CM1602 : Data Structures and Algorithms for AI

5. Queue

Lecture 5 | R. Sivaraman

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

MODULE CONTENT

Lecture	Topic
Lecture 01	Introduction to Fundamentals of Algorithms
Lecture 02	Analysis of Algorithms
Lecture 03	Array and Linked Lists
Lecture 04	Stack
Lecture 05	Queue
Lecture 06	Searching algorithms and Sorting algorithms
Lecture 07	Trees
Lecture 08	Maps, Sets, and Lists
Lecture 09	Graph algorithms

Learning Outcomes

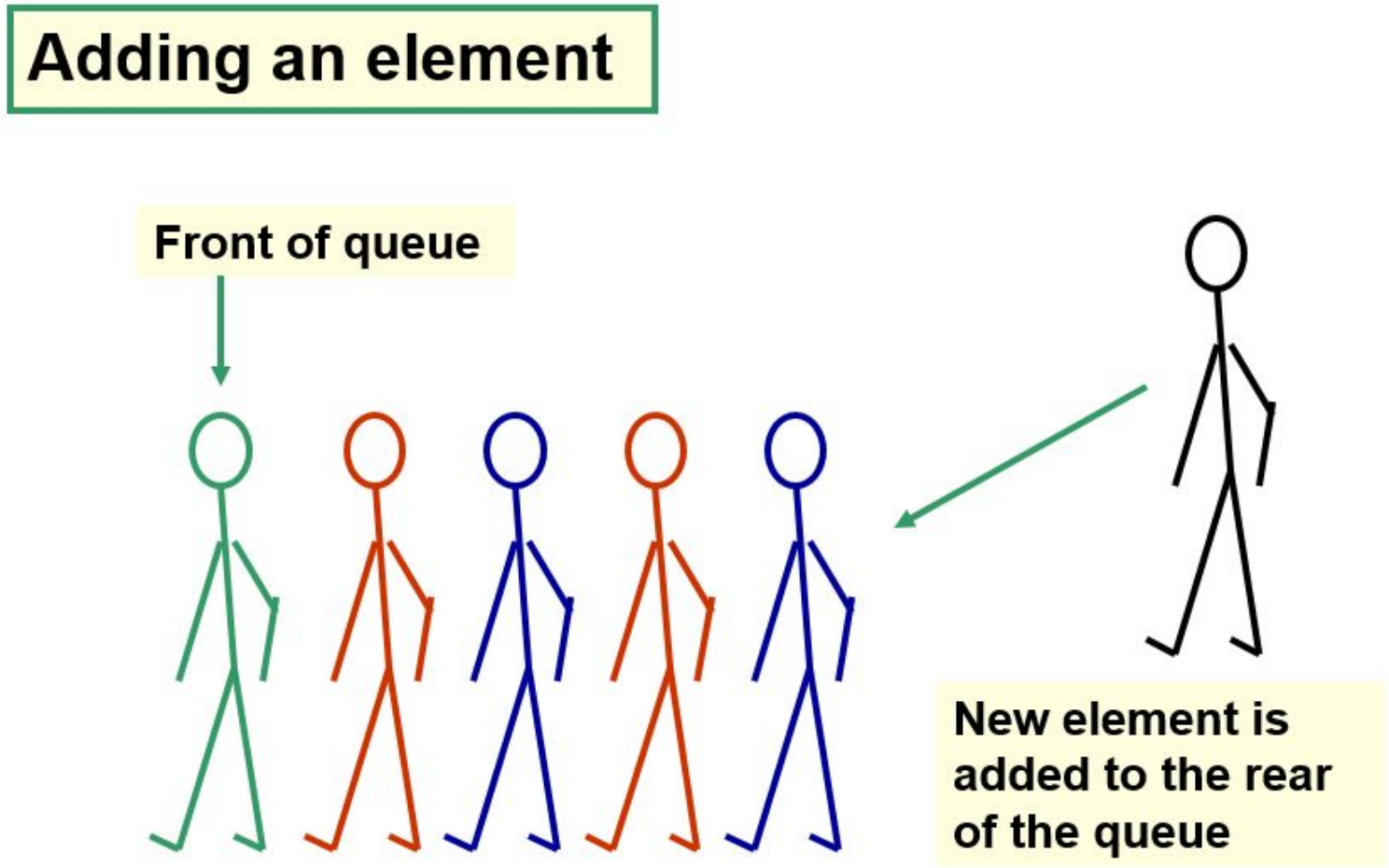
- LO1 : Describe the fundamental concepts of algorithms and data structures.
- LO3 : Apply appropriate data structures given a real-world problem to meet requirements of programming language APIs.
- On completion of this lecture, students are expected to be able to:
  - Describe Queues, types of queues and when queue is used
  - Implement Queue for a real life scenario

Introduction

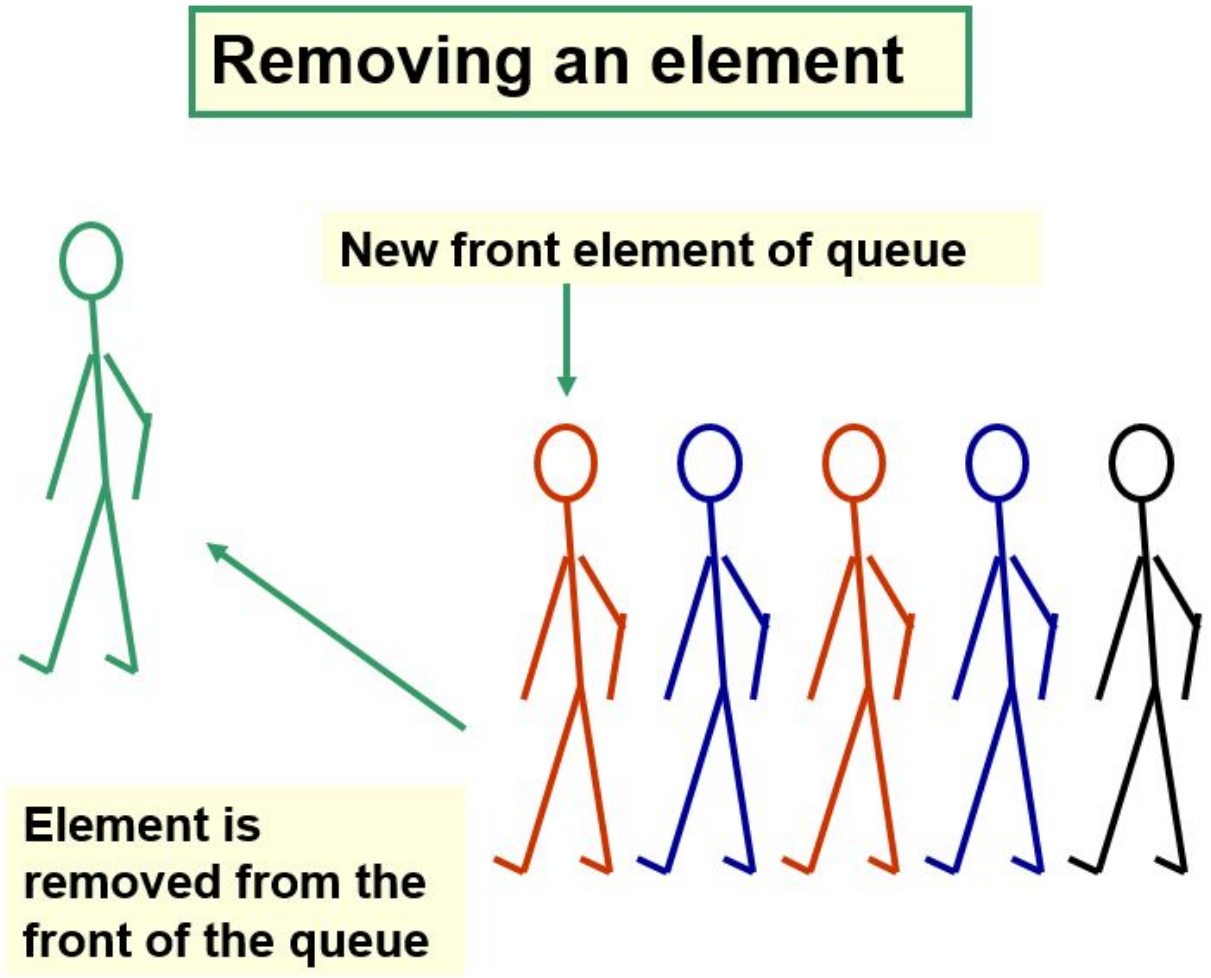
Introduction

- Queue: a collection whose elements are added to the rear of the queue and removed from the front of the queue
- A queue is a FIFO (first in, first out) data structure
- Any waiting line is a queue:
  - The check-out line at a grocery store
  - The cars at a stop light
  - An assembly line

Introduction



Introduction



Uses of Queues in Computing

- For any kind of problem involving FIFO data
- Printer queue (e.g. printer in MC 235)
- Keyboard input buffer
- GUI event queue (click on buttons, menu items)
- To encode messages (more on this later)

Queue Class

- Definitions:
  - Front
  - Rear
  - noOfItems
  - maxSize
  - queueArray

Queue Class

- Methods:
  - Enqueue
  - Dequeue
  - isFull
  - isEmpty

Queue Class

```
1  class Queue {
2      private int front;
3      private int rear;
4      private int noOfItems;
5      private int maxSize;
6      private int queueArray[];
7
8      public Queue(int size)
9      {
10         maxSize = size;
11         front = 0
12         rear = -1;
13         noOfItems = 0;
14         queueArray = new int[maxSize];
15     }
```

IsFull and IsEmpty methods

```
18      public boolean isFull()
19      {
20          return (rear == maxSize-1);
21      }
22
23
24      public boolean isEmpty()
25      {
26          return (noOfItems == 0);
27      }
28
```

Enqueue

- Method to add an element to the queue
- The element is added to the rear of the queue
- Steps:
  - Check if the queue is not full
  - Increment the rear
  - Add the element to the rear
  - Increment the noOfItems

Enqueue

```
30      public void enqueue(int item)
31      {
32          if (isFull())
33          {
34              System.out.println("The Queue is full");
35          }
36          else
37          {
38              queueArray[++rear] = item;
39              noOfItems++;
40          }
41      }
```

Dequeue

- Method to delete an element to the queue
- The element is deleted from the front of the queue
- Steps:
  - Check if the queue is not empty
  - Decrease the noOfItems
  - Return the value on the front
  - Increment the front

Dequeue

```
44      public int dequeue()
45      {
46          if (isEmpty())
47          {
48              System.out.println("The Queue is Empty");
49          }
50          else
51          {
52              noOfItems--;
53              return queueArray[front++];
54          }
55      }
```

Types of Queue

Simple Queue

- It defines the simple operation of queue in which insertion occurs at the rear of the list and deletion occurs at the front of the list

Types of Queue

Circular Queue

- Insertion and deletion is similar to simple queue but the Last node is connected back to the first node.
- It is an abstract data type and It is also known as Ring buffer

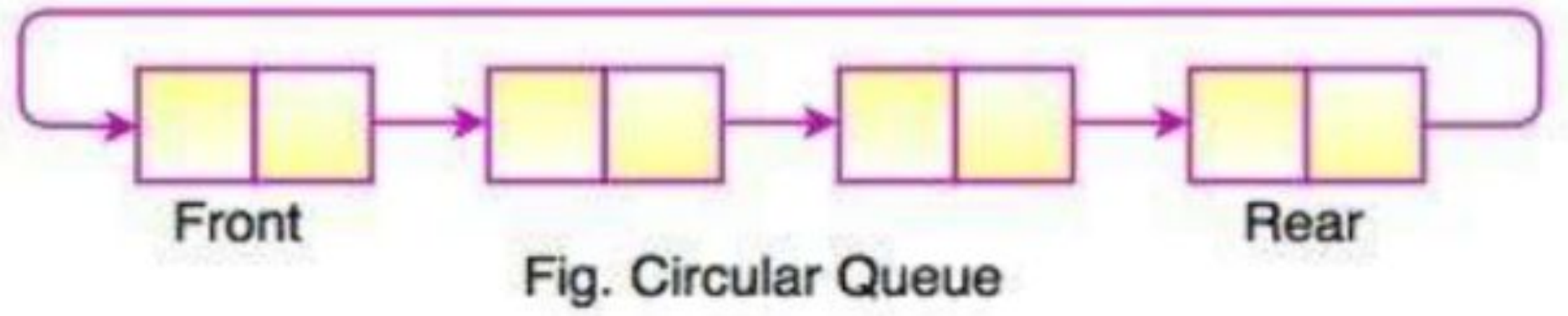


Fig. Circular Queue

Types of Queue

Circular Queue

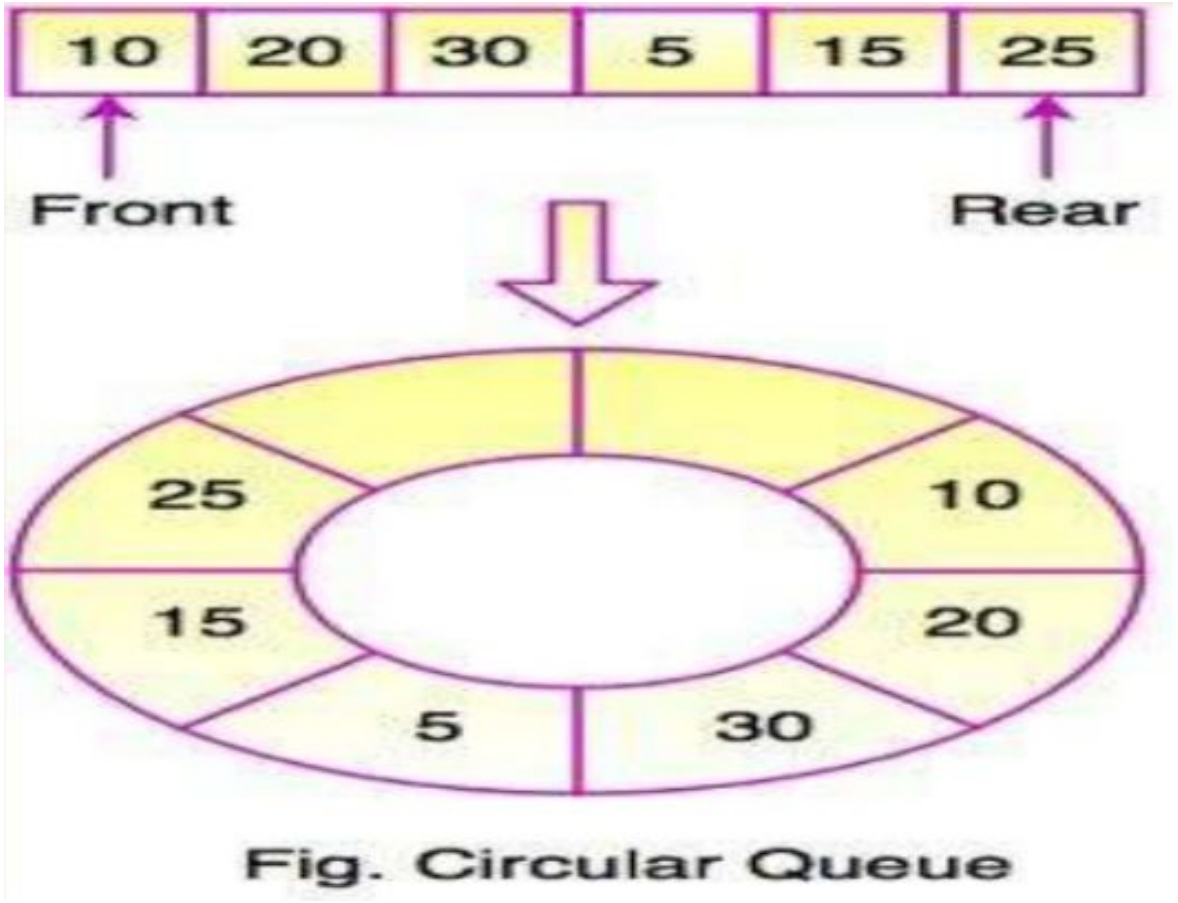


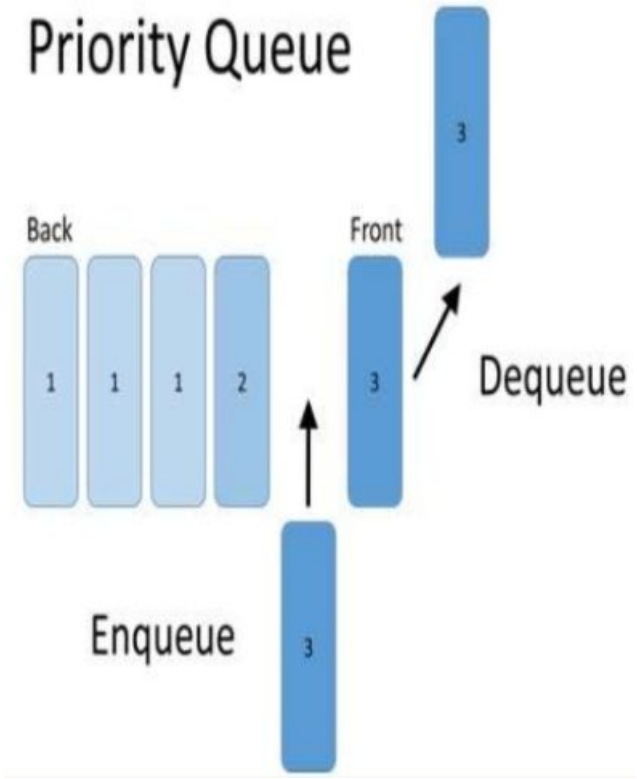
Fig. Circular Queue

Types of Queue

Priority Queue

- Priority Queue contains data items which have some preset priority
- While removing, data item with highest priority is removed first
- Insertion is performed in the order of arrival.

Types of Queue



Queue Implementation using Linked List

```
1  class QNode {
2      int key;
3      QNode next;
4
5      // constructor to create a new linked list node
6      public QNode(int key)
7      {
8          this.key = key;
9          this.next = null;
10     }
11 }
```

Queue Implementation using Linked List

```
14  class Queue {
15      QNode front, rear;
16
17      public Queue()
18      {
19          this.front = this.rear = null;
20      }
21
```

Queue Implementation using Linked List

```
22      // Method to add an key to the queue.
23      void enqueue(int key)
24      {
25
26          // Create a new LL node
27          QNode temp = new QNode(key);
28
29          // If queue is empty, then new node is front and rear both
30          if (this.rear == null) {
31              this.front = this.rear = temp;
32              return;
33          }
34
35          // Add the new node at the end of queue and change rear
36          this.rear.next = temp;
37          this.rear = temp;
38      }
```

Queue Implementation using Linked List

```
40      // Method to remove an key from queue.
41      void dequeue()
42      {
43          // If queue is empty, return NULL.
44          if (this.front == null)
45              return;
46
47          // Store previous front and move front one node ahead
48          QNode temp = this.front;
49          this.front = this.front.next;
50
51          // If front becomes NULL, then change rear also as NULL
52          if (this.front == null)
53              this.rear = null;
54      }
```