SWIFT
Data Structure - Queue

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Queue

큐(Queue)는 스택과 마찬가지로 삽입, 삭제의 위치와 방법이 제한되어있는 유한 순서 리스트 형태의 자료구조입니다.

스택과 다르게 큐는 한쪽에서는 추가, 다른 한쪽에서는 삭제가 이루어지느 구조입니다.

큐는 우리 일상 생활에서 특정 장소에서 줄을 서서 대기하는 것과 같은 구조입니다.

즉 먼저 대기한 사람이 가장 먼저 나가는 FIFO(First - In - First - Out)구조입니다.

Queue

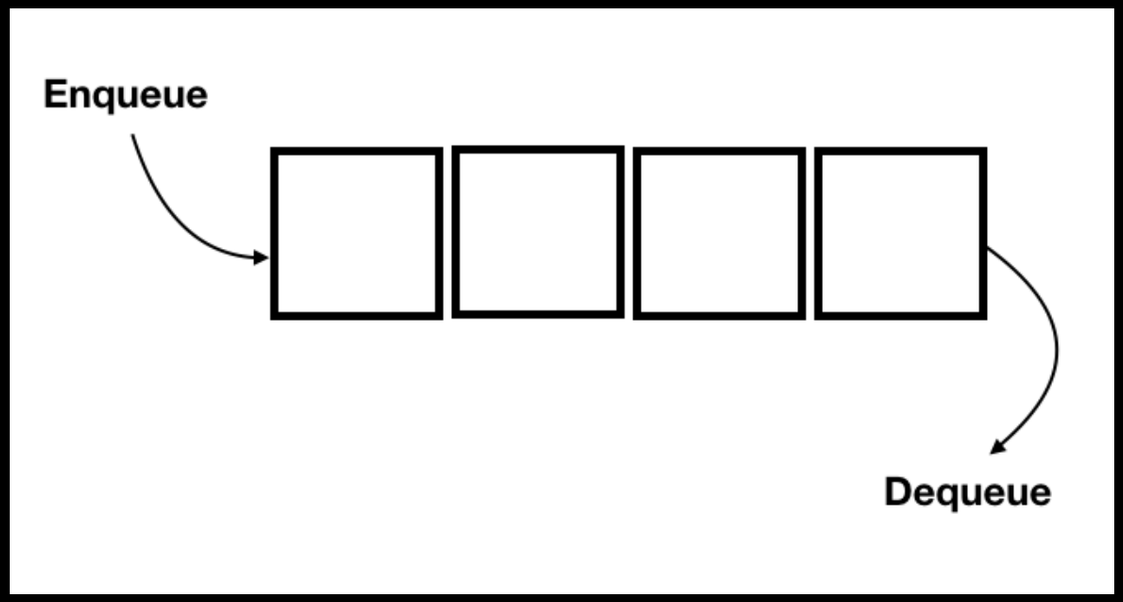
큐(Queue)는 구성하는 방식에 따라서 아래와 같은 방식 등이 있습니다.

1. Array를 사용해서 큐 구현
2. Doubly Linked List를 사용해 큐 구현
3. Two Stack을 사용해서 큐 구현
4. Ring Buffer를 사용해서 큐 구현

Concept

Array(배열)를 사용한 Queue의 기본 동작 흐름은 아래와 같습니다.

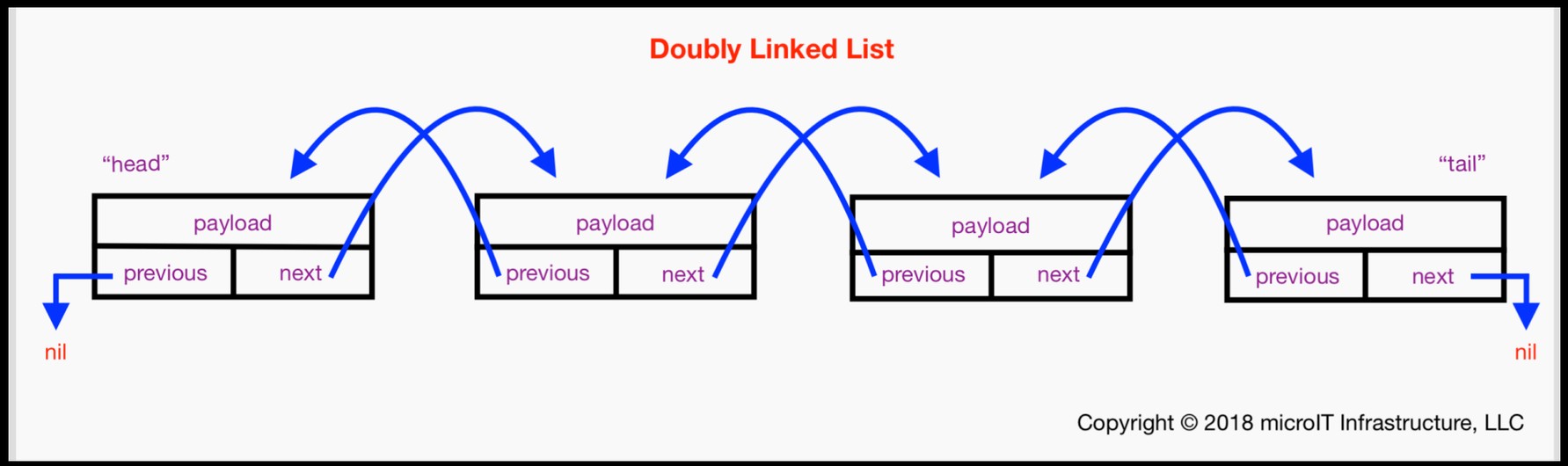
- 특정 크기의 배열을 초기에 설정
- Enqueue(추가) 속도는 O(1)
- Dequeue(삭제) 시에는 삭제 이후에 있는 요소들이 한칸 앞으로 이동, 속도는 O(n)
- 공간이 꽉차면 2배씩 배열 크기를 늘림, 스위프트의 배열에서는 먼저 크게 늘려놓으니 빈 공간에 추가할 땐 수행속도가 빠름



Concept

Doubly Linked List를 사용한 Queue의 기본 동작 흐름은 아래와 같습니다.

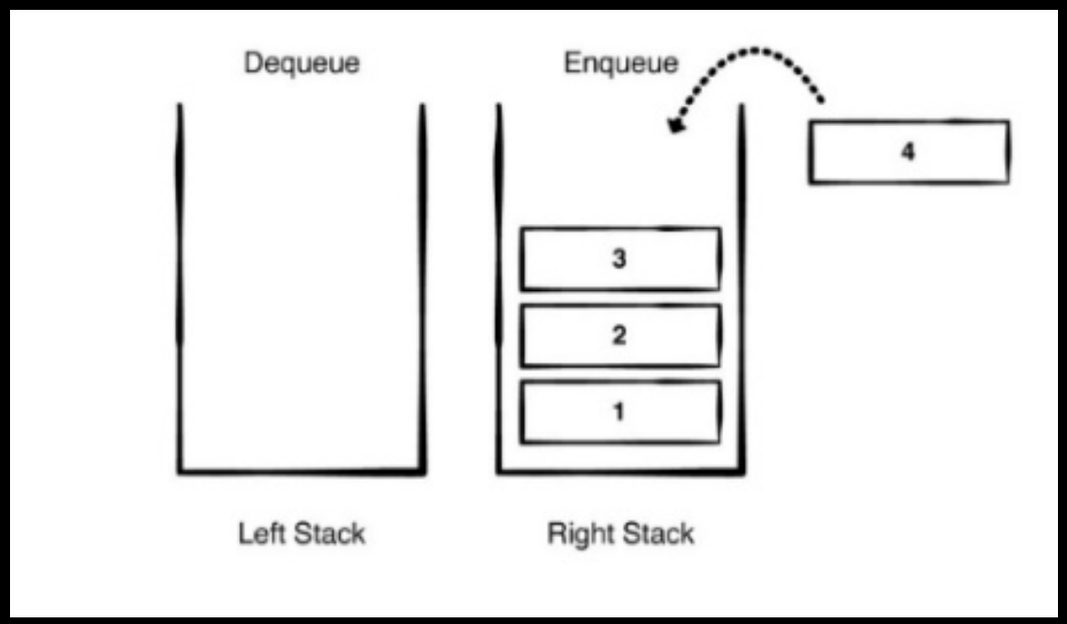
- 링크드 리스트를 활용하여 큐를 구현
- 추가, 삭제 속도가 O(1)으로 빠름
- 삭제 시에 데이터 재정렬 과정이 필요없음



Concept

Two Stack을 사용한 Queue의 기본 동작 흐름은 아래와 같습니다.

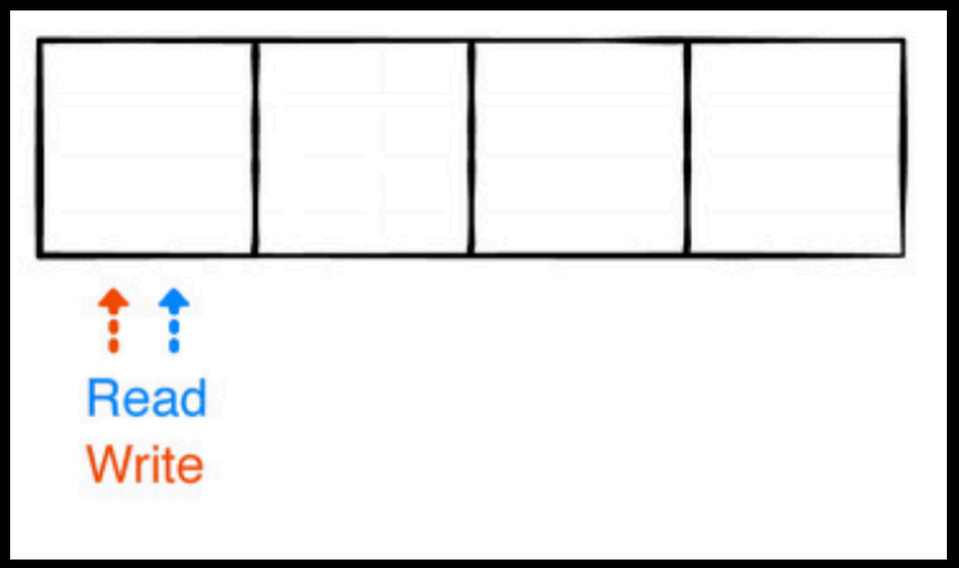
- 두 개의 스택으로 큐를 구현
- Right Stack은 Enqueue, Left Stack은 Dequeue 전용 스택
- 추가(Enqueue) 시에 우측 스택에 추가, 수행 속도는 O(1)
- 삭제(Dequeue) 시에 우측 스택의 요소를 거꾸로 정렬 후 좌측 스택 이동 후 마지막 요소 삭제(popLast), 수행 속도는 O(1)

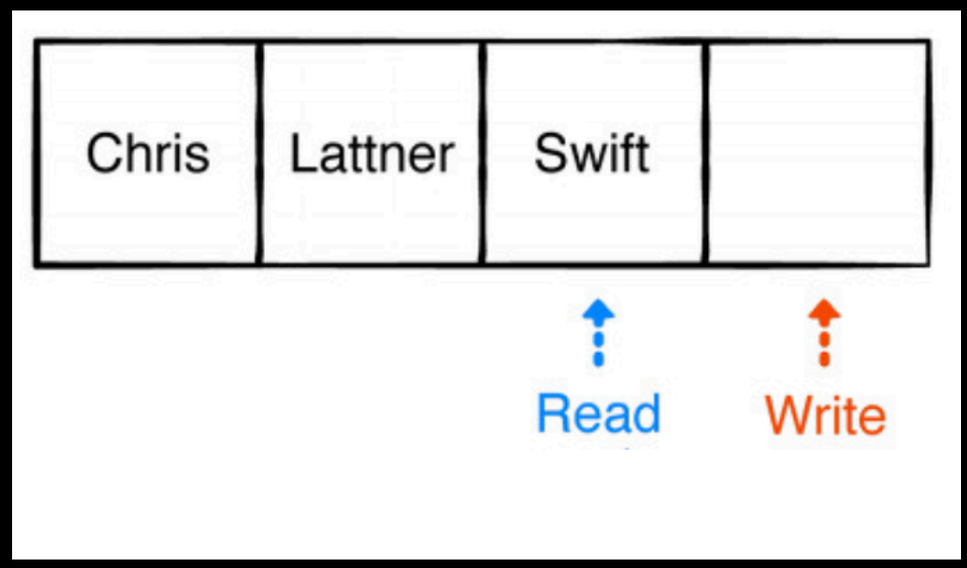


Concept

Ring Buffer를 사용한 Queue의 기본 동작 흐름은 아래와 같습니다.

- 고정된 크기의 배열을 사용
- 배열이 큐의 값들로 가득 차면 다시 첫 번째 아이템으로 돌아옴
- 추가는 Write Index, 삭제는 Read Index를 사용
- 큐의 공간이 꽉차면 더이상 추가 불가
- 크기가 고정되 있어서 큐에 들어오는 값의 개수가 예측 불가능할 경우엔 사용할 수 제한





Features

Queue의 특징을 살펴보면 아래와 같습니다.

- 데이터를 한 곳에서는 삽입 한 곳에서는 삭제가 되는 구조
- FIFO(First In First Out) 방식
- 입력된 순서대로 순차적으로 처리
- 운영체제의 작업 스케줄링 등 순차적인 처리 분야에서 활용

Implementation

Swift를 활용하여 Queue를 구현해보겠습니다. 본 강의에서는 가장 단순한 구조인 배열을 활용한 큐를 살펴보도록 하겠습니다.

우선 필요한 메소드는 아래와 같습니다.

- init : 리스트를 초기화하는 함수
- enqueue : 데이터 입력
- dequeue : 첫번째 데이터 삭제
- peak : 첫번째 데이터 반환
- removeAll : 모든 데이터 삭제
- count : 현재 리스트의 크기를 반환
- capacity : 리스트의 최대 허용 공간 크기를 반환
- isFull : 리스트가 모두 찼는지 체크
- isEmpty : 현재 리스트의 크기가 비어있는지 체크

Implementation

```
class Queue<T> {
    internal var array = Array<T>()
    public init() {}

    public func enqueue(_ element: T) {
        array.append(element)
    }

    public func dequeue() -> T? {
        return array.removeFirst()
    }

    public func peek() -> T? {
        return array.first
    }

    public func removeAll() {
        array.removeAll()
    }
}
```

Implementation

```
extension Queue {
    public var count: Int {
        return array.count
    }

    public var capacity: Int {
        get { return array.capacity }
        set { array.reserveCapacity(newValue) }
    }

    public func isFull() -> Bool {
        return count == array.capacity
    }

    public func isEmpty() -> Bool {
        return array.isEmpty
    }
}
```

Implementation

데이터 반복(Iterator) 제어를 위하여 아래의 프로토콜들을 체택합니다.

```
public struct QueueIterator<T> : IteratorProtocol {
    var currentElement: [T]

    public mutating func next() -> T? {
        if !self.currentElement.isEmpty {
            return currentElement.removeFirst()
        }else {
            return nil
        }
    }
}

extension Queue : Sequence {
    public func makeIterator() -> QueueIterator<T> {
        return QueueIterator(currentElement: self.array)
    }
}
```

Implementation

```swift
let queue = Queue<Int>()

queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
queue.enqueue(4)
queue.enqueue(5)

// 현재 데이터 카운트 : 5
print(queue.count)

// 최대 리스트 크기 : 8 (Swift의 Array는 기본 8개씩 크기를 할당하며 동적으로 2배씩 늘어난다.)
print(queue.capacity)
print(queue.isFull()) // false

for node in queue {
    print(node)
    // 1
    // 2
    // 3
    // 4
    // 5
}
```

Implementation

```
queue.enqueue(6)
queue.enqueue(7)
queue.enqueue(8)

print(queue.capacity) // 8
print(queue.isFull()) // true

queue.dequeue() // 첫 번째 요소 삭제

for node in queue {
    print(node)
    // 2
    // 3
    // 4
    // 5
    // 6
    // 7
    // 8
}
```

References

[1] [스위프트 : 자료구조] 큐 : Queue : Sequence, Collection 프로토콜 지향 큐 구현하기 : https://the-brain-of-sic2.tistory.com/39

[2] [Swift 자료구조 ch05] Queue : https://kor45cw.tistory.com/241

[3] [Swift 자료구조 ch06] Circular Queue : https://kor45cw.tistory.com/242

[4] Swift로 작성해보는 기본 자료구조 - Stack, Queue : https://wlaxhrl.tistory.com/87

[5] 자료구조 - 큐(Queue) with Swift : https://lazyowl.tistory.com/125

References

[6] [자료구조] 큐 Queue / 배열을 이용한 큐의 구현 : https://ppomelo.tistory.com/57

[7] Swift Algorithm Club: Swift Queue Data Structure : https://torpedo87.github.io/algorithm/swift-queue-data-structure.html

[8] [Data Structure] 자료구조 - 큐(Queue) : https://palpit.tistory.com/145

[9] [자료구조]스택과 큐. (stack and queue) : https://winplz.tistory.com/entry/자료구조스택과-큐-stack-and-queue

[10] [Data Structure] 자료구조 - 큐(Queue) : https://palpit.tistory.com/145

Thank you!