데이터구조 3장

03-1. 큐개념과 연산

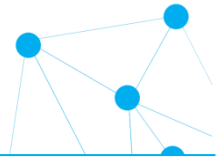
큐(QUEUE)

- 큐 : 먼저 들어온 데이터가 먼저 나가는 자료구조
- **선입선출(FIFO: First-In First-Out)**
  - 가장 먼저 들어온 데이터가 가장 먼저 나감

  (예) 매표소의 대기열

큐는 FIFO 구조'의 자료구조이다. 때문에 먼저 들어간 것이 먼저 나오는, 일종의 줄서기에 비유할 수 있는 자료구조이다.

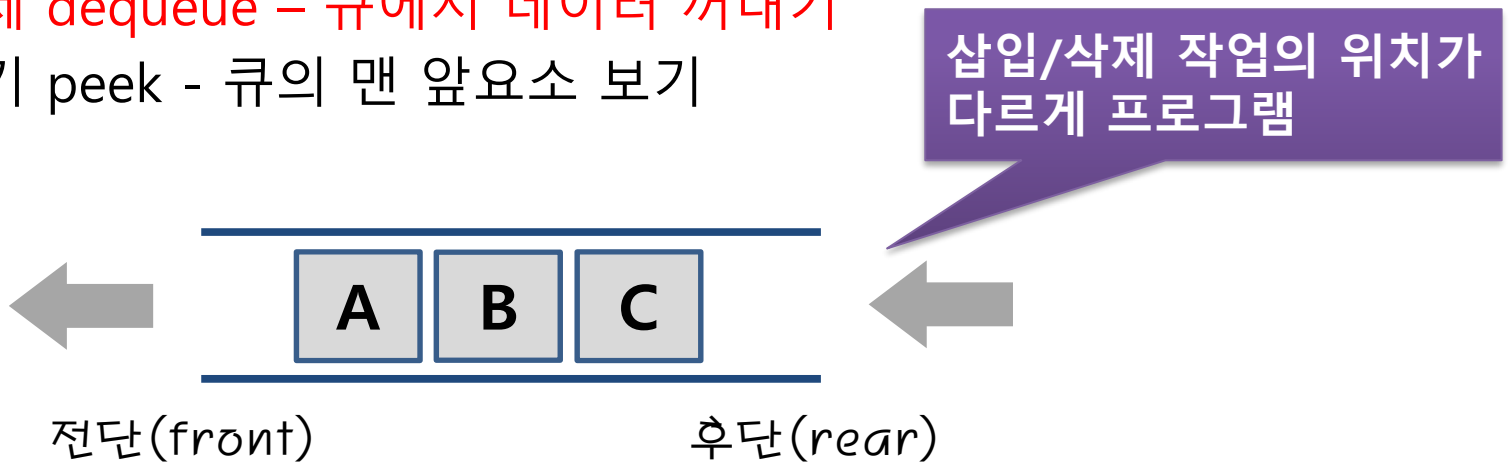
큐의 용도



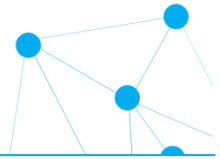
- 컴퓨터 장치들 간의 데이터 송수신용 버퍼
  - 키보드→컴퓨터, 프린터→컴퓨터(네트워크프린터 출력) 등
- 컴퓨터 데이터 통신 패킷 처리
- 컴퓨터를 이용한 시뮬레이션(은행, 공항 등의 대기열)
- 운영체제의 프로세스 관리(CPU 작업 스케줄링)
- 실시간 시스템의 인터럽트 처리
- 이진트리의 레벨순회

큐의 구조

- 이해가 필요한 용어
  - 전단 : front (머리) → 데이터가 큐에서 삭제되는 위치
  - 후단 : rear (꼬리) → 데이터가 큐에 삽입되는 위치
  - 요소, 항목
  - 공백상태, 포화상태
  - 삽입 enqueue – 큐에 데이터 넣기
  - 삭제 dequeue – 큐에서 데이터 꺼내기
  - 보기 peek - 큐의 맨 앞요소 보기



큐의 추상 자료형



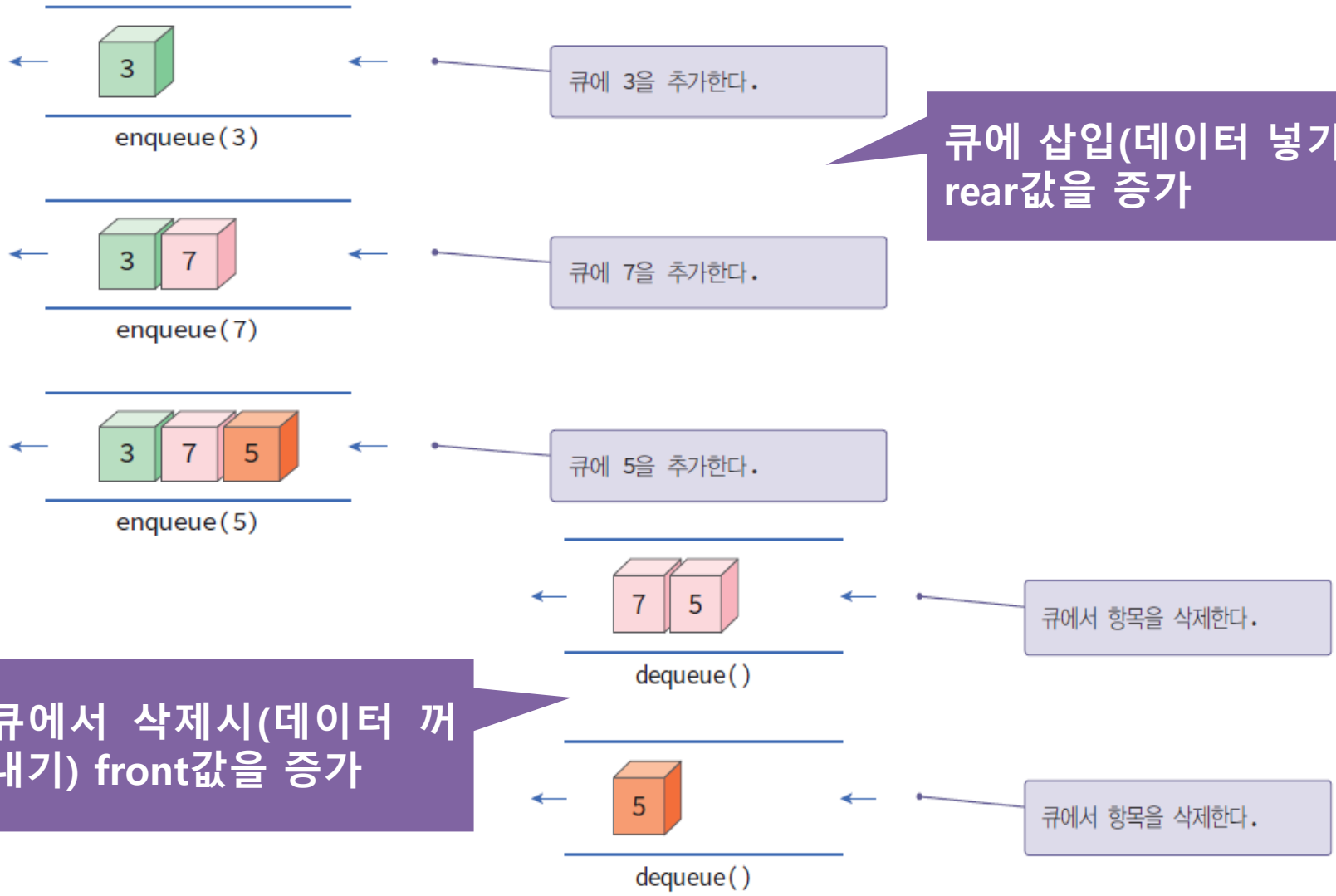
- **QUEUE ADT**
  - 삽입과 삭제는 FIFO순서를 따른다.
  - 삽입은 큐의 후단(꼬리)에서, 삭제는 전단(머리)에서 이루어진다.

데이터: 선입선출(FIFO)의 접근 방법을 유지하는 요소들의 모음

연산:

- enqueue(q,e): 주어진 요소 e를 큐의 맨 뒤에 추가한다.
- dequeue(q): 큐가 비어있지 않으면 맨 앞 요소를 삭제하고 반환한다.
- is_empty(q): 큐가 비어있으면 true를 아니면 false를 반환한다.
- peek(q): 큐가 비어있지 않으면 맨 앞 요소를 삭제하지 않고 반환한다.
- is_full(q): 큐가 가득 차 있으면 true을 아니면 false을 반환한다.

큐의 기본 연산

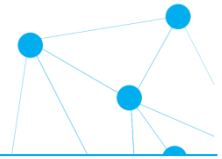


큐에 삽입(데이터 넣기)시 rear값을 증가

큐에서 삭제시(데이터 꺼내기) front값을 증가

03-2. 다양한 큐

큐의 여러 형태



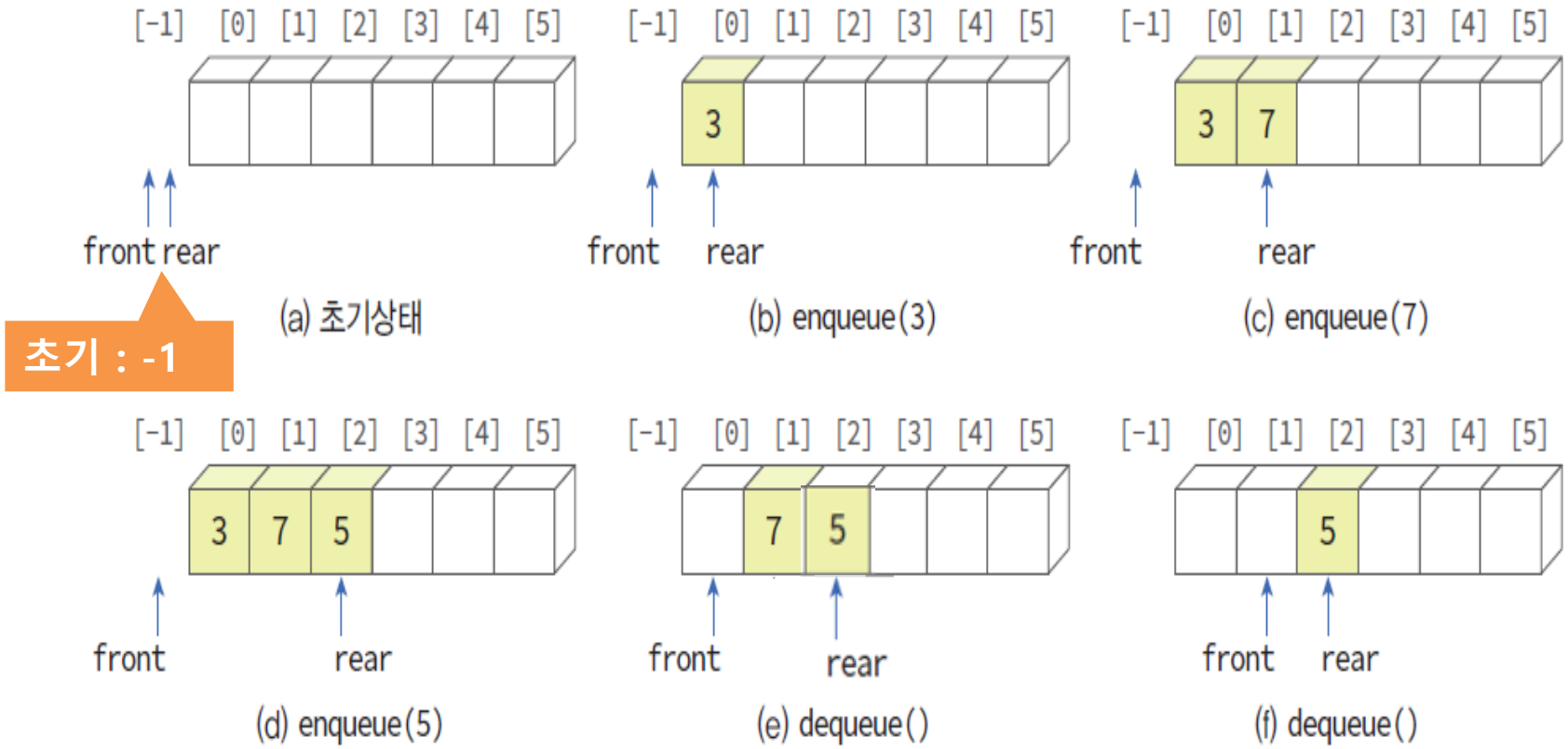
- **선형큐**
- **원형큐**
- **덱**

❖ 큐의 구현 방법

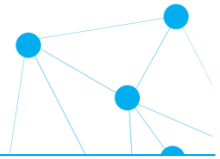
- 방법1 : 배열을 이용한 구현(수업시간에 다루는 범위)
- 방법2 : 연결리스트를 이용한 구현(연결리스트는 4장에서)

선형큐

- 배열을 선형으로 사용하여 큐를 구현



배열구현 선형큐의 연산



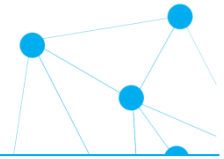
```
is_empty(q):

if front == rear then
    return TRUE
else
    return FALSE
```

```
is_full(q):

if rear == (MAX_QUEUE_SIZE-1) then
    return TRUE
else
    return FALSE
```

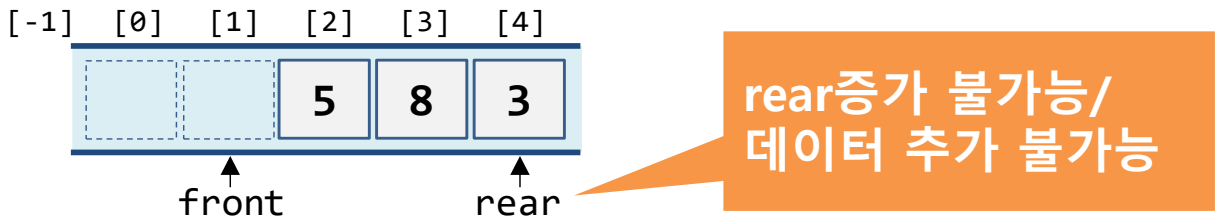
```
enqueue(q, x):

if is_full(q) then
    error " 포화상태"
else
    rear ← rear+1
    queue[rear] ← x
```

```
dequeue(q):

if is_empty(q) then
    error "공백상태"
else
    front ← front+1
    e ← queue[front]
    return e
```

선형큐 : 문제점

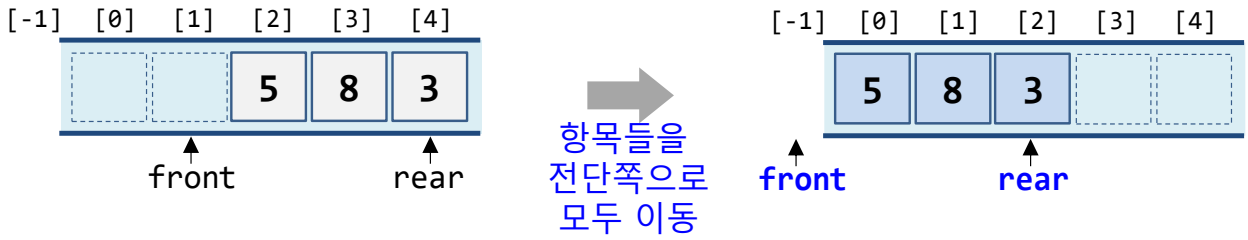


- dequeue 작업을 통해 큐가 비어 있음에도 새로운 데이터를 추가할 수 없다.



선형큐 : 문제점

- 해결?
  ① 데이터 자체를 이동 : 구현이 복잡

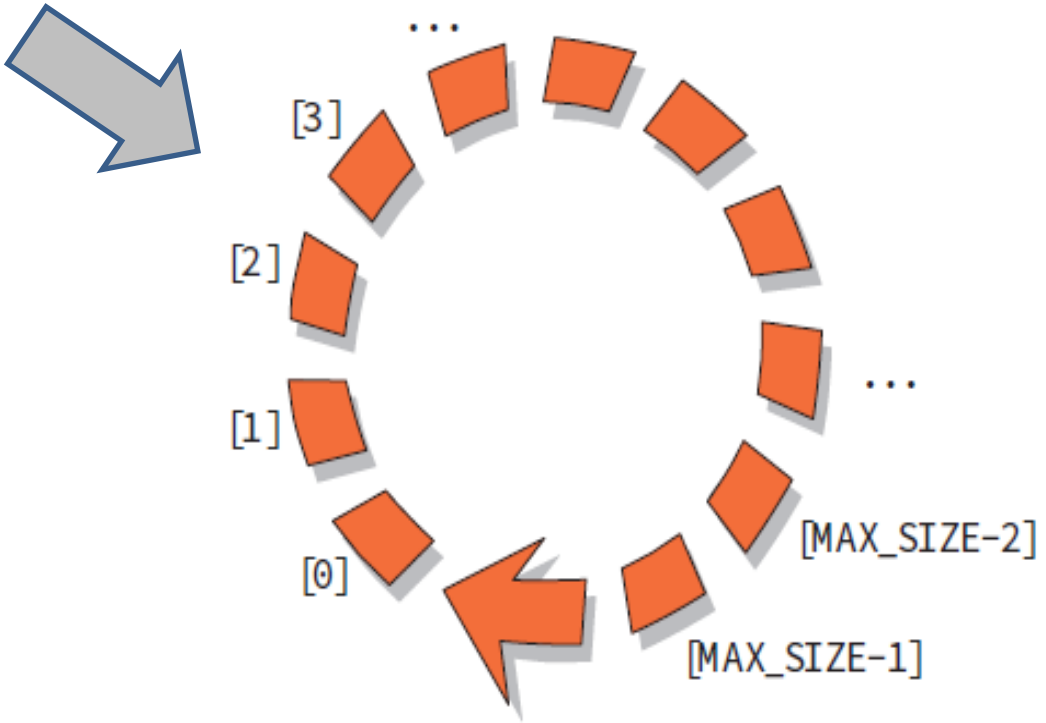


② 꼬리 rear를 인덱스 0인 위치로 이동 → 원형큐 구현

원형큐

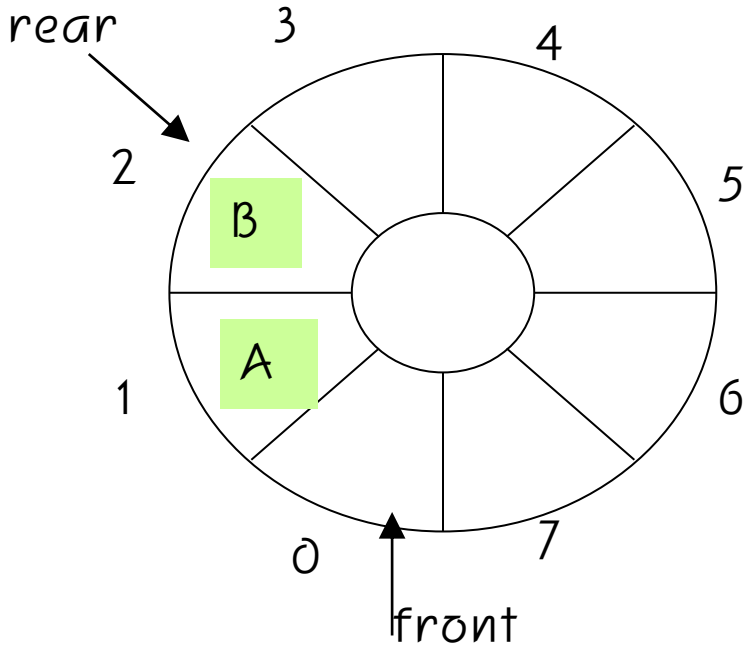
- 배열의 시작과 끝을 연결한 구조로 큐를 구현

[0] **[MAX_SIZE-1]**

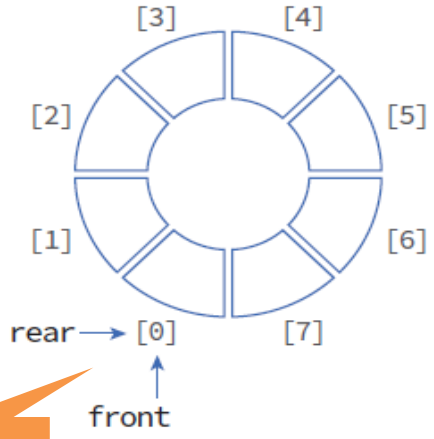


원형큐의 구조

- 큐의 전단과 후단을 관리하기 위한 2개의 변수 필요
  : (초기 위치가 선형큐와 다름에 주의)
  - front : 첫번째 요소 하나 앞의 인덱스
  - rear : 마지막 요소의 인덱스

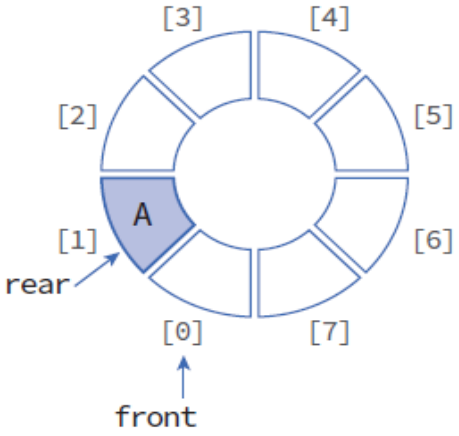


원형큐

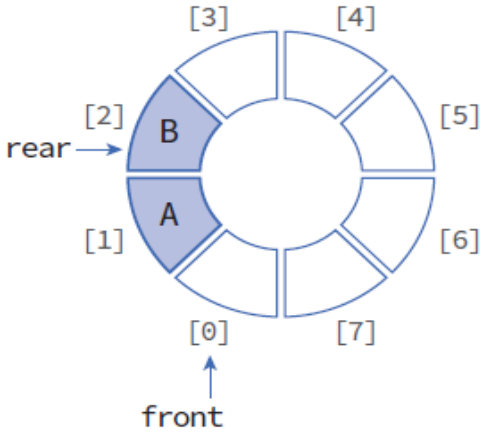


초기 : 0

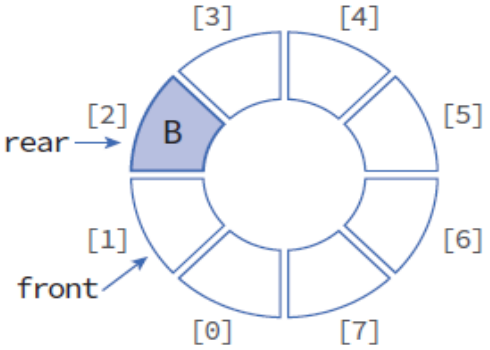
(a) 초기상태



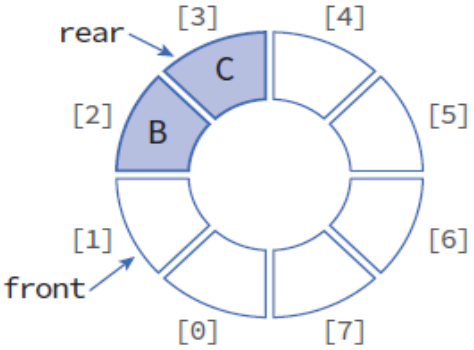
(b) A 삽입



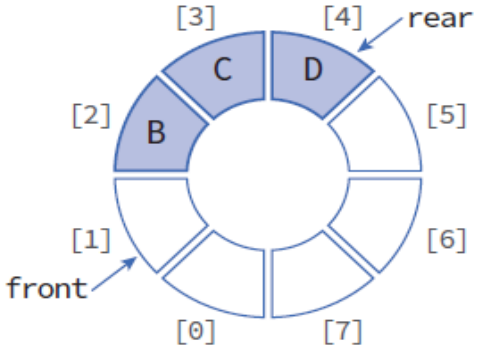
(c) B 삽입



(d) 삭제

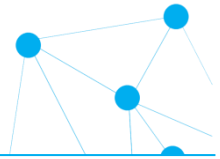


(e) C 삽입



(f) D 삽입

원형큐의 연산



```
enqueue(q, x):

if is_full(q) then
    error " 포화상태"
else
    rear ← (rear+1) mod MAX_QUEUE_SIZE
    data[rear] ← x
```

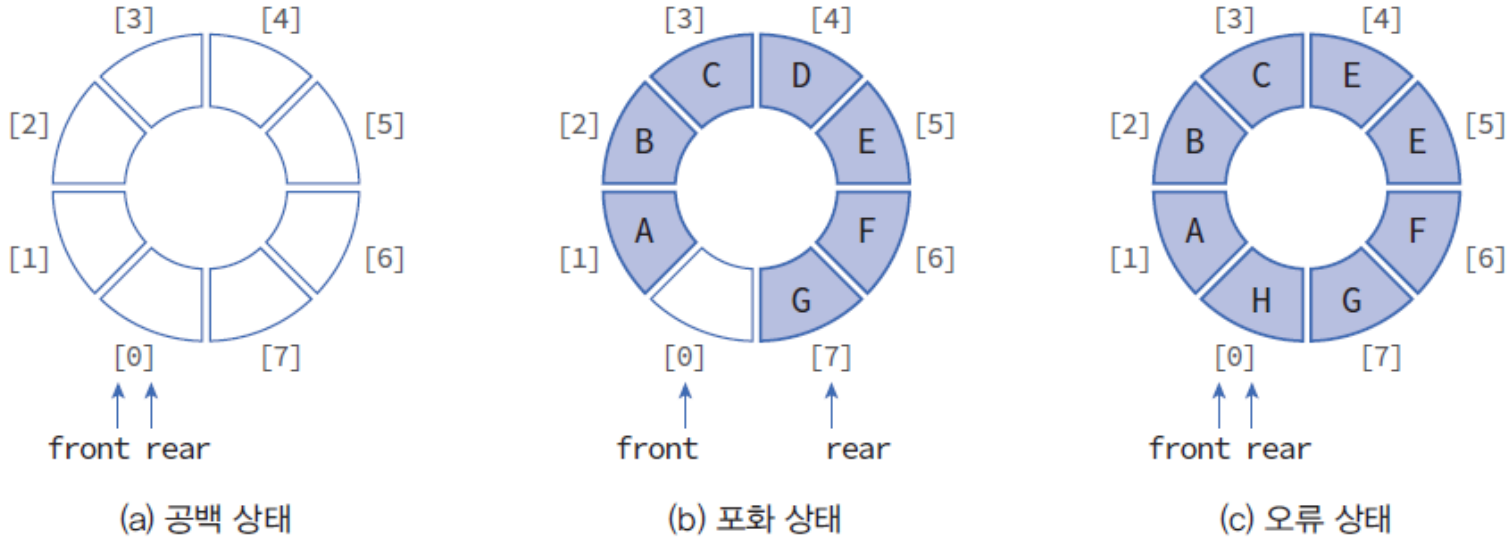
나머지(modulo) 연산을 사용하여 인덱스를 원형으로 회전시킨다.

```
dequeue(q):

if is_empty(q) then
    error "공백상태"
else
    front ← (front+1) mod MAX_QUEUE_SIZE;
    return data[front]
```

원형큐의 연산

```
is_empty(q):

return front == rear
```

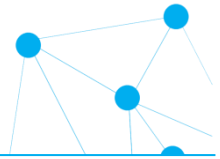
```
is_full(q):

return
   (rear+1)%MAX_QUEUE_SIZE == front
```



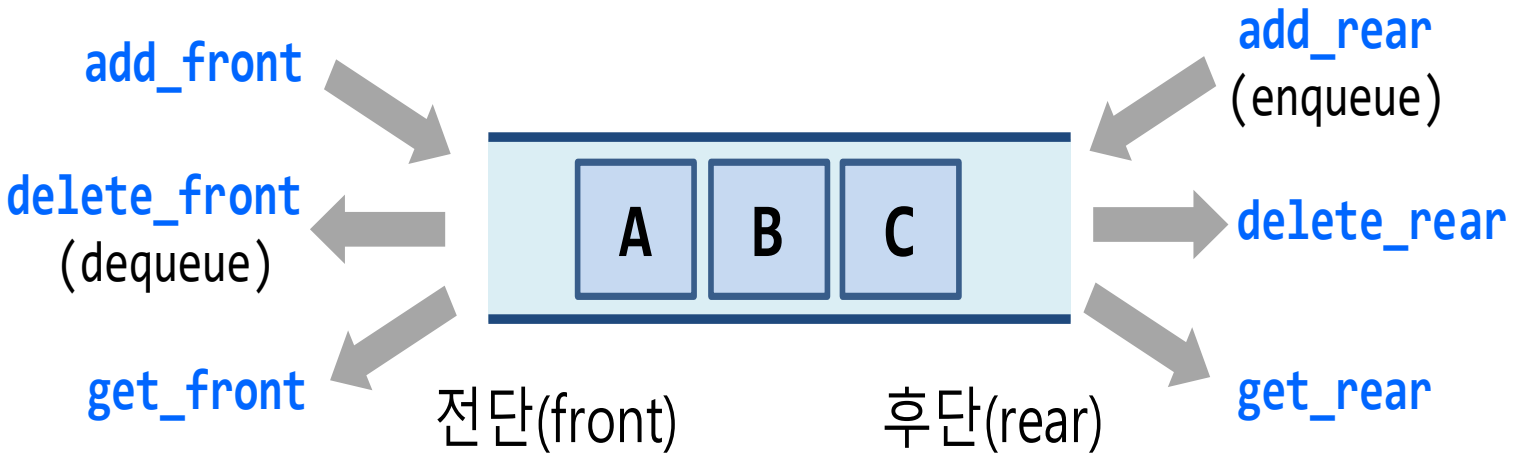
공백/포화 상태 구분을 위해 하나의 공간을 비워둠

03-3. 덱

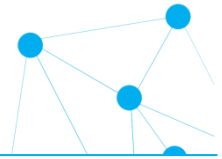
덱(데크, deque)



- 덱(deque)은 ***double-ended queue***의 줄임말
  - 전단(front)와 후단(rear)에서 모두 삽입과 삭제가 가능한 큐
  - 스택과 큐를 혼합한 형태 : 스택과 큐 동시구현에 사용



덱의 추상 자료형



- 큐와 데이터는 동일
  - 연산은 추가됨

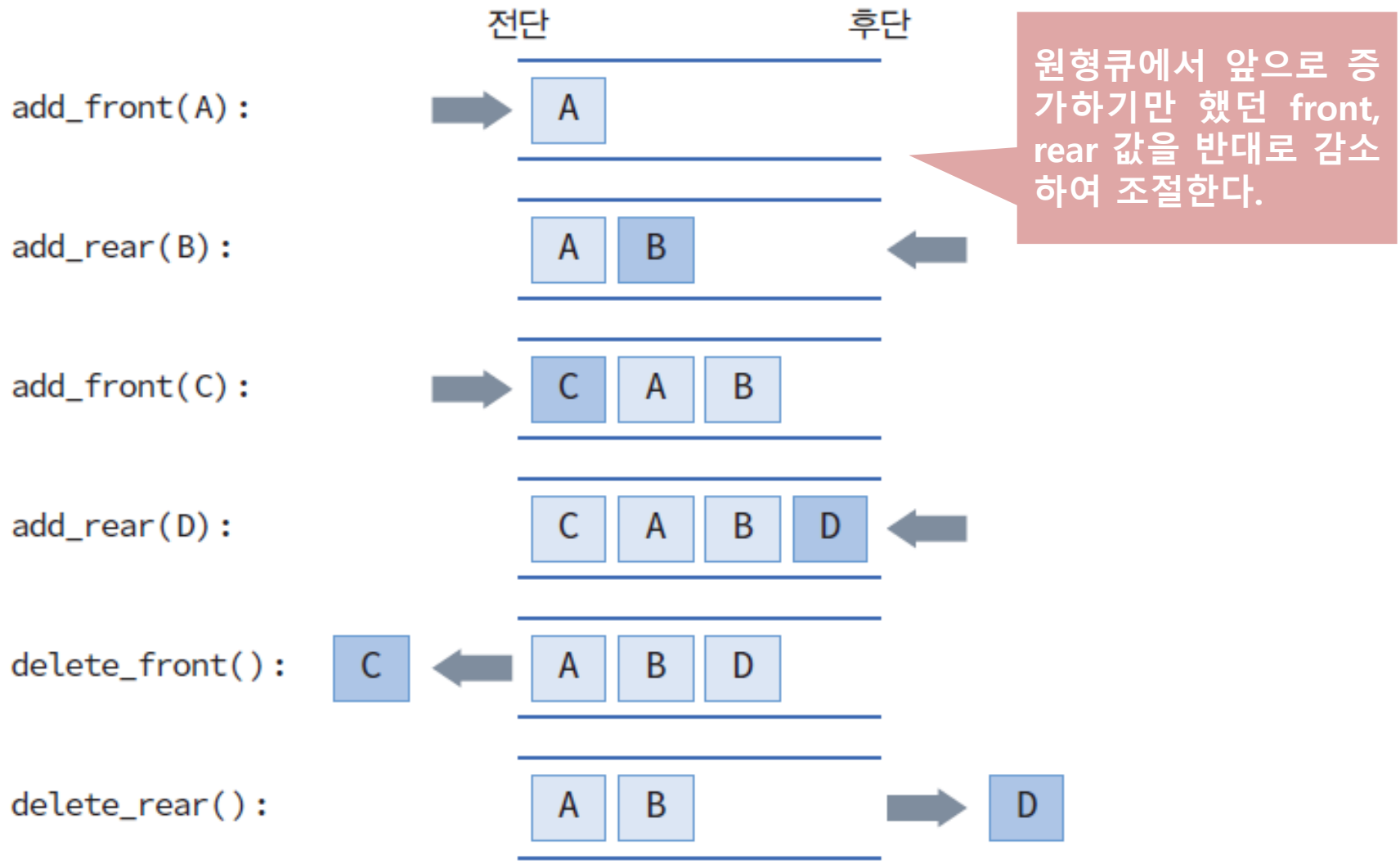
- **DEQUE의 ADT**

데이터: 전단과 후단을 통한 접근을 허용하는 요소들의 모음

연산:

- add_front(dq,e): 주어진 요소 e를 덱의 맨 앞에 추가한다.
- delete_front(dq): 전단 요소를 삭제하고 반환한다.
- add_rear(dq, e): 주어진 요소 e를 덱의 맨 뒤에 추가한다.
- delete_rear(dq): 후단 요소를 삭제하고 반환한다.
- is_empty(dq): 공백 상태이면 TRUE를 아니면 FALSE를 반환한다.
- is_full(dq): 덱이 가득 차 있으면 TRUE를 아니면 FALSE를 반환한다.

덱의 연산


전단
후단
add_front(A):
A
add_rear(B):
A
B
add_front(C):
C
A
B
add_rear(D):
C
A
B
D
delete_front():
C
A
B
D
delete_rear():
A
B
D
원형큐에서 앞으로 증가하기만 했던 front, rear 값을 반대로 감소하여 조절한다.

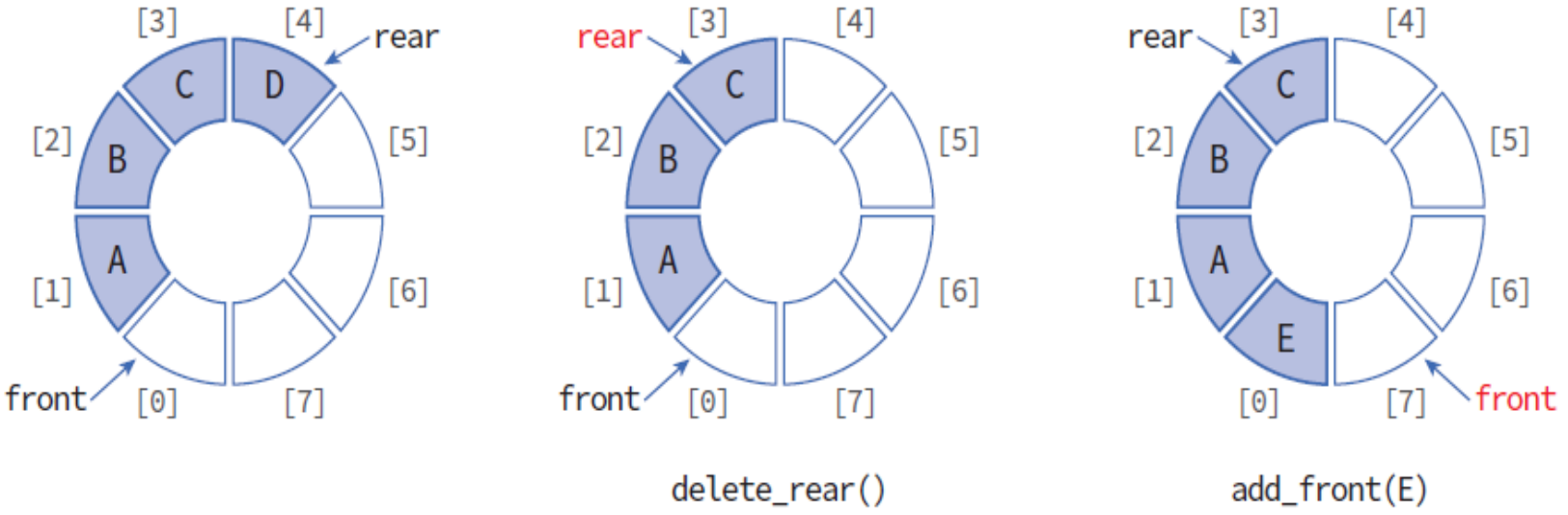
원형 덱의 연산

- 덱에서 추가된 연산
  - 반대방향의 회전이 필요

```
front ← (front-1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
rear ← (rear-1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
```

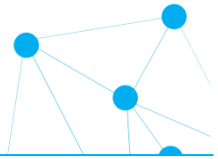
add_front()

delete_rear()

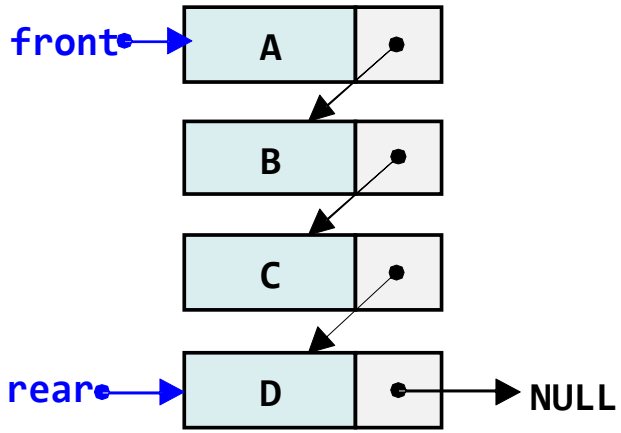


03-4. 큐 구현 및 응용

연결리스트를 이용한 큐의 구현



- 연결 리스트를 이용한 큐



연결 리스트를 이용한 큐

- 덱을 리스트로 구현할 경우 선행노드와 후속노드의 정보가 필요하므로 이중연결리스트로 구현

큐의 응용

- 서로 다른 속도로 실행되는 두 프로세스 간의 상호작용을 조화시키는 버퍼 역할 담당

큐의 응용 : 시뮬레이션

- 시스템의 특성을 시뮬레이션하여 분석하는데 이용
  - 고객에 대한 서비스를 수행하는 서버와 서비스를 받는 고객들로 이루어진 모델에서 고객들은 제한된 수의 서비스를 받기 위해 대기행렬(큐로 구현)에서 기다림