CSE2010 자료구조론

Week 5: Queue 1

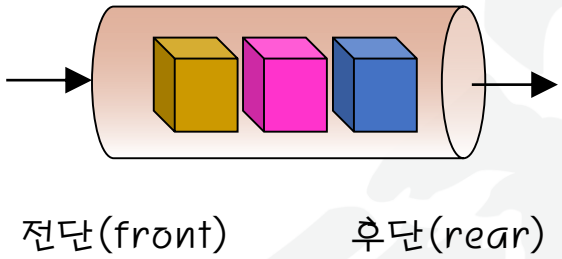
ICT융합학부 한진영

큐(queue)란?

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

큐 ADT

- 삽입과 삭제는 FIFO순서를 따름
- 삽입은 큐의 후단(rear)에서, 삭제는 전단(front)에서 이루어짐

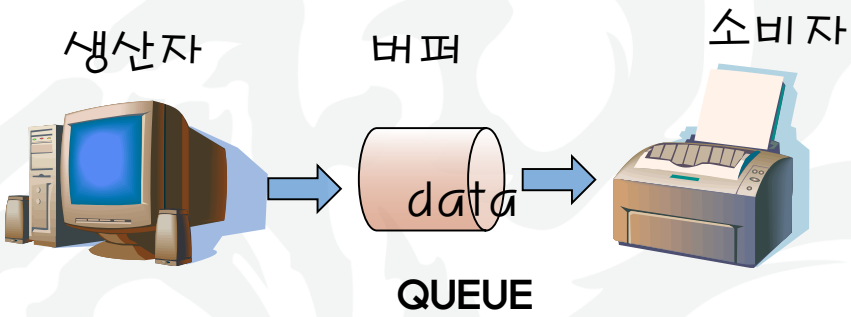


·객체: n개의 element형으로 구성된 요소들의 순서있는 모임
·연산:

- **create()** ::= 큐를 생성한다.
- **init(q)** ::= 큐를 초기화한다.
- **is_empty(q)** ::= 큐가 비어있는지를 검사한다.
- **is_full(q)** ::= 큐가 가득 찼는가를 검사한다.
- **enqueue(q, e)** ::= 큐의 뒤에 요소를 추가한다.
- **dequeue(q)** ::= 큐의 앞에 있는 요소를 반환한 다음 삭제한다.
- **peek(q)** ::= 큐에서 삭제하지 않고 앞에 있는 요소를 반환한다.

큐의 응용

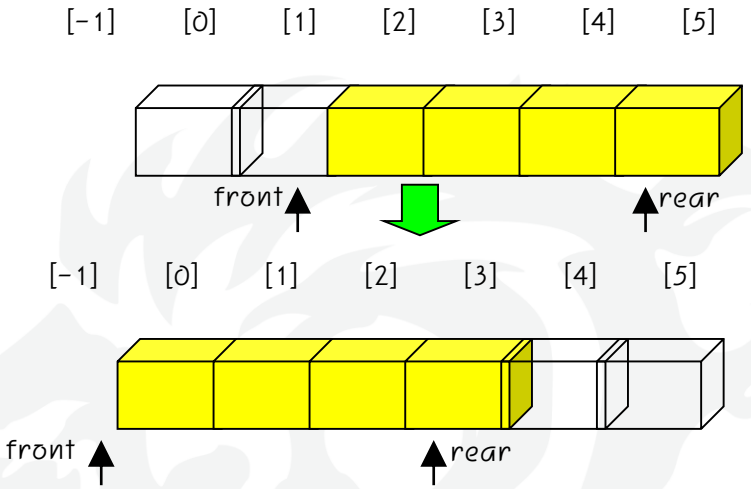
- 직접적인 응용
  - 시뮬레이션의 대기열(공항에서의 비행기들, 은행에서의 대기열)
  - 통신에서의 데이터 패킷들의 모델링에 이용
  - 프린터와 컴퓨터 사이의 버퍼링



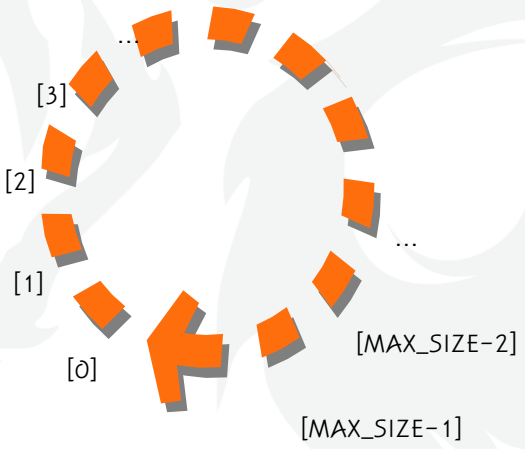
- 간접적인 응용
  - 스택과 마찬가지로 프로그래머의 도구
  - 많은 알고리즘에서 사용됨

배열을 이용한 큐

- 선형큐: 배열을 선형으로 사용하여 큐를 구현
  - 삽입을 계속하기 위해서는 요소들을 이동시켜야 함
  - 문제점이 많아 사용되지 않음

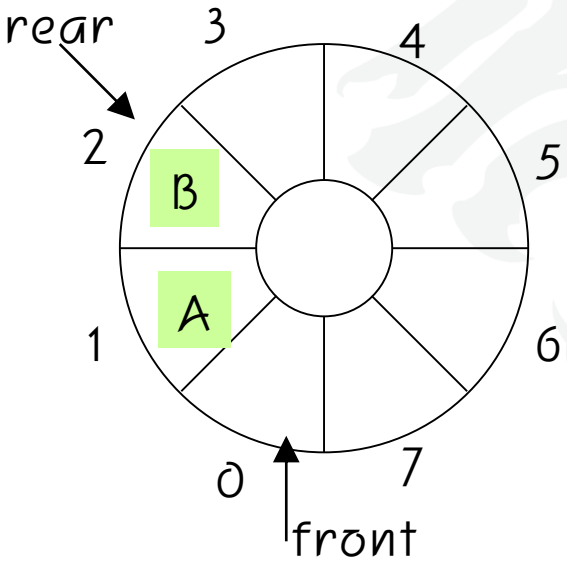


- 원형큐: 배열을 원형으로 사용하여 큐를 구현
  - Front와 rear값이 배열의 끝인 [MAX_QUEUE_SIZE-1]에 도달하면 다음에 증가되는 값은 0이 되도록 구현
  - 개념상 원형일뿐, 실제로 원형 모양의 배열은 아님



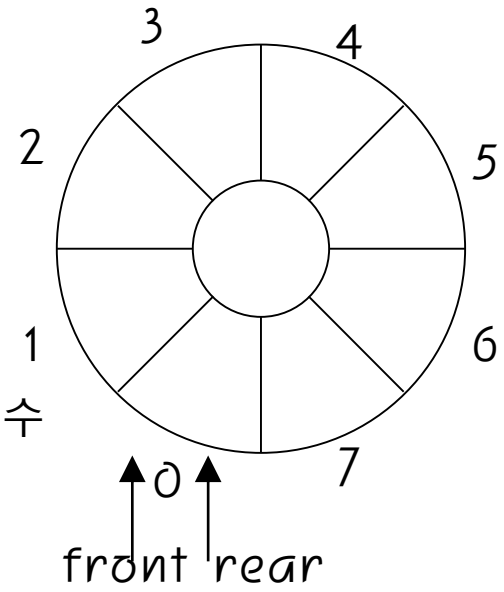
원형 큐의 구조

- 큐의 전단과 후단을 관리하기 위한 2개의 변수 필요
  - front: 첫번째 요소 하나 앞의 인덱스, 삭제와 관련된 변수
  - rear: 마지막 요소의 인덱스, 삽입과 관련된 변수
  - 초기값은 0임(-1 아님)

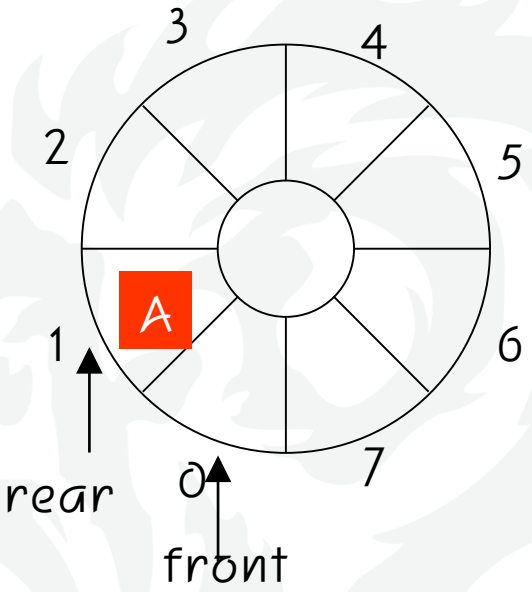


원형 큐 동작(1)

front와 rear 변수
초기값은 0

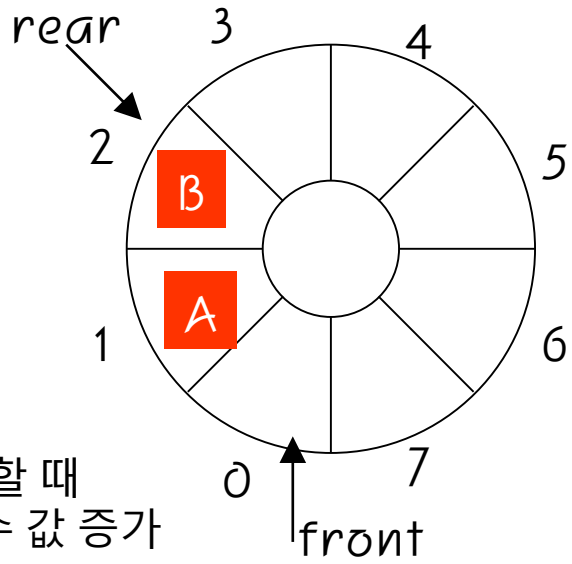


(a) 초기상태



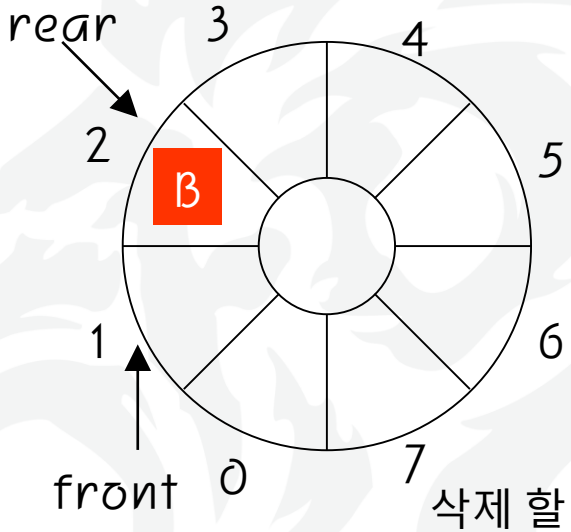
(b) A 삽입

원형 큐 동작(2)



삽입을 할 때
rear 변수 값 증가

(c) B 삽입

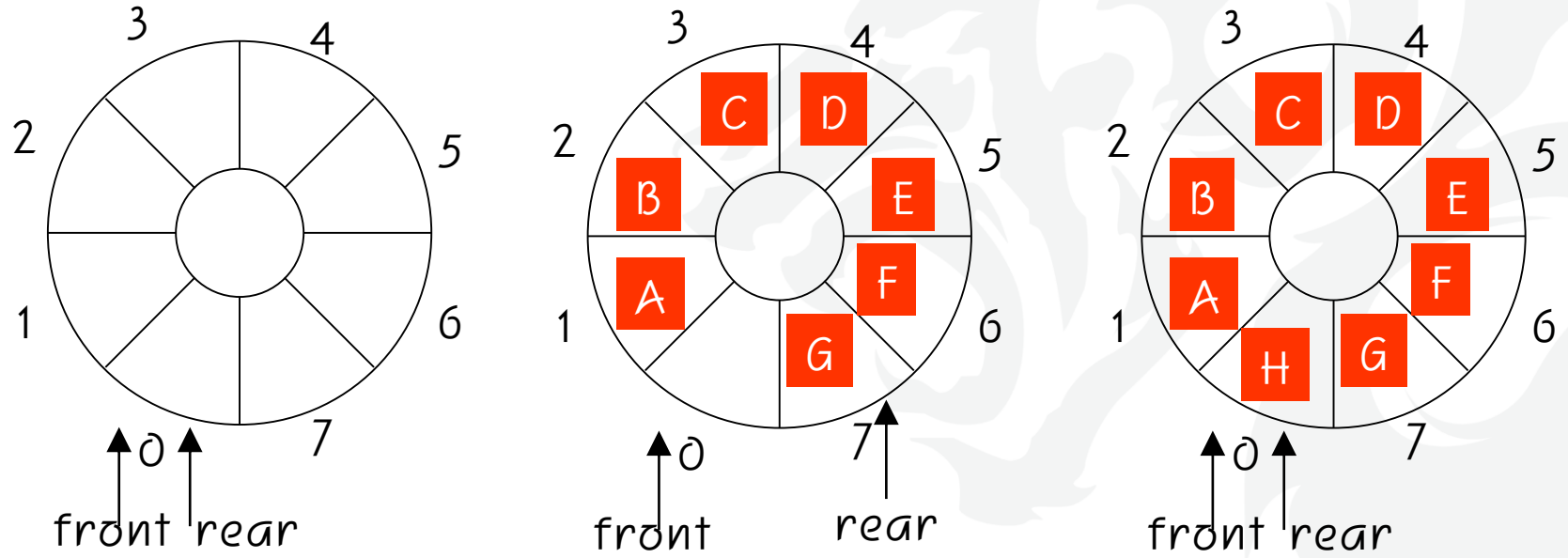


삭제 할 때
front 변수 값 증가

(d) A 삭제

원형 큐의 공백 & 포화 상태

- 공백상태: front == rear (front변수와 rear 변수 값이 같으면)
- 포화상태: front % MAX_QUEUE_SIZE==(rear+1) % MAX_QUE_SIZE
- 공백상태와 포화상태를 구별하기 위하여 하나의 공간은 항상 비움



(a) 공백상태 (b) 포화상태 (c) 오류상태

원형 큐의 삽입 & 삭제 알고리즘

- 나머지(mod) 연산을 사용하여 인덱스를 원형으로 회전시킴
  - front <- (front+1) mod MAX_QUEUE_SIZE;
  - rear <- (rear+1) mod MAX_QUEUE_SIZE;

```
원형 큐에서의 삽입 알고리즘
enqueue(Q, x)

rear ← (rear+1) mod MAX_QUEUE_SIZE;
Q[rear] ← x;

원형 큐에서의 삭제 알고리즘
dequeue(Q)

front ← (front+1) mod MAX_QUEUE_SIZE;
return Q[front];
```

원형 큐의 구현(1)

- Preliminary
  - 나머지(mod) 연산자 : %
  - front: 첫 번째 요소로부터 시계 방향으로 하나 앞에 위치함. 삭제시 먼저 front를 증가시키고 그 위치에서 데이터를 꺼내와야 함
  - rear: 마지막 요소. 삽입시 무조건 rear를 하나 증가시키고, 증가된 위치에 데이터를 넣어야 함
  - 공백상태 검출: front == rear
  - 포화상태 검출: front == (rear+1)%MAX_QUEUE_SIZE

원형 큐의 구현(2)

```
#define MAX_QUEUE_SIZE 100
typedef int element;
typedef struct {
    element  queue[MAX_QUEUE_SIZE];
    int front, rear;
} QueueType;
// 초기화 함수
Void init(QueueType *q)
{
     q→front = q→rear = 0;
}
// 공백 상태 검출 함수
int is_empty(QueueType *q)
{
     return (q->front == q->rear);
}
// 포화 상태 검출 함수
int is_full(QueueType *q)
{
     return ((q->rear+1)%MAX_QUEUE_SIZE == q->front);
}
```

원형 큐의 구현(3)

```
// 삽입 함수
void enqueue(QueueType *q, element item)
{
     if( is_full(q) )
          error("큐가 포화상태입니다");
     q->rear = (q->rear+1) % MAX_QUEUE_SIZE;
     q->queue[q->rear] = item;
}
// 삭제 함수
element dequeue(QueueType *q)
{
     if( is_empty(q) )
          error("큐가 공백상태입니다");
     q->front = (q->front+1) % MAX_QUEUE_SIZE;
     return q->queue[q->front];
}
// 피크 함수
element peek(QueueType *q)
{
     if( is_empty(q) )
          error("큐가 공백상태입니다");
     return q->front[(q->front+1) % MAX_QUEUE_SIZE];
}
```

Week 5: Queue 1