Due on 18th Sep 2020 / A6

Week 6-, circular linked list, doubly linked list

CED19I027

1. **Show how a circular linked list can be useful to implement Circular Queue (CQ) ADT (Abstract Data Type). Compare and contrast with array based implementation. Which one would you recommend to implement CQ and why?**

For implementing a circular queue using circular linked list we can do the following:

- For enqueue operation: insert the element at the beginning every time
- For dequeue operation: delete the element in the End every time
- Other all operations like display queue etc are the same only.

Circular Queue using CLL	Circular Queue using Array
Size of the queue is flexible because of dynamic memory allocation. (No overflow condition)	Size of the queue is fixed (Overflow happens when the array is fully filled)
No Random access	Random access is possible
Memory Wastage is much lesser	Memory wastage is there but is less as it is circular queue

I would recommend using a Circular Linked List to implement a Circular Queue as it is very flexible and allows users to modify the queue more efficiently.
In case the user forgets to enqueue in proper order ,they can make use of functions like “insert from middle” and “delete from middle” to edit the order instead of deleting all elements and enqueueing again.

2. **Deque- double ended queue (may be pronounced ‘deck’):**
Unlike a queue, in deque, both insertion (enqueue) and deletion (dequeue) operations can be made at either end of the list. Suggest the suitable data structure to implement deque so that operations enqueue and dequeue can be done at both the ends in O(1) (constant) time. Write the code/algorithm to do this.

A circular queue can be used to implement a queue of this type.
Here we have 4 functions instead of simple enqueue and dequeue functions
ALGORITHM:

```
enqueueRear()
{

   if(front==-1&&rear==-1)
   {   printf("Enter the integer to be enqueued:\n");
      scanf("%d",&x);
      front=0;
      rear=0;
      queue[rear]=x;

   }
   else if((rear+1)%SIZE==front)
   {
      printf("Given circular queue is full.\n");
   }
   else
   {
      printf("Enter the integer to be enqueued:\n");
      scanf("%d",&x);
      rear=(rear+1)%SIZE;
      queue[rear]=x;
   }
}


enqueueFront()
{
   if(front==-1&&rear==-1)
   {   printf("Enter the integer to be enqueued:\n");
      scanf("%d",&x);
      front=0;
      rear=0;
      queue[rear]=x;

   }
   else if((rear+1)%SIZE==front)
   {
      printf("Given circular queue is full.\n");
   }
   else
```

```
    {
       printf("Enter the integer to be enqueued:\n");
       scanf("%d",&x);
      if(front==0)
       {
         front=SIZE-1;
       }
       else
       {
        front=(front-1)%SIZE;
       }
       queue[rear]=x;
    }

}

dequeueFront()
{
   if(front==-1&&rear==-1)
   {
      printf("Given queue is empty\n");
   }
   else if(front==rear)
   {
      printf("Dequeued integer : %d\n",queue[front]);
      front=-1;
      rear=-1;
   }
   else
   {
      printf("Dequeued integer : %d\n",queue[front]);
      front=(front+1)%SIZE;
   }
}


dequeueRear()
{
  if(front==-1&&rear==-1)
   {
      printf("Given queue is empty\n");
   }
   else if(front==rear)
   {
```

```
        printf("Dequeued integer : %d\n",queue[front]);
        front=-1;
        rear=-1;
    }
    else
    {
        printf("Dequeued integer : %d\n",queue[front]);
     if(rear==0)
        {
          rear=SIZE-1;
        }
        else
        {
         rear=(rear-1)%SIZE;
        }


    }

}
```

3. **What is the worst case time complexity of inserting a node in a doubly linked list?**

   For inserting a node
   a) At the beginning : O(1) because already head pointer is pointing the first node and there is no need to traverse the whole list or there is no relation with number of nodes (n) of the list
   b) At the end: O(n) because we should traverse from 1st node to last (nth) node and then insert
   c) At a random position: If we are finding the node by searching for the value stored in it , then it is O(n) .

4. **Circular Queue:**

   Given a circular array-based queue 'q' capable of holding '7' objects. Show the final Contents of the array after the following code is executed: Array index start from '0'

```
for (int k = 1; k <= 6; k++)
q.enqueue(k);
for (int k = 1; k <= 3; k++)
{
        q.dequeue();
        q.enqueue(q.dequeue());
        q.enqueue(k);
```

}

1	4	2	6	3(rear)	-	2(front)

5. Write an algorithm to delete last node in the doubly linked list

```
struct node *temp,*head ,*ptr;\\head pointer points to the first node always\\
\\prev stores prev node address to temp node and next stores next node address to temp node\\
int deleteE(struct node* next,struct node* prev)
{
temp=head;
while(temp->next!=0)
{
  temp=temp->next;
}
ptr=temp;
temp=temp->prev;
free(ptr);
temp->next=NULL;
}
```

6. What is the functionality of the following piece of pseudo code on DLL?

```
public int function()
{
        Node temp = tail.getPrev();
```

```
            tail.setPrev(temp.getPrev());
            temp.getPrev().setNext(tail);
            size--;

}
```

inside the function,

first step: A temporary node pointer will point to the node lying previously to the tail node.(to the last 2nd node)

second step: The previous node to last node is set as the last 3rd node

third step: we access the node which is previous to temp node ,i.e,the last third node and assign its next node as tail node.

After these 3 steps , we have **removed the last 2nd node from the doubly linked list** but it can still be accessed through temp pointer .

The last 3 elements will now look as shown below: