently utilize memory and can continue to enqueue elements even when the rear pointer wraps around to the beginning. This avoids overflow errors and optimizes the use of available space.
* Applications.
* Linear Queue :- Linear queues are commonly used in scenarios where the order of data processing is important, such as printing tasks, CPU scheduling, and breadth-first search algorithms.
* Circular Queue :- Circular queues are suitable for scenarios where data needs to be processed in a circular manner, such as buffering data in devices with limited memory, implementing a queue in a circular buffer, or managing tasks in a round-robin scheduling algorithm.