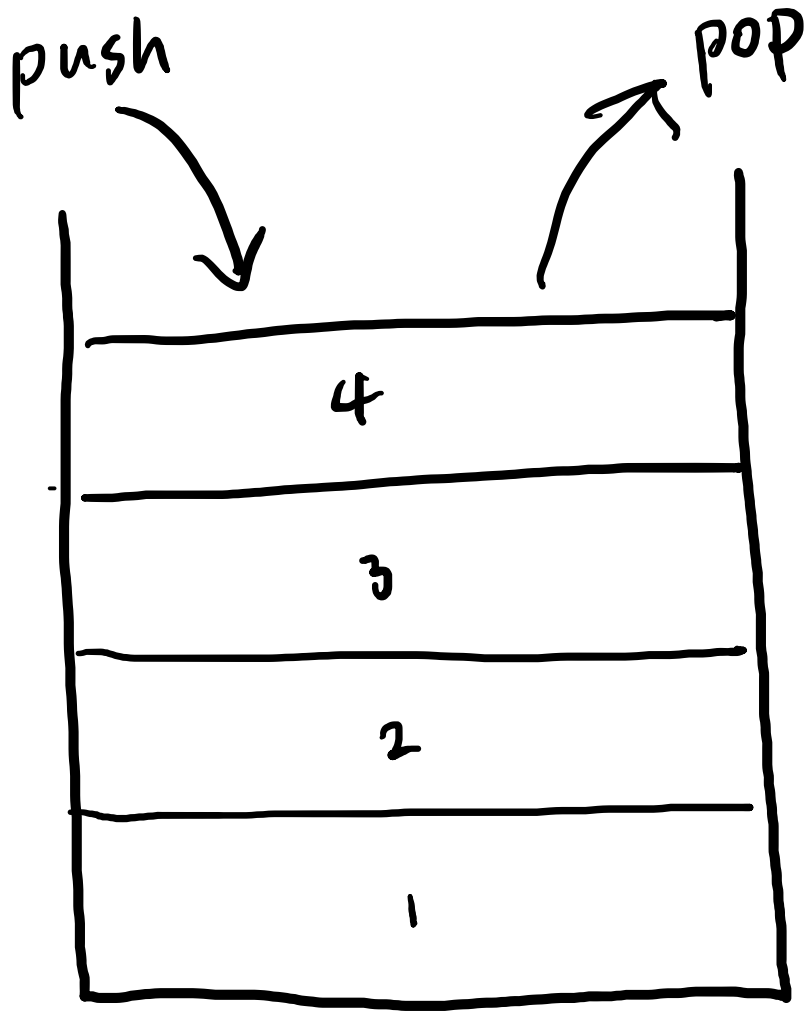
스택과 큐

스택과 큐

- 자료를 넣고, 빼는 동작을 하는 자료구조
- 시간 복잡도 (Big-O) 는,
  - 삽입: O(1)
  - 삭제: O(1) (궁금한 점: 큐의 삭제를 O(n) 이라고 말하는 블로그도 있던데, 왜일까요?)
  - 검색: O(n)

스택

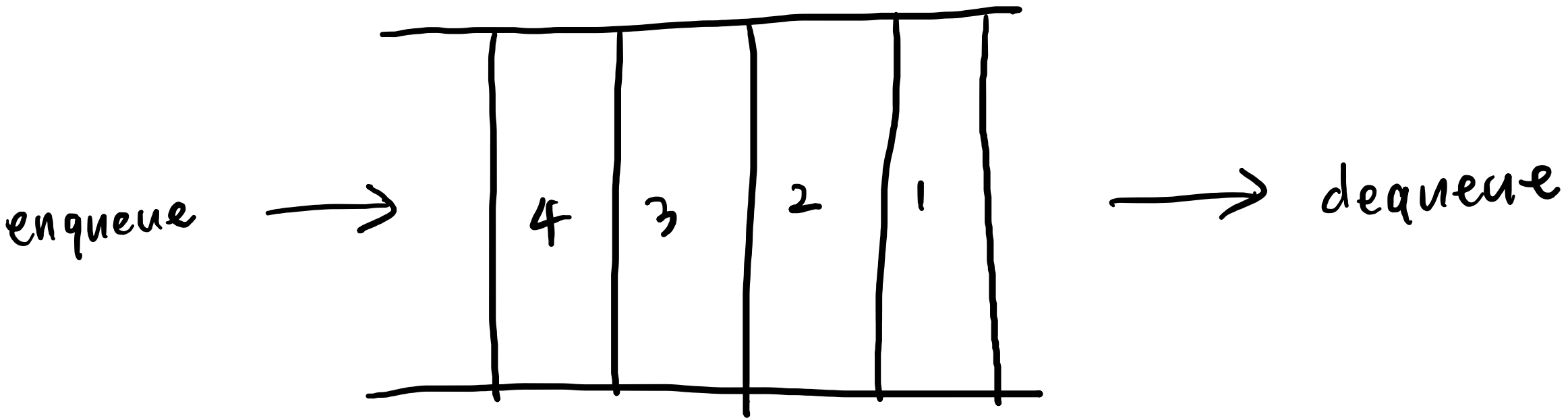


1 → 2 → 3 → 4

Last In, First Out

4 → 3 → 2 → 1

큐



1 → 2 → 3 → 4

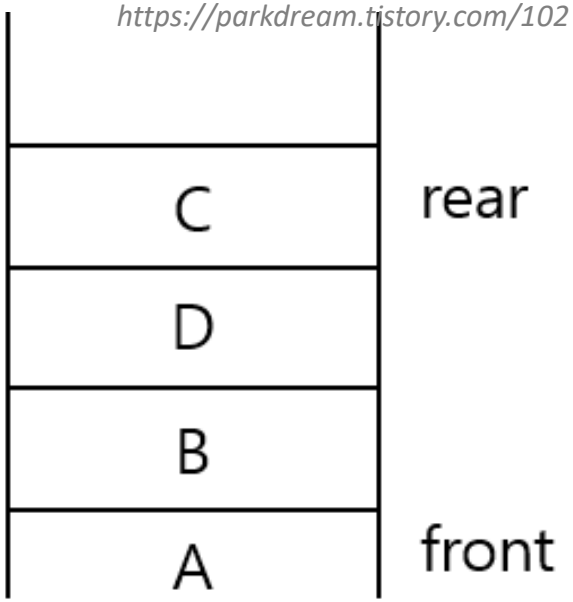
First In, First Out

1 → 2 → 3 → 4

큐: 종류

선형 큐

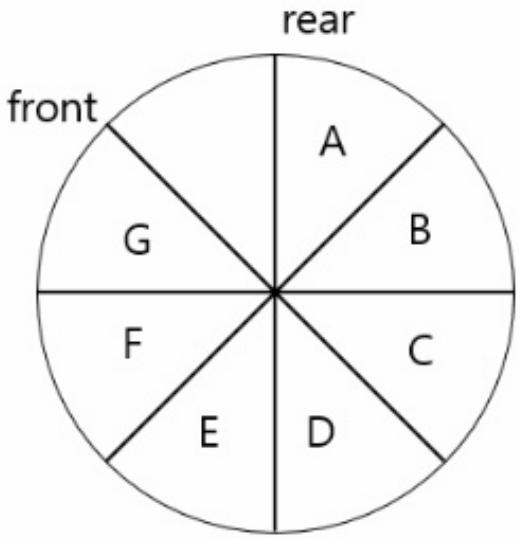
https://parkdream.tistory.com/102



1차원 배열을 이용한 큐입니다. 보통 시작과 끝을 가리키는 변수가 있어서 공백, 포화 상태를 확인합니다.

원형 큐

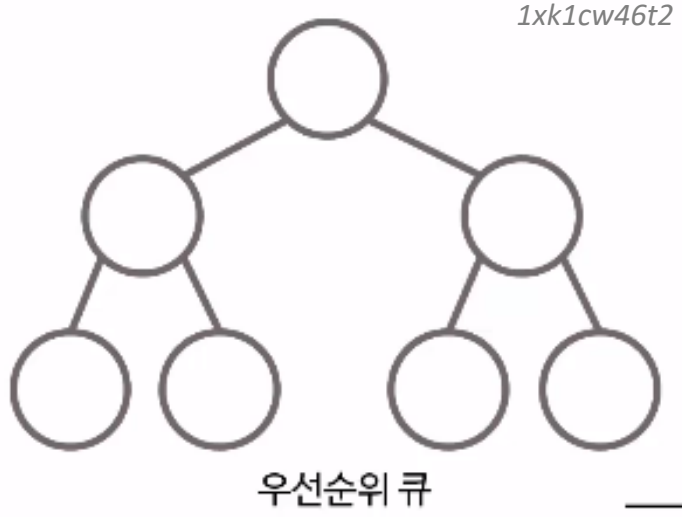
https://parkdream.tistory.com/102



dequeue 할 때마다 빈 공간이 생기는 선형 큐의 단점을 보완합니다. 시작과 끝을 구분하기 위한 공간 하나가 필요합니다.

우선순위 큐

https://velog.io/@pa324/%EC%9A%B0%EC%84%A0%EC%88%9C%EC%9C%84-%ED%81%90-1xk1cw46t2



FIFO가 아니며, 우선순위가 높은 순으로 드나듭니다. 삽입, 삭제시 원소를 재배치하므로 메모리 낭비가 큽니다.

스택: 파이썬

1. 파이썬 기본 문법
   - 자료 넣기push: `stack.append(value)` ➔ 맨 뒤에 값 삽입
   - 자료 빼기pop: `stack.pop()` ➔ 맨 뒤의 값부터 제거 (LIFO)

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

https://docs.python.org/ko/3/tutorial/datastructures.html#using-lists-as-stacks

스택: 파이썬

2. 기본 queue 모듈
   - `queue.LifoQueue()`

class queue.**LifoQueue**(*maxsize=0*)

LIFO 큐의 생성자. *maxsize*는 큐에 배치할 수 있는 항목 수에 대한 상한을 설정하는 정수입니다. 일단, 이 크기에 도달하면, 큐 항목이 소비될 때까지 삽입이 블록 됩니다. *maxsize*가 0보다 작거나 같으면, 큐 크기는 무한합니다.

https://docs.python.org/ko/3/library/queue.html#queue.LifoQueue

큐: 파이썬

1. 파이썬 기본 문법
   - 자료 넣기enqueue: `q.append(value)` ➔ 맨 뒤에 값 삽입
   - 자료 빼기dequeue: `q.pop(0)` ➔ 맨 앞의 값부터 제거 (FIFO)

큐: 파이썬

2. 기본 queue 모듈
   - `queue.Queue()` – 선형 큐
   - `queue.PriorityQueue()` – 우선순위 큐

https://docs.python.org/ko/3/library/queue.html

class queue.**Queue**(*maxsize=0*)

FIFO 큐의 생성자. *maxsize*는 큐에 배치할 수 있는 항목 수에 대한 상한을 설정하는 정수입니다. 일단, 이 크기에 도달하면, 큐 항목이 소비될 때까지 삽입이 블록 됩니다. *maxsize*가 0보다 작거나 같으면, 큐 크기는 무한합니다.

class queue.**LifoQueue**(*maxsize=0*)

LIFO 큐의 생성자. *maxsize*는 큐에 배치할 수 있는 항목 수에 대한 상한을 설정하는 정수입니다. 일단, 이 크기에 도달하면, 큐 항목이 소비될 때까지 삽입이 블록 됩니다. *maxsize*가 0보다 작거나 같으면, 큐 크기는 무한합니다.

class queue.**PriorityQueue**(*maxsize=0*)

우선순위 큐의 생성자. *maxsize*는 큐에 배치할 수 있는 항목 수에 대한 상한을 설정하는 정수입니다. 일단, 이 크기에 도달하면, 큐 항목이 소비될 때까지 삽입이 블록 됩니다. *maxsize*가 0보다 작거나 같으면, 큐 크기는 무한합니다.

가장 낮은 값을 갖는 항목이 먼저 꺼내집니다 (가장 낮은 값을 갖는 항목은 `sorted(list(entries))[0]` 에 의해 반환되는 항목입니다). 항목의 전형적인 패턴은 `(priority_number, data)` 형식의 튜플입니다.

data 요소를 비교할 수 없으면, 데이터는 데이터 항목을 무시하고 우선순위 숫자만 비교하는 클래스로 감쌀 수 있습니다:

```
from dataclasses import dataclass, field
from typing import Any

@dataclass(order=True)
class PrioritizedItem:
    priority: int
    item: Any=field(compare=False)
```

class queue.**SimpleQueue**

상한 없는 FIFO 큐의 생성자. 단순 큐에는 작업 추적과 같은 고급 기능이 없습니다.

큐: 파이썬

3. collections의 deque 모듈

리스트를 큐로 사용하는 것도 가능한데, 처음으로 넣은 요소가 처음으로 꺼내지는 요소입니다 (《first-in, first-out》); 하지만, 리스트는 이 목적에는 효율적이지 않습니다. 리스트의 끝에 덧붙이거나, 끝에서 꺼내는 것은 빠르지만, 리스트의 머리에 덧붙이거나 머리에서 꺼내는 것은 느립니다 (다른 요소들을 모두 한 칸씩 이동시켜야 하기 때문입니다).

https://docs.python.org/ko/3/tutorial/datastructures.html#using-lists-as-queues

큐를 구현하려면, 양 끝에서의 덧붙이기와 꺼내기가 모두 빠르도록 설계된 collections.deque 를 사용하세요. 예를 들어:

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

collections.deque 의 삭제 (popleft) 는 시간 복잡도가 O(1)로 매우 빠르다고 합니다.
(https://daimhada.tistory.com/107)

감사합니다.