TECHNO INTERNETIONAL NEWTOWN

NAME : ROHAN SAMANTA
ROLL:18731722017
STREAM:CSE(CYBER SECURITY)
TOPIC:QUEUE
SEMESTER:3RD
YEAR:2022-26 (2ND)

TOPIC:QUEUE

Introduction:

This technical report provides an in-depth overview of the queue data structure and its associated algorithms implemented in the C programming language. A queue is a normal data structure that follows the First-In-First-Out (FIFO) principle, making it particularly useful for scenarios where elements need to be processed in the order they were added. This report covers the basic concepts of queues, their implementation using arrays and linked lists, as well as essential queue algorithms with their respective complexities.

A queue is a linear data structure that supports two primary operations: enqueue (insertion) and dequeue (removal). The enqueue operation sum of an component to the rear of the queue, while the dequeue operation removes an component from the front of the queue. This behavior ensures that the oldest component are processed first, making queues suitable for tasks like managing printer job queues, breadth-first graph traversal, and more

What is queue?

A queue is a linear data structure in computer science that follows the First In First Out (FIFO) principle. It is used to store and retrieve elements in a specific order, where the first element added to the queue will be the first one to be removed.

Importance

Queues are widely used in computer science for various applications, such as process scheduling, resource allocation, and network packet management. They also play a crucial role in implementing algorithms like breadth-first search and depth-first search, which are used in graph theory and artificial intelligence.

Types of Queues

Circular Queue:

A circular queue is a data structure that uses a circular buffer to store and retrieve elements. It follows the First-In-First-Out (FIFO) principle, where the first element added to the queue is the first one to be removed. When the buffer is full, new elements overwrite the oldest ones.

Priority Queue:

A priority queue is a data structure that stores elements with associated priorities and retrieves them in order of priority. The highest priority element is always at the front of the queue and is the first one to be removed. Priority queues can be implemented using various algorithms, such as binary heaps, Fibonacci heaps, or balanced search trees.

Double-Ended Queue:

A double-ended queue, also known as a deque, is a data structure that allows elements to be added or removed from both ends. It can be used as a stack or a queue, depending on the order of operations. Deques can be implemented using arrays, linked lists, or other data structures.

Implementation:

Array Implementation:

To implement a queue using an array, we need to keep track of the front and rear of the queue. The front points to the first element in the queue, and the rear points to the last element in the queue. We also need to keep track of the size of the queue, which is the number of elements currently in the queue. Here is an example of a queue implementation using an array in C:

```
#define MAX_SIZE 100 typedef struct { int art [MAX_SIZE]; int front; int rear; int size; } Queue; void enqueue(Queue *q, int Val) { if (q->size == MAX_SIZE) { print ("Queue is full.\ n"); return; } q->art[q->rear] = Val; q->rear = (q->rear + 1) % MAX_SIZE; q->size++; } int dequeue(Queue *q) { if (q->size == 0) { print ("Queue is empty.\n"); return -1; } int Val = q->art[q->front]; q->front = (q->front + 1) % MAX_SIZE; q->size--; return Val; }
```

Linked List-based Queue:

The linked list-based queue is another implementation of a queue using a linked list. The enqueue operation adds an element to the end of the linked list, and the dequeue operation removes the first element of the linked list. The time complexity of enqueue and dequeue operations is O(1).

```
#include < studio. h >
struct node {int data; struct node * next;
};struct node * front;
struct node * rear;
void insert(struct node * per, int item) { print = (struct node * )
malloc(size of(struct node));
    if (print == NULL) {print("\ overflow\ n");
   return; } else {per - > data = item; if (front == NULL) {
  front = per; rear = per; front - > next = NULL;
  rear - > next = NULL;} else {rear - > next = per;
 rear = prewar - > next = NULL;
 }}}int main() {struct node * head = NULL; insert(head, 10);
    insert(head, 20);print ("front element: %d", front - > data);
    return 0;
}
```

Queue Fundamentals:

Explain the fundamental concept of a queue:

- **Definition**: Define what a queue is. A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle, where the element then has been in the queue the longest is the first to be removed.
- **Operations**: Describe the primary operations associated with queues:
  - **enqueue**: Adding an element to the rear (end) of the queue.
  - **dequeue**: Removing the element from the front (beginning) of the queue.
  - **front**: Retrieving the component at the front without removing it.
  - **rear**: Retrieving the element at the rear without removing it.
  - **Is Empty**: Checking if the queue is empty.
  - **size**: Determining the number of component in the queue.

Queue Applications:

Highlight the practical applications of queues in different domains:

- **Task Scheduling**: Explain how queues are used in scheduling tasks or processes in operating systems.
- **Breadth-First Search**: Describe how queues play a crucial role in the breadth-first search algorithm for traversing graphs.
- **Print Spooling**: Discuss how queues are employed in print spooling systems to manage print jobs.
- **Browsing History**: Explain how a queue can be used to implement a browsing history feature in web browsers

Time Complexity Analysis:

Analyze the time complexities of various queue operations for different implementations:

- Provide a table or graph comparing the time complexities of enqueue, dequeue, and other operations for array-based and linked list-based queues.

Formatting and Style:

•Use clear headings and subheadings to organize your report.
•Use diagrams, figures, and code snippets where necessary to enhance understanding.
•Maintain a consistent writing style and ensure proper grammar and punctuation.
•Proofread the report for errors before finalizing it.
Remember to tailor your report to your intended audience. If your audience has a strong technical background, you can delve deeper into algorithmic complexities and implementation details. If your audience is more general, focus on explaining concepts in a clear and approachable manner.

Conclusion:

Reinforce the importance of queues in data structures and algorithms . Highlight the versatility of queues in solving various problems. Mention any challenges or limitations associated with queues. In conclusion, the queue data structure is a fundamental concept in computer science algorithms. Its applications are numerous, and it is an essential tool for solving many real-world problems. Queues are useful data structures for implementing algorithms that require a FIFO ordering, such as breadth-first search and scheduling tasks. While other data structures like stacks, linked lists, and arrays have similar functionalities, they have different trade-offs in terms of memory usage, access time, and insertion/removal time.

REFERENCES:

LIST THE SOURCES YOU USED TO GATHER INFORMATION FOR YOUR REPORT, INCLUDING TEXTBOOKS, ACADEMIC PAPERS, ONLINE PORTAL, AND ANY OTHER RELEVANT MATERIALS.