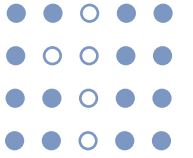


알고리즘 스터디 2주차

발표자 장수현

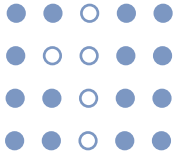
알고리즘 스터디 2주차

목차

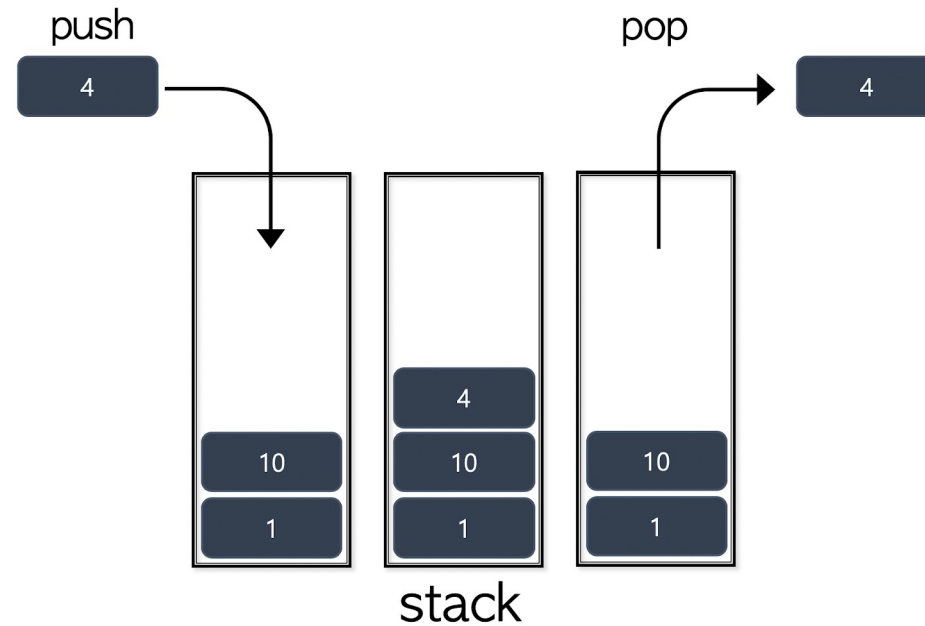


1. 스택(Stack)
2. 큐(Queue)
3. 덱(Deque)
4. 힙(Heap)& 우선순위 큐

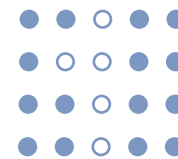
스택(Stack)



- LIFO(후입선출)
- 먼저 들어갈수록 늦게 나오고 늦게 들어갈수록 일찍 나온다
- 기본 리스트로 구현 가능하다

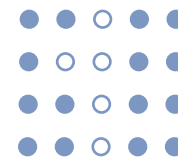


스택 기본 연산



- **push** – 스택의 가장 위(뒤)에 원소를 삽입
-> 리스트.append(원소)
- **pop** – 스택의 가장 위(뒤)에 있는 원소를 삭제하고 원소를 반환한다
-> 리스트.pop()
- **peek** – 스택의 가장 위(뒤)에 있는 원소를 반환한다 (삭제는 하지 않는다)
-> 리스트[-1]
- **empty** – 스택이 비어있는지 확인, 비어있다면 1 아니라면 0
-> not bool(리스트)

스택 구현



[코드]

```
stack = [1, 2, 3, 4, 5]

# push
stack.append(6)
print(stack)

# pop
stack.pop()
print(stack)

# peek
print(stack[-1])

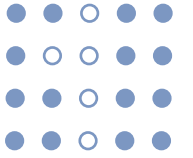
# empty
print(not bool(stack))

# size
print(len(stack))
```

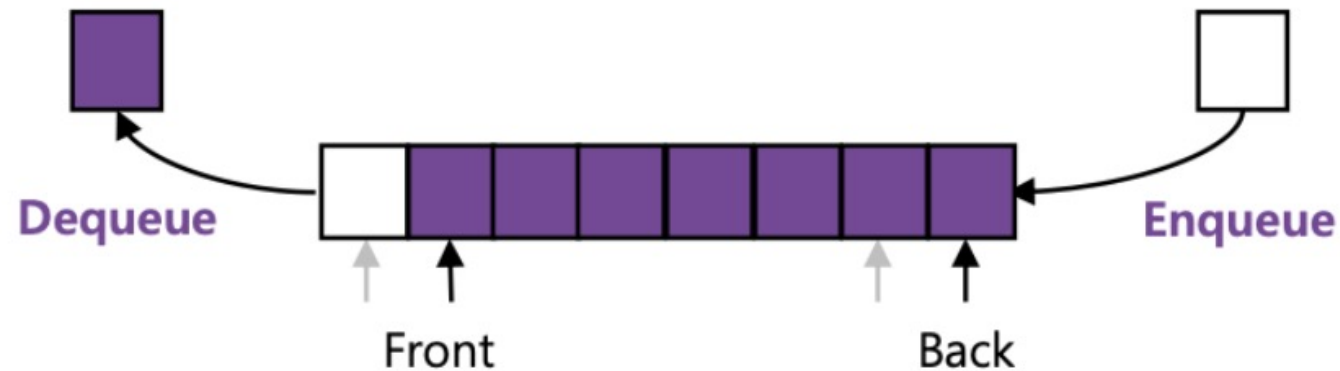
[출력 결과]

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5]
5
False
5
```

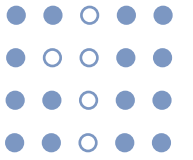
큐(Queue)



- FIFO(선입선출)
- 먼저 들어가면 먼저 나오고 늦게 들어가면 늦게 나온다
- 기본 리스트와 내장모듈로 구현 가능하다

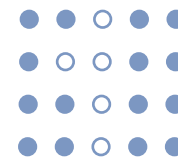


큐 기본 연산



- **enqueue** – 큐의 가장 뒤에 원소를 삽입
-> 리스트.append(원소)
- **dequeue** – 큐의 가장 앞에 있는 원소를 삭제하고 원소를 반환한다
-> 리스트.pop(0)
- **peek** – 큐의 가장 위(앞)에 있는 원소를 반환한다 (삭제는 하지 않는다)
-> 리스트[0]
- **empty** – 스택이 비어있는지 확인, 비어있다면 1 아니라면 0
-> not bool(리스트)

큐 구현 - 리스트



[코드]

```
queue = [1, 2, 3, 4, 5]

# enqueue
queue.append(6)
print(queue)

# dequeue
queue.pop(0)
print(queue)

# peek
print(queue[0])

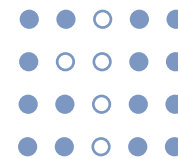
# empty
print(not bool(queue))

# size
print(len(queue))
```

[출력 결과]

```
[1, 2, 3, 4, 5, 6]
[2, 3, 4, 5, 6]
2
False
5
```


큐 구현 - 내장모듈



[코드]

```
from queue import Queue

queue = Queue()

# enqueue
queue.put(1)
queue.put(2)
queue.put(3)

# dequeue
queue.get()

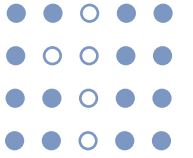
# empty
print(queue.empty())

# size
print(queue.qsize())
```

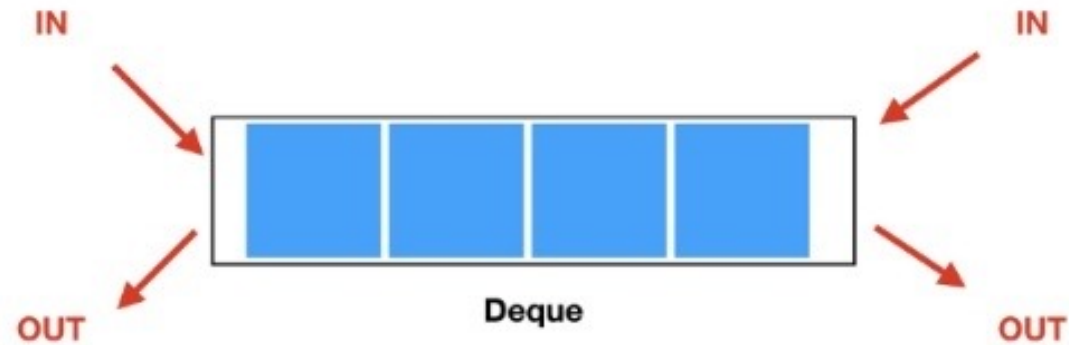
[출력 결과]

```
[1, 2, 3]
[2, 3]
True
0
```

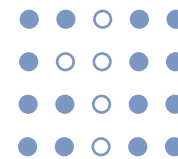
덱(*Deque*)



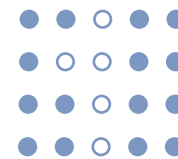
- 앞과 뒤에서 모두 삽입, 삭제가 가능한 구조
- 내장모듈로 구현 가능하다



덱 기본 연산



- **삽입** – `덱.appendleft(원소)`, `덱.append(원소)`
- **삭제** – `덱.popleft()`, `덱.pop()`
- **반환** – `덱[0]`, `덱[-1]`
- **밀기** – `덱.rotate(음수)`, `덱.rotate(양수)`
- **empty** – `not bool(덱)`



[코드]

```
from collections import deque

d = deque([1, 2, 3])

# 앞에 삽입
d.appendleft(0)
print(list(d))

# 뒤에 삽입
d.append(4)
print(list(d))

# 앞에 삭제
d.popleft()
print(list(d))

# 뒤에 삭제
d.pop()
print(list(d))
```

```
# 중간에 삽입
d.insert(1, 7)
print(list(d))

# 왼쪽으로 밀기
d.rotate(-2)
print(list(d))

# 오른쪽으로 밀기
d.rotate(2)
print(list(d))

# 가장 앞의 요소와 가장 뒤의 요소 반환
print(d[0], d[-1])

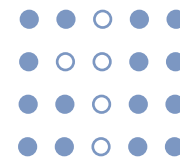
# empty
print(not bool(d))

# size
print(len(d))
```

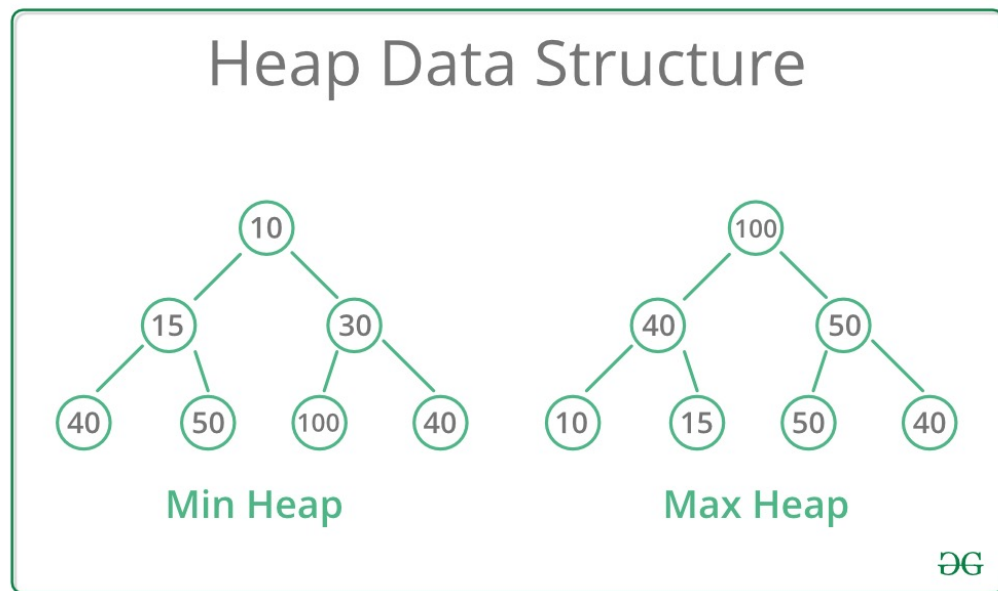
[출력 결과]

```
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3]
[1, 7, 2, 3]
[2, 3, 1, 7]
[1, 7, 2, 3]
1 3
False
4
```

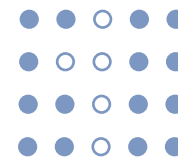
힙(Heap) & 우선순위 큐



- 힙은 최소값과 최대값을 빠르기 찾기 위한 이진트리
- 우선순위 큐는 들어간 순서에 상관없이 우선순위가 높은 데이터가 먼저 나오는 것
- 힙은 내장 모듈로 구현할 수 있고 우선순위 큐는 힙으로 구현 가능하다

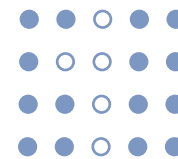


힙 기본 연산



- **push** – 힙에 원소를 삽입
-> `heapq.heappush(리스트, 원소)`
- **pop** – 힙에서 우선순위가 가장 높은 원소를 삭제하고 원소를 반환한다
-> `heapq.heappop(리스트)`
- **peek** – 힙에서 우선순위가 가장 높은 원소를 반환한다 (삭제는 하지 않는다)
-> `리스트[0]`
- **empty** – 힙이 비어있는지 확인, 비어있다면 1 아니라면 0
-> `not bool(리스트)`

힙 구현 - 최소 힙



- heapq(내장 모듈)은
최소힙이 기본이다

[코드]

```
import heapq
heap = [4, 3, 2]

# push
heapq.heappush(heap, 1)
print(heap)

# pop
heapq.heappop(heap)
print(heap)

# peek
print(heap[0])

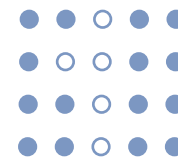
# empty
print(not bool(heap))

# size
print(len(heap))
```

[출력 결과]

```
[1, 4, 2, 3]
[2, 4, 3]
2
False
3
```

힙 구현 - 최대 힙



- (우선순위, 값) 구조의 튜플을 힙에 추가한다
- 우선순위에는 -를 붙인다

[코드]

```
import heapq
heap = [(-4, 4), (-3, 3), (-2, 2)]

# push
heapq.heappush(heap, (-1, 1))
print(heap)

# pop
heapq.heappop(heap)
print(heap)

# peek
print(heap[0])

# empty
print(not bool(heap))

# size
print(len(heap))
```

[출력 결과]

```
[(-4, 4), (-3, 3), (-2, 2), (-1, 1)]
[(-3, 3), (-1, 1), (-2, 2)]
(-3, 3)
False
3
```