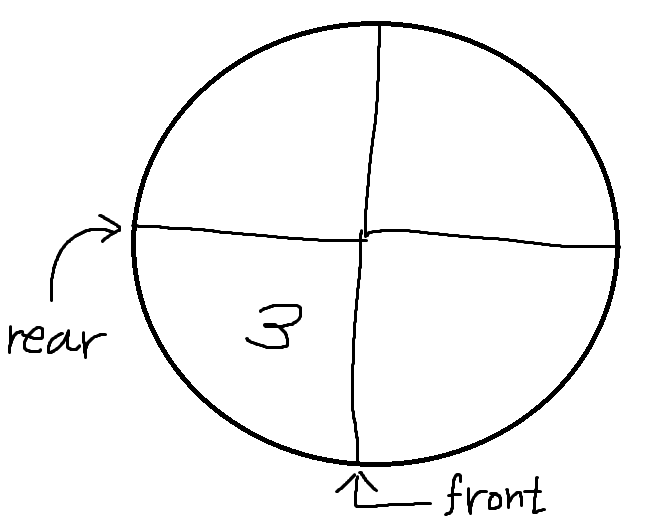
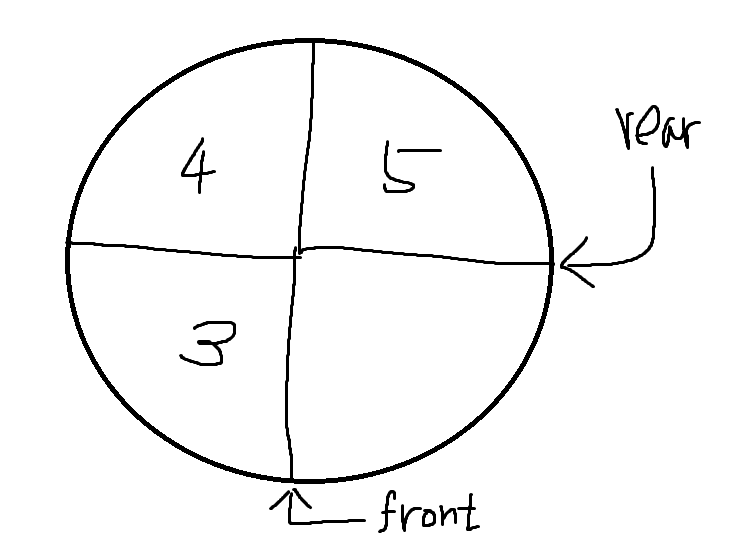


선형 큐 구조보다 어렵다. 위 그림에서 pop을 2번 하게 되면



이 상태에서 다시 push를 두번 하게 되면

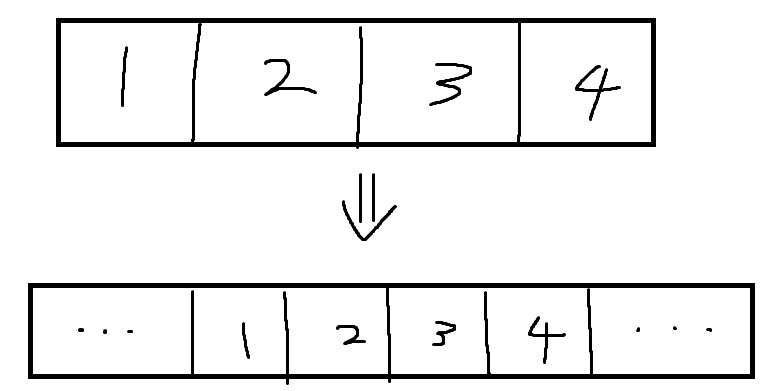


이렇게 front가 rear보다 앞에 있게되는 신기한 상황이 나온다.

보통 이 상황은 rear = (rear+1)%capacity; // capacity = 4; 를 이용해 처리한다.

중요한 것은 선형 큐에 비교할 때, 공간을 낭비하지 않는다는 점이다.

원형 큐의 핵심은 ‘공간을 어떻게 동적으로 늘리는가’ 이다. 아래 그림을 보자.



저번 선형 큐에서 공간을 늘릴 때처럼, 똑같이 capacity\*2를 해준 모습이다.

중요한 것은, 늘린 뒤 front가 어디에 있든 위 구조는 순환하게 코드를 짰다. 즉, front는 내가 정하기 나름이라는 것이다..!

정리하자면, 위 선형구조는 하나의 작업처럼 보이지만, 실은 두 가지의 작업으로 구별할 수 있다.

1) 정해진 용량 안에서 순환하며 front와 rear를 바꿀 것

2) capacity를 늘려야 할 때, 선형 큐처럼 늘린 후 front와 rear를 다시 잡을 것.

위 두개를 고려하며 코드를 짜면 어렵지않게 끝낼 수 있다.

**추가 고민)** empty와 full은 어떻게 구별해야 할까?

cq.begin() 🡪 front, cq.end() 🡪 rear라 하자,

열심히 원소를 추가해서 rear = (rear + 1)%capacity;를 capacity까지 실행한다면

capacity = 10일 때 front == 0; rear == 9 인 순간에서 고민을 해봐야 한다.

위 식대로면 rear 🡪 0이 되고 front == rear면서 array에 원소가 가득찬 상태가 된다.

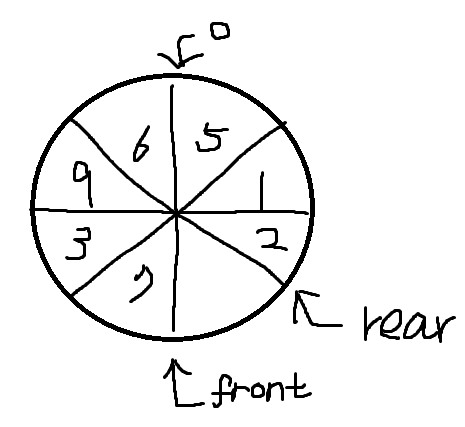
java같은거면 null check라도 하겠지만, 여긴 element가 초기화 된건지 아닌지 판단할 방법이 없다.

해결책 🡪 capacity의 마지막 부분을 포기한다!

만약 front == (rear+1)%capacity; 를 진행해서 true가 나오면 capacity 늘리기 작업을 진행하는 것이다.

**추가 고민)** capacity를 동적으로 늘린 후, 원소들을 어떤 방식으로 옮겨야 할까?

이전 선형 큐에서 capacity를 늘릴 땐, 별 고민 없이 memcpy를 써서 원소들을 옮겼지만… 이번엔 생각이 살짝 필요하다. 아래 그림을 보자.



그림은 capacity = 8; rear = 3; front = 4; 인 상황이다.

생각없이 memcpy를 적용시키면 front부터 capacity를 뚫고 복사해버리는 대참사가 벌어진다.

memcpy를 적용시키되, [4] ~ [7] 까지 복사하고, [0] ~ [3]까지 다시 복사해주면 완벽하다..!