Queue

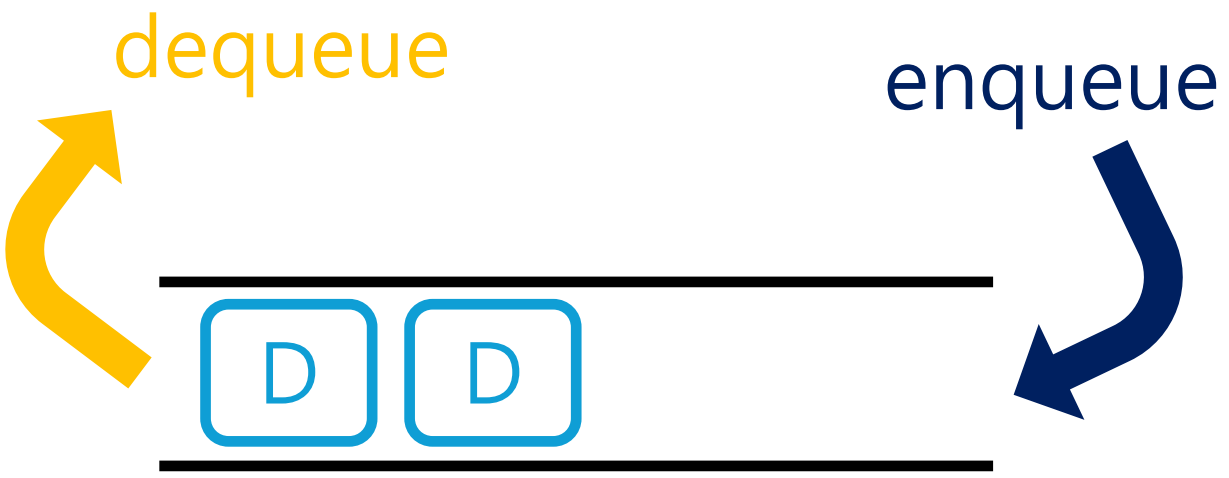
즐겁고 알찬 자료구조 튜터링

Queue

Queue

- 먼저 들어간 것이 먼저에 나오는 FIFO(First-in, First-out)구조의 자료 구조

- 큐의 기본적인 연산
  - enqueue
  - dequeue

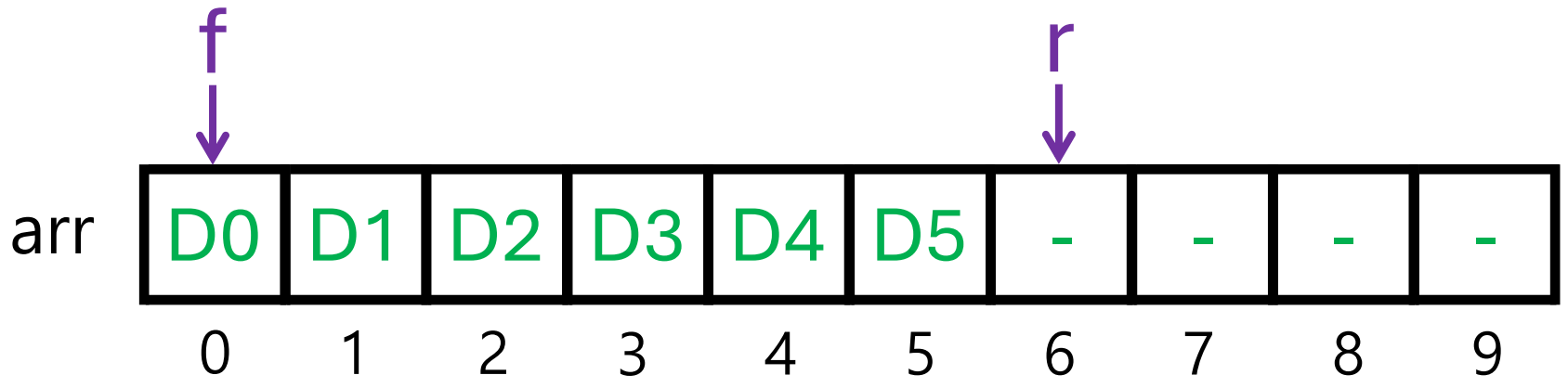


Queue의 ADT

- int size() : 현재 queue에 있는 원소 개수를 반환
- bool empty() : queue에 원소가 비어있는지 여부를 반환
- T &front() : queue에서 가장 먼저 저장된 원소를 반환
- void enqueue(T &data) : queue에 인자로 받은 data를 저장하고, stack의 용량을 초과한 경우 예외 발생
- void dequeue() : queue에 가장 먼저 저장된 요소를 삭제하고, queue가 비어있는 경우 예외 발생

배열 기반의 Queue

- 클래스의 멤버 변수로 배열을 가지고, 이를 Queue로 사용
- 멤버 변수 f(front)와 r(rear)을 사용해 원소의 끝과 끝을 가리킴



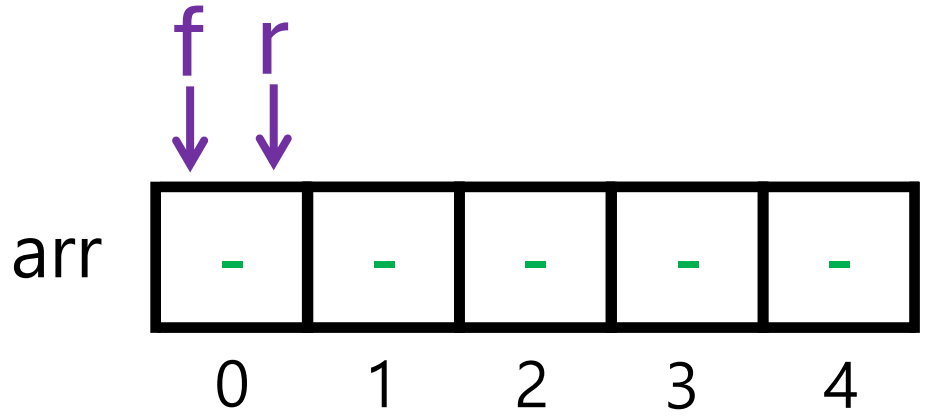
배열 기반의 Queue

```
1 template <typename T>
2 class ArrayQueue {
3 private:
4     T *arr;
5     int cap, n;
6     int f, r;
```

```
1 ArrayQueue(int cap) {
2     this->cap = cap;
3     arr = new T[cap];
4     f = r = 0;
5     n = 0;
6 }
```

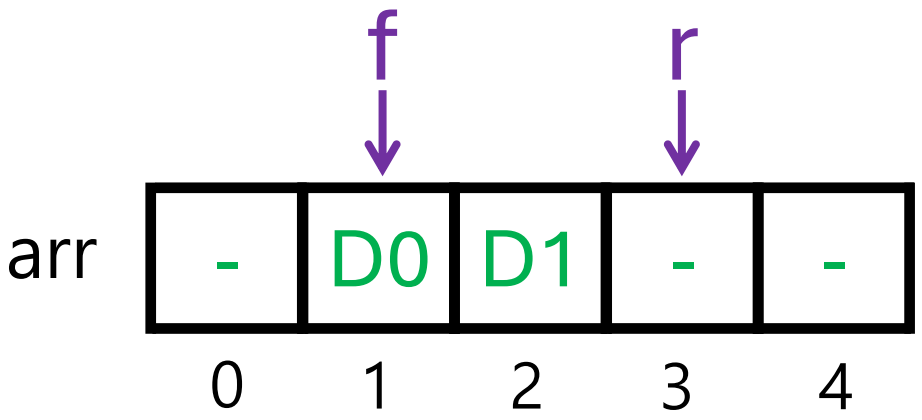
size, empty

```
1 int size() const {
2     return n;
3 }
4
5 bool empty() const {
6     return n == 0;
7 }
```



front, enqueue, dequeue

```
1 T &front() {
2     if (empty()) {
3         throw "Queue is empty";
4     }
5
6     return arr[f];
7 }
```

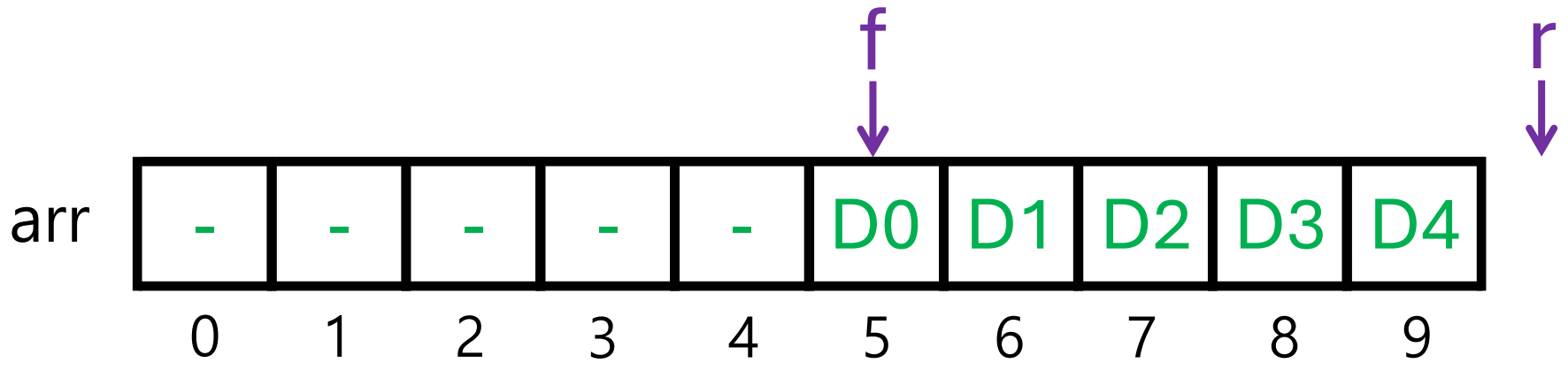



```
1 void enqueue(const T &data) {
2     if (size() == cap) {
3         throw "Queue is full";
4     }
5
6     arr[r++] = data;
7     n++;
8 }
```

```
1 void dequeue() {
2     if (empty()) {
3         throw "Queue is empty";
4     }
5
6     f++;
7     n--;
8 }
```

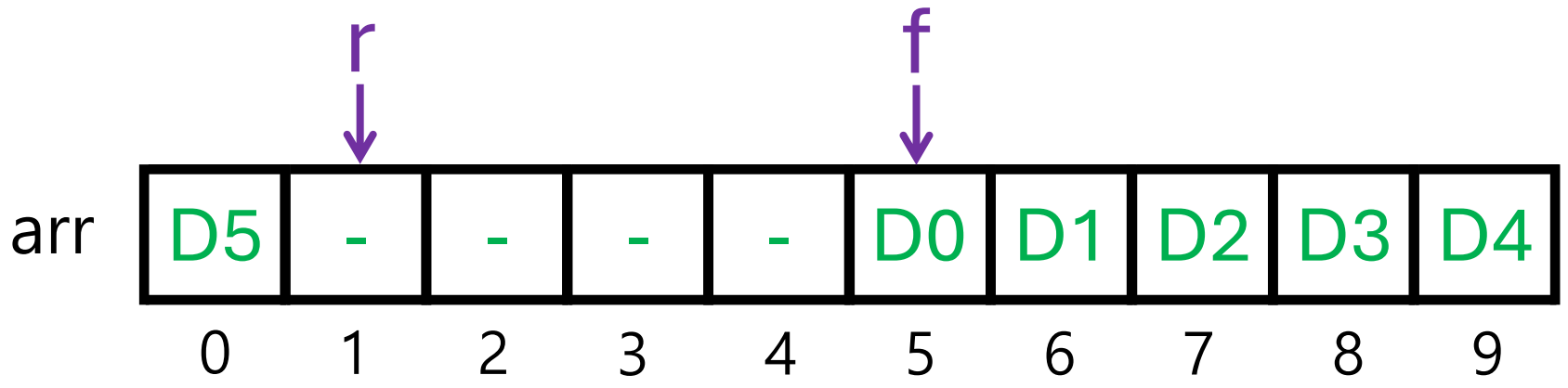
Queue에서 배열의 크기가 부족하면?

- 배열 기반으로 queue을 구현한 경우 배열 크기만큼 원소를 enqueue하지 못함
- stack과 달리 배열에서 한번 사용한 영역은 다시 사용하지 않아 공간 낭비가 심함



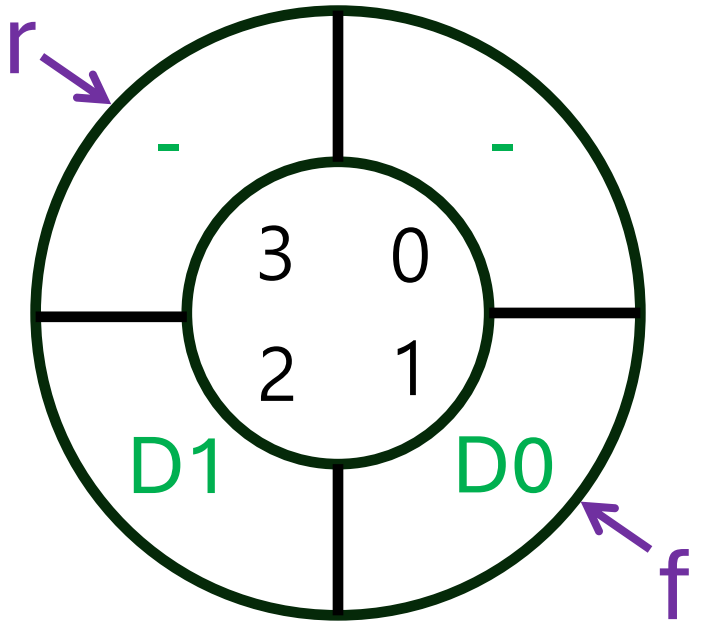
Circular Queue

- rear가 배열의 끝 다음을 가리키게 되면, 배열 처음의 처음으로 돌아가 이전에 원소를 dequeue한 공간을 사용



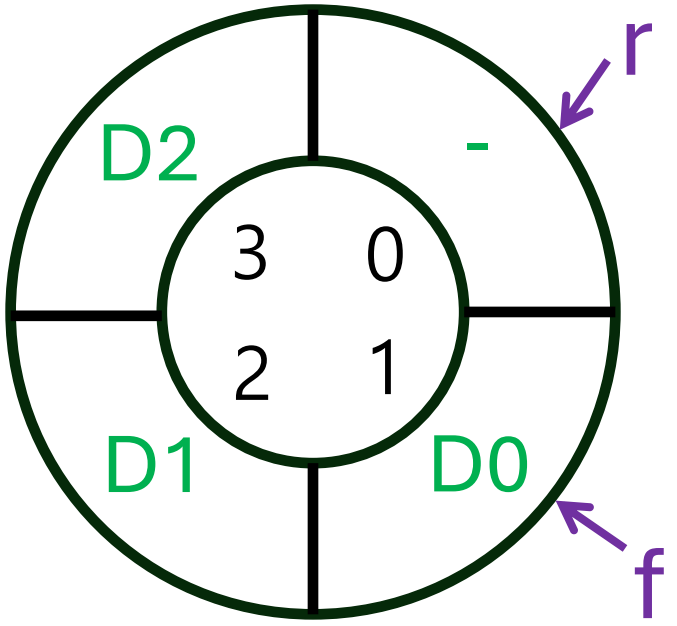
Circular Queue

- rear가 배열의 끝 다음을 가리키게 되면, 배열 처음의 처음으로 돌아가 이전에 원소를 dequeue한 공간을 사용



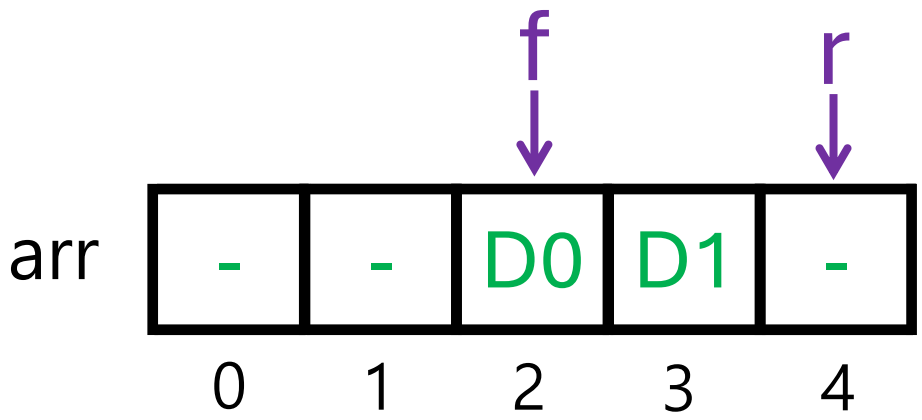
Circular Queue

- rear가 배열의 끝 다음을 가리키게 되면, 배열 처음의 처음으로 돌아가 이전에 원소를 dequeue한 공간을 사용

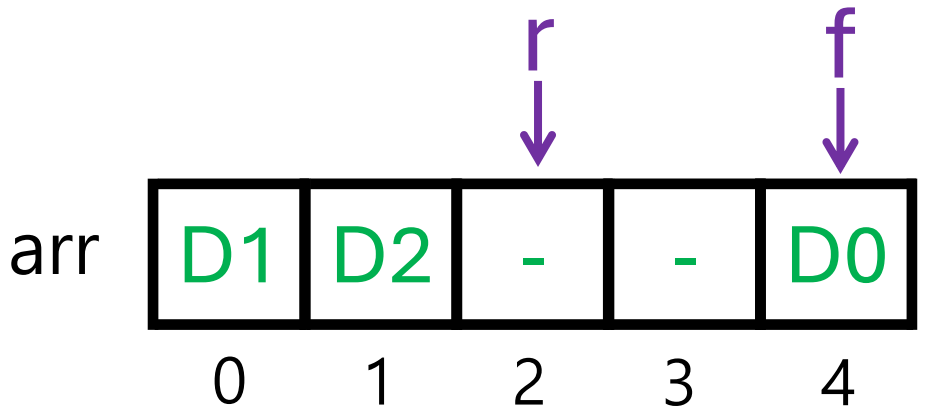


Circular Queue의 enqueue, dequeue

```
1 void enqueue(const T &data) {
2     if (size() == cap) {
3         throw "Queue is full";
4     }
5
6     arr[r] = data;
7     r = (r + 1) % cap;
8     n++;
9 }
```




```
1 void dequeue() {
2     if (empty()) {
3         throw "Queue is empty";
4     }
5
6     f = (f + 1) % cap;
7     n--;
8 }
```

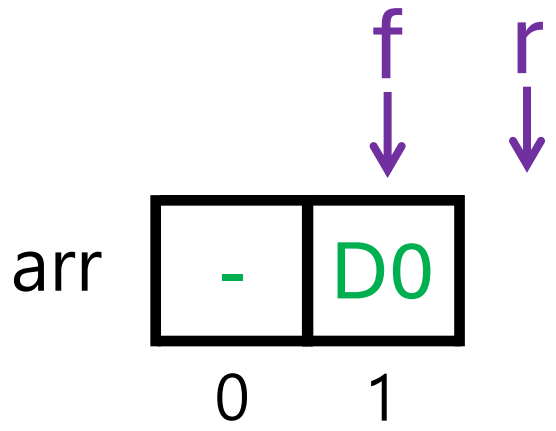


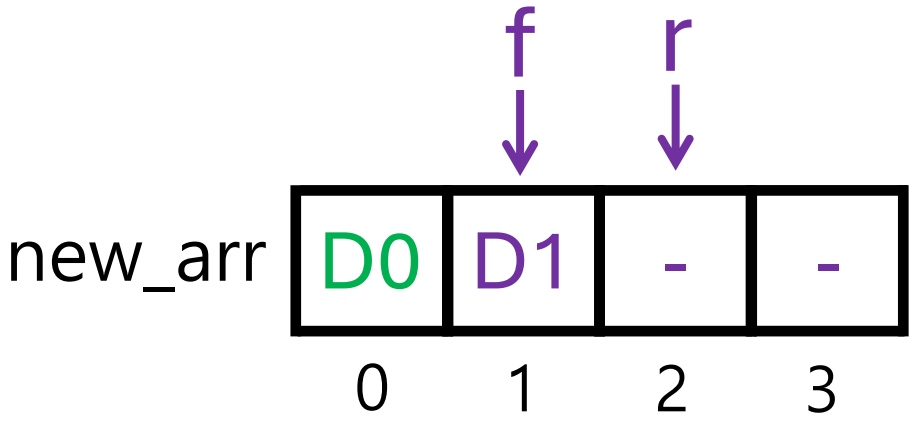
배열의 길이가 증가하는 Queue

- 배열의 크기 이상으로 원소를 enqueue하는 경우 배열의 크기를 2배로 늘려 새로운 배열 할당
- 기존의 배열에 있는 원소를 크기가 증가한 새로운 배열에 옮겨주고, 새로운 배열을 queue으로 사용

예외 없는 enqueue

```
1  void enqueue(const T &data) {
2      if (size() == cap) {
3          cap *= 2;
4          T *new_arr = new T[cap];
5          for (int i = 0; i < n; i++) {
6              new_arr[i] = arr[(f + i) % n];
7          }
8
9          delete[] arr;
10         arr = new_arr;
11         f = 0;
12         r = n;
13     }
14
15     arr[r] = data;
16     r = (r + 1) % cap;
17     n++;
18 }
```

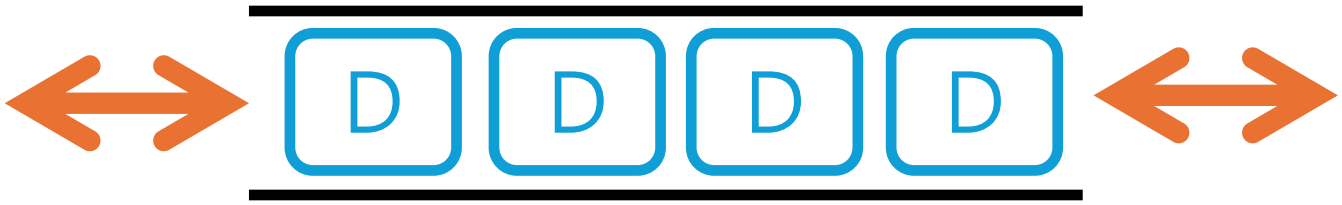




Double-Ended Queue

Double-Ended Queue

- 큐의 front와 rear 모두에서 삽입과 삭제 연산이 가능
- 양쪽 방향으로 모두 입출력이 가능한 자료구조
- Stack으로도, Queue로도 사용 가능



deque의 ADT

- int size() : 현재 deque에 있는 원소 개수를 반환
- bool empty() : deque에 원소가 비어있는지 여부를 반환
- T &front() : deque에서 가장 앞에 저장된 원소를 반환
- T &back() : deque에서 가장 뒤에 저장된 원소를 반환

deque의 ADT

- void insertFront(T &data) : deque에서 가장 앞에 원소를 삽입
- void insertBack(T &data) : deque에서 가장 뒤에 원소를 삽입
- void eraseFront() : deque에서 가장 앞에 있는 원소를 삭제
- void eraseBack() : deque에서 가장 뒤에 있는 원소를 삭제