Data Structures and its Applications

Dinesh Singh
Department of Computer Science & Engineering

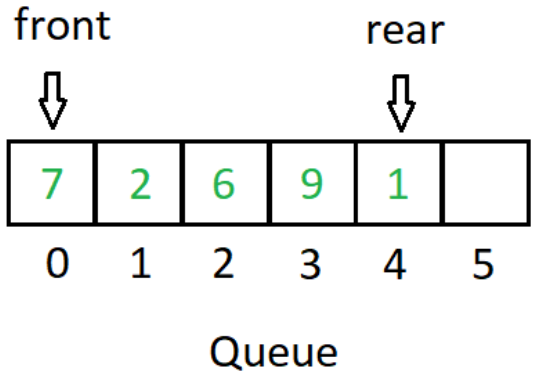
DATA STRUCTURES AND ITS APPLICATIONS

Basic Structure of a Queue , Implementation using an Array

Dinesh Singh
Department of Computer Science & Engineering

Queue Data Structure - definition

- A Queue is an ordered collection of items from which items may be deleted at one end (called the front of the queue)and into which items may be inserted at the other end (called the rear of the queue).



Types Queue

- Different Types of Queues
  - Simple Queue
  - Circular Queue
  - Priority Queue
  - Dequeue

- Implementation
  - Sequential Representation (Arrays)
  - Linked Representation (Linked Lists)

Queue Operations

- Three primitive operations can be applied to the queue
- Insert (q,x) : inserts x at the rear of the queue q
- x = remove (q) : deletes front element from the queue and set x to its contents
- empty(q) : returns true or false depending on whether the queue contains any elements.
- Insert operation cannot be performed if the queue has reached the maximum size.
- The result of an illegal attempt to insert an element into a queue which has reached its maximum size is called overflow.
- The remove operation can be applied only if the queue is non empty.
- The result of an illegal attempt to remove an element from an empty queue is called underflow.
- The empty operation is always applicable.

Structure of a Simple 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 (x, &q)
- remove (&q)
- empty(&q)

Simple Queue – Insert Operation

Inserting into a queue

1. Check the queue for overflow condition
2. If the queue is not in overflow condition, increment the rear pointer and insert the element at a location indicated by the rear pointer.
3. If this is the first element in the queue, initialise front to 0.
4. Return 1

Simple Queue – Implementation of Insert

```
int insert(int x,struct queue *q)
 {
  //check queue overflow
 if(q->rear==MAXQUEUE - 1)
 {
  printf("Queue overflow..\n");
  return -1;
 }
 (q->rear)++;
 q->items[q->rear]=x;
 if(q->front==-1)//if first element
   q->front=0;
 return 1;
}
```

Simple Queue – Remove Operation

Removing element from the queue

1. Check the queue for underflow condition
2. If the queue is not in underflow condition, remove the element pointed by the front pointer into x.
3. If this was the only element in the queue, initialise front and rear to -1.
4. Return x

Simple Queue – Implementation of remove

```
int remove(struct queue *q)
 {
  int x;

  if(q->front==-1)
  {
    printf("Queue empty..\n");//underflow
    return -1;
  }
   x=q->items[q->front];
   if(q->front==q->rear)//only one element in queue
    q->front=q->rear=-1;
   else
    (q->front)++;
   return x;
 }
```

Simple Queue – Implementation of empty and display

```
int empty ( struct queue * q)
{
  if (q->front ==-1)
    return 1;
  return -1;
}
display (struct queue * q)
{
 int i;
 if(q->front==-1)
   printf(”Empty queue..\n”)
 else
 {
   for ( i = q->front; i<=q->rear;i++)
     printf(”%d”,q->items[i]);
 }
}
```

Simple Queue – Another Implementation

Implementation where queue represented as an array , front and rear are separate variables

```
#define MAXQUEUE 100

int q[MAXQUEUE]
int front, rear
front = rear = -1
```

Function calls to implement the operations

- insert(x, q, &front, &rear) ;
- remove (q , &front, &rear);
- empty(q, &front)

Simple Queue – Another Implementation

Implementation where queue represented as an array , front and rear are separate variables

```
int insert(int x,int *q,int *f,int *r)
 {
  //check queue overflow
 if(*r==MAXQUEUE - 1)
 {
  printf("Queue overflow..\n");
  return -1;
 }
 (*r)++;
 q[*r]=x;
 if(*f==-1)//if first element
  *f=0;
 return 1;
}
```

Simple Queue – Another Implementation

Implementation where queue represented as an array , front and rear are separate variables

```
int remove(int *q,int *f,int *r)
 {
  int x;
  if(*f==-1)
  {
    printf("Queue empty..\n");
    return -1;
  }
   x=q[*f];
   if(*f==*r)//only one element in queue
     *f=*r=-1;
   else
     (*f)++;
   return x;
 }
```

Simple Queue – Another Implementation

Implementation where queue represented as an array , front and rear are separate variables

```
display (int *q, int f, int r)
{
  int i;
  if(f==-1)
    printf(”Empty queue..\n”)
  else
  {
    for ( i = f; i<= r; i++)
      printf(”%d”,q[i]);
  }
}
```

THANK YOU

Dinesh Singh
Department of Computer Science & Engineering

dineshs@pes.edu

+91 8088654402