**큐**

**서문**

큐는 스택과 다르게 FIFIO의 구조를 취하는 자료구조이다. LIFO를 이해 했다면 FIFO가 뭔지 금방 감이 올 것이다. 큐는 FIFO 즉 first in first out의 구조를 취하는 자료 구조이다. 즉 먼저 온사람이 먼저 밖으로 나가는 것이다. 우리가 줄을 서거나 공장의 컨베이어 벨트가 큐를 사용한 적절한 예이다. 만약 줄을 섰는데, 스택의 구조를 활용한다고 하면 얼마나 큰일 나겠는가 .

**큐의 구조**

큐를 클래스를 이용하여, 만들어보자 큐는 stack과 달리 앞과 뒤를 표현해주는 front와 rear라는 변수가 있다. Stack과 마찬가지로 생성자 함수의 전달인자로 이 큐의 size를 지정한다.

아이템이 큐 배열에 추가되면 rear가 1씩 증가 된다. 그리고 size변수와 rear가 일치하면 더 이상 큐에 아이템 추가를 멈출 수 있다. 반대로 아이템이 큐 배열에서 제거 될 때는 front가 일씩 증가하도록 합니다.

**큐의 유틸리티**

큐는 배열에 아이템을 추가하는 enqueue, 아이템을 배열에서 제거하는 deque가 있다. 좀더 편한 사용을 위해 큐가 꽉 차있는지 확인 하는 isFull과 내용물이 비어있는지 확인 하는 isEmpty함수를 제작 할 것이다.

isEmpty메서드는 스택과는 조금 다른 조건인데, front값이 rear값을 초과하게 된다면 해당 배열이 빈 것이라 판단한다.

삽입을 담당하는 enqueue메서드는 삽입할 때 rear를 1씩 증가 시켜서 해당 인덱스에 아이템을 추가하는 메서드이다. Enqueue의 조건은 rear가 maxSize까지 도달 하기 전까지 가능하다.

삭제를 담당하는 dequeue메서드는 queue[front]에 있는 아이템을 반환함과 동시에 front를 1씩 증가시키는 기능을 가지고 있다.

**실습**

스택과 동일하게 front,rear,queue변수는 캡슐화로 숨기도록 하겠다.

class QueType{

constructor(size){

let queue = [];

let front = 0;

let rear = -1;

const maxSize = size;

const isEmpty = ()=>{

return rear === -1 || front > rear;

}

const isFull = ()=>{

return rear + 1 === maxSize;

}

this.enque = (item) => {

if(isFull()){

console.log("큐가 가득 찼습니다.");

}else{

queue[++rear] = item;

console.log("정상적으로 아이템이 삽입되었습니다.");

}

}

this.deque = ()=>{

if(isEmpty()){

return "Queue가 비어있습니다.";

}else{

console.log("정상적으로 아이템이 제거 되었습니다. ");

return queue[front++];

}

}

}

}

const queue = new QueType(3);

queue.enque(10);

queue.enque(20);

queue.enque(30);

queue.enque(40);

console.log(queue.deque());

console.log(queue.deque());

console.log(queue.deque());

console.log(queue.deque());

**큐의 활용사례**

Cpu의 연산 처리

통신의 대기열

프린터 출력 시 문서 순서

**원형큐**

지금 만든 큐는 한번 꽉차면 더 이상 재활용 할 수 없는 선형적인 구조를 띄고 있는데, 이를 선형 큐, 리니어 큐라고 합니다. 그래서 이 단점을 보완하고자 FIFO의 구조를 그대로 띄고, 재 사용이 가능한 원형큐라는 녀석이 만들어졌습니다.

이 원형큐는 나머지 연산을 통해서 rear와 프론트를 계산하여, size이상으로 넘어가지 않고, 0부터 다시 시작 할 수 있게 합니다. 다만 이 계산을 하려면 항상 프론트가 가리키고 있는 곳은 아이템을 비우고 있어야 합니다.

class QueType{

constructor(size){

let queue = [];

let front = 0;

let rear = 0;

const maxSize = size;

const isEmpty = ()=>{

return rear === front;

}

const isFull = ()=>{

return front === (rear +1) % maxSize;

}

this.enque = (item) => {

if(isFull()){

console.log("큐가 가득 찼습니다.");

}else{

rear = (rear + 1) % maxSize;

console.log(rear);

queue[rear] = item;

console.log("정상적으로 아이템이 삽입되었습니다.");

console.log(queue)

}

}

this.deque = ()=>{

if(isEmpty()){

return "Queue가 비어있습니다.";

}else{

front = (front + 1) % maxSize;

console.log("정상적으로 아이템이 제거 되었습니다. ");

return queue[front];

}

}

}

}

const queue = new QueType(3);

queue.enque(10);

queue.enque(20);

queue.enque(30);

queue.enque(40);

console.log(queue.deque());

console.log(queue.deque());

console.log(queue.deque());

console.log(queue.deque());