2025 겨울방학 알고리즘 스터디

큐,덱

컴퓨터 공학과 20230546 서보경

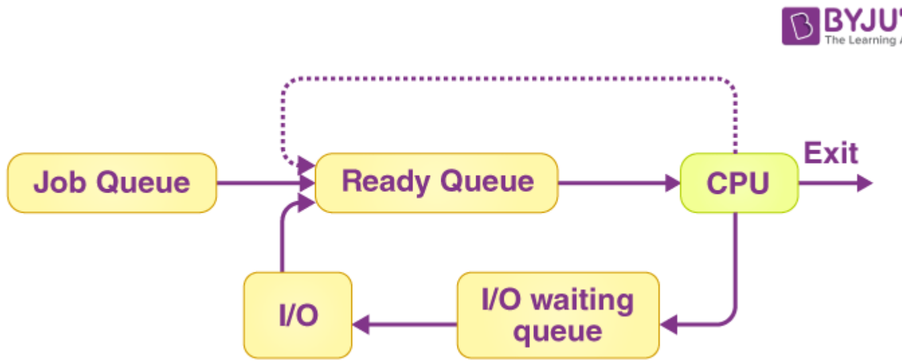
목차

1. 큐란?
2. 덱이란?
3. 원형 배열 다루기

큐란?

큐(Queue)는 선입선출(First In, First Out, FIFO) 원칙을 따르는 선형 자료구조이다. 선입선출이란 가장 먼저 추가된 데이터가 가장 먼저 제거되는 구조를 의미한다. 큐는 한쪽 끝에서는 데이터를 삽입하고, 반대쪽 끝에서는 데이터를 제거할 수 있다.

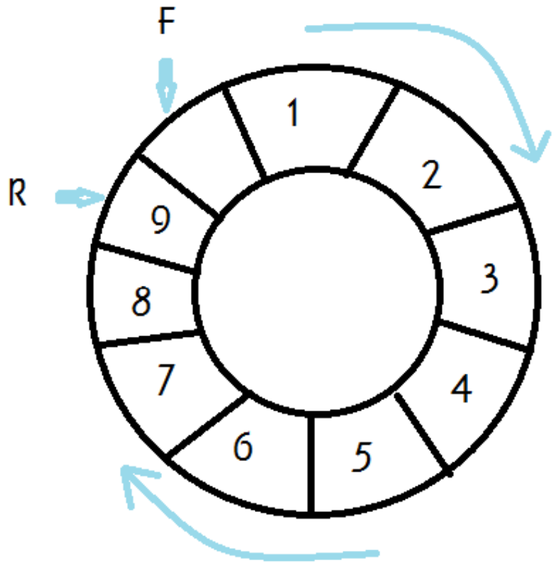
은행에서 먼저 온 사람부터 업무를 처리하는 걸 상상해보자.

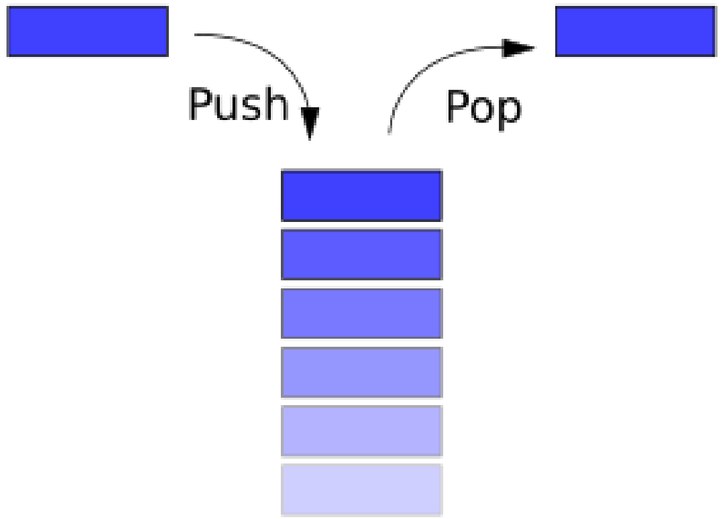


주로 운영체제의 스케줄링 기법에서 사용된다. (라운드 로빈)

덱이란?

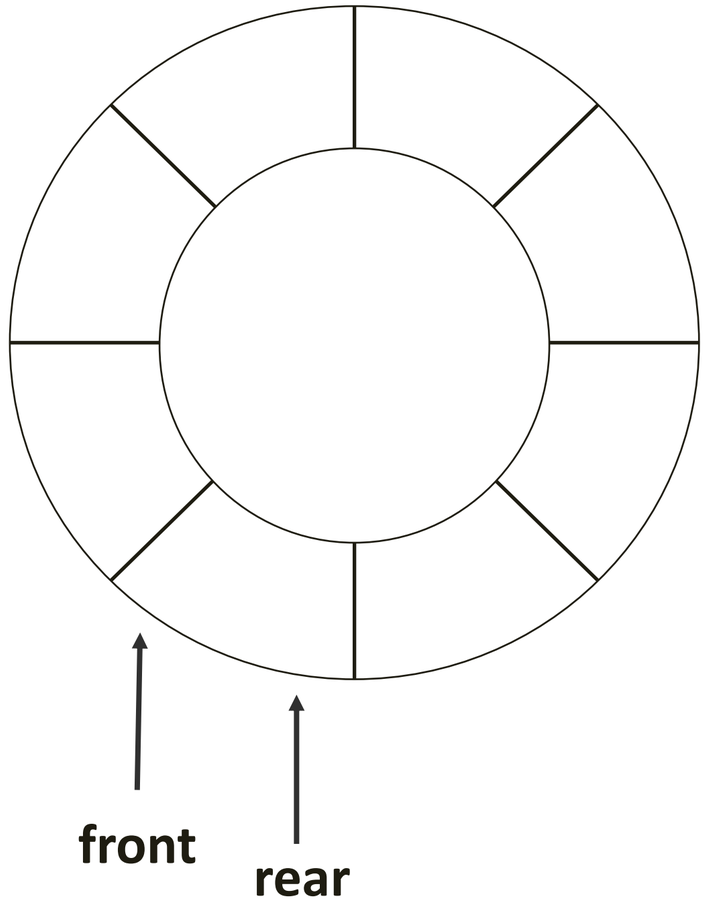
덱(Deque)은 양방향 큐(Double-Ended Queue)로, 양쪽 끝에서 데이터를 삽입하거나 제거할 수 있는 선형 자료구조이다. 덱은 데이터의 삽입과 제거가 앞쪽과 뒤쪽 양쪽 끝에서 모두 가능하다는 점에서 큐나 스택보다 유연하다.





Queue + Stack = Deque!

큐의 사용법

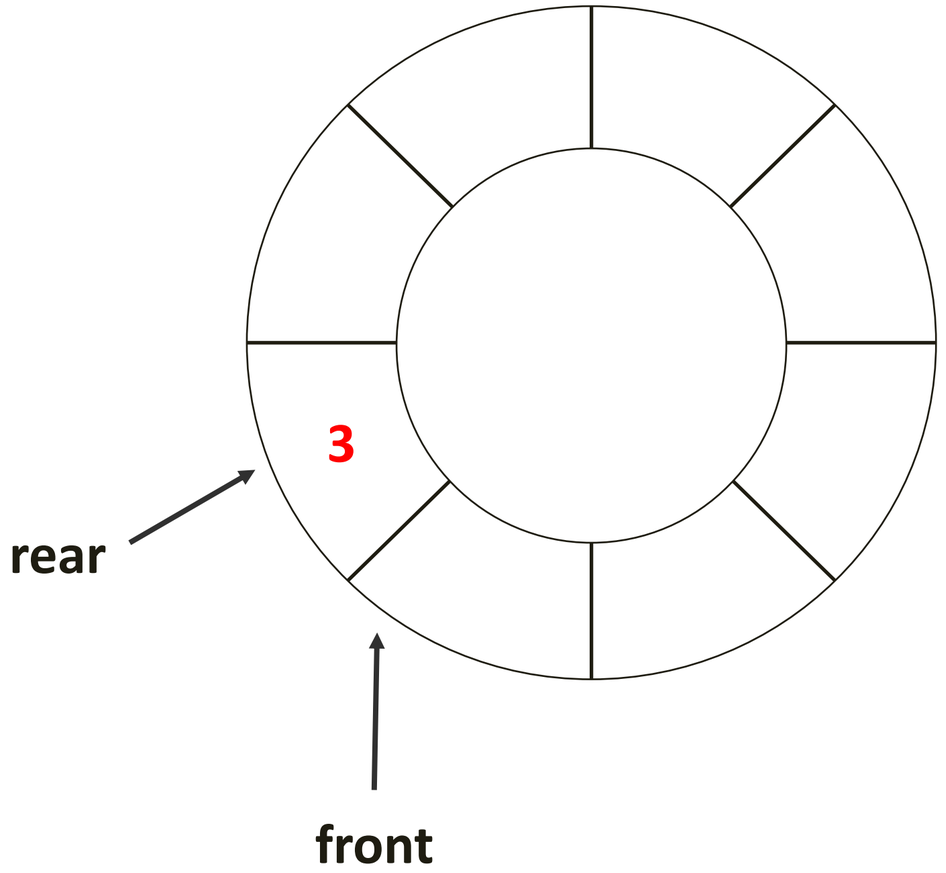


큐는 주로 배열을 통해 구현하게 된다.

이 때, 제일 앞 원소와 제일 뒤 원소를 가리키는 포인터도 따로 두는데, 각각 front, rear 라고 하겠다.

두 포인터를 모두 0으로 초기화 한다. 이는 큐가 비었다는 것을 의미하기 위해서이다.

큐의 사용법



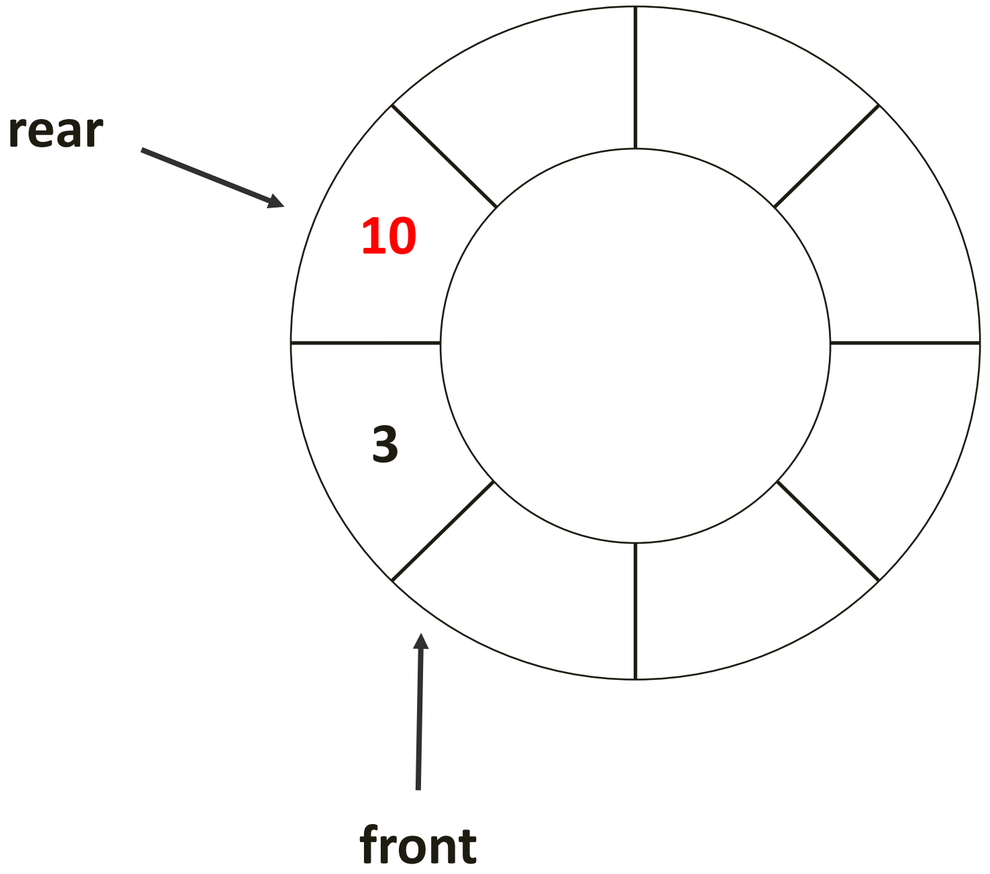
query : push 3

큐에 3를 넣는 연산이다.

현재, front 포인터는 0, rear 포인터는 0이므로 큐가 비었다는 걸 알 수 있다.

rear 포인터를 '먼저' 하나 늘리고 그곳에 3을 집어 넣는다.

큐의 사용법



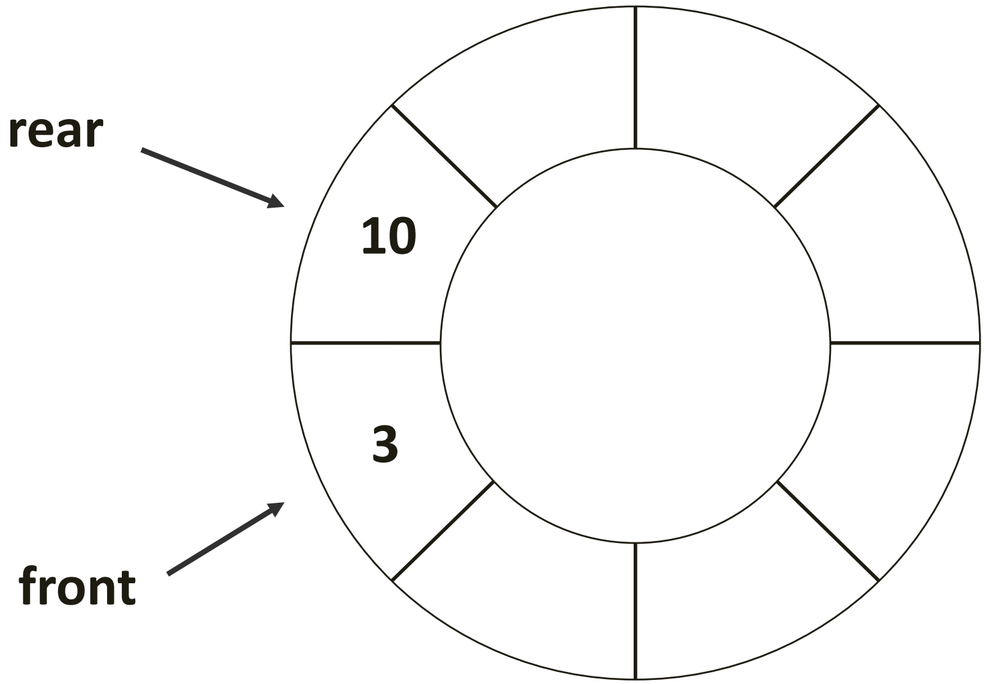
query : push 10

큐에 10를 넣는 연산이다.

현재 rear + 1이 front가 아니므로 큐가 가득 차지 않았다. (포인터로 empty와 full 구분)

rear 포인터를 '먼저' 하나 늘리고 그곳에 10을 집어 넣는다.

큐의 사용법



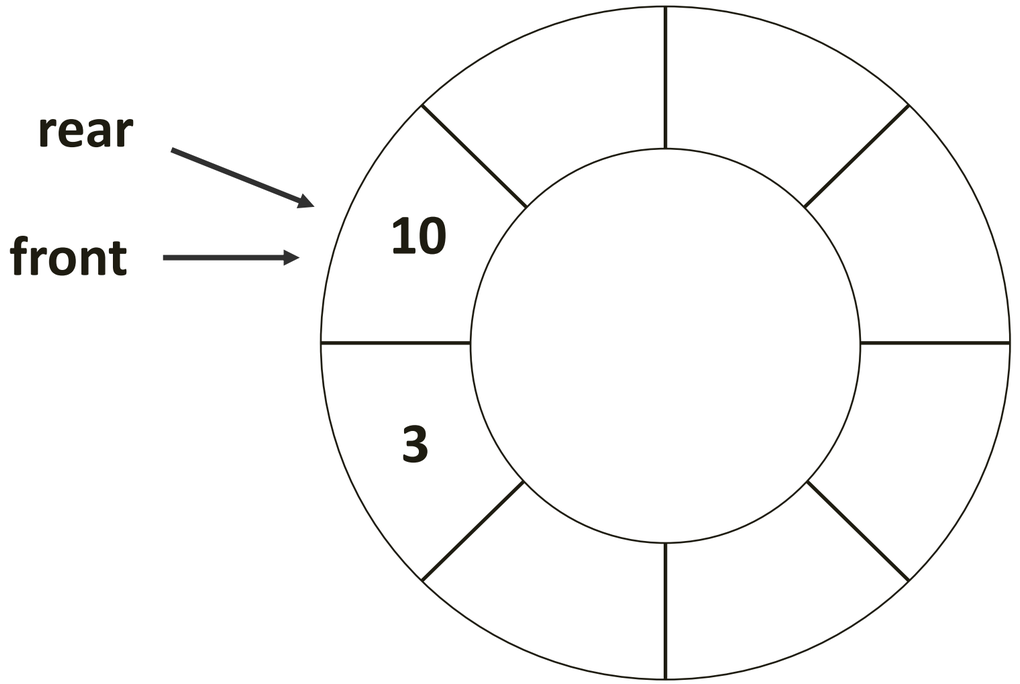
query : pop

큐에서 제일 앞에 있는 원소를 지우는 연산이다.

현재 front와 rear가 다르므로 큐가 비지 않았다. (2 , 0)

front 포인터를 늘린다. (스택과 같이 실제로 지우진 않지만 포인터로 지워졌다고 표현)

큐의 사용법



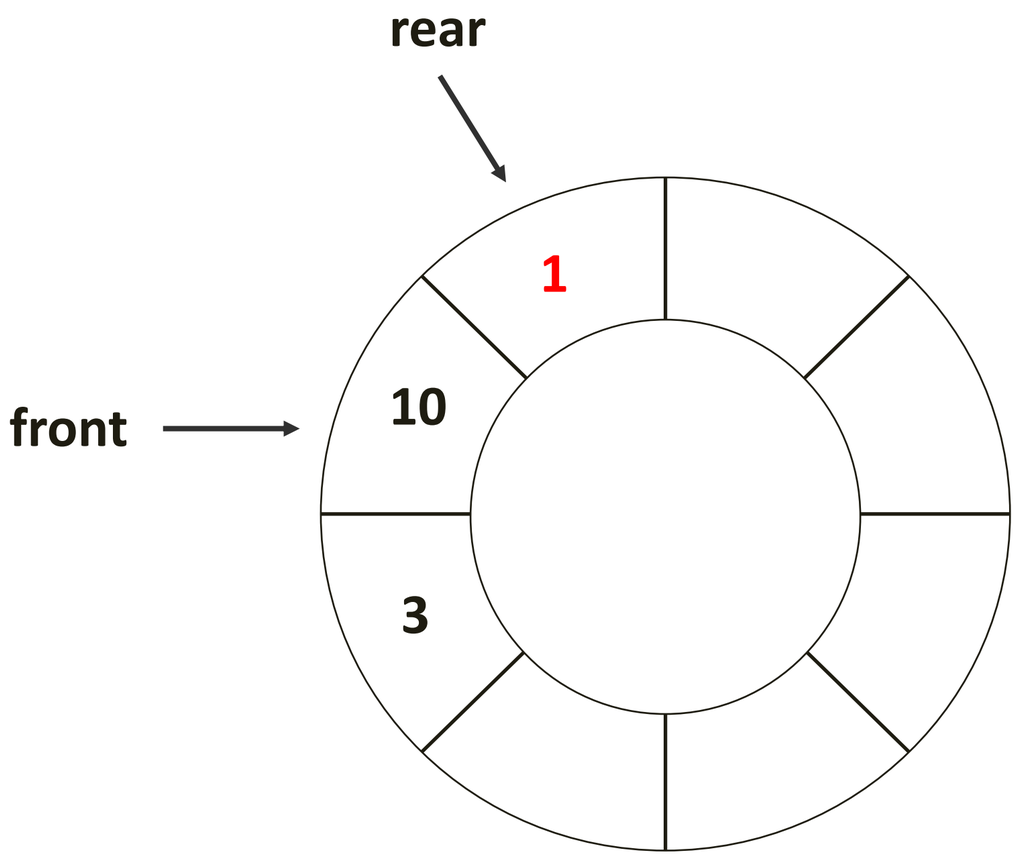
query : pop

큐에서 제일 앞에 있는 원소를 지우는 연산이다.

현재 front와 rear가 다르므로 큐가 비지 않았다. (2 , 1)

front 포인터를 늘린다. (스택과 같이 실제로 지우진 않지만 포인터로 지워졌다고 표현)

큐의 사용법



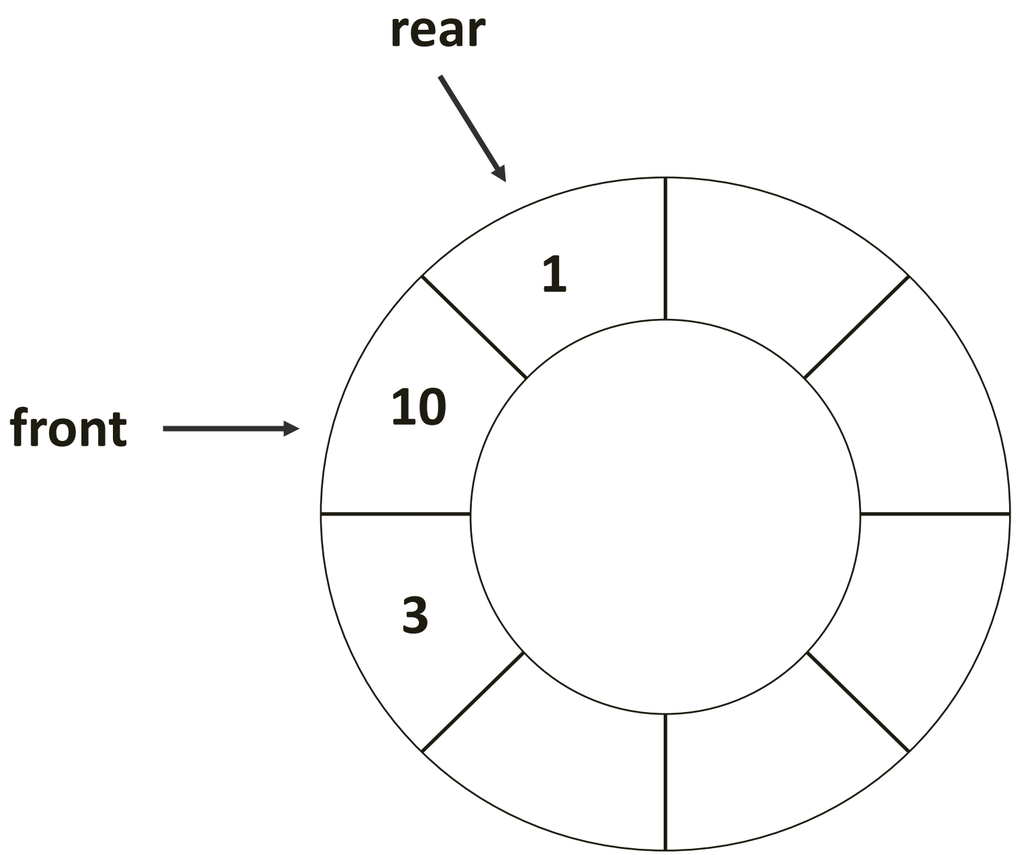
query : push 1

큐에 1을 넣는 연산이다.

현재 rear + 1이 front가 아니므로 큐가 가득 차지 않았다.

rear 포인터를 하나 늘리고 그곳에 1을 집어 넣는다.

큐의 사용법

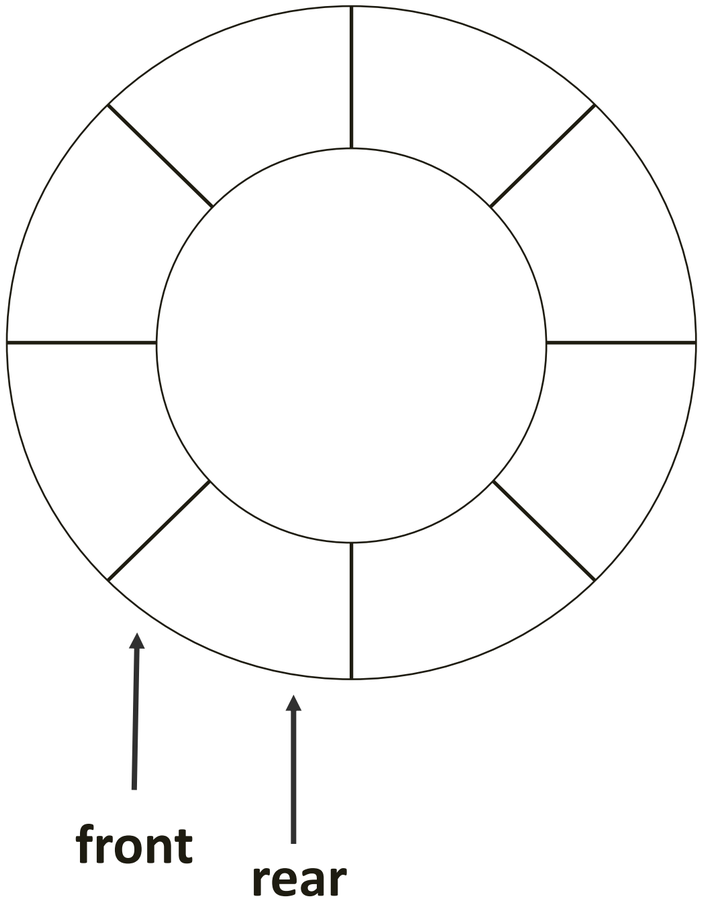


큐를 사용할 때 주의할 점

스택과 비슷하게 항상 포인터(front, rear)를 적절히 옮겨야 한다.
rear은 추가용, front는 삭제용이라고 생각하면 된다.

이때 구현과 이해의 용이성을 위해서 원형으로 구현하게 되는데 front == rear 이면 큐가 비었다는 것을 의미하고, (front + 1) % Queue_size == rear 이면 큐가 가득 찼다는 걸 의미한다.

덱의 사용법

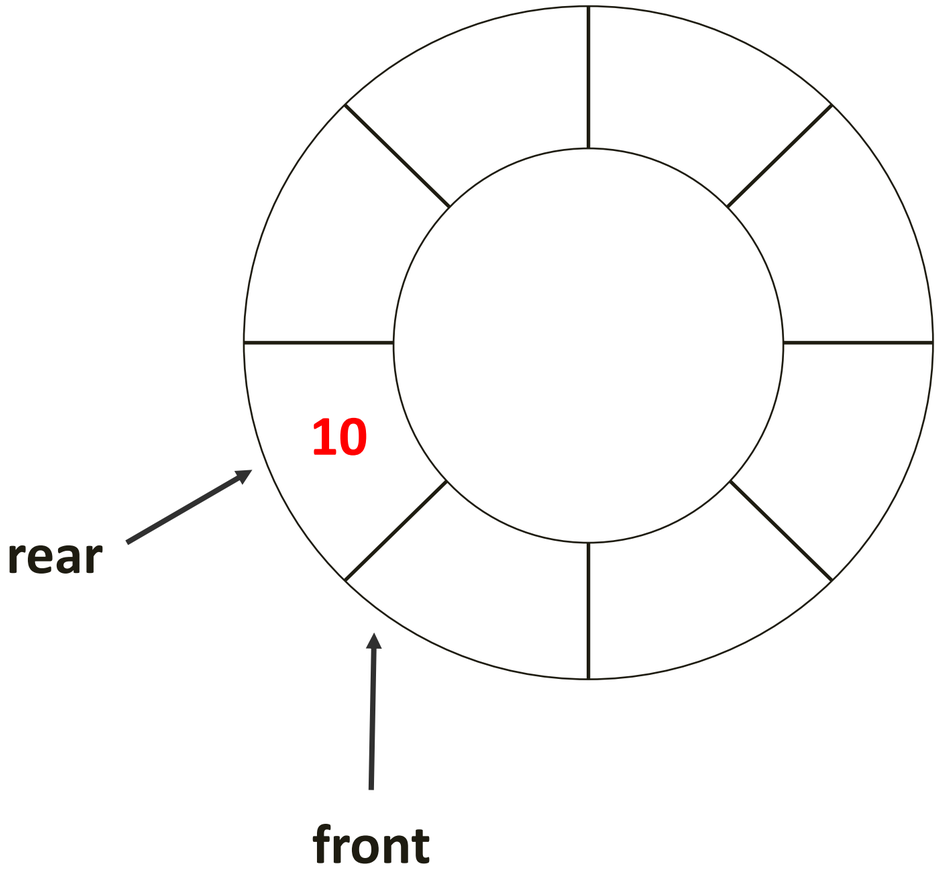


덱도 주로 배열을 통해 구현하게 된다.

큐와 같이 제일 앞 원소와 제일 뒤 원소를 가리키는 포인터인 front와 rear을 가진다.

덱이 비었다는걸 나타내기 위해서 front와 rear은 0으로 초기화 해준다.

덱의 사용법



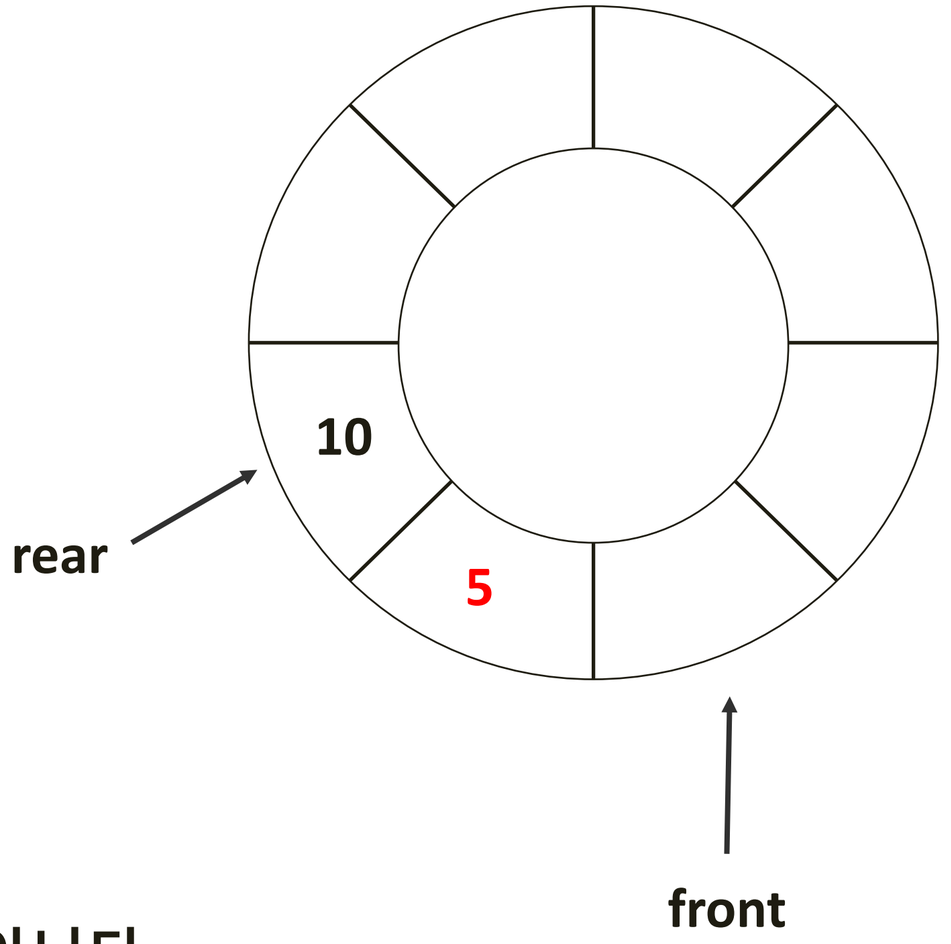
query : push_front 10

덱 앞쪽에 10을 넣는 연산이다.

현재 rear + 1이 front가 아니므로 덱이 전부 찬 상태가 아니다.

rear 포인터를 ‘먼저’ 늘리고 거기에 10을 넣어준다.

덱의 사용법



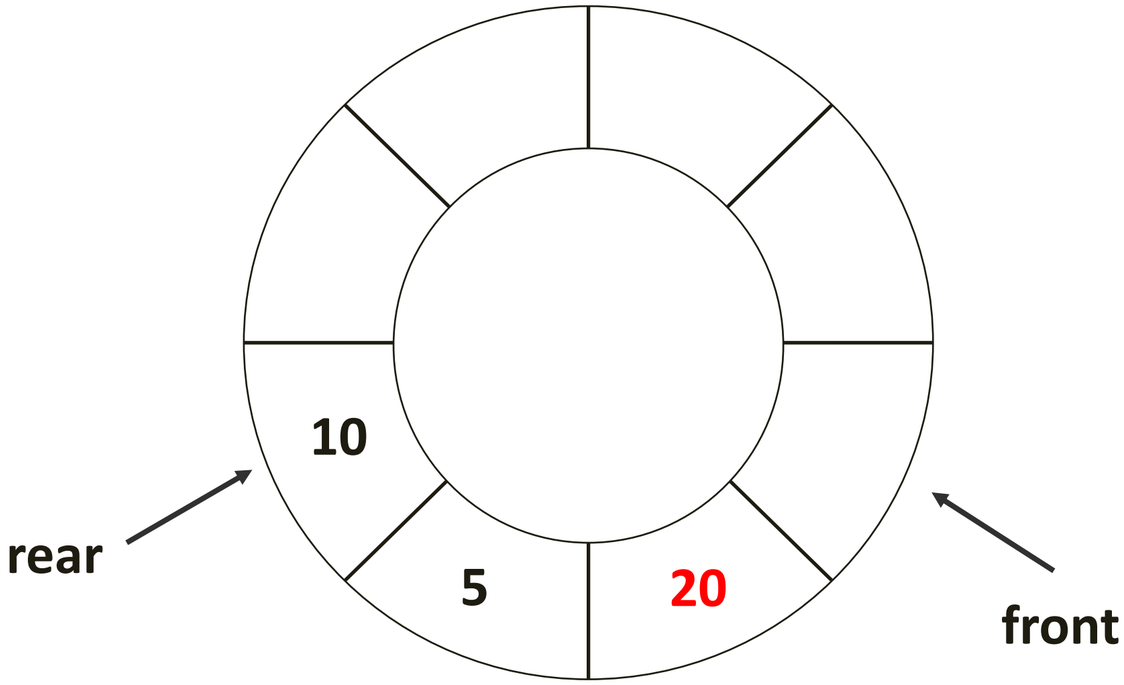
query : push_back 5

덱 뒤쪽에 5을 넣는 연산이다.

현재 rear + 1이 front가 아니므로 덱이 전부 찬 상태가 아니다.

front 위치에 '먼저' 5를 넣어주고, front 포인터를 하나 줄여준다. (front - 1)

덱의 사용법



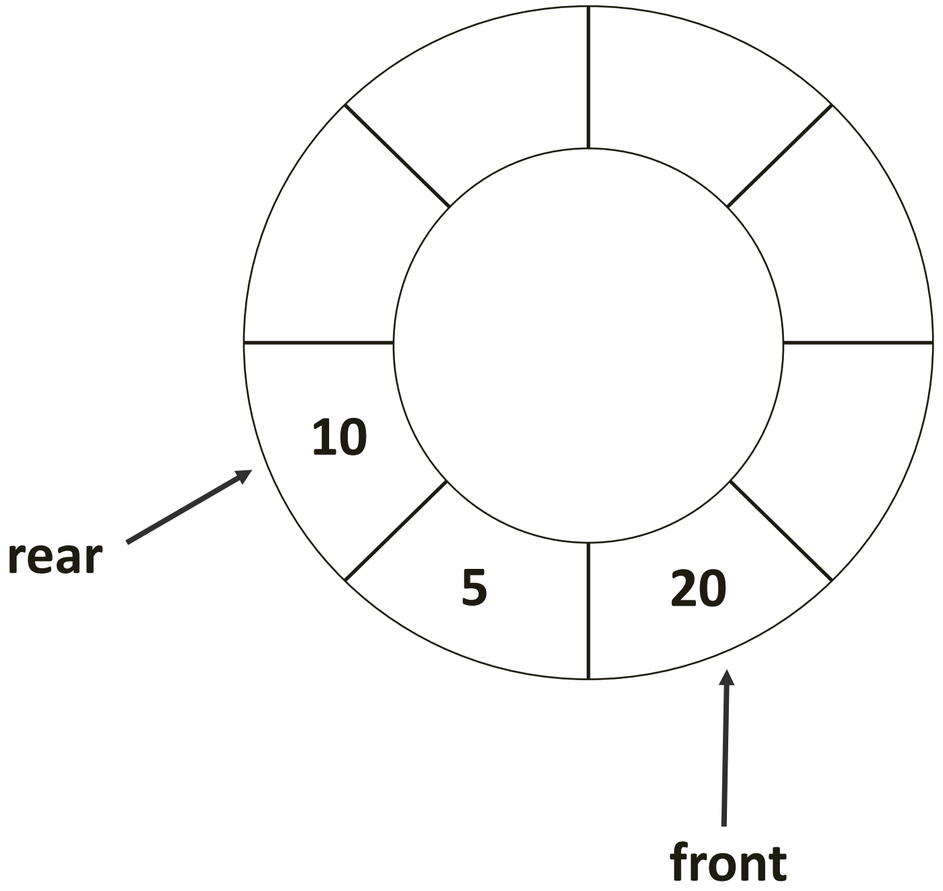
query : push_back 20

덱 뒤쪽에 20을 넣는 연산이다.

현재 rear + 1이 front가 아니므로 덱이 전부 찬 상태가 아니다.

front 위치에 '먼저' 5를 넣어주고, front 포인터를 하나 줄여준다.

덱의 사용법



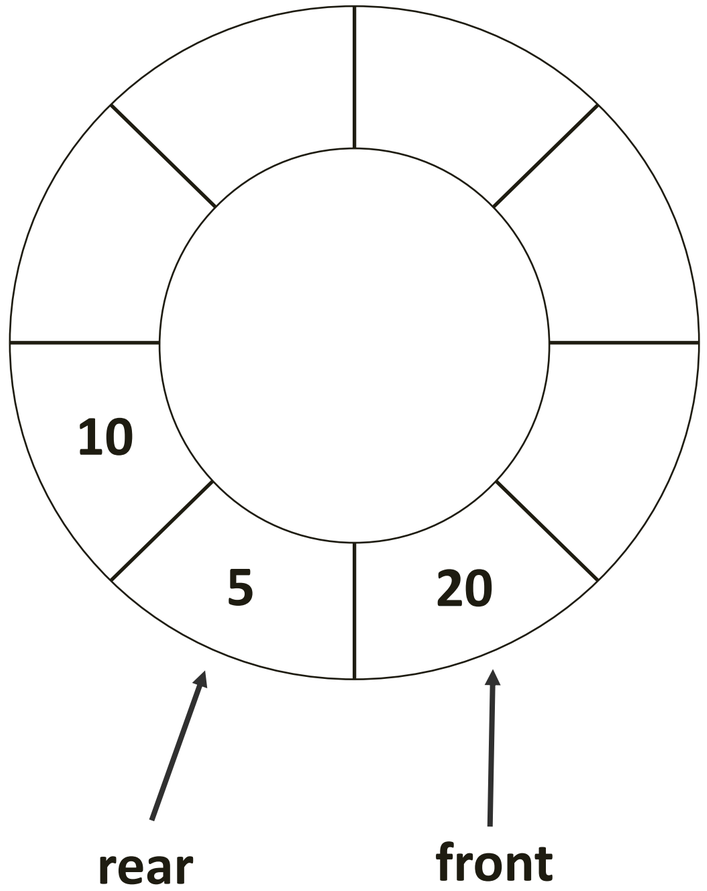
query : pop_front

덱에서 제일 앞에 있는 원소를 지우는 연산이다.

현재 front와 rear가 다르므로 덱이 비지 않았다. (1, 6)

front 포인터를 늘린다. (동일하게 실제로 지우진 않고 포인터만 옮기기)

덱의 사용법



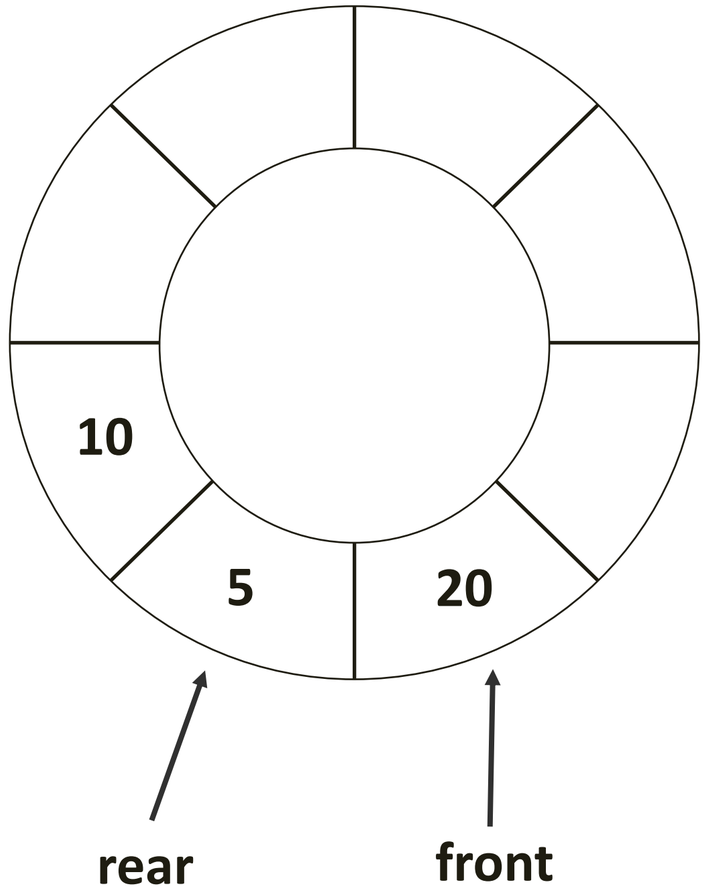
query : pop_back

덱에서 제일 뒤에 있는 원소를 지우는 연산이다.

현재 front와 rear가 다르므로 덱이 비지 않았다. (1, 7)

rear 포인터를 줄인다. (front는 옮겨야 원소를, rear은 현재 위치에서 원소를 가진다는게 차이점)

덱의 사용법

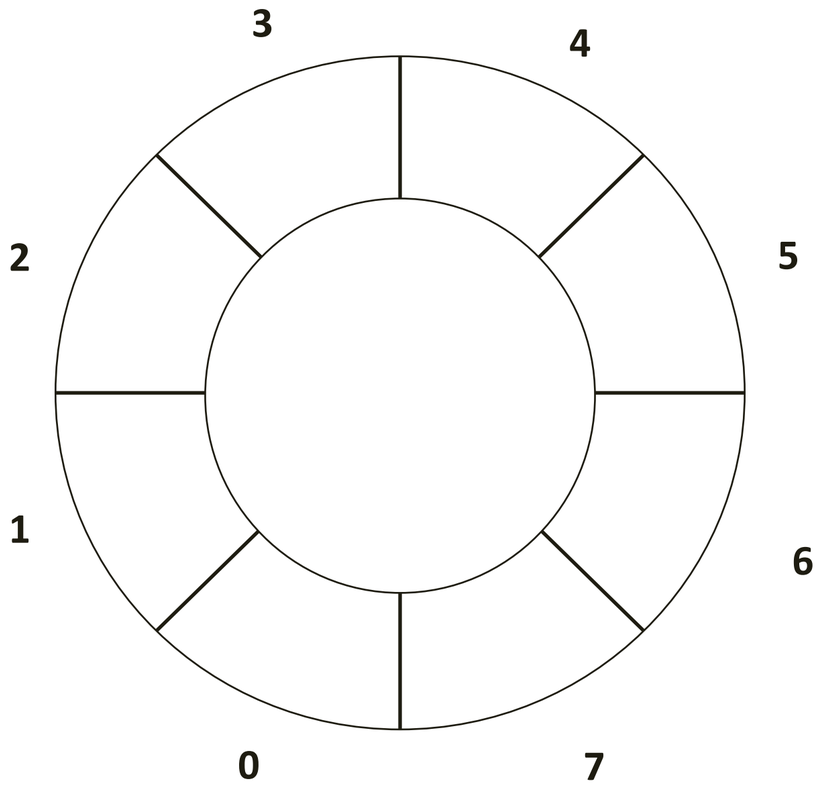


덱을 사용할때 주의점

일단 기본적으로 큐와 비슷한 성질을 가진다.

그러나 포인터를 양의 방향으로 옮기던 큐와 달리 덱은 음의 방향으로 움직이기도 하기 때문에 원형 배열에 대한 이해가 필요하다. 또한, rear 포인터에서는 현재 위치에 원소가 있고 front 포인터에서는 다음 위치에 원소를 가진다는 점을 유의해서 pop 연산을 해야한다. (C언어 기준)

원형 배열 다루는 법



모두가 알다 싶이, 원형 배열이라는 구조는 우리가 편하게 쓰기 위해서 만들어낸 환상의 배열이다.

우리의 환상으로 만들어 냈지만, 실제로 많은 부분에서 원형 배열을 이용하곤 한다.
어떻게 구현할까?

나머지 연산 이용하기

배열의 최대 크기를 이용한 나머지 연산으로 끝에서 처음으로 가는 구조를 만들 수 있다!
오른쪽의 원형 배열(큐)를 예시로 들자면, 7에서 한 칸 더 가면 원래 배열에선 8이지만,
배열의 사이즈인 8만큼 나눠주게 된다면 0으로 다시 돌아온다!

ex) (7 + 1) % 8 = 0 | pos = (cur + 1) % SIZE

그럼 반대 방향으로 돌아가는 것도 유효할까?

아쉽지만 반대로 돌아가는건 조금 더 정밀한 조정이 필요하다.
가령, 현재 인덱스가 0(처음)이고, 7(끝)으로 돌아가고자 한다. 여기서 한 칸 뒤로 가는 순간
(0 -1) % 8 = -1 이 되어버려서 음수 값이 되어버린다.
나머지 연산에서는 이런 경우를 방지하기 위해서 제수를 더하고 나머지 연산을 하는 형태를
다룬다.

ex) (0 – 1 + 8) % 8 = 7 | pos = (cur – 1 + SIZE) % SIZE

* 원형 배열을 사용하기 위해선, 배열을 0-based로 쓰는 편이 좋다. *
ex) 1-based 배열이고 size가 3이며 현재 위치가 4인경우
pos = (3 + 1) % 4 = 0
분명 4로 가야 하는데 0으로 돌아와 버린다. 심지어 0은 유효하지 않은 위치이다!