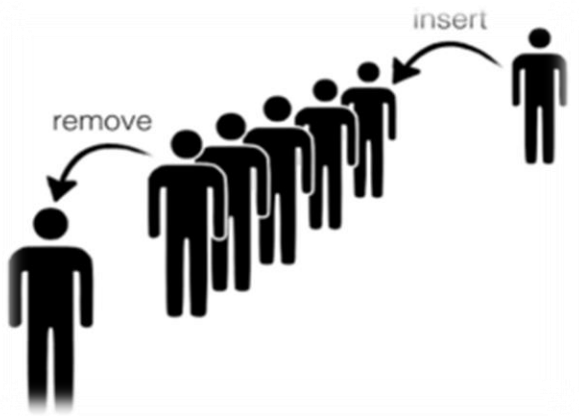
Database and Data Structures II



Queue Data Structure

Dilini Thamarasee

Queues

- FIFO (First In First Out)
- Items are deleted at one end called „front“
- Items are inserted at other end called „rear“
- Elements are sorted by insertion order

Operations in a queue

- Enqueue
  - Put the item at the end of the queue
- Dequeue
  - Remove the first item from the front end of the queue.
- Peek/firstEl
  - Find the value of the first item without removing it

Convention

- The rear pointer advances by 1 before an insertion, while the front pointer stays put.
- The front pointer increases by 1 after a deletion, while the rear pointer stays put.
- When the number of elements in the queue is zero, then queue is empty.
- Re-initialize the queue such that front = 0 and rear = -1
- When rear = maxsize-1,
- If front = 0, the queue is full and an insertion results in an overflow
- If front is not equal to 0, then there are unused nodes.

Queue implementation through Array Java Code for Queue

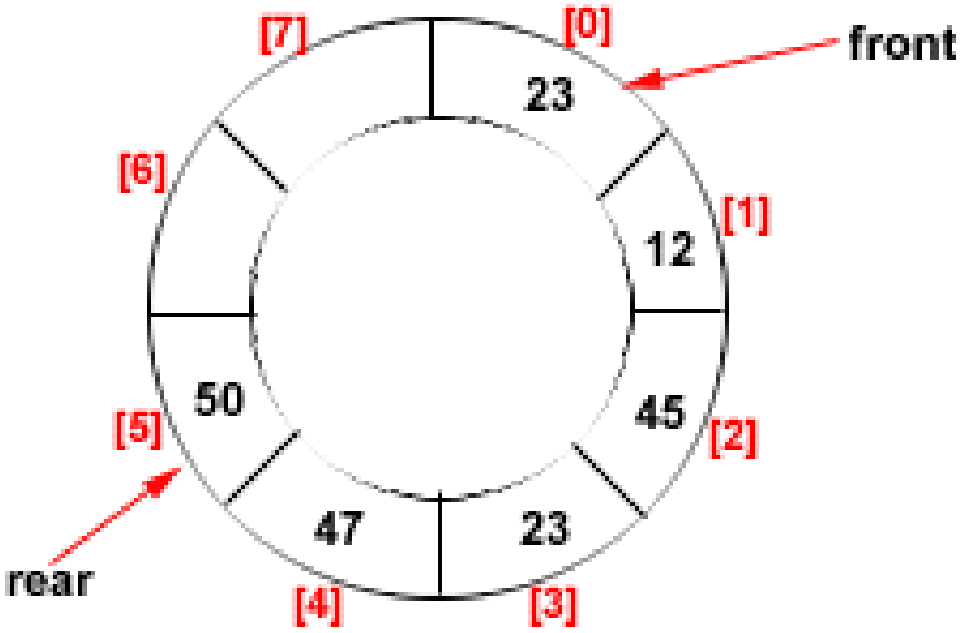
```
public class Queue {
      int queue[] = new int[5];  //array queue
      int size;
      int front;
      int rear;
      public void enqueue(int data) {  //inserting values
         queue [rear]= data;
         rear = rear+1;
         size = size+1;
      }
      public void show()
      {
         System.out.print("Elements :");
         for(int i=0;i<size;i++)
         {
            System.out.print(queue[front+i] + " ");
         }
      }
      public int dequeue()
      {
         int data = queue[front];
         front = front+1;
         size = size- 1;
         return data;
      }
}
```

Queue implementation through Array Java Code for Queue (contd.)

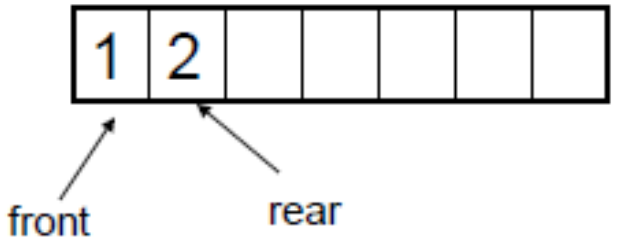
```
public class Driver {
    public static void main(String[] args) {
        Queue q= new Queue();
        q.enqueue(5);
        q.enqueue(2);
        q.enqueue(4);
        q.show();
        q.dequeue();

        q.show();

    }
}
```

Circular Queues

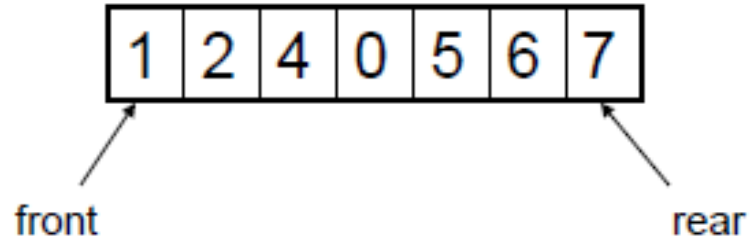
- A specific implementation of a queue.
- This queue is not straight but circular.



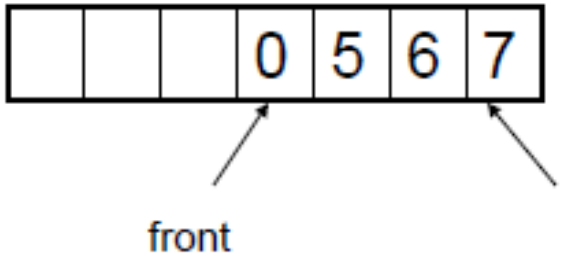
Initial state



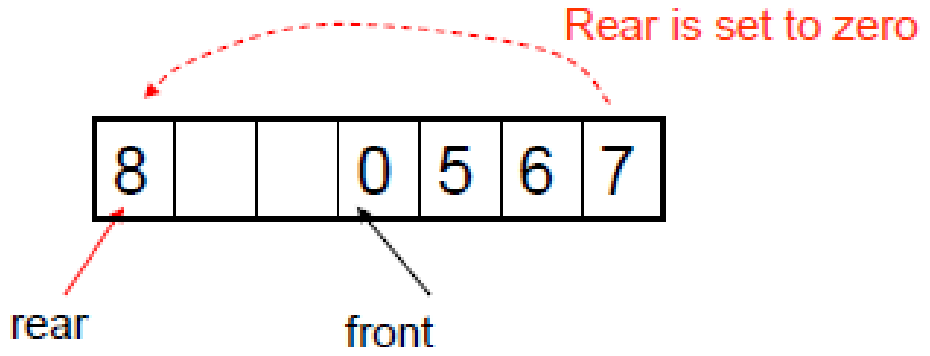
After adding 5 elements one by one



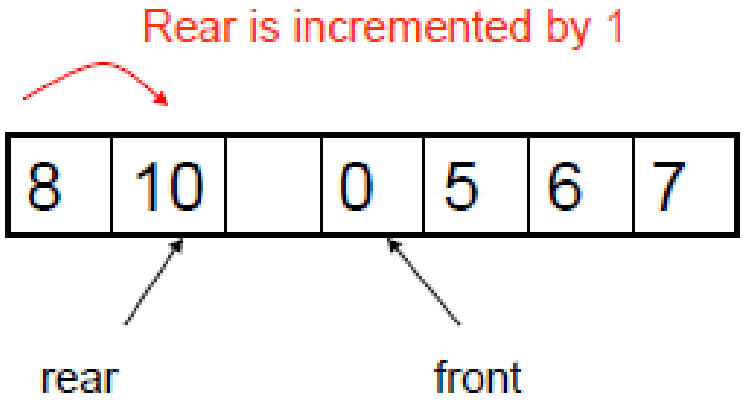
After deleting 3 elements one by one



Add new element 8 into the queue



Add 10 into the queue



Circular Queue Algorithm

- **Algorithm for Insertion**
- Step 1:
  - If the number of Items in the queue is maxsize(size of an array)then no more items can be inserted (queue is full). Quit.
- Step 2:
  - If the “rear” of the queue is pointing to the last position, then make the “rear” value as 0.
  - Otherwise, increment the “rear” value by one.
- Step 3:
  - Insert the new value for the queue position pointed by the “rear”

Circular Queue Algorithm

- **Algorithm for Deletion**
- Step 1:
  - If the number of Items in the queue is zero then no more items to be deleted (Queue is empty). Quit.
  - Otherwise, go to Step 2.
- Step 2:
  - Delete the “front” element.
- Step 3:
  - If the “front” is pointing to the last position of the queue
  - Make the “front” point to the first position (zero index) in the queue and quit.
  - Otherwise, Increment the “front” position by one.

Java Code for Circular Queue

```
class Queue
  {
  private int maxSize;          <-- Maximum queue size
  private int[] queArray;
  private int front;            <-- front locator
  private int rear;             <-- rear locator
  private int nItems;
//--------------------------------------------------------------
  public Queue(int s)       // constructor
    {
    maxSize = s;
    queArray = new int[maxSize];
    front = 0;
    rear = -1;
    nItems = 0;
    }
```

Java Code for Circular Queue (Contd.)

```
public void insert(int j)   // put item at rear of queue
    {
    if(rear == maxSize-1)
      rear = 0;
    queArray[++rear] = j;
     nItems++;
    }
//--------------------------------------------------------------
  public int remove()       // take item from front of queue
    {
    int temp = queArray[front++];
    if(front == maxSize)
     front = 0;
    nItems--;
return temp;
    }
```

deal with wraparound

increment rear and insert

get value and incr front

deal with wraparound

one less item

Java Code for Circular Queue

```
public int peekFront()     // peek at front of queue
    {
    return queArray[front];
    }
public boolean isEmpty()    // true if queue is empty
    {
    return (nItems==0);
    }
  public boolean isFull()     // true if queue is full
    {
    return (nItems==maxSize);
    }
public int size()          // number of items in queue
    {
    return nItems;
    }
```

Deque

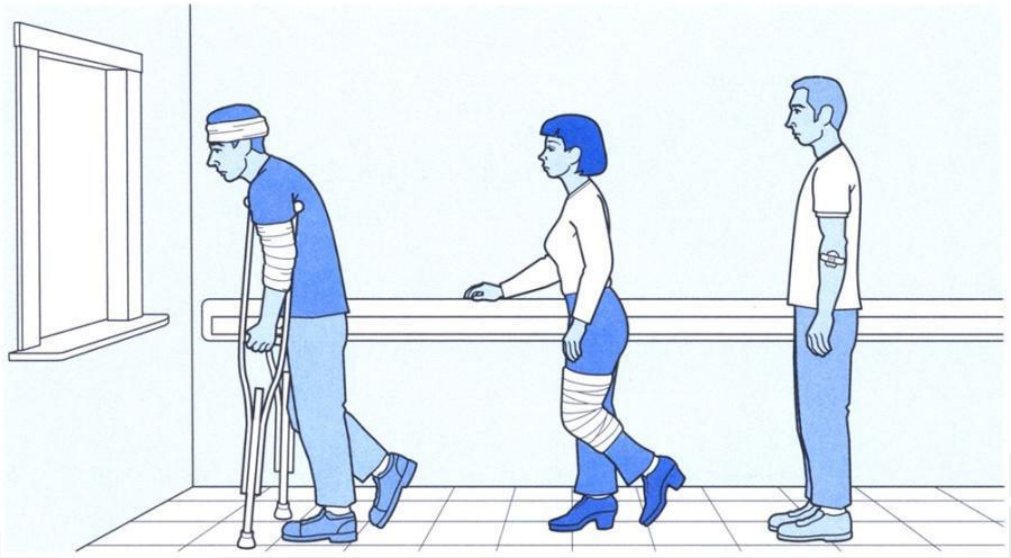
- Called as double-ended queue or deck.
- Elements can be added to or removed from either the front (head) or rear (back).
- Deque ADT is more general than Stack or Queue ADT.
- There might be methods like,
  - addFirst(e), removeFirst()
  - insertLast(e), removeLast()
  - Additionally,
  - first(), last(), size(), isEmpty()

Deque operation and their effects

Method	Return Value	DequeContents
addLast(5)	-	(5)
addFirst(3)	-	(3, 5)
addFirst(7)	-	(7, 3, 5)
first()	7	(7, 3, 5)
removeLast()	5	(7, 3)
size()	?	?
removeLast()	?	?
removeFirst()	?	?
addFirst(6)	?	?
last()	?	?
isEmpty()	?	?

Priority Queues

- Items are ordered by a priority value, at the insertion.
- Item with the highest priority is always at the front.
- Remove the element from the queue that has the highest priority, and return it.



Priority Queue ADT

- insert (k, v)
  - create an entry with key kand value vin the priority queue.
  - Key indicates the priority
- min()
  - Returns (but does not remove) a priority queue entry (k, v) having minimal key
  - Returns null if the queue is empty

Priority Queue ADT

- removeMin()
  - Removes and returns an entry (k, v) having minimal key from the priority queue
  - Returns null if the queue is empty
- size()
  - Returns number of entries in the priority queue
- isEmpty()
  - Returns a Boolean indicating whether the priority queue is empty

Set of operations and their effects

Method	Return Value	Priority Queue Contents
insert(5, A)	-	{(5, A)}
inser(9, C)	-	{(5, A), (9,C)}
insert(3, B)	-	{(3, B), (5, A), (9,C)}
min()	(3, B)	{(3, B), (5, A), (9,C)}
size()	(3, B)	{(5, A), (9,C)}
removeMin()	?	?
insert(7,D)	?	?
removeMin()	?	?
removeMin()	?	?
removeMin()	?	?
removeMin()	?	?
isEmpty()	?	?

Priority Queue Algorithm

- **Algorithm for Insertion**
- Step 1:
  - In number of items in the queue is maxsize(size of an array) then no more items can be inserted (queue is full). Quit.
  - Otherwise, go to Step 2

Priority Queue Algorithm

- Step 2:
  - If initially there are no more elements, insert new item at first position zero (0) index
  - Otherwise, if new item is larger than existing one‟s shift those elements upward one by one till the larger one is found.
- Step 3:Insert item to that new location

Priority Queue Algorithm

- **Algorithm for Deletion:**
- Step 1:
  - If number of items in the queue is zero then no more items to be deleted. Quit.
  - Otherwise go to step-2.
- Step 2:
  - Delete the “front” element.

Task

- Assume that you have a set of numbers like 5,4,3,7,1,6
- Consider the priorities of these numbers are equivalent to number itself.
- Hint: Assume1 has higher priority than the 2.
- Implement a priority queue with Inserting operation to illustrate the above scenario.

Thank You