**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**

**A Report on**

**COMP 202: Data Structures and Algorithms**

**Mini Project**

**Submitted by:**

Bisheshwor Neupane(35)

Gaurab Shrestha(43)

Umesh Shrestha(47)

**Submitted to:**

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

**Submission Date:**

January 19, 2019

**Task:**  Implement a queue using a circularly linked list with the following operations: • enqueue – Inserts an element at the end of the queue

• dequeue – Displays and deletes an element from the front of the queue

• rear – Displays the last element of the queue • front – Displays the first element of the queue

• isEmpty – Returns true if the queue is empty and false otherwise

• isFull – Returns true if the queue is full and false otherwise

What are the time complexities of these operations in your implementation? You are supposed to use the linked list implementation done in Lab 2.

**Implementation:**

**Algorithm:**

Enqueue(element)

1. Create a new node,newNode
2. newNode->info=data
3. newNode->next=NULL
4. TAIL->next = newNode
5. TAIL = newNode
6. If(HEAD==NULL)
7. HEAD=TAIL
8. else
9. TAIL->next = HEAD
10. endif

Dequeue()

1. Node\* nodeToDelete
2. nodeToDelete =HEAD
3. int num = HEAD->info
4. HEAD = nodeToDelete->next
5. TAIL->next = HEAD
6. return num

Front()

1. return HEAD->info

Rear()

1. return TAIL->info

isEmpty()

1. if (HEAD==NULL && TAIL == NULL)
2. return true
3. else
4. return false
5. endif

isFull()

1. int p=0
2. Node\* temp = HEAD
3. while(temp!=NULL)
4. p++
5. temp = temp-> next
6. if(temp==HEAD)
7. break
8. endif
9. endwhile
10. if(p<maxSize)
11. return false
12. else
13. return true
14. Endif

Traverse()

1. Node\* temp = HEAD
2. while(temp!=NULL)
3. Display temp->info
4. temp = temp-> next
5. if(temp==HEAD)
6. return
7. endif
8. endwhile

**Time Complexity of the program:**

Before implementing the program, we have considered the maximum number of elements that can be enqueued to be 7. But, let us assume it to be ‘n’ for finding time complexity of the program.

|  |  |  |
| --- | --- | --- |
| **S.No** | **Program** | **Steps** |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21. | int main(){  cout<<"\nQueue Implementation:"<<endl;  Queue qu;  qu.isEmpty();  qu.enqueuetohead(4);  qu.dequeue();  for(int i=1;i<15;i++){ //in general to run loop for i<n ;  if(qu.isFull() == false)  qu.enqueue(i);  else{  cout<<"Queue is full !";  break; }  if(qu.isFull() == true)  cout<<" queue full ";  else  cout<<"Not full ";  qu.isEmpty();  qu.showitems();  cout<<"\n Rear and front:";  cout<<qu.rear()<<" ";  cout<< qu.front();  return 0; } | 0  1  0  4  6  6  n<=8  (4n+7)n  7n  0  1  1  4n+7  1  0  0  4  3n+1  1  1  1  0  O(n^2) |

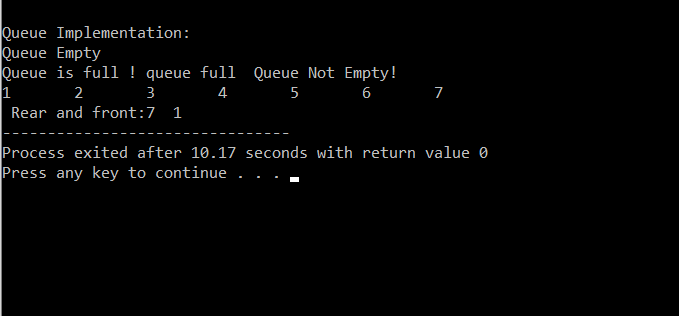
If we execute the loop of step 7 until ‘i<15’, then only 7 elements are added in queue and remaining cant be added. This is because the ‘maxSize’ has already been defined to 7 . Hence, for the condition ‘i<15’ , the time complexity becomes constant for the case. But, in generalized form, for ‘i<n’ such that ‘n ’ can range from 0 to undefined, the time complexity of total runtime of the program becomes O(n^2).But still, for this implementation the value of n becomes constant, suppose K . Hence, for the above program, the time complexity is constant.

**Screenshots**:

The main block:



**Output window:**



Hence, the implementation works correctly.