SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

APS 기본

SAMSUNG SW ACADEMY FOR YOUTH

APS(Algorithm Problem Solving)

스택 (Stack) & 큐 (Queue)

- 큐
- 선형큐
- 큐의 활용

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Queue

큐(Queue)의 특성

- 스택과 마찬가지로 삽입과 삭제의 위치가 제한적인 자료구조
  - 큐의 뒤에서는 삽입만 하고, 큐의 앞에서는 삭제만 이루어지는 구조

- 선입선출구조(FIFO : First In First Out)
  - 큐에 삽입한 순서대로 원소가 저장되어, 가장 먼저 삽입(First In)된 원소는 가장 먼저 삭제(First Out)된다.

큐의 예: 서비스 대기행렬

큐의 구조 및 기본연산

큐의 선입선출 구조

큐의 기본 연산

- 삽입 : enQueue
- 삭제 : deQueue

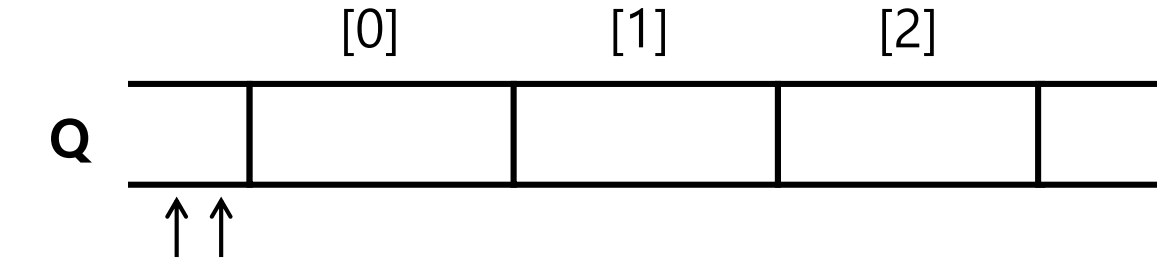
큐의 주요 연산

- 큐의 사용을 위해 필요한 주요 연산은 다음과 같음

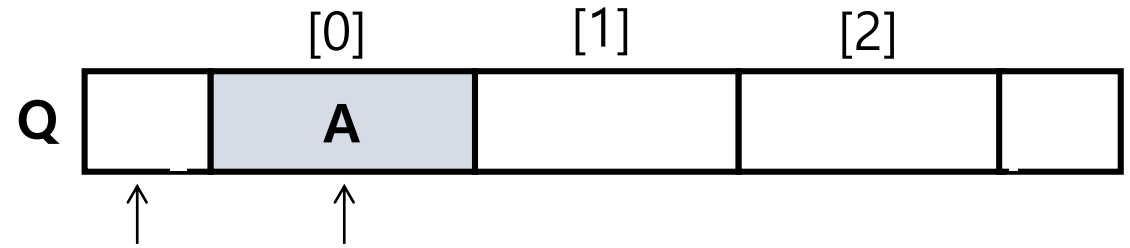
연산	기능
enQueue(item)	큐의 뒤쪽(rear 다음)에 원소를 삽입하는 연산
deQueue()	큐의 앞쪽(front)에서 원소를 삭제하고 반환하는 연산
createQueue()	공백 상태의 큐를 생성하는 연산
isEmpty()	큐가 공백상태인지를 확인하는 연산
isFull()	큐가 포화상태인지를 확인하는 연산
Qpeek()	큐의 앞쪽(front)에서 원소를 삭제 없이 반환하는 연산

큐의 연산 과정

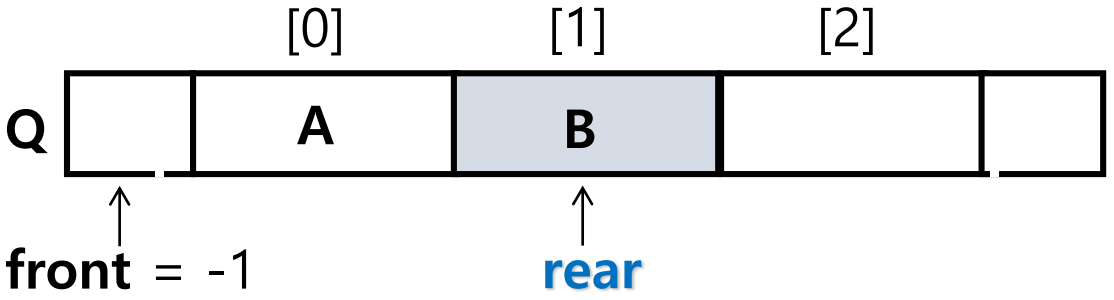
1) 공백 큐 생성 : createQueue();



2) 원소 A 삽입 : enQueue(A);

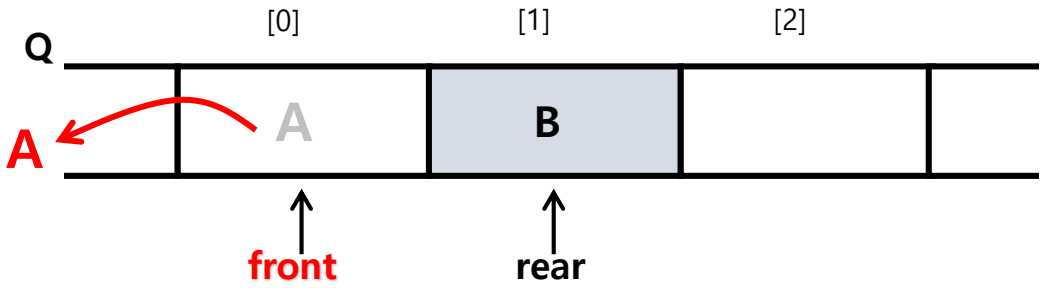


3) 원소 B 삽입 : enQueue(B);

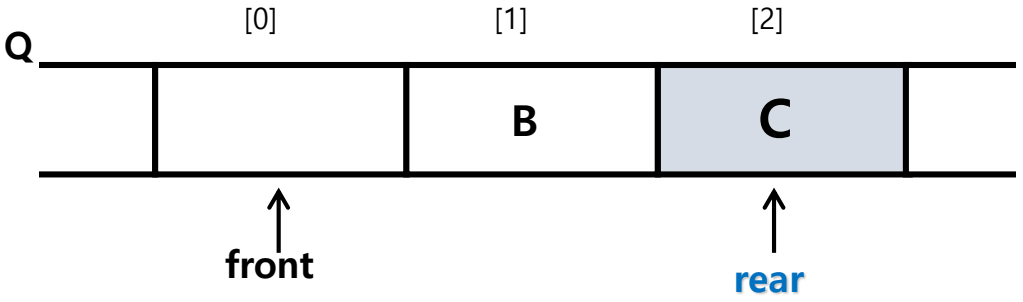


큐의 연산 과정

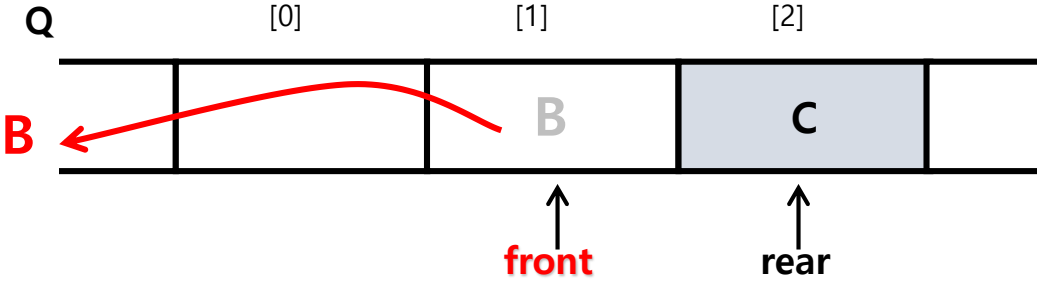
4) 원소 반환/삭제 : deQueue();



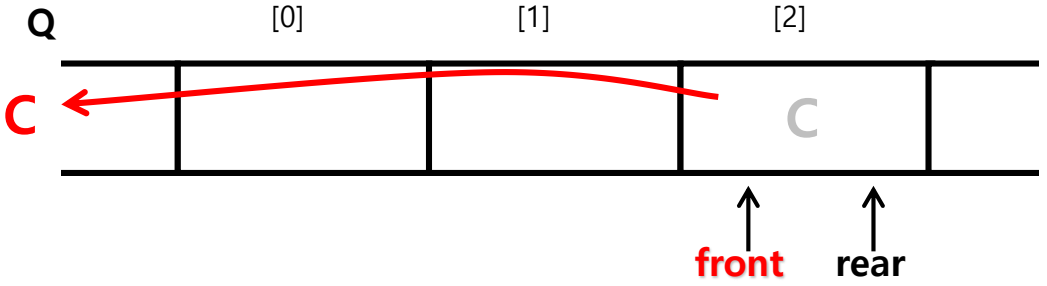
5) 원소 C 삽입 : enQueue(C);



6) 원소 반환/삭제 : deQueue();



7) 원소 반환/삭제 : deQueue();



SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

선형 큐
(Linear Queue)

큐의 구현

선형큐

- 1차원 배열을 이용한 큐
  - 큐의 크기 = 배열의 크기
  - front : 마지막으로 삭제된 인덱스
  - rear : 저장된 마지막 원소의 인덱스

- 상태 표현
  - 초기 상태 : front = rear = -1
  - 공백 상태 : front = rear
  - 포화 상태 : rear = n-1 (n : 배열의 크기, n-1 : 배열의 마지막 인덱스)

큐의 구현

초기 공백 큐 생성

- 크기 n인 1차원 배열 생성
- front와 rear를 -1로 초기화

삽입 : enQueue(item)

- 마지막 원소 뒤에 새로운 원소를 삽입하기 위해

1) rear 값을 하나 증가시켜 새로운 원소를 삽입할 자리를 마련

2) 그 인덱스에 해당하는 배열원소 Q[rear]에 item을 저장

```
enQueue(item) {
    if (isFull()) print("Queue_Full")
    else {
        rear ← rear + 1;
        Q[rear] ← item;
    }
}
```

삭제 : deQueue()

- 가장 앞에 있는 원소를 삭제하기 위해

  1) front 값을 하나 증가시켜 큐에 남아있는 첫 번째 원소 이동

  2) 새로운 첫 번째 원소를 리턴 함으로써 삭제와 동일한 기능함

```
deQueue() {
    if (isEmpty()) print("Queue_Empty");
    else {
        front ← front + 1;
        return Q[front];
    }
}
```

공백상태 및 포화상태 검사 : isEmpty(), isFull()

- 공백상태 : front = rear
- 포화상태 : rear = n-1 (n : 배열의 크기, n-1 : 배열의 마지막 인덱스)

```
isEmpty() {
    if(front == rear) return true;
    else return false;
}
isFull() {
    if (rear == n-1) return true;
    else return false;
}
```

검색 : Qpeek()

- 가장 앞에 있는 원소를 검색하여 반환하는 연산
- 현재 front의 한자리 뒤(front+1)에 있는 원소, 즉 큐의 첫 번째에 있는 원소를 반환

```
Qpeek() {
        if (isEmpty()) print(“Queue_Empty”);
        else return Q[front+1]
}
```

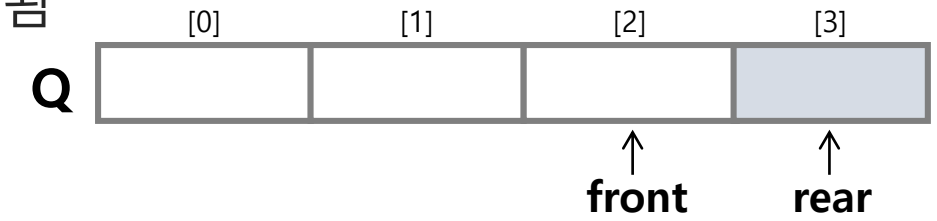
<연습문제1>

- **큐를 구현하여 다음 동작을 확인해 봅시다.**
  - 세 개의 데이터 1, 2, 3을 차례로 큐에 삽입하고
  - 큐에서 세 개의 데이터를 차례로 꺼내서 출력한다.
    - 1, 2, 3이 출력 되야 함.

선형 큐 이용시의 문제점

잘못된 포화상태 인식

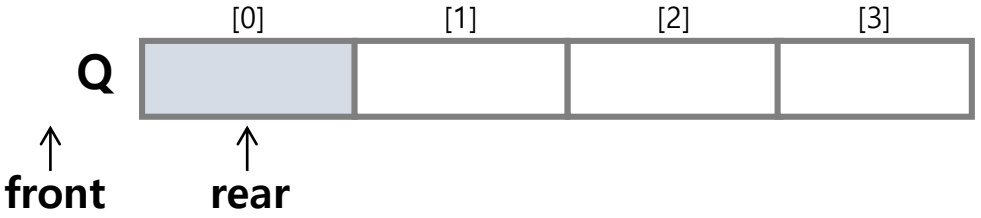
- 선형 큐를 이용하여 원소의 삽입과 삭제를 계속할 경우, 배열의 앞부분에 활용할 수 있는 공간이 있음에도 불구하고, rear=n-1 인 상태 즉, 포화상태로 인식하여 더 이상의 삽입을 수행하지 않게 됨



<포화상태로 잘못 인식>

해결방법 1

- 매 연산이 이루어질 때마다 저장된 원소들을 배열의 앞부분으로 모두 이동시킴
- 원소 이동에 많은 시간이 소요되어 큐의 효율성이 급격히 떨어짐

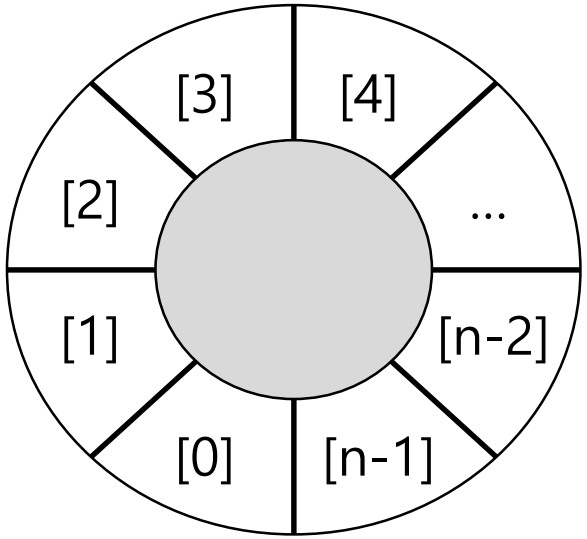


<큐의 원소들을 이동>

선형 큐 이용시의 문제점

해결방법 2

- 1차원 배열을 사용하되, 논리적으로는 배열의 처음과 끝이 연결되어 원형 형태의 큐를 이룬다고 가정하고 사용
- 원형 큐의 논리적 구조



SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

큐 활용

큐의 활용 : 버퍼(Buffer)

버퍼

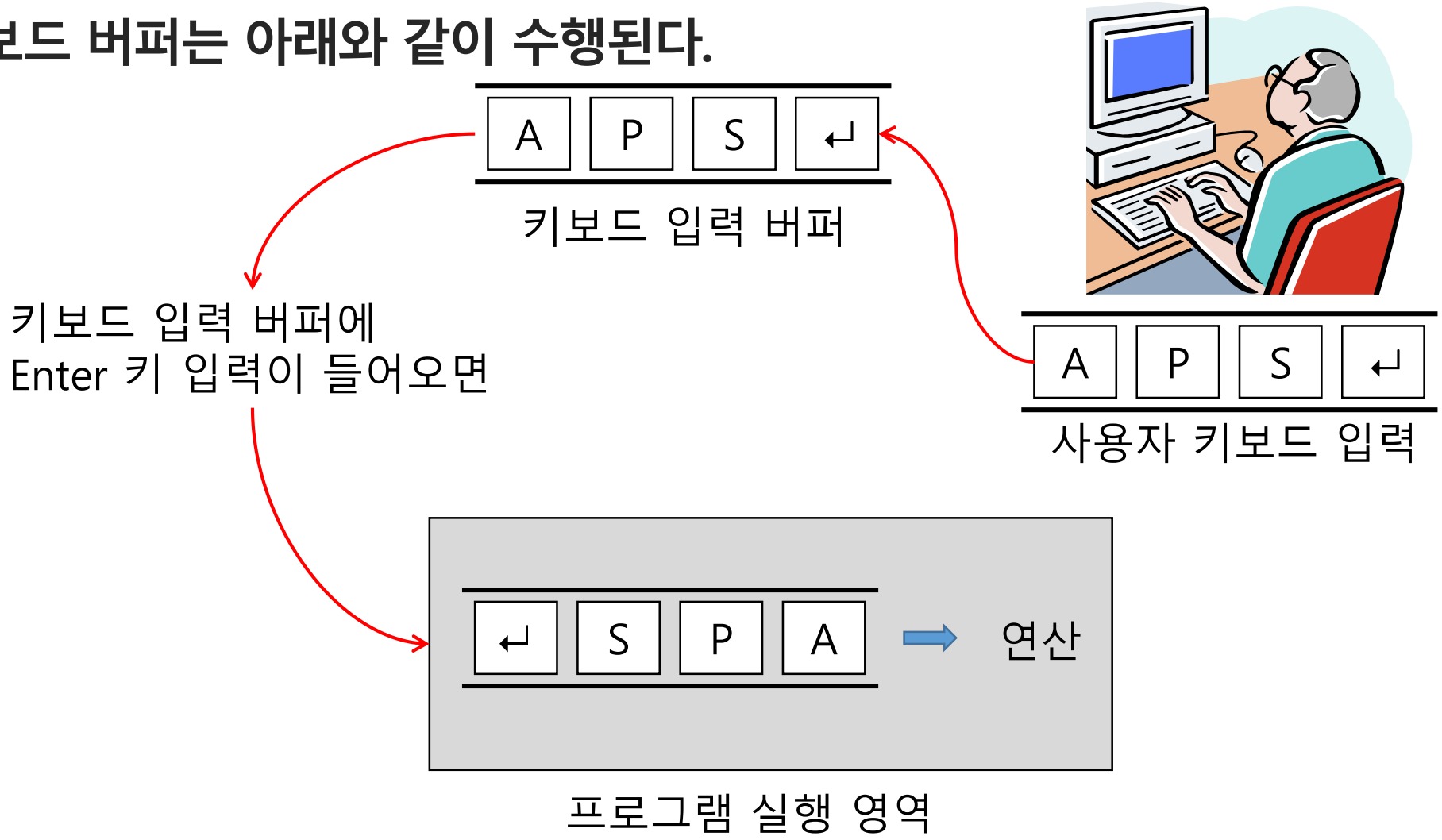
- 데이터를 한 곳에서 다른 한 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 메모리의 영역
- 버퍼링 : 버퍼를 활용하는 방식 또는 버퍼를 채우는 동작을 의미한다.

버퍼의 자료 구조

- 버퍼는 일반적으로 입출력 및 네트워크와 관련된 기능에서 이용된다.
- 순서대로 입력/출력/전달되어야 하므로 FIFO 방식의 자료구조인 큐가 활용된다.

키보드 버퍼

- 키보드 버퍼는 아래와 같이 수행된다.



SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미