NAME: GAURAV KISHOR PATIL

DIV: 2 ROLL NO:54

BATCH: C

Experiment No.5
Implement Priority Queue ADT using array
Date of Performance:
Date of Submission:

Experiment No. 5: Priority Queue

Aim: To Implement Priority Queue ADT using array

Objective:

Circular Queues offer a quick and clean way to store FIFO data with a maximum size

Theory:

A priority queue is a type of queue that arranges elements based on their priority values. Elements with higher priority values are typically retrieved before elements with lower priority values.

In a priority queue, each element has a priority value associated with it. When you add an element to the queue, it is inserted in a position based on its priority value. For example, if you add an element with a high priority value to a priority queue, it may be inserted near the front of the queue, while an element with a low priority value may be inserted near the back..

Operations of a Priority Queue:

A typical priority queue supports the following operations:

1) Insertion in a Priority Queue

When a new element is inserted in a priority queue, it moves to the empty slot from top to bottom and left to right. However, if the element is not in the correct place then it will be compared with the parent node. If the element is not in the correct order, the elements are swapped. The swapping process continues until all the elements are placed in the correct position.

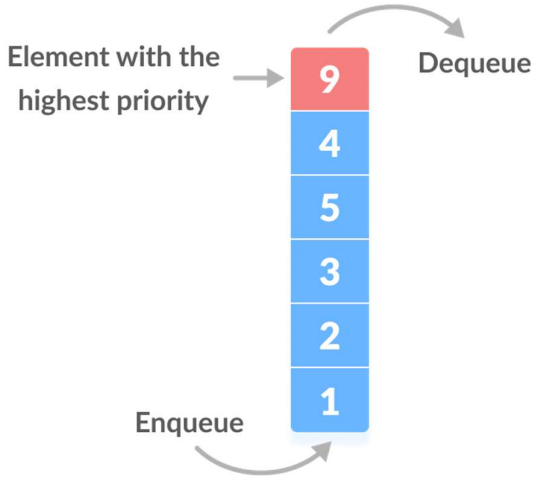
2) Deletion in a Priority Queue

As you know that in a max heap, the maximum element is the root node. And it will remove the element which has maximum priority first. Thus, you remove the root node from the queue. This removal creates an empty slot, which will be further filled with new insertion. Then, it compares the newly inserted element with all the elements inside the queue to maintain the heap invariant.

3) Peek in a Priority Queue

This operation helps to return the maximum element from Max Heap or the minimum element from Min Heap without deleting the node from the priority queue.

Priority Queue



Algorithm

Algorithm : PUSH(HEAD, DATA, PRIORITY):

Step 1: Create new node with DATA and PRIORITY

Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end. Else goto Step 5.

Step 3: NEW -> NEXT = HEAD

Step 4: HEAD = NEW

Step 5: Set TEMP to head of the list

Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY > PRIORITY

Step 7: TEMP = TEMP -> NEXT

[END OF LOOP]

Step 8: NEW -> NEXT = TEMP -> NEXT

Step 9: TEMP -> NEXT = NEW

Step 10: End

POP(HEAD):

Step 1: Set the head of the list to the next node in the list. HEAD = HEAD -> NEXT.

Step 2: Free the node at the head of the list

Step 3: End

PEEK(HEAD):

Step 1: Return HEAD -> DATA

Step 2: End

Code:

```
#include<stdio.h>
#include<conio.h>
#define MAX 5
int array[MAX];
int front,rear;
void initialize()
{
	front=rear=-1;
}
void enqueue(int x)
{
	int i,j;
	if(rear==-1)
	{
		front=rear=0;
		array[front]=x;
	}
	else if(rear+1==MAX)
	{
		printf("Queue is full....cannot add\n");
	}
```

```
        else
        {
                for(i=front;i<=rear;i++)
                {
                        if(x<array[i]) break;
                }
                rear++;
                for(j=rear;j>i;j--)
                {
                        array[j]=array[j-1];
                }
                array[i]=x;
        }

}
int dequeue()
{
        int x=-1;
        if(front==-1)
        printf("Queue is empty\n");
        else
        {
```

```
            x=array[front];
            front++;
            if(front>rear) front=rear=-1;
        }
        return x;
}
int empty()
{
        if(rear==-1) return 1;
        else return 0;
}
int full()
{
        if(rear==MAX-1) return 1;
        return 0;
}
void display()
{
        int i=front;
        if(front==-1) printf("Queue is empty\n");
        else
        {
```

```
                printf("Queue contains:\n");
                for(;i<=rear;i++)
                {
                        printf("%d\n",array[i]);
                }
        }
}
void main()
{
        int choice,x;
        clrscr();
        initialize();
        do
        {
                printf("1.Enqueue an element\n2.Dequeue an element\n3.Display\n4.Check if queue is empty\n5.Check if queue is full\n6.Exit\n");
                printf("_____\n");
                printf("Enter Your Choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:printf("Enter element to enqueue:");
                        scanf("%d",&x);
```

```
                    enqueue(x);
                    break;
                    case 2:x=dequeue();
                    if(x!=-1) printf("Element deleted from queue is%d\n",x);
                    break;
                    case 3:display();
                    break;
                    case 4:if(empty()) printf("Queue is empty\n");
                    else printf("Queue is not empty\n");
                    break;
                    case 5:if(full()) printf("Queue is full\n");
                    else printf("Queue is not full\n");
                    break;
                    case 6:break;
                    default:printf("Invalid choice\n");
                }
        }while(choice!=6);
        getch();
}
```

Output:

```
4.Check if queue is empty
5.Check if queue is full
6.Exit

_____________________________________
Enter Your Choice:1
Enter element to enqueue:45
1.Enqueue an element
2.Dequeue an element
3.Display
4.Check if queue is empty
5.Check if queue is full
6.Exit

_____________________________________
Enter Your Choice:3
Queue contains:
34
45
1.Enqueue an element
2.Dequeue an element
3.Display
4.Check if queue is empty
5.Check if queue is full
6.Exit

_____________________________________
Enter Your Choice:
```

Conclusion:

A priority queue is a special type of queue in which each element is associated with a priority value. Elements are served based on their priority, with higher priority elements being served first. If elements with the same priority occur, they are served according to their order in the queue.

The time complexity of the program is O(n) but the same program can be redo with the help of heap concept to minimize the time complexity and maximize the efficiency.