Data Structures and its Applications

Dinesh Singh
Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Circular Queue - Implementation

Dinesh Singh
Department of Computer Science & Engineering

Circular Queues - definition

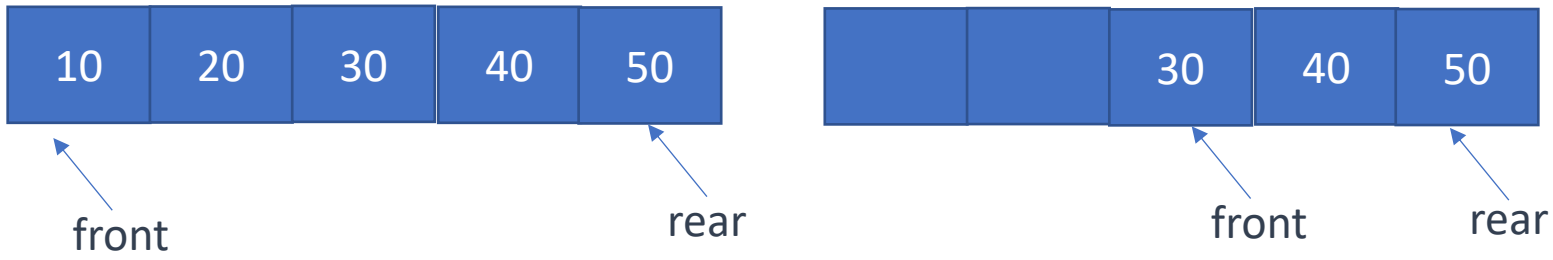
- Circular Queue is a linear data structure, which follows the principle of FIFO(First In First Out), but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

- In a simple queue, once the queue is completely full, it's not possible to insert more elements. Even if we perform remove operation on the queue to remove some of the elements, until the queue is reset, no new elements can be inserted

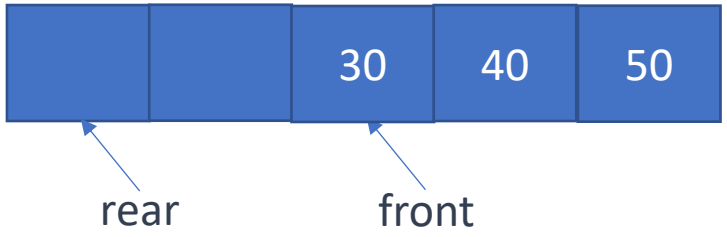
Data Structures and its Applications

Drawback of a simple Queue

Structure of the simple queue



Cannot insert even after two elements are removed and Space available in the front.



It is possible to insert in a circular queue by moving the rear To the beginning of the queue

Implementation of operations

- To insert into the queue : Finding the rear index

```
rear = (rear + 1) % size
If(rear=front)
   cannot insert
else
   insert at rear index
```

- For eg. If size = 5, front = 2 and rear = 4
- rear = (4 + 1) % 5 = 0,
- The new element gets inserted at index 0

- For eg. If size = 5, front = 0 and rear = 4
- rear = (4 + 1) % 5 = 0,
- rear = front , therefore cannot insert

Implementation of operations

- To remove from the queue :

  remove the element pointed by front , move the front

  front = (front + 1) % size

For eg. If size = 5, front = 2 and rear = 4

- front = (2 + 1) % 5 = 3,
- front moves to index 3 after removal of the element,

- For eg. If size = 5, front = 4 and rear = 2
- front = (4 + 1) % 5 = 0,
- Front moves to 0 after removal of the element

Structure of a circular Queue – Sequential Representation

```
#define MAXQUEUE 100
struct queue
{
  int items [MAXQUEUE];
  int front, rear;
} ;

struct queue q;
q.rear = q.front = -1;
```

Functions to implement the operations

- insert (&q,x)
- remove (&q)

Data Structures and its Applications

Implementation of operations - insert

```
int qinsert(struct queue *q, int x)
{

 //check for queue overflow
      if((q->r+1)%MAXQUEUE==q->f)
      {
           printf("Queue Overflow..\n");
            return -1;
       }
      else
       {
            q->rear=(q->rear+1)%size;     //get the rear index
            q->item[q->rear]=x;    //insert at rear index
            if(q->front==-1)  //if first element
              q->front=0;     // make front point to 0
            return 1;
             }
    }
```

Implementation of operations - insert

```
//ANOTHER WAY TO IMPLEMENT INSERT
int qinsert(int *q,int *f, int *r, int size, int x)
{
 // f ,r are pointers to front and rear of the queue, size is the max size of queue
 //check for queue overflow
      if((*r+1)%size==*f)
      {
           printf("Queue Overflow..\n");
            return -1;
       }
      else
       {
            *r=(*r+1)%size;     //get the rear index
            q[*r]=x;    //insert at rear index
            if(*f==-1)  //if first element
              *f=0;     // make front point to 0
            return 1;
             }
   }
```

Data Structures and its Applications

Implementation of operations - remove

```
int remove(struct queue *q)
 {
   int x;
  if(q->front==-1)  //check for empty queue
  {
   printf("Queue empty..\n");
   return -1;
  }
else
  {
    x=q->items[q->front];
    if(q->front==q->rear)//only one element
      q->front=q->rear=-1;
 else
      q->frontf=(q->front+1)%MAXQUEUE;   //increment the front
    return x;
   }
}
```

Data Structures and its Applications

Implementation of operations - remove

```
//ANOTHER WAY TO IMPLEMENT REMOVE
int remove(int *q, int *f, int *r,int size)
 {
   int x;
  if(*f==-1)  //check for empty queue
  {
   printf("Queue empty..\n");
   return -1;
  }
else
  {
    x=q[*f];
    if(*f==*r)//only one element
      *f=*r=-1;
 else
      *f=(*f+1)%size;   //increment the front
    return x;
   }
}
```

Implementation of operations - display

```
void display(struct queue q)
 {
  if(q.front==-1)
    printf("\nQueue empty..\n");
  else
  {
    while(q.front!=q.rear) //increment front till it reaches rear
    {
      printf("%d ",q.items[q.front]);
      q.front=(q.front+1)%MAXQUEUE;
    }
   printf("%d ",q->items[q->front]); // display the last element
  }
}
```

Implementation of operations - display

```
void display(int *q, int f, int r, int size)
 {
  if(f==-1)
    printf("\nQueue empty..\n");
  else
  {
    while(f!=r) //increment front till it reaches rear
    {
      printf("%d ",q[f]);
      f=(f+1)%size;
    }
   printf("%d ",q[f]); // display the last element
  }
}
```

THANK YOU

Dinesh Singh
Department of Computer Science & Engineering

dineshs@pes.edu

+91 8088654402