Lecture-57

(Queues)

⭐ Introduction to Queues :-

→ Data structure which supports FIFO (First come First out) / FCFS (First come First serve)

→ Queue is a linear Data Structure.

⭐ Types of operations on Queues

1) Enqueue :- help to add a new element in the ~~the~~ Queue.

2) Dequeue :- this operation helps us to remove a new element in the queue.

3) isfull 4) isEmpty

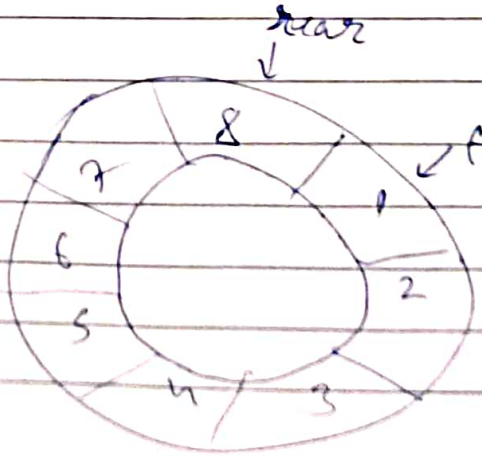
5) front → Gives us the element which comes first.

⭐ Types of Queues :-

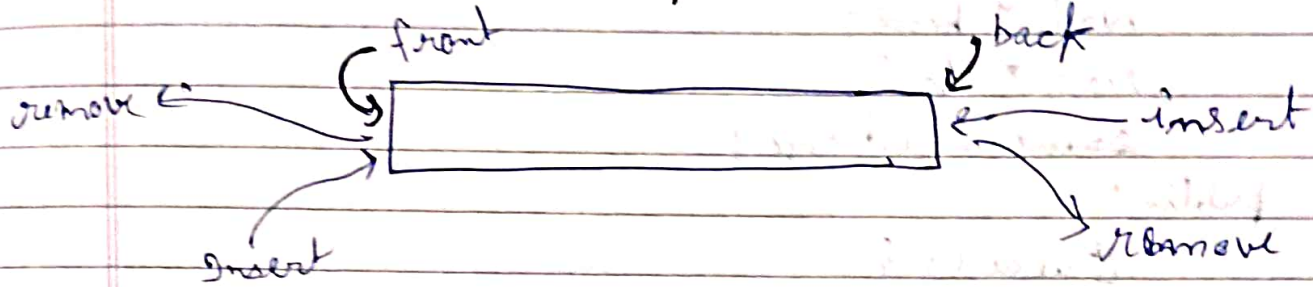
1) Simple

2) Priority Queue

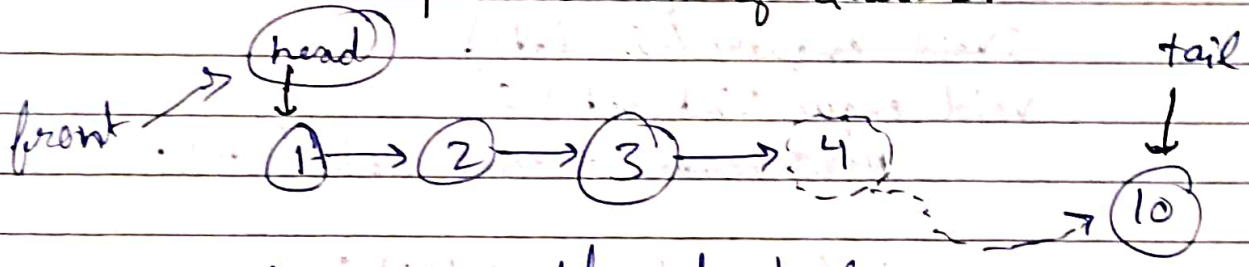
3) Circular Queue



4) Double Ended queue / Deque



⭐ Linked List Implementation of Queues:



enqueue → add at tail

Dequeue → remove from head

front → head → data

Lecture 50

#Code

```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node * next;
    Node (Node int data) {
        this -> data = data;
        this -> next = NULL;
    }
}
```

```
Class Queue {
    Node * head;
    Node * tail;
    int size;
public:
    Queue() {
        this -> head = NULL;
        this -> tail = NULL;
        this -> size = 0;
    }
    Void enque(int data) {
        Node* newNode = new Node*(data);

        if (this -> head == NULL) {
            this -> head = this -> tail = newNode;
        } else {
            this -> tail -> next = newNode;
            this -> tail = newNode;
        }
        this -> size++;
    }
    Void deque() {
        if (this -> head == NULL) {
            return;
        } else {
            Node * oldhead = this -> head;
            Node * newhead = this -> head -> next;
            this -> head = newhead;
            if (this -> head == NULL) this -> tail = NULL;
            oldhead -> next = NULL
            delete oldhead;
            this -> size--
        }
    }
```

```
int getsize(){
        return this->size;
}
bool isempty(){
        return this->head == NULL;
}
int main(){
        Queue qu;

int front(){
        if (this -> head == NULL) return -1;
        return this->head -> data;
}
int main(){
        Queue qu;
        qu.enqueue(10);                 //10
        qu.enqueue(20);                  //10 20
        qu.enqueue(30);                  //10 20 30
        qu.dequeue();               //20 30
        qu.enqueue(40);                //20 30 40

        while(not qu.isEmpty()){
                cout << qu.front() << " ";
                qu.dequeue();
        }
        return 0;
}
```

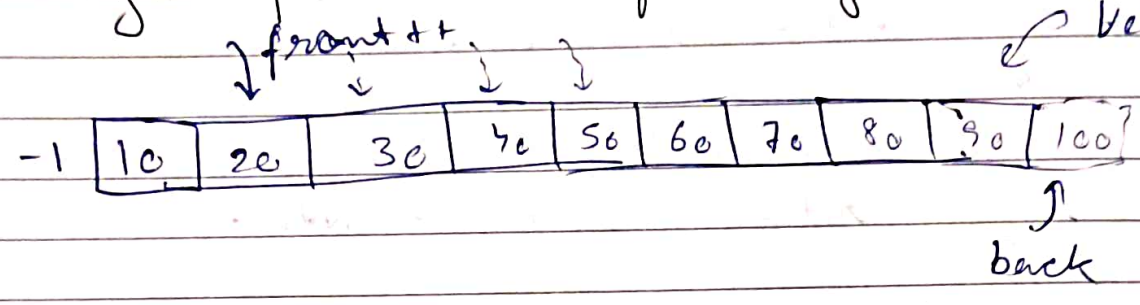
oldh - head
rem -> head -> next
free(oldh)
head -> ...

FREEMIND

Date

Page

Array implementation of ~~Arrays~~ Queues



V. pushback (n)

V.back

qu. enque (10) - (20) - (30) - deque

#Code

```
#include<vector>
using namespace std;

class Queue{
    ~~this->back~~ int front;
    int back;
    vector<int> v;
public:
    Queue(){
        this->back=-1;
        this->front=-1;
    }
    Void enqueue(int data){
     this->v.push_back(data);
        this->back++;
    if(this->back==0) this->front=0;
    Void dequeue(){
        if(this->front==this->back){
            this->front=-1;
            this->back=-1; this->v.clear();
        }else{
            this->front++
    }
```

```
int getfront() {
    if (this->front = -1) return -1;
    return this->V[this->front];
}
bool isEmpty() {
    return this->front == -1;
}
};
int main() {
    Queue qu;
    qu.enqueue(10);
    } same as prev.

    while (not qu.empty()) {
    cout<<qu.getfront()<<" ";
    qu.dequeue();
    }
    return 0;
}
```

Note: Similar to stacks there is an inbuilt library for Queues. So, we can use it instead of making it by arrays/linked list.

```
#include <queue>
using namespace std;
int main() {
    queue<int> qu;
    qu.push(10);
    qu.push(20);
    qu.pop();
    while (not qu.empty) {
        cout << qu.front() << " "; qu.pop();
```

FREEMIND

Date

Page

⭐ Advantage of array implementation of queues

ll → space efficiency { Good

ll → pointer → deletion

queue → STL

Q) Reverse the elements of a queue.

① Move elements from queue to stack
② Move elements back from stack to queue

Ans)

```
#include <queue>
#include <stack>
using namespace std;
int main() {
    queue<int> input;
    input.push(10);
    input.push(20); (30); (40);
    while(not input.empty()) {
        st.push(input.front());
        input.pop();
    }
    while(not st.empty()) {
        input.push(st.top());
        st.pop();
    }
    while(not input.empty()) {
        cout << input.front() << " ";
        input.pop();
    }
    return 0;
}
```

Lecture-58

(Queues)

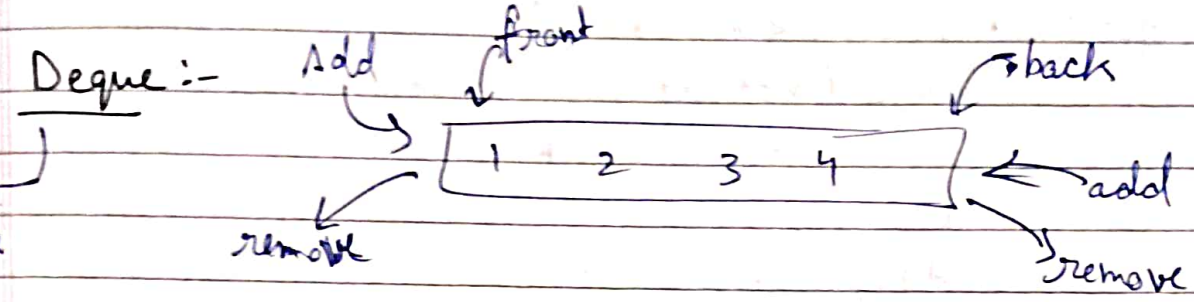
Interview Questions

FREEMIND

Date

Page

Deque :-



Double ended queue

⇒ we can add/remove elements from both sides.

{Check inbuilt funct of different libraries at cppreference}

- push_back();
- push_front();
- pop_back();
- pop_front();

Qs You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window moves right by one position. Return the max sliding window which basically contains the max element in each window.

[1, 2, -1, -3, 5, 3, 6, 7]

[0 1 2] [1 2 3] [3] [6]

Sol:-

```
#include <deque>
#include <vector>
using namespace std;

vector<int> maxSlidingWindow(vector<int>& arr, int k){
    deque<int> dq;
    vector<int> res;
```

Time → O(n)
Space → O(k)

FREEMIND
Date
Page

```
for(int i=0; i<k; i++) {
    while (not dq.empty and arr[dq.back()] < arr[i]){
        dq.pop_back();
    }
    dq.push_back(i);
}
res.push_back(arr[dq.front()]);
for(int i=k; i<arr.size(); i++){
    int curr = arr[i];
    if (dq.front() == (i-k)) dq.pop_front();
    while (not dq.empty() and arr[dq.back()] < arr[i]) {
        dq.pop_back();
    }
    dq.push_back(i);
    res.push_back(arr[dq.front()]);
}
return res;
}
int main() {

}
```

@ We are given a stack data structure with push and pop operations, the task is to implement a queue using instances of stack data structure and operations on them.

queue → fifo stack → lifo

(3) (*) Push efficient (enqueue) → O(1), pop (dequeue) can be bad

(*) Pop efficient (dequeue) → O(1), push (enqueue) can be bad

Push Efficient Queue

```
#include <iostream>
#include <stack>
using namespace std;

class Queue {
    stack<int> st;
public:
    Queue() { }

    void push(int x) {          // queue. enqueue
        this->st.push(x)        // Time -> O(1)
    }
    void pop() {                // dequeue
     // Time -> O(n)

        if (this->st.empty()) return;
        stack<int> temp;
        while (this->st.size() > 1) {
            temp->st.push(
            temp.push(st.top());
            st.pop();
        }
        // now stack size is 1 and we are at bottom element.
        this->st.top();
        while (!temp.empty()) {
            this->st.push(temp.top())
            temp.pop();
        }
    }
    bool empty() {
        return this->st.empty();
    }
```

```
int front() {
    if (this->st.empty()) return;
    stack<int> temp;
    while (this->st.size()>1) {
        temp.push(st.top());
        st.top();
    }
    int result = this->st.top();
    while (not temp.empty()) {
        this->st.push(temp.top());
        temp.pop()
    }
    return result;
}
};
int main() {
    Queue qu;
    qu.push(10);------(40);
    qu.pop();
    while(not qu.empty()) {
        cout << qu.front() << endl;
        qu.pop();
    }
    return 0;
}
```

{pop efficient Queue}

```
void push(int n) {
    stack<int> temp;
    while(!this->st.empty()) {
        temp.push(st.top())
        st.pop();
    }
```

(OW)

Date

Page

```
this->st.push(x);

    while (not temp.empty()) {

        this->st.push(temp.top())
        temp.pop;

    }
}
Void pop() {
    if (this->st.empty) return;        O(1)
  this->st.pop();
}
bool empty() {
    return this->st.empty();
}
int front() {
    if (this->st.empty()) return INT_MIN;
    return this->st.top();
}
};
int main() {
```

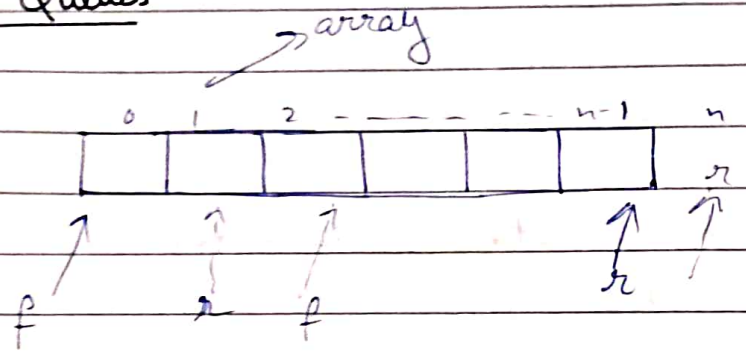
H.W

Q1) → We are given a stack data structure with push and pop operations, the task is to implement a queue using instances of stack data structure and operations on them.

f = front
r = rear

Date
Page

Circular Queues



(r + 1) % n
(f + 1) % n [0, n-1]

#Code

```
# include <vector>
using namespace std;

class Queue {
    int front;
    int back;
    vector<int> v;
    int cs;          // current size
    int ts;          // total size
public:
    Queue(int n) {
        v.resize(n);
        this->ts = n;
        this->back = n-1;
        this->front = 0;
        this->cs = 0;
    }
    Void enqueue(int data) {
        if (isfull()) return;
        this->back = (this->back+1) % this->ts
        this->v[this->back] = data;
        this->cs++;
```

Date

Page

```
Void dequeue (){
    if (isEmpty()) return;
    this -> front = (this -> front + 1) % this -> fs;
    this -> cs --;
}

int getfront (){
    if (this -> front == -1) return -1;
    return this -> v [this -> front];
}

bool isempty (){
    return this -> cs == 0;
}
bool isFull (){
    return this -> cs == this -> fs;
}
```