The Queue Data Structure

In this lecture, the following topics are covered:

- ❑ The concept of Queue
- ❑ Queue Representation using Arrays
- ❑ Queue Representation using Linked Lists

1. The Concept of Queue

- They're also used to model real-world situations such as people waiting in line at a bank, airplanes waiting to take off, or data packets waiting to be transmitted over the Internet.
- There are various queues quietly doing their job in your computer's (or the network's) operating system:
    - There's a printer queue where print jobs wait for the printer to be available.
    - A queue also stores keystroke data as you type at the keyboard. This way, if you're using a word processor but the computer is briefly doing something else when you hit a key, the keystroke won't be lost; it waits in the queue until the word processor has time to read it. Using a queue guarantees the keystrokes stay in order until they can be processed.
- A queue is a FIFO (First-In, First-Out) data structure in which the element that is inserted first is the first one to be taken out.
- The elements in a queue are added at one end called the REAR and removed from the other end called the FRONT.
- Queues can be implemented by using either **arrays** or **linked lists**.
- In this section, we will see how queues are implemented using each of these data structures.

2. Array Representation of Queues

- Queues can be easily represented using linear arrays.
- As stated earlier, every queue has front and rear variables that point to the position from where deletions and insertions can be done, respectively.

- The array representation of a queue is shown in Fig. 8.1.

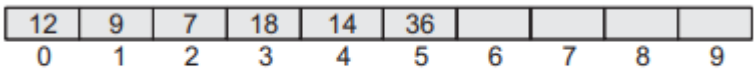

Figure 8.1 Queue

2.1 Operations on Queues

- In Fig. 8.1, FRONT = 0 and REAR = 5.
- Suppose we want to add another element with value 45, then REAR would be incremented by 1 and the value would be stored at the position pointed by REAR.
- The queue after addition would be as shown in Fig. 8.2. Here, FRONT = 0 and REAR = 6.
- Every time a new element has to be added, we repeat the same procedure.
- If we want to delete an element from the queue, then the value of FRONT will be incremented.
- Deletions are done from only this end of the queue.
- The queue after deletion will be as shown in Fig. 8.3. Here, FRONT = 1 and REAR = 6.

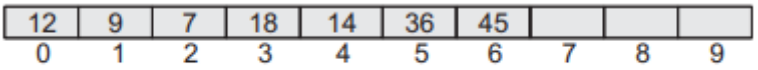

Figure 8.2 Queue after insertion of a new element

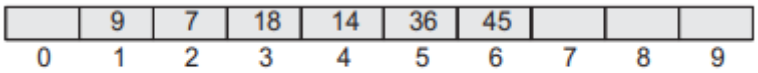

Figure 8.3 Queue after deletion of an element

- However, before inserting an element in a queue, we must check for overflow conditions.
- An overflow will occur when we try to insert an element into a queue that is already full.
- When REAR = MAX – 1, where MAX is the size of the queue, we have an overflow condition.
- Note that we have written MAX – 1 because the index starts from 0.
- Similarly, before deleting an element from a queue, we must check for underflow conditions. An underflow condition occurs when we try to delete an element from a queue that is already empty.
- If FRONT = –1 and REAR = –1, it means there is no element in the queue.

- Let us now look at Figs 8.4 and 8.5 which show the algorithms to insert and delete an element from a queue.
- Figure 8.4 shows the algorithm to insert an element in a queue.
    - In Step 1, we first check for the overflow condition.
    - In Step 2, we check if the queue is empty. In case the queue is empty, then both FRONT and REAR are set to zero, so that the new value can be stored at the 0th location. Otherwise, if the queue already has some values, then REAR is incremented so that it points to the next location in the array.
    - In Step 3, the value is stored in the queue at the location pointed by REAR.
- Figure 8.5 shows the algorithm to delete an element from a queue.
    - In Step 1, we check for underflow condition.
    - An underflow occurs if FRONT = –1 or FRONT > REAR. However, if queue has some values, then FRONT is incremented so that it now points to the next value in the queue.

```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

Figure 8.4 Algorithm to insert an element in a queue

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

Figure 8.5 Algorithm to delete an element from a queue

3. A Linked List Representation of Queues

- We have seen how a queue is created using an array.
- Although this technique of creating a queue is easy, its drawback is that the array must be declared to have some fixed size.
- If we allocate space for 50 elements in the queue and it hardly uses 20–25 locations, then half of the space will be wasted.
- And in case we allocate less memory locations for a queue that might end up growing large and large, then a lot of re-allocations will have to be done, thereby creating a lot of overhead and consuming a lot of time.

- In case the queue is a very small one or its maximum size is known in advance, then the array implementation of the queue gives an efficient implementation.
- But if the array size cannot be determined in advance, the other alternative, i.e., the linked representation is used.
- In a linked queue, every element has two parts, one that stores the data and another that stores the address of the next element.
- The START pointer of the linked list is used as FRONT.
- Here, we will also use another pointer called REAR, which will store the address of the last element in the queue.
- All insertions will be done at the rear end and all the deletions will be done at the front end. If FRONT = REAR = NULL, then it indicates that the queue is empty. The linked representation of a queue is shown in Fig. 8.6.

Figure 8.6 Linked queue

3.1 Operations on Linked Queues

- A queue has two basic operations: insert and delete.
- The insert operation adds an element to the end of the queue, and the delete operation removes an element from the front or the start of the queue.
- Apart from this, there is another operation peek which returns the value of the first element of the queue.

Insert Operation

- The insert operation is used to insert an element into a queue.
- The new element is added as the last element of the queue.
- Consider the linked queue shown in Fig. 8.7.

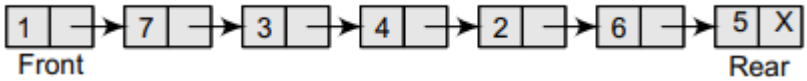

Figure 8.7 Linked queue

  - To insert an element with value 9, we first check if FRONT=NULL.
  - If the condition holds, then the queue is empty. So, we allocate memory for a new node, store the value in its data part and NULL in its next part.
  - The new node will then be called both FRONT and rear.
  - However, if FRONT != NULL, then we will insert the new node at the rear end of the linked queue and name this new node as rear.
  - Thus, the updated queue becomes as shown in Fig. 8.8.

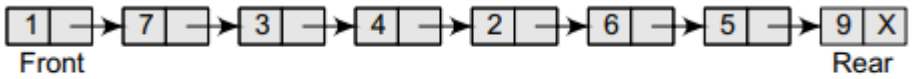

Figure 8.8 Linked queue after inserting a new node

- Figure 8.9 shows the algorithm to insert an element in a linked queue.
- In Step 1, the memory is allocated for the new node.
- In Step 2, the DATA part of the new node is initialized with the value to be stored in the node.
- In Step 3, we check if the new node is the first node of the linked queue. This is done by checking if FRONT = NULL. If this is the case, then the new node is tagged as FRONT as well as REAR.
- Also NULL is stored in the NEXT part of the node (which is also the FRONT and the REAR node).
- However, if the new node is not the first node in the list, then it is added at the REAR end of the linked queue (or the last node of the queue).

```
Step 1: Allocate memory for the new node and name
        it as PTR
Step 2: SET PTR -> DATA = VAL
Step 3: IF FRONT = NULL
            SET FRONT = REAR = PTR
            SET FRONT -> NEXT = REAR -> NEXT = NULL
        ELSE
            SET REAR -> NEXT = PTR
            SET REAR = PTR
            SET REAR -> NEXT = NULL
        [END OF IF]
Step 4: END
```

Figure 8.9 Algorithm to insert an element in a linked queue

Delete Operation

- The delete operation is used to delete the element that is first inserted in a queue, i.e., the element whose address is stored in FRONT.
- However, before deleting the value, we must first check if FRONT=NULL because if this is the case, then the queue is empty and no more deletions can be done.
- If an attempt is made to delete a value from a queue that is already empty, an underflow message is printed.
- Consider the queue shown in Fig. 8.10.
    - To delete an element, we first check if FRONT=NULL.
    - If the condition is false, then we delete the first node pointed by FRONT.

- The FRONT will now point to the second element of the linked queue. Thus, the updated queue becomes as shown in Fig. 8.11.

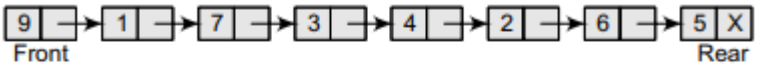

Figure 8.10 Linked queue

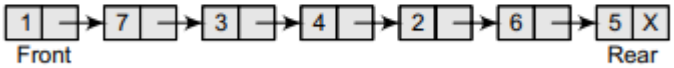

Figure 8.11 Linked queue after deletion of an element

- Figure 8.12 shows the algorithm to delete an element from a linked queue.
- In Step 1, we first check for the underflow condition.
- If the condition is true, then an appropriate message is displayed, otherwise in Step 2, we use a pointer PTR that points to FRONT.
- In Step 3, FRONT is made to point to the next node in sequence.
- In Step 4, the memory occupied by PTR is given back to the free pool.

```
Step 1: IF FRONT = NULL
            Write "Underflow"
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = FRONT
Step 3: SET FRONT = FRONT -> NEXT
Step 4: FREE PTR
Step 5: END
```

Figure 8.12 Algorithm to delete an element from a linked queue