3장. 큐

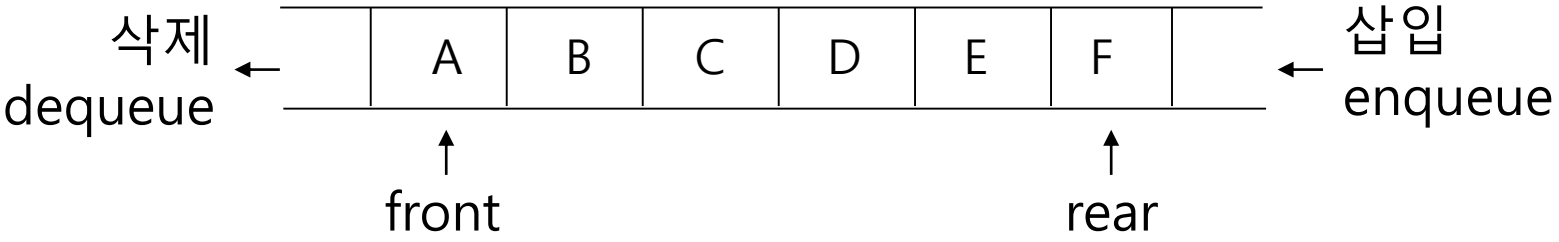
- 큐의 의미
- 큐의 연산
- 배열을 이용한 큐 구현
- 연결리스트를 이용한 큐 구현
- 큐의 응용

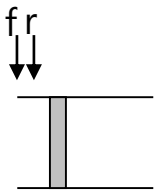
학습목표

- 큐의 의미
- 큐의 연산
- 구현 관점에서 큐의 종류
- 배열을 이용한 큐 운용
- 연결리스트를 이용한 큐 운용

큐란?

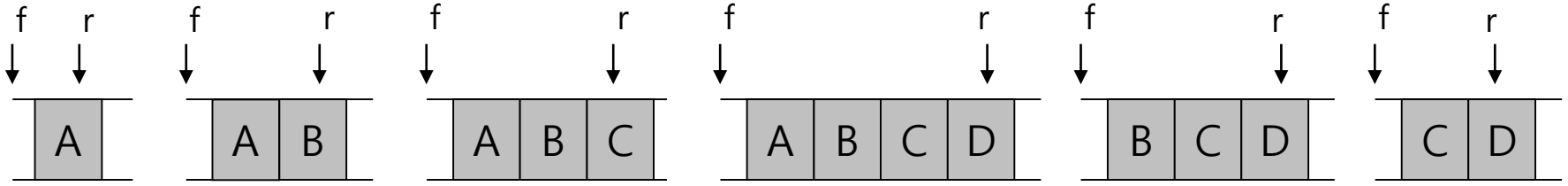
- **먼저 입력된 자료를 먼저 출력할 목적으로 구성한 자료구조 =>First In First Out**
  - 자료 간에 먼저/나중 관계만 표시할 수 있으면 된다. => 선형 구조
- **데이터 저장 구조 : 선형 구조**
- **큐 관련용어**
  - front / rear / enqueue /dequeue





작업 전

위 아래 두 표현에서
유효 데이터 범위 차이는?

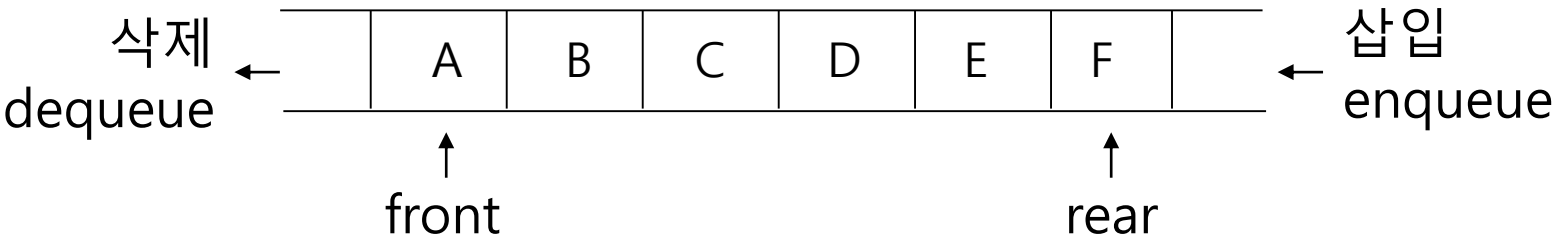


Queue ADT

데이터: 선입선출(FIFO)의 접근 방법을 유지하는 항목들의 모음

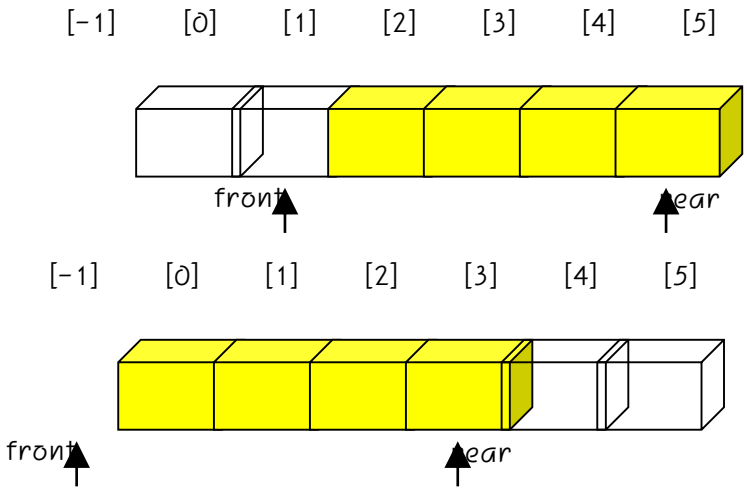
연산

- Queue(): 비어 있는 새로운 큐를 만든다.
- isEmpty(): 큐가 비어있으면 True를 아니면 False를 반환한다.
- enqueue(x): 항목 x를 큐의 맨 뒤에 추가한다.
- dequeue(): 큐의 맨 앞에 있는 항목을 꺼내 반환한다.
- peek(): 큐의 맨 앞에 있는 항목을 삭제하지 않고 반환한다.
- size(): 큐의 모든 항목들의 개수를 반환한다.
- clear(): 큐를 공백상태로 만든다.



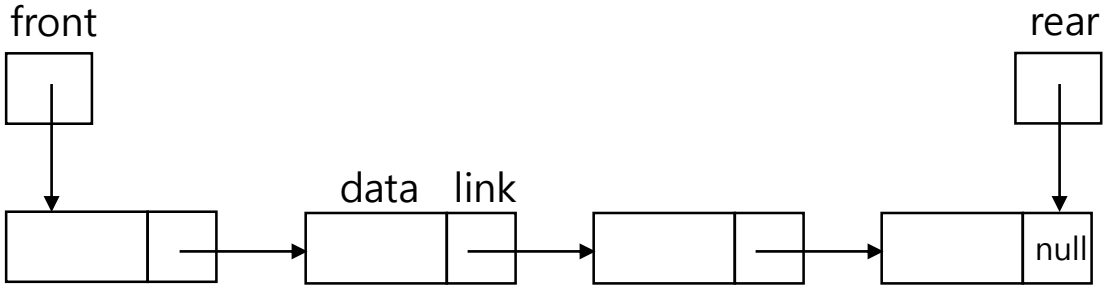
구현 관점에서의 큐

배열

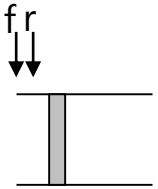


큐의 초기 상태 모습은?
front의 값(가리키는 위치)는?
rear의 값(가리키는 위치)는?
배열 구현 위 그림 상태에서 삽입 가능한가?
각각의 구현에서 삽입할 수 없는 상황은?
포화(full) 상태의 모습은?
각각의 공백(empty) 상태의 모습은?
배열 큐의 불편을 극복하는 방법은?

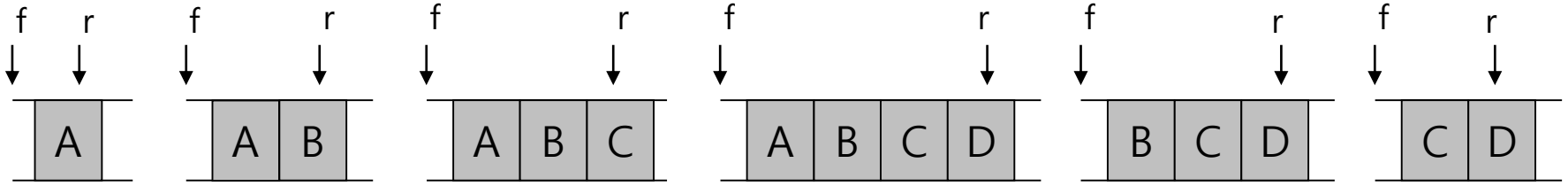
연결리스트



배열에 의한 큐



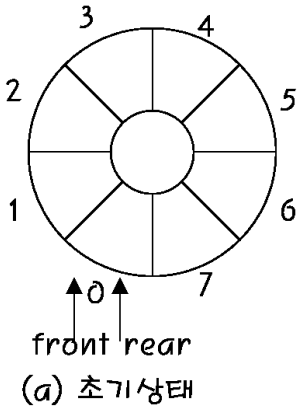
작업 전/초기 상태



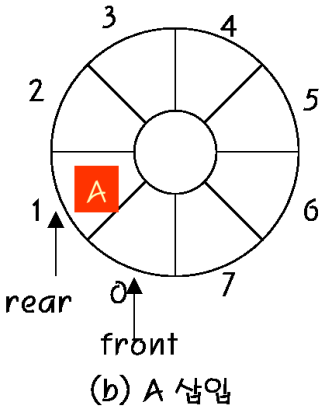
삽입(A) 삽입(B) 삽입(C) 삽입(D) 삭제 삭제

배열 원형 큐

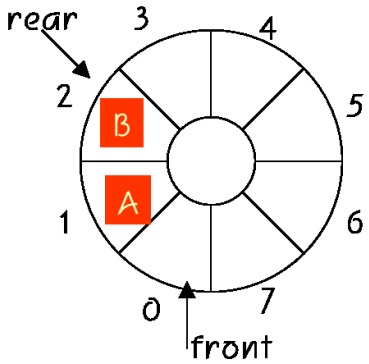
❖ 이동의 문제점 해결을 위해 배열을 원형으로 운영

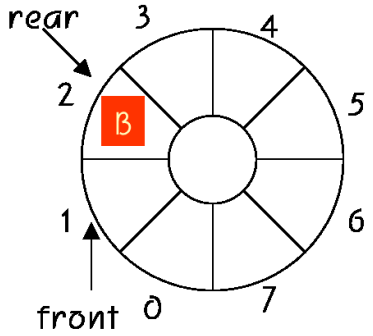


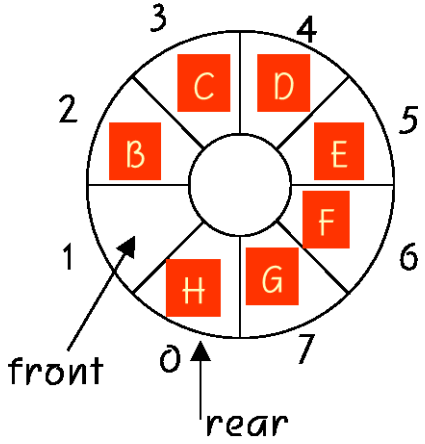
(a) 초기상태



(b) A 삽입



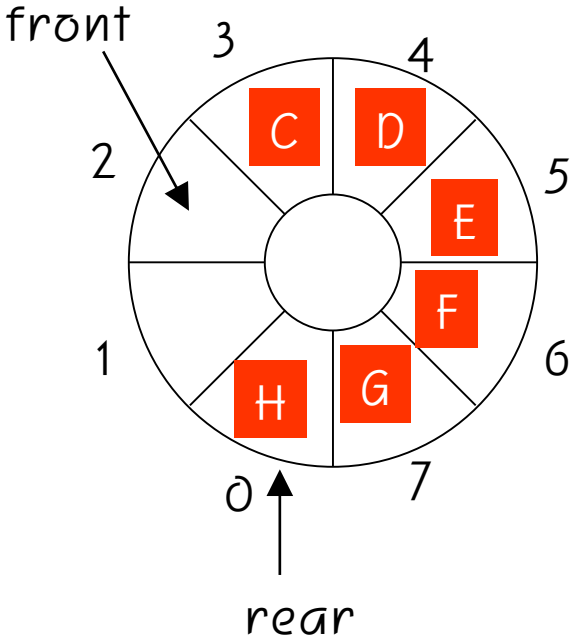




- 원형 큐의 구현 : 실제 front 공간 하나가 비지만, 이동 않는 편의를 위해 그 공간을 희생!!!
  모듈러(modular : 나머지 계산 연산)을 통한 rear/front 값 제어.
  - 초기화 : front = rear = 0 (공백큐) //0 이외에 1, 2, …n-1도 가능
  - 공백 상태 : if (front == rear) then ~~
  - 원소 삭제 : front = (front + 1) % n 후 front 위치 원소 삭제
  - 포화 상태 : if((rear+1) % n == front) then ~~ // % : 나머지 계산 연산자
  - 원소 삽입 : rear=(rear+1)%n, 후 rear 위치에 원소 저장
  - queueShow( ) : if (rear != front) then 보여줄 것이 있다. //front 다음 부터 rear까지 출력

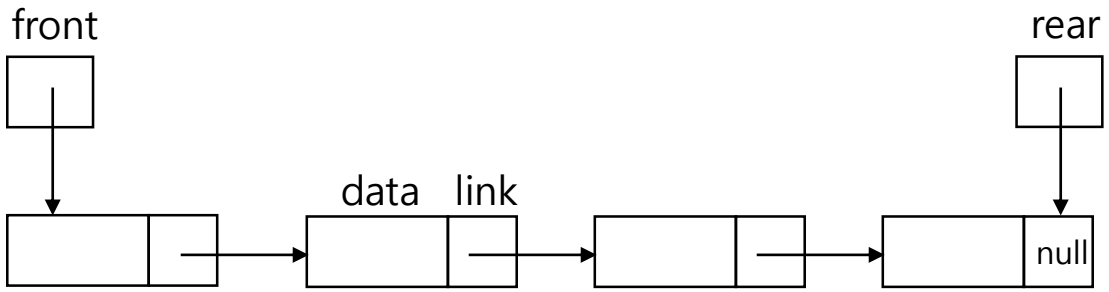
배열 원형 큐

rear, front 운영



- mod(modulus, %) 연산자 이용
  - 삽입을 위해 먼저 rear를 증가시킬 때 : rear = (rear+1) % n
    - rear 값은 n-1 다음에 n이 되지 않고, 다시 0으로 되돌아감
  - 삭제를 위해 front를 증가시킬 때 : front = (front+1) % n
    - rear와 마찬가지로, front 값은 n-1 다음에 0이 되어 원형으로 순환

연결리스트 큐



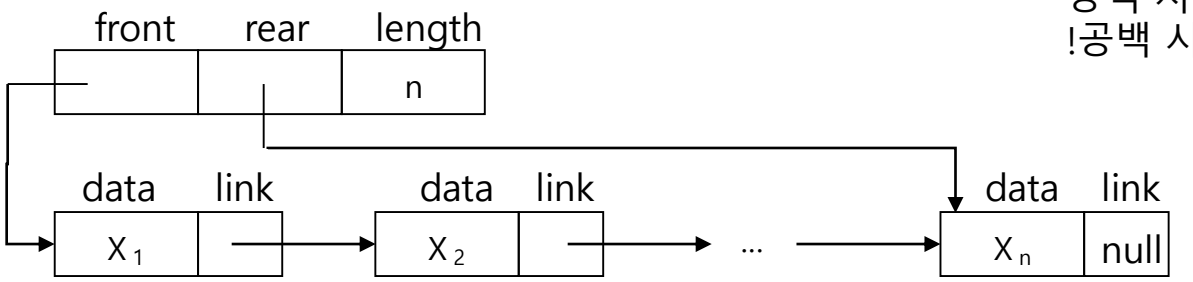
노드 구조 및 포인터 준비

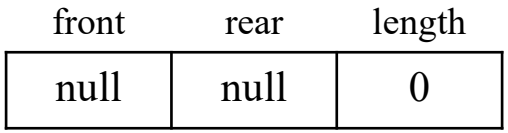
노드 생성 함수?

초기화?

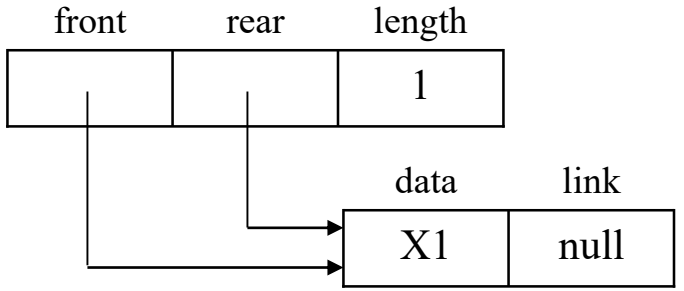
공백 상태?
다수에서 1개 삭제 및 마지막 노드 삭제 연산?

포화 상태?
공백 시 삽입 연산?
!공백 시 삽입 연산?





공백 상태

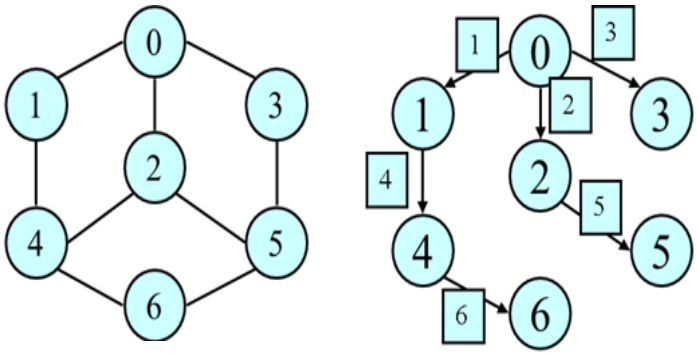


하나의 노드를 가진 연결 큐

큐의 응용

너비 우선 탐색

- 너비우선 탐색은 시작 정점(노드)을 출발점으로 하여, 현재 노드를 방문한 후 현 정점에 인접한 모든 정점들을 우선 방문하는 방법이다. 이후의 모든 행위도 같은 방법으로 이루어진다.
- 그림 1의 그래프의 경우 출발점을 0으로 하면 그림 2에 표시한 바와 같이 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 순으로 방문하게 된다.

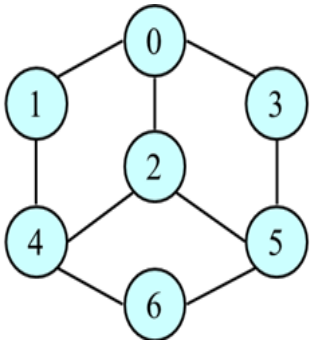


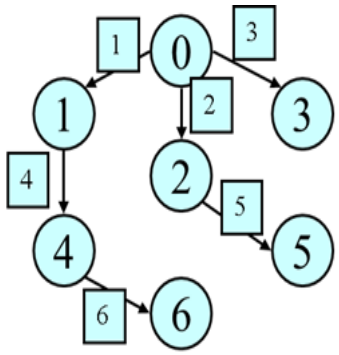
1) 탐색 순서 제어용 큐와
방문 여부 표시용 배열을 준비.

2) 출발점을 큐에 넣고,

3) 큐가 비(空)거나 모든 노드를 방문할 때까지,
큐에서 데이터를 꺼내어 방문한 노드이면 버리고,
방문하지 않았으면 방문하고(방문표시병행), 방문한
노드의 후행자를 방문 선(先)순위부터 큐에 저장하고,
이를 반복.

큐의 응용





< 0을 출발 노드로 할 때 >

< Queue f:-1 r:0 > : 0
방문순서 (1) 방문노드 (0)
< Queue f:0 r:1 > : 1
< Queue f:0 r:2 > : 1 2
< Queue f:0 r:3 > : 1 2 3
방문순서 (2) 방문노드 (1)
< Queue f:1 r:4 > : 2 3 0
< Queue f:1 r:5 > : 2 3 0 4
방문순서 (3) 방문노드 (2)
< Queue f:2 r:6 > : 3 0 4 0
< Queue f:2 r:7 > : 3 0 4 0 4
< Queue f:2 r:8 > : 3 0 4 0 4 5
방문순서 (4) 방문노드 (3)
< Queue f:3 r:9 > : 0 4 0 4 5 0
< Queue f:3 r:10 > : 0 4 0 4 5 0 5
방문순서 (5) 방문노드 (4)
< Queue f:5 r:11 > : 0 4 5 0 5 1
< Queue f:5 r:12 > : 0 4 5 0 5 1 2
< Queue f:5 r:13 > : 0 4 5 0 5 1 2 6
방문순서 (6) 방문노드 (5)
< Queue f:8 r:14 > : 0 5 1 2 6 2
< Queue f:8 r:15 > : 0 5 1 2 6 2 3
< Queue f:8 r:16 > : 0 5 1 2 6 2 3 6
방문순서 (7) 방문노드 (6)
< Queue f:13 r:17 > : 2 3 6 4
< Queue f:13 r:18 > : 2 3 6 4 5

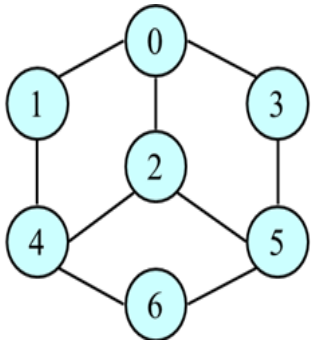
< 1을 출발 노드로 할 때 >

< Queue f:-1 r:0 > : 1
방문순서 (1) 방문노드 (1)
< Queue f:0 r:1 > : 0
< Queue f:0 r:2 > : 0 4
방문순서 (2) 방문노드 (0)
< Queue f:1 r:3 > : 4 1
< Queue f:1 r:4 > : 4 1 2
< Queue f:1 r:5 > : 4 1 2 3
방문순서 (3) 방문노드 (4)
< Queue f:2 r:6 > : 1 2 3 1
< Queue f:2 r:7 > : 1 2 3 1 2
< Queue f:2 r:8 > : 1 2 3 1 2 6
방문순서 (4) 방문노드 (2)
< Queue f:4 r:9 > : 3 1 2 6 0
< Queue f:4 r:10 > : 3 1 2 6 0 4
< Queue f:4 r:11 > : 3 1 2 6 0 4 5
방문순서 (5) 방문노드 (3)
< Queue f:5 r:12 > : 1 2 6 0 4 5 0
< Queue f:5 r:13 > : 1 2 6 0 4 5 0 5
방문순서 (6) 방문노드 (6)
< Queue f:8 r:14 > : 0 4 5 0 5 4
< Queue f:8 r:15 > : 0 4 5 0 5 4 5
방문순서 (7) 방문노드 (5)
< Queue f:11 r:16 > : 0 5 4 5 2
< Queue f:11 r:17 > : 0 5 4 5 2 3
< Queue f:11 r:18 > : 0 5 4 5 2 3 6

스택의 응용

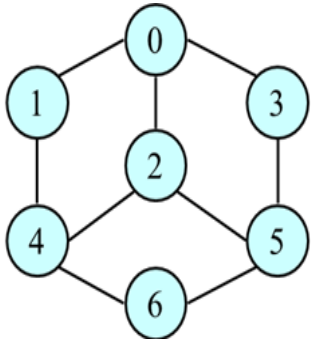
깊이 우선 탐색

- 시작 정점(노드)을 출발점으로 하여, 현재 노드를 방문한 후 현 정점에 인접한 노드들 중 우선순위가 가장 높은 노드를 먼저 방문하는 방법이다. 인접 노드가 없으면(후행 노드가 없으면) 전 단계에서 방문하지 않은 노드 중 우선 순위가 가장 높은 노드를 방문한다.
- 이와 같은 방문 순서는 스택을 이용하면 쉽게 구현된다. 다만 방문한 노드의 인접노드를 스택에 저장할 때 방문 순서가 가장 낮은 노드부터 저장하면 된다.




```
while( 1 ) {                              //깊이우선 탐색 시작
        data = pop();
        if (data == -999)                          //삭제할 것이 없으면 종료
                break;
        if (visited[data] == 1)  //방문한 노드
                continue;
        else{
                printf("₩n 방문노드 ( %d ) ",  data);
                visited[data] = 1;
                for(int x=6; x>=0; x--){
                        if (graph[data][x] == 1)
                                push(x);
                }
        }
}
```

스택의 응용



< 0을 출발 노드로 할 때 >

< stack - top(0)> : 0
< stack - top(-1)> :
방문순서 (1) 방문노드 (0)
< stack - top(0)> : 3
< stack - top(1)> : 3 2
< stack - top(2)> : 3 2 1
< stack - top(1)> : 3 2
방문순서 (2) 방문노드 (1)
< stack - top(2)> : 3 2 4
< stack - top(3)> : 3 2 4 0
< stack - top(2)> : 3 2 4
< stack - top(1)> : 3 2
방문순서 (3) 방문노드 (4)
< stack - top(2)> : 3 2 6
< stack - top(3)> : 3 2 6 2
< stack - top(4)> : 3 2 6 2 1
< stack - top(3)> : 3 2 6 2
< stack - top(2)> : 3 2 6
방문순서 (4) 방문노드 (2)
< stack - top(3)> : 3 2 6 5
< stack - top(4)> : 3 2 6 5 4
< stack - top(5)> : 3 2 6 5 4 0
< stack - top(4)> : 3 2 6 5 4
< stack - top(3)> : 3 2 6 5
< stack - top(2)> : 3 2 6
방문순서 (5) 방문노드 (5)
< stack - top(3)> : 3 2 6 6
< stack - top(4)> : 3 2 6 6 3
< stack - top(5)> : 3 2 6 6 3
2
< stack - top(4)> : 3 2 6 6 3
< stack - top(3)> : 3 2 6 6
방문순서 (6) 방문노드 (3)
< stack - top(4)> : 3 2 6 6 5
< stack - top(5)> : 3 2 6 6 5
0
< stack - top(4)> : 3 2 6 6 5
< stack - top(3)> : 3 2 6 6
< stack - top(2)> : 3 2 6
방문순서 (7) 방문노드 (6)
< stack - top(3)> : 3 2 6 5
< stack - top(4)> : 3 2 6 5 4
< stack - top(3)> : 3 2 6 5
< stack - top(2)> : 3 2 6
< stack - top(1)> : 3 2
< stack - top(0)> : 3
< stack - top(-1)> :

< 2를 출발 노로로 할 때 >
방문순서 (1) 방문노드 (2)
방문순서 (2) 방문노드 (0)
방문순서 (3) 방문노드 (1)
방문순서 (4) 방문노드 (4)
방문순서 (5) 방문노드 (6)
방문순서 (6) 방문노드 (5)
방문순서 (7) 방문노드 (3)

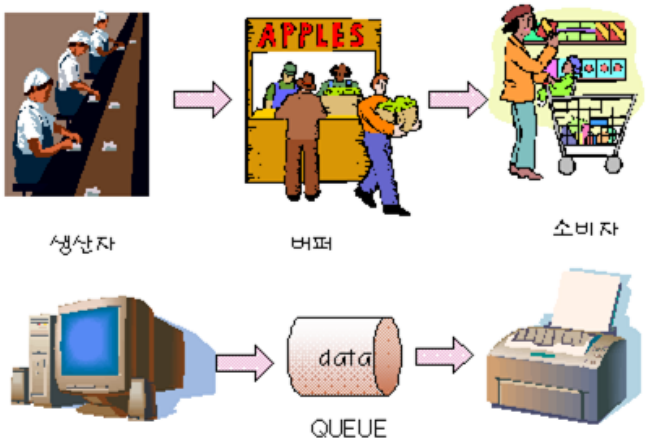
< 4를 출발 노드로 할 때 >
방문순서 (1) 방문노드 (4)
방문순서 (2) 방문노드 (1)
방문순서 (3) 방문노드 (0)
방문순서 (4) 방문노드 (2)
방문순서 (5) 방문노드 (5)
방문순서 (6) 방문노드 (3)
방문순서 (7) 방문노드 (6)

< 3을 출발 노드로 할 때 >
방문순서 (1) 방문노드 (3)
방문순서 (2) 방문노드 (0)
방문순서 (3) 방문노드 (1)
방문순서 (4) 방문노드 (4)
방문순서 (5) 방문노드 (2)
방문순서 (6) 방문노드 (5)
방문순서 (7) 방문노드 (6)

큐의 응용

시스템 버퍼

- 서로 다른 속도로 실행되는 두 프로세스 간의 상호 작용을 조화시키는 버퍼

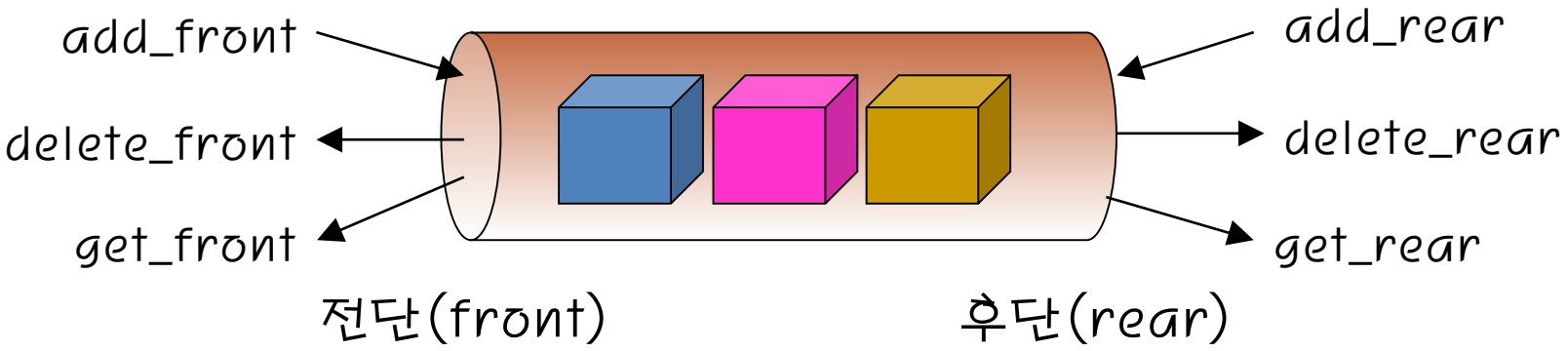


시뮬레이션

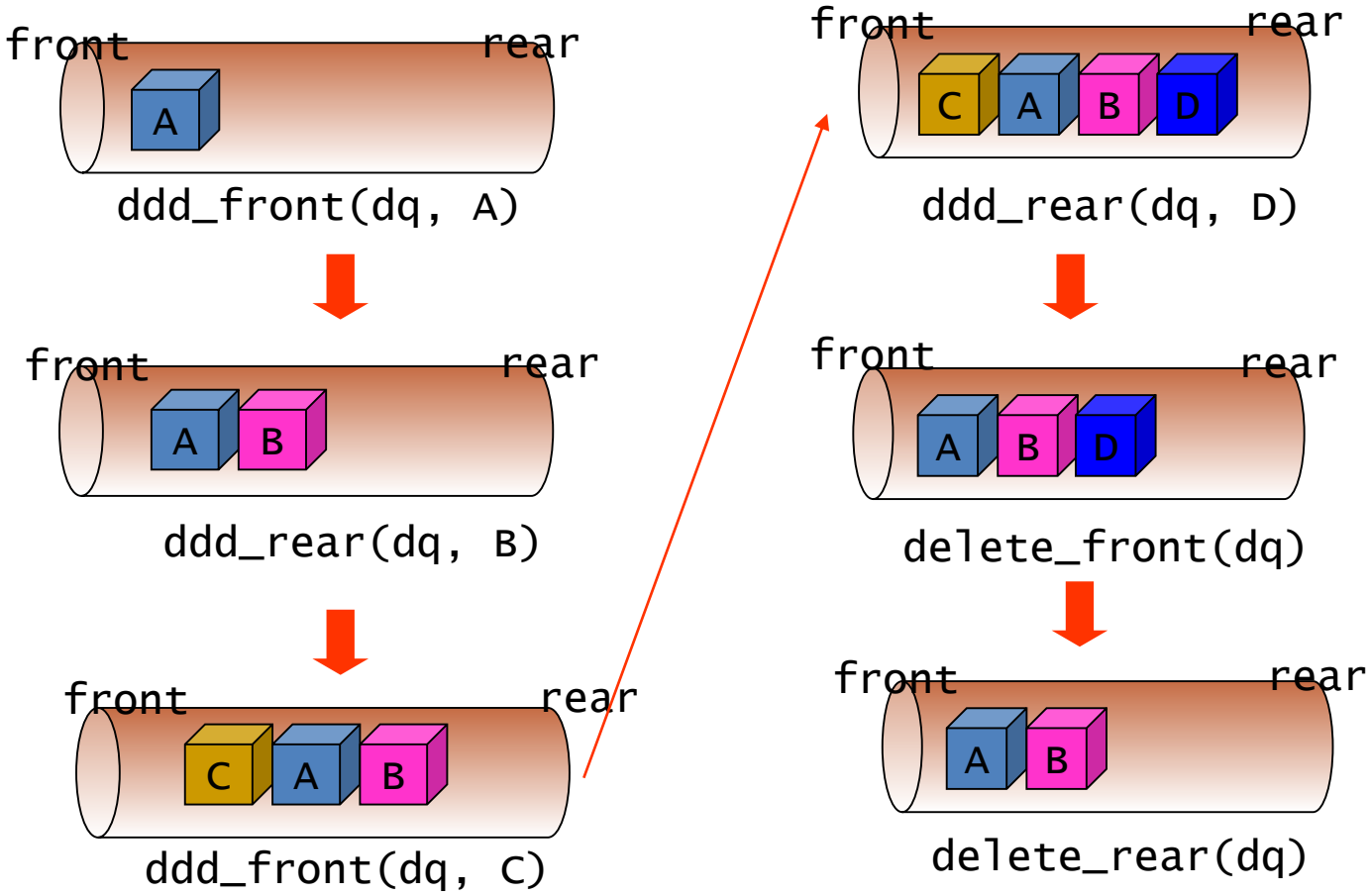
- 큐잉이론에 따라 시스템의 특성을 시뮬레이션. 예) 은행 시뮬레이션

덱

- 덱(deque)은 double-ended queue의 줄임말로서 큐의 전단(front)와 후단(rear)에서 모두 삽입과 삭제가 가능한 큐



덱의 연산


front
rear
A
ddd_front(dq, A)
front
rear
A
B
ddd_rear(dq, B)
front
rear
C
A
B
ddd_front(dq, C)
front
rear
C
A
B
D
ddd_rear(dq, D)
front
rear
A
B
D
delete_front(dq)
front
rear
A
B
delete_rear(dq)