ADVANCED DATA STRUCTURE

Over the Pre-defined Data Structure

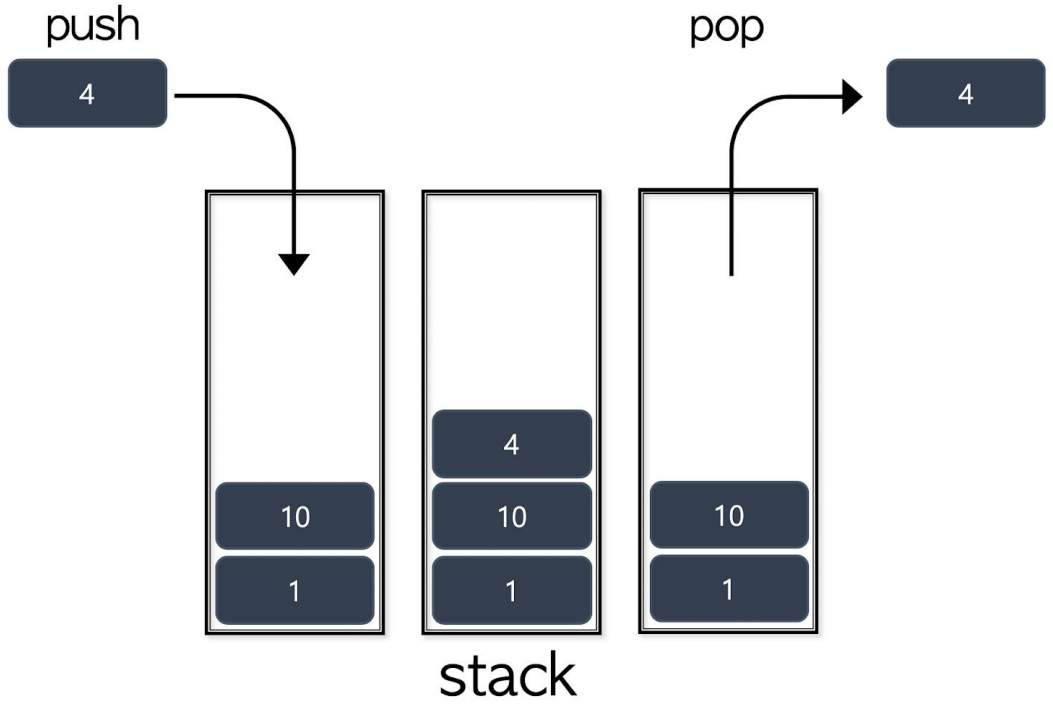
파이썬에는 기본적으로 몇 가지 자료구조가 사전에 선언

- list, set, dict 등등

좀 더 강력한 자료구조를 사용하고 싶다면...?

- 기존 자료구조를 활용
- Python Standard Library로 대부분 해결!
  - Stack & Queue
  - Linked List
  - 우선 순위 큐
  - 기본값이 있는 dictionary
  - 개수를 세는 자료구조
  - ...

Stack

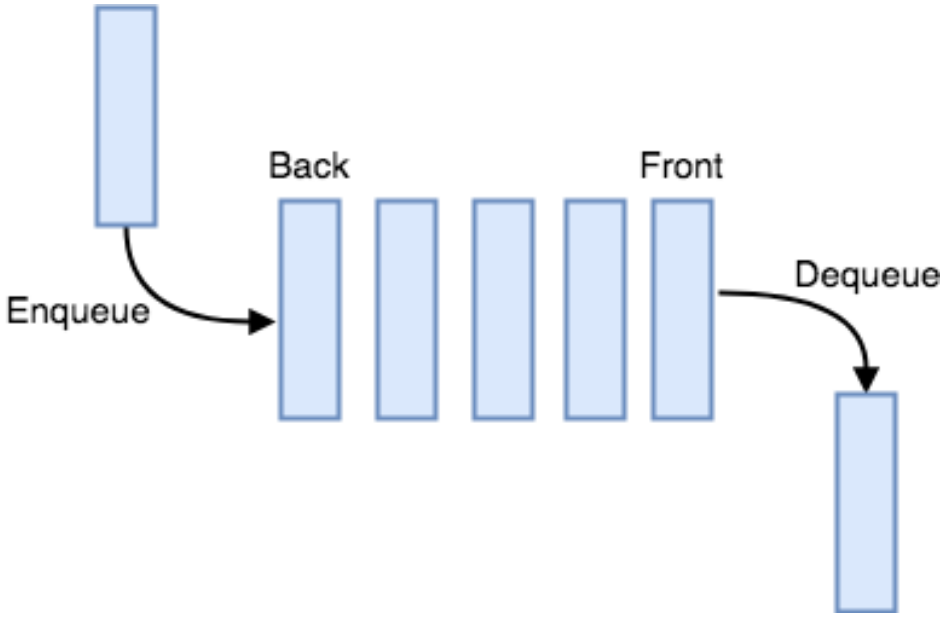



```
>>> a = [1, 10]

>>> a.append(4)
>>> a.append(20)

>>> a.pop()
20
>>> a.pop()
4
```

스택 구조는 기존 List를 활용

- 동적 배열이기 때문에 push와 pop이 *O(1)*

Queue




```
>>> a = [1, 10]

>>> a.insert(0, 20)
>>> a.insert(0, 15)

>>> a.pop()
10
>>> a.pop()
1
```

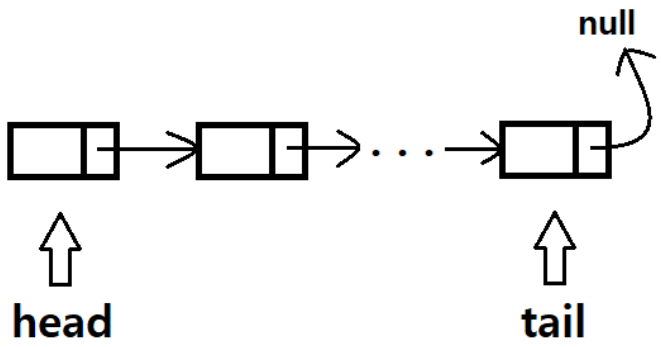
큐 구조를 기존 List로 만들 경우

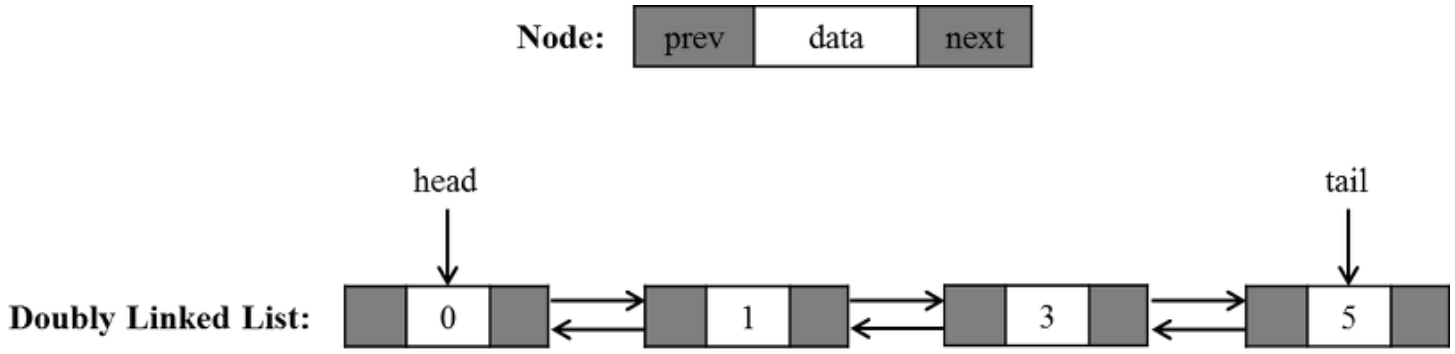
- List는 한 쪽 방향으로 열려 있는 동적 배열

  → 처음 위치 삽입 혹은 삭제가 ***O(N)*** 걸림

- 다른 형태의 데이터 구조가 필요

Linked List





Queue 구조 구현을 위해선 연결 리스트 자료 구조가 필요

- 양쪽으로 자유로운 입출력
- 중간 참조는 오래 걸리나, 큐 구조에선 상관 없음

파이썬 ***collection*** Library의 ***deque***를 사용!

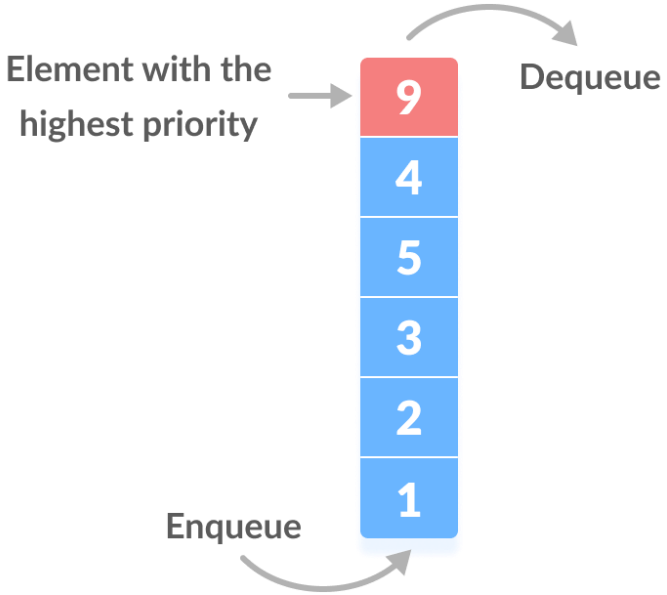
Deque

```
>>> from collections import deque                # deque import
>>> queue = deque([10, 5, 12])                   # deque 생성

>>> queue.appendleft(16)                         # 왼쪽 삽입
>>> queue.pop()                                  # 오른쪽 삭제
12

>>> queue.append(20)                             # 오른쪽 삽입
>>> queue.popleft()                              # 왼쪽 삭제
16

>>> queue
deque([10, 5, 20])
>>> deque(reversed(queue))                       # deque 뒤집기 O(N)
deque([20, 5, 10])
```

이중 연결 리스트의 함수를 지원

https://docs.python.org/ko/3/library/collections.html#collections.deque

Priority Queue



최소/최대값을 빠르게 구할 수 있을까?

- min/max 함수 사용 → $O(N)$
- 이진 검색을 위해선 내부가 정렬되어 있어야함

내부가 항상 정렬된 자료구조를 사용하고 싶다면?

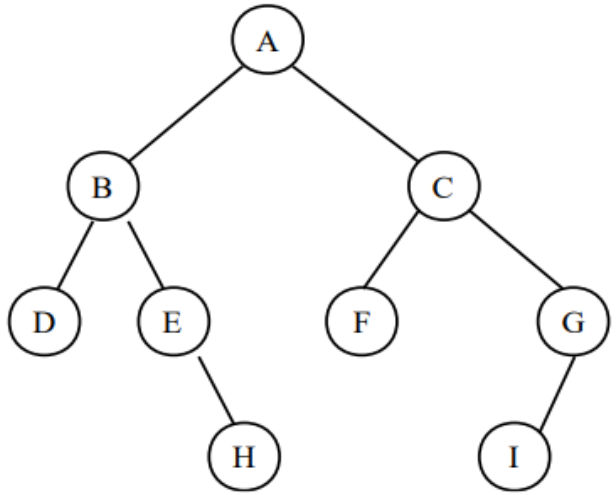
- 입력할 때마다 sorted 사용 → **값 삽입/삭제: $O(N \log N)$**
- 리스트 정렬 후 값을 중간에 삽입/삭제 → $O(N)$

→ Priority Queue 가 필요한 순간

파이썬 ***heapq*** Library의 ***heapq***를 사용!

Heapq

```
>>> import heapq
>>> queue = [5, 2, 8, 4]
>>> heapq.heapify(queue)            # Heap 초기화, O(N log N)
>>> queue                           # Heap[k] <= heap[2*k+1], heap[2*k+2]
[2, 4, 8, 5]
>>> queue[0]                        # Heap top: 가장 작은 값 위치, Min Heap
2

>>> heapq.heappush(queue, 3)        # Heap Push, O(log N)
>>> heapq.heappush(queue, 6)
>>> queue[0]
2
>>> item = heapq.heappop(queue)     # Heap Pop, O(log N)
>>> item, queue[0]
(2, 3)

>>> item = heapq.heappushpop(queue, 7)      # Push하고 Pop하는 것보다 빠름
>>> item, queue[0]
(3, 4)
```



리스트를 이진 트리 형식으로 사용 → 값 삽입/삭제: $O(\log N)$

Defaultdict

```
>>> d = {"first": 0}

>>> d["second"]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'second'

>>> d.get("second", "없어요")
'없어요'
```

파이썬 기본 dictionary는 키가 없을 경우 에러

- 에러 발생을 막기 위해선 get 메소드의 default 값 지정 필요

https://docs.python.org/ko/3/library/collections.html#collections.defaultdict

Defaultdict

```
text = """
유구한 역사와 전통에 빛나는 우리 대한국민은
3·1운동으로 건립된 대한민국임시정부의 법통과 불의에 항거한 4·19민주이념을 계승하고,
조국의 민주개혁과 평화적 통일의 사명에 입각하여 정의·인도와 동포애로써 민족의 단결을 공고히 하고,
모든 사회적 폐습과 불의를 타파하며,
자율과 조화를 바탕으로 자유민주적 기본질서를 더욱 확고히 하여 정치·경제·사회·문화의 모든 영역에 있어서 각인의 기회를 균등히 하고,
능력을 최고도로 발휘하게 하며, 자유와 권리에 따르는 책임과 의무를 완수하게 하여,
안으로는 국민생활의 균등한 향상을 기하고 밖으로는 항구적인 세계평화와 인류공영에 이바지함으로써
우리들과 우리들의 자손의 안전과 자유와 행복을 영원히 확보할 것을 다짐하면서
1948년 7월 12일에 제정되고 8차에 걸쳐 개정된 헌법을 이제 국회의 의결을 거쳐 국민투표에 의하여 개정한다.
1987년 10월 29일.
"""
```

```
characters = {}

for char in text:
    count = characters.get(char, None)

    if count is None:
        characters[char] = 0

    characters[char] += 1
```

```
from collections import defaultdict

characters = defaultdict(int) # 기본값 int()
# characters = defaultdict(lambda: 0)
for char in text:
    characters[char] += 1
```

Dictionary의 기본 값을 지정 가능

https://docs.python.org/ko/3/library/collections.html#collections.defaultdict

Counter

```
text = """
유구한 역사와 전통에 빛나는 우리 대한국민은
3·1운동으로 건립된 대한민국임시정부의 법통과 불의에 항거한 4·19민주이념을 계승하고,
조국의 민주개혁과 평화적 통일의 사명에 입각하여 정의·인도와 동포애로써 민족의 단결을 공고히 하고,
모든 사회적 폐습과 불의를 타파하며,
자율과 조화를 바탕으로 자유민주적 기본질서를 더욱 확고히 하여 정치·경제·사회·문화의 모든 영역에 있어서 각인의 기회를 균등히 하고,
능력을 최고도로 발휘하게 하며, 자유와 권리에 따르는 책임과 의무를 완수하게 하여,
안으로는 국민생활의 균등한 향상을 기하고 밖으로는 항구적인 세계평화와 인류공영에 이바지함으로써
우리들과 우리들의 자손의 안전과 자유와 행복을 영원히 확보할 것을 다짐하면서
1948년 7월 12일에 제정되고 8차에 걸쳐 개정된 헌법을 이제 국회의 의결을 거쳐 국민투표에 의하여 개정한다.
1987년 10월 29일.
"""
```

```
from collections import defaultdict

characters = defaultdict(int) # 기본값 int()
# characters = defaultdict(lambda: 0)
for char in text:
    characters[char] += 1
```

```
from collections import Counter

characters = Counter(text)
print(characters)
```

뭔가를 세는 데 최적화된 데이터 구조

https://docs.python.org/ko/3/library/collections.html#collections.defaultdict

Counter

Dictionary 처럼 생성 및 관리

```
>>> c = Counter({"Korean": 2, "English": 3})
>>> c
Counter({'English': 3, 'Korean': 2})

>>> c.keys()                                # 일반적인 dict과 차이 없음
Dict_keys([ ' Korean ' ,  ' English ' ])
>>> c.values()
Dict_values([2, 3])
>>> c[ " Korean " ]
2

>>> list(c.elements())                      # 모든 요소 반환
['Korean', 'Korean', 'English', 'English', 'English']
```

kwargs로 생성

```
>>> c = Counter(Korean=2, English=3)
>>> c
Counter({'English': 3, 'Korean': 2})
```

https://docs.python.org/ko/3/library/collections.html#collections.Counter

Counter

집합 연산 지원

```
>>> from collections import Counter
>>> a = Counter([1, 1, 2, 2, 2, 3])
>>> b = Counter([2, 3, 3, 4])

>>> a + b                                    # 횟수 더하기
Counter({2: 4, 3: 3, 1: 2, 4: 1})

>>> a & b                                    # 교집합
Counter({2: 1, 3: 1})

>>> a | b                                    # 합집합
Counter({2: 3, 1: 2, 3: 2, 4: 1})

>>> a - b                                    # 차집합
Counter({1: 2, 2: 2})
```

https://docs.python.org/ko/3/library/collections.html#collections.Counter

Named Tuple

```python
class Coords3D:
    def __init__(self, x, y, z):
        self._x = x
        self._y = y
        self._z = z

    @property
    def x(self):
        return self._x

    @property
    def y(self):
        return self._y

    @property
    def z(self):
        return self._z
```

데이터만을 담기 위한 클래스 사용

- 클래스 선언 및 getter 작성 필요
- 숫자로 Indexing 불가능
  - __getitem__ 작성 필요

→ 귀찮음

- 간단한 데이터 구조인데 굳이 클래스를..?

https://docs.python.org/ko/3/library/collections.html#collections.namedtuple

Named Tuple

```
point = (10, 20, 30)

print(point[0]) # 뭐가 x였지…?
print(point[1])
```

- 튜플로 관리할 시 쉽게 자료구조를 만들 수는 있음
- 그러나 관리하기에는 불편
  - Attribute명으로 접근이 불가능

```
from collections import namedtuple

# 새로운 타입 생성
Coords3D = namedtuple("Coords3D", ['x', 'y', 'z'])

point = Coords3D(10, 20, z=30)
print(point.x)          # Attribute 이름으로 참조
Print(point[1])         # Index로 참조
print(*point)           # Tuple Unpacking

point[1] += 1           # Error 발생
```

Named Tuple

- 각 튜플 원소에 이름 붙이기 가능
- 클래스가 아니라 튜플이다
  - Unpacking 가능

https://docs.python.org/ko/3/library/collections.html#collections.namedtuple

Dataclass

```python
from dataclasses import dataclass

@dataclass
class Coords3D:
    x: float
    y: float
    z: float = 0

    def norm(self) -> float:
        return (self.x ** 2 + self.y ** 2 + self.z ** 2) ** .5

point = Coords3D(10, 20, z=30)
print(point)
print(point.norm())
```

Pythonic한 데이터 클래스를 위해선 ***dataclasses***의 ***dataclass*** 데코레이터 활용

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