스택 / 큐

탐색 : 많은 양의 데이터 중에서 원하는 데이터를 찾는 과정

→ 그래프, 트리 등의 자료구조 안에서 탐색을 하는 문제! ex) DFS / BFS

자료구조 : '데이터를 표현하고 관리하고 처리하기 위한 구조'
기본 자료구조인 스택과 큐는 다음의 두 핵심적인 함수로 구성됨

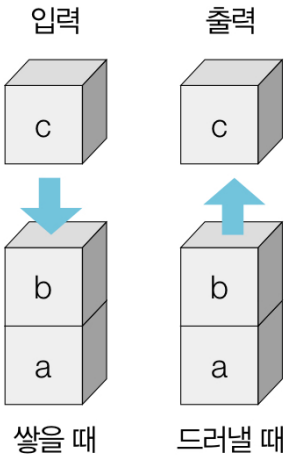
- 삽입(Push) : 데이터를 삽입
- 삭제(Pop) : 데이터를 삭제

+) 오버플로: 특정한 자료구조가 수용할 수 있는 데이터의 크기를 이미 가득 찬 상태에서 삽입 연산을 수행할때 발생 → 저장 공간을 벗어나 데이터가 넘쳐흐를 때 발생

언더플로: 데이터가 전혀 들어있지 않은 상태에서 삭제 연산을 수행하면 발생

스택

= 박스 쌓기/ 나중에 넣은 데이터를 먼저 반환 | Last In First Out (LIFO)



- 선입후출 구조 or 후입선출 구조

- 파이썬에서는 스택을 이용할때 별도의 라이브러리 없이 기본 리스트에서 append()와 pop() 메서드만 사용하면 됨!
- `append()`: 리스트의 가장 뒤쪽에 데이터를 삽입
- `pop()`: 리스트의 가장 뒤쪽에서 데이터 꺼냄, return이 있으면서, 리스트가 변함

[프로그래머스] Lv2. 올바른 괄호

```
def solution(s):
    answer = True
    stack = []

    if s[0] == ')':
        return False

    for i in s:
        if len(stack) == 0 and i == ')':
            return False

        if i == '(':
            stack.append(i)
        else:
            stack.pop()


    return True if len(stack) == 0 else False
```

큐

= 대기줄 = 먼저 온 사람은 먼저 들어가기! = 나중에 온 사람일수록 나중에 들어가기! = 선입선출
| FIFO (First In First Out)

- 큐 구현을 위해 **deque 라이브러리** 사용 : 스택과 큐의 장점을 모두 채택, 리스트 자료형에 비해 효율적임!

```
from collections import deque
```

- deque 객체를 리스트 자료형으로 바꾸고 싶으면 list() 메서드 이용!
- `append()`를 이용해 **put,** `pop(0)` or `popleft()` 을 이용해 **get** !

[프로그래머스] Lv2. 다리를 지나는 트럭

```
# 다리(queue)에 있는 트럭을 하나씩 빼고, 다리 무게에 여유있으면 트럭을 추가
# 무게에 여유가 없으면, 무게 0짜리인 트럭 넣어주기!
# 이때, sum 이용하면 시간초과걸리므로, bridge_weightr같은 다리무게 변수 추가해주기!

from collections import deque

def solution(bridge_length, weight, truck_weights):
    answer = 0
    bridge_weight = 0
    truck_list = deque(truck_weights)
    in_bridge = deque([0] * bridge_length, maxlen = bridge_length)

    while truck_list or bridge_weight != 0:
        answer += 1
        bridge_weight -= in_bridge[0]
        if truck_list and (bridge_weight + truck_list[0] <= weight):
            bridge_weight += truck_list[0]
            in_bridge.append(truck_list.popleft())
        else:
            in_bridge.append(0)
    return answer
```