알고리즘 특강
기본 자료구조

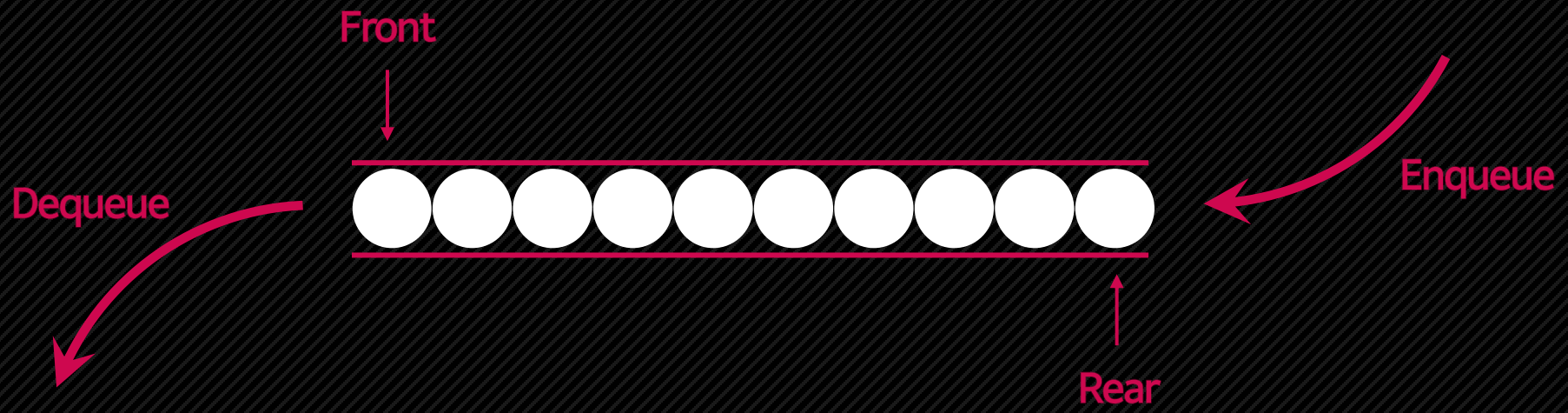
기본적인 자료구조인 큐와 스택, 해시, 그리고 힙에 대해 알아봅시다.
이후 다양한 알고리즘을 공부할 때 기본이 되니 잘 숙지하셔야 합니다.

스택

Stack

- FILO (First In, Last Out) 구조를 띄고 있는 자료구조로, 삽입과 삭제 연산이 동일한 한군데에서 발생함.
- 삽입/삭제 연산에 있어 시간복잡도가 O(1)임.
- 이전에 활용한 데이터를 역으로 추적하거나, 처음 들어온 데이터보다 나중에 들어온 데이터가 빨리 나가야 할 때 사용.
- Python에서는 LifoQueue라는 구현체가 있으나, List의 pop을 활용하면 Stack 처럼 사용할 수 있음.

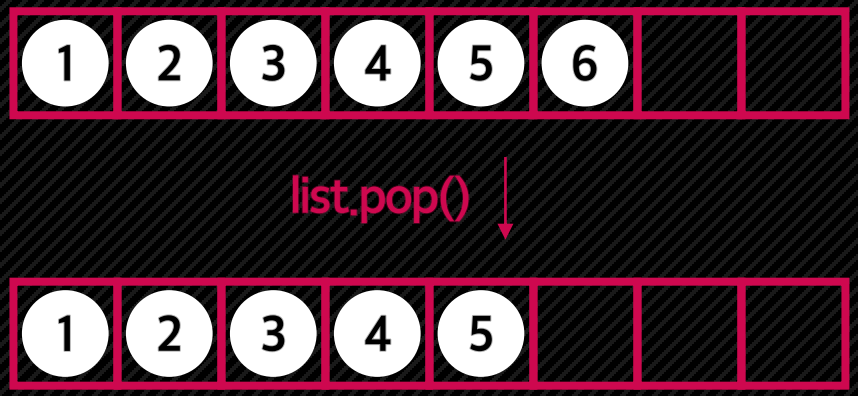
큐



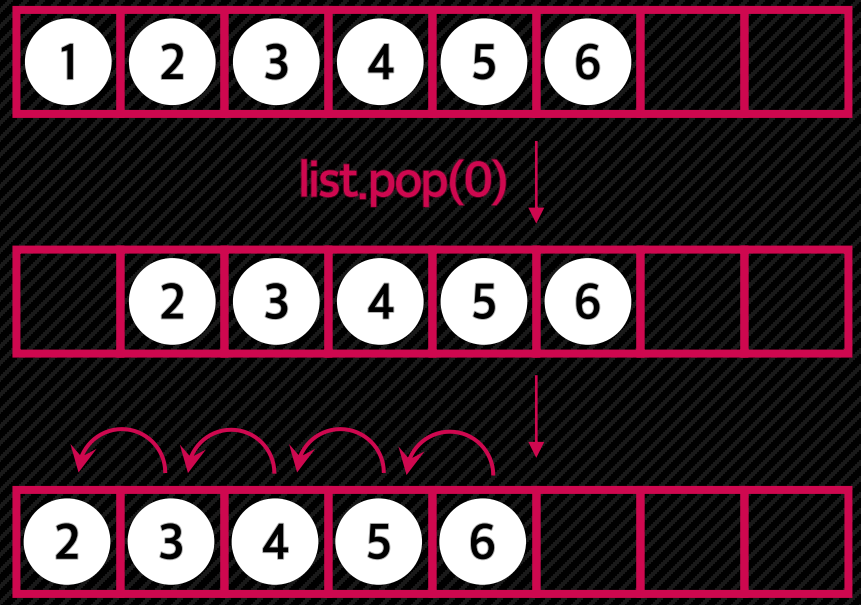
Queue

- FIFO (First In, First Out) 구조를 띄고 있는 자료구조로, 삽입과 삭제 연산이 서로 다른 한군데에서 발생함.
- 삽입/삭제 연산에 있어 시간복잡도가 O(1)임.
- 맨 앞을 front, 맨 뒤를 rear 라고 하고, 삽입 연산을 enqueue, 삭제 연산을 dequeue라고 함.
- 주로 순차적으로 진행되어야 하는 일을 스케줄링 할 때 사용 됨.

List의 삭제



끝 원소만 삭제 → O(1)

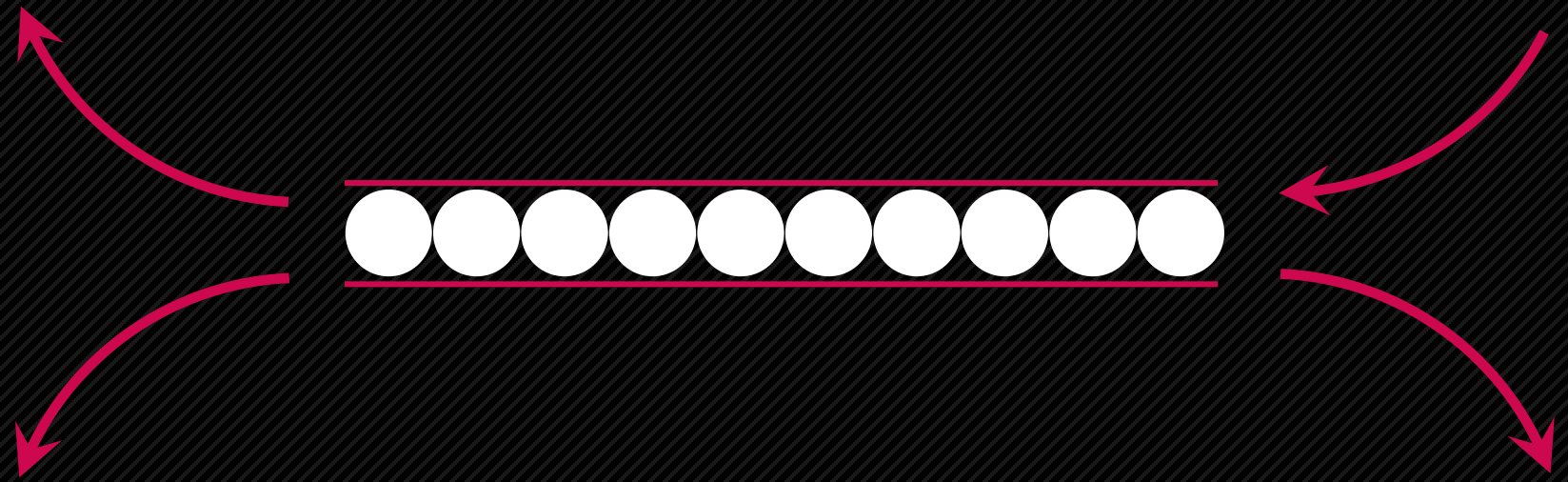


삭제 후 모든 원소 이동 → O(N)

List를 Stack 처럼 사용하는 것은 괜찮지만, Queue처럼 사용하지 말 것!!!

큐가 있긴 한데…

```python
import queue

listQueue = []
For I in range(5):
    listQueue.append(i)

listQueue.pop(0) # 추천하지 않음!!!


normalQueue = queue.Queue()
# normalQueue = queue.Queue(maxsize = 0)
for i in range(5):
    normalQueue.put(i)


while normalQueue.qsize():
    print(normalQueue.get())

if normalQueue.empty():
    print("Queue is empty!!")
```



Deque

- Queue/Stack의 구조를 합친 구조로, 양쪽에서 삽입/삭제를 모두 할 수 있음.
- 삽입/삭제 연산에 있어 시간복잡도가 O(1)임.

큐 VS 덱

```python
import queue

normalQueue = queue.Queue()
for i in range(5):
    normalQueue.put(i)

while normalQueue.qsize():
    print(normalQueue.get())

if normalQueue.empty():
    print("Queue is empty!!")
```

```python
from collections import deque

dq = deque("1234")

dq.append("5")
# dq.appendleft("0")

dq.rotate(1)

while len(dq):
    print(dq[0])
    dq.popleft()
```

- Deque는 iterable한 데이터를 기반으로 선언 가능하며, 왼쪽에서 진행하는 연산은 left를 붙임.
- Queue는 멀티 스레드 환경에 최적화 되었으며, Deque의 경우 속도 측면에서 더 빠름.
- 코딩테스트 환경에서는 Deque를 쓰는 것을 권장.

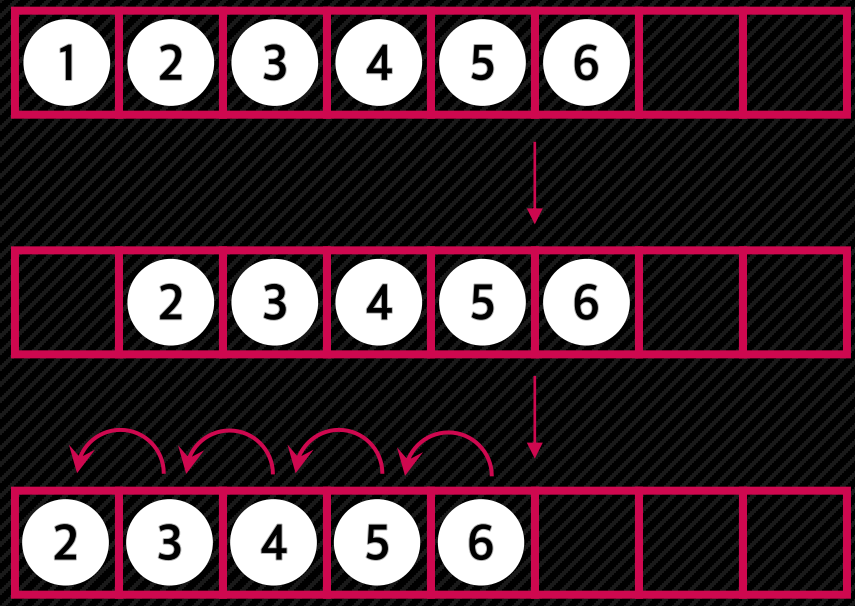
파이썬에서의 덱

```
from collections import deque

dq = deque()
dq = deque('12345')

dq.append(6) # 1 2 3 4 5 6
dq.appendleft(0) # 0 1 2 3 4 5 6

removed = dq.pop() # 0 1 2 3 4 5
print(removed) # 6
removed = dq.popleft() # 1 2 3 4 5
print(removed) # 0

dq.rotate(1) # 5 1 2 3 4
dq.rotate(-2) # 2 3 4 5 1
```

- Deque를 사용하기 위해서 import 함.
- Deque의 선언: iterable한 데이터를 받아 생성.
- 대부분의 연산은 List와 유사하나,
  앞에서 진행되는 연산은 left를 붙인다.
- pop()/popleft() 메소드는 제거한 값을 리턴함.
- rotate(n): dq.appendleft(dq.pop()) * n (n > 0)
  dq.append(dq.popleft()) * |n| (n < 0)

큐/스택은 어디에 쓸까?

Queue



- 순서대로 처리해야 하는 일을 수행할 때.
- 처리 과정에서 데이터를 제거해야 하는데, 제거해야 하는 위치가 맨 끝이 아닐 때.

큐 연습문제

Silver 4 - 카드 2 (#2164)

요약

- N장의 카드가 있다. 각각의 카드는 차례로 1부터 N까지의 번호가 붙어 있으며,
1번 카드가 제일 위에, N번 카드가 제일 아래인 상태로 순서대로 카드가 놓여 있다.
- 카드가 한 장 남을 때 까지, 맨 위에 있는 카드를 바닥에 버린다.
그 다음, 제일 위에 있는 카드를 제일 아래에 있는 카드 밑으로 옮기는 행위를 반복한다.
- N이 주어졌을 때, 제일 마지막에 남는 카드를 구하는 프로그램을 작성하시오.

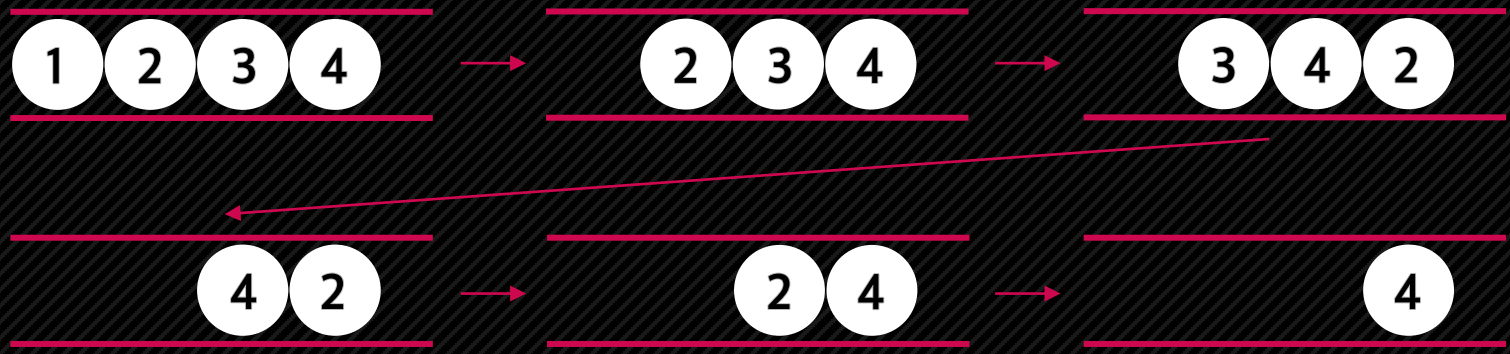
제약조건

- N의 범위는 1 <= N <= 500,000 이다.

연습문제

Silver 4 - 카드 2 (#2164)

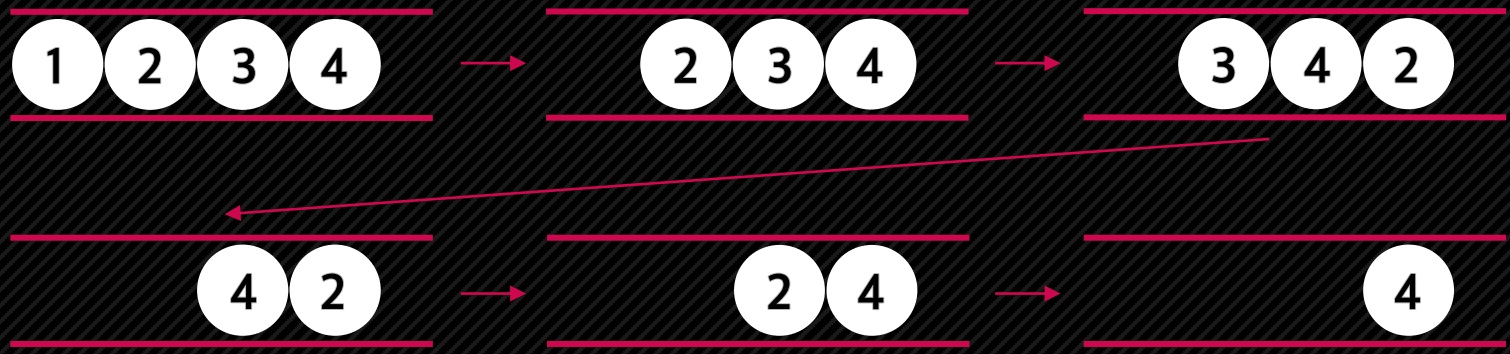
N = 4



연습문제

Silver 4 - 카드 2 (#2164)

N = 4



- 순서대로 처리해야 하는 일을 수행할 때.
- 처리 과정에서 데이터를 제거해야 하는데, 제거해야 하는 위치가 맨 끝이 아닐 때.

연습문제

Silver 4 - 카드 2 (#2164)

```
function solution(number) do
    set queue = []

    foreach(i <- 1 ... number) do
        enqueue i to queue
    end

    while loop (size of queue is 1) {
        dequeue of queue
        set anotherNumber = front of queue
        dequeue of queue
        enqueue anotherNumber to queue
    }

    return front of queue

end
```

Dequeue를 두 번 할 때 마다 큐의 길이가 1씩 감소 →
O(2 * N) = O(N)

스택 연습문제

Silver 4 - 제로 (#10773)

요약

- 데이터를 입력 받다가, 중간에 잘못된 데이터가 들어오면 0이 들어온다.
- 0이 들어오면 잘못된 데이터가 들어온 것 이므로, 값을 삭제한다.
- 이후 남아있는 값들의 합을 구하라.

제약조건

- 전체 수의 수는 1 <= K <= 100,000 이다.
- 최종적으로 적어낸 수의 합은 $2^{31}-1$ 보다 같거나 작다.

스택 연습문제

Silver 4 - 제로 (#10773)

요약

- 데이터를 입력 받다가, 중간에 잘못된 데이터가 들어오면 0이 들어온다.
- 0이 들어오면 잘못된 데이터가 들어온 것 이므로, 값을 삭제한다.
- 이후 남아있는 값들의 합을 구하라.

Stack

- FILO (First In, Last Out) 구조를 띄고 있는 자료구조로, 삽입과 삭제 연산이 동일한 한군데에서 발생함.
- 삽입/삭제 연산에 있어 시간복잡도가 O(1)임.
- 이전에 활용한 데이터를 역으로 추적하거나, 처음 들어온 데이터보다 나중에 들어온 데이터가 빨리 나가야 할 때 사용.

스택 연습문제

Silver 4 - 제로 (#10773)

```
function solution(number) do
    set stack = []

    foreach (numberElement in number) do
        if numberElement != 0 do
            push numberElement to stack
        end else do
            pop numberElement to stack
        end
    end

    set answer = 0

    while loop(stack is not empty) {
        answer += top of stack
        pop element of stack
    }

    return answer
end
```

숫자를 순차적으로 push하고, 순차적으로 pop함. →
O(2 * N) = O(N)

그럼 이제…

Silver 2 - AC (#5430)

요약

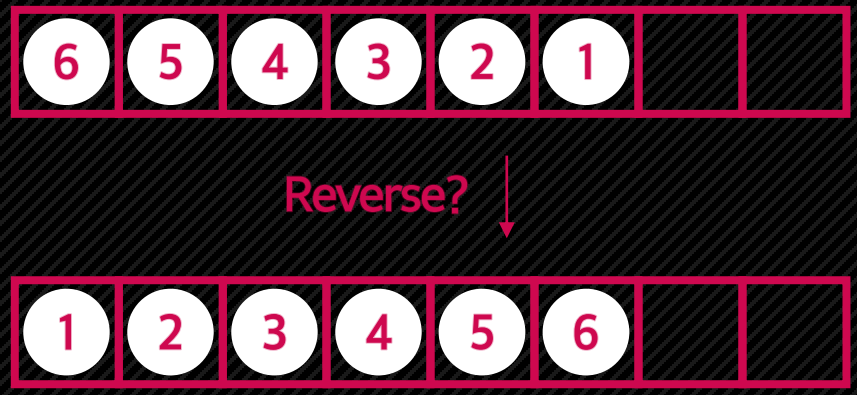
- 배열에서 'R' (뒤집기) 연산과 'D' (버리기) 연산을 수행하려고 한다.
- R을 실행하면 배열 내 모든 원소의 순서가 뒤집히고, D를 수행하면 첫 번째 숫자를 버린다.
- 배열의 초기값과 수행할 함수가 주어진다면, 최종 결과를 출력하는 프로그램을 작성하시오.

제약조건

- 테스트 케이스의 수의 범위는 1 <= T <= 100 이다.
- 수행할 함수의 수는 1 <= P <= 100,000 이다.
- 배열 내 숫자의 개수는 0 <= P <= 100,000 이다.

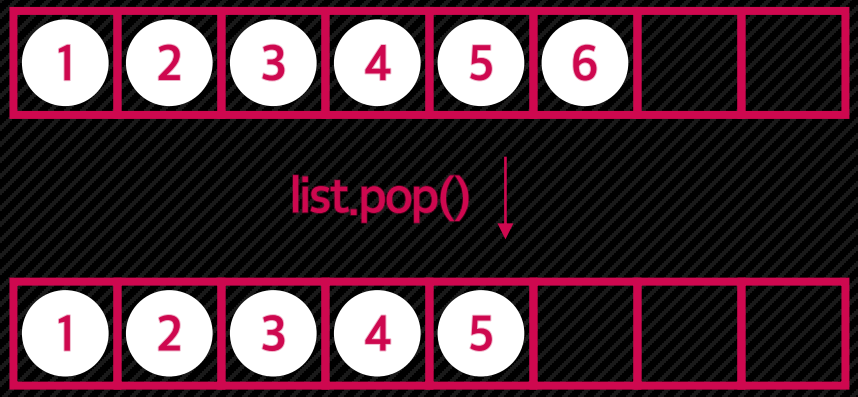
언제 뒤집고 앉아있어?

Silver 2 - AC (#5430)

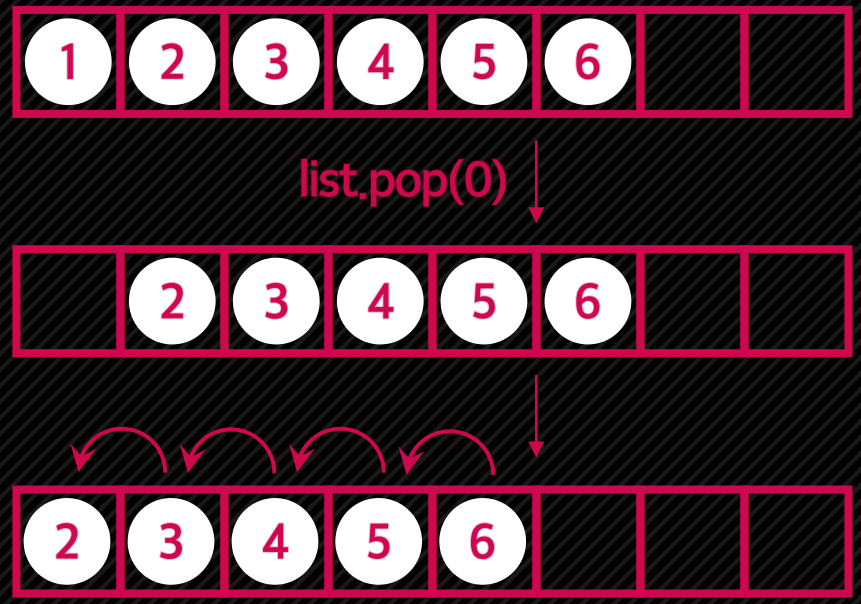


그냥 빼면 큰일난다!

Silver 2 - AC (#5430)



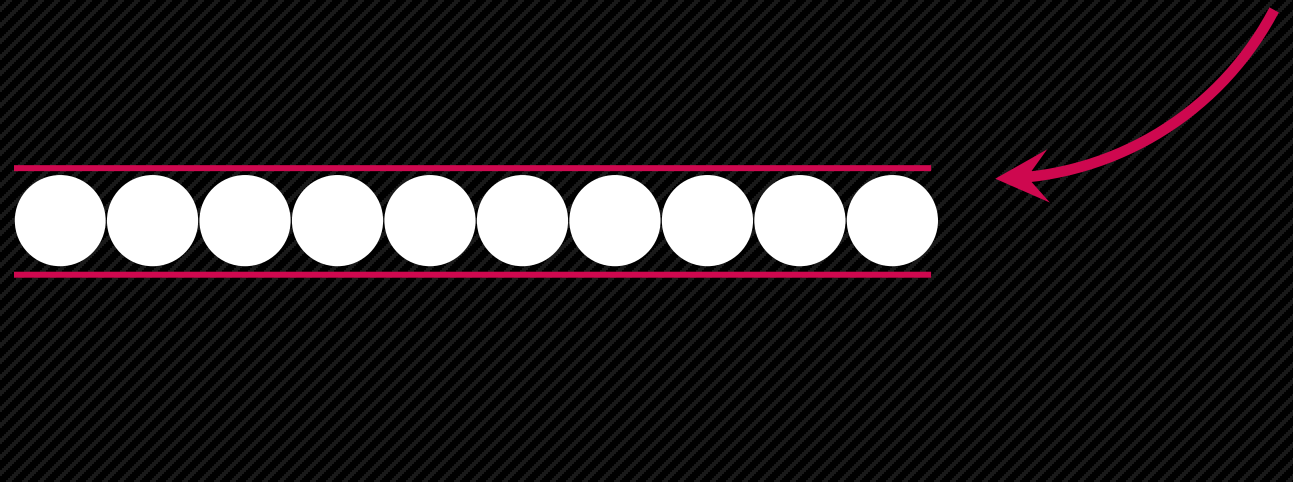
끝 원소만 삭제 → O(1)



삭제 후 모든 원소 이동 → O(N)

앞 뒤로 뺀다고?

Silver 2 - AC (#5430)



Deque

- Queue/Stack의 구조를 합친 구조로, 양쪽에서 삽입/삭제를 모두 할 수 있음.
- 삽입/삭제 연산에 있어 시간복잡도가 O(1)임.

이것까지 봐보자!

Silver 2 - 외계인의 기타 연주 (#2841)

요약

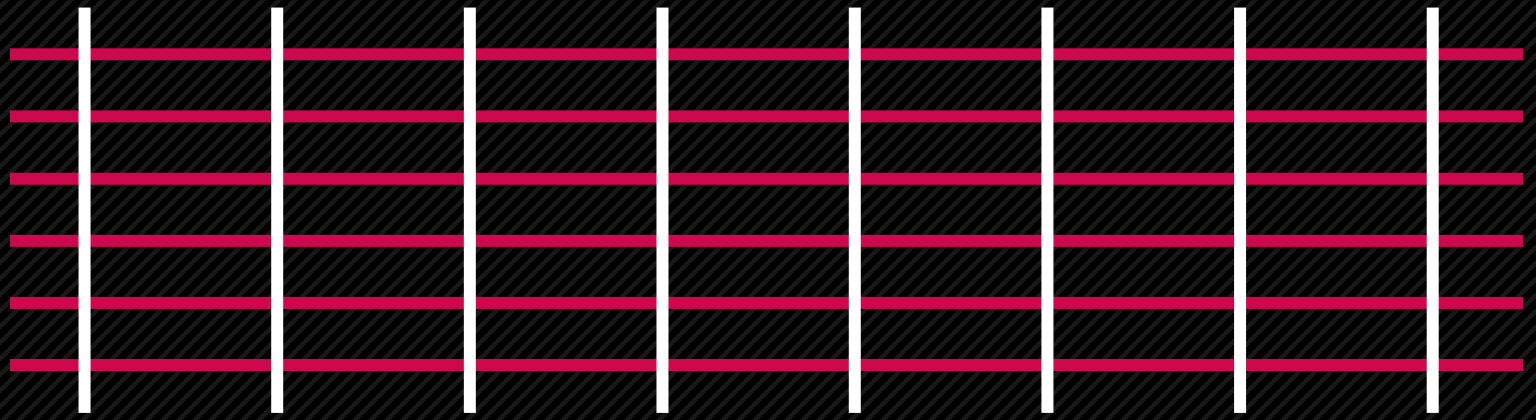
- 손가락을 수십억개 갖고 있는 외계인이 기타를 치려고 한다.
- 기타는 6개의 줄이 있고, 각각의 줄은 P개의 프렛으로 나뉘어진다.
- 프렛을 누른 상태로 줄을 튕기면 음을 연주할 수 있는데, 어떤 줄의 프렛을 여러 개 누르고 있다면 가장 높은 음이 들린다.
- 손가락으로 프렛을 누르거나 떼는 것을 한 번 움직였다고 할 때, 횟수를 최소화하는 방법을 구하는 프로그램을 작성하시오.

제약조건

- 멜로디에 포함하는 음의 수는 $1 <= N <= 500,000$ 이다.
- 한 줄에 있는 프렛의 수는 $2 <= P <= 300,000$ 이다.

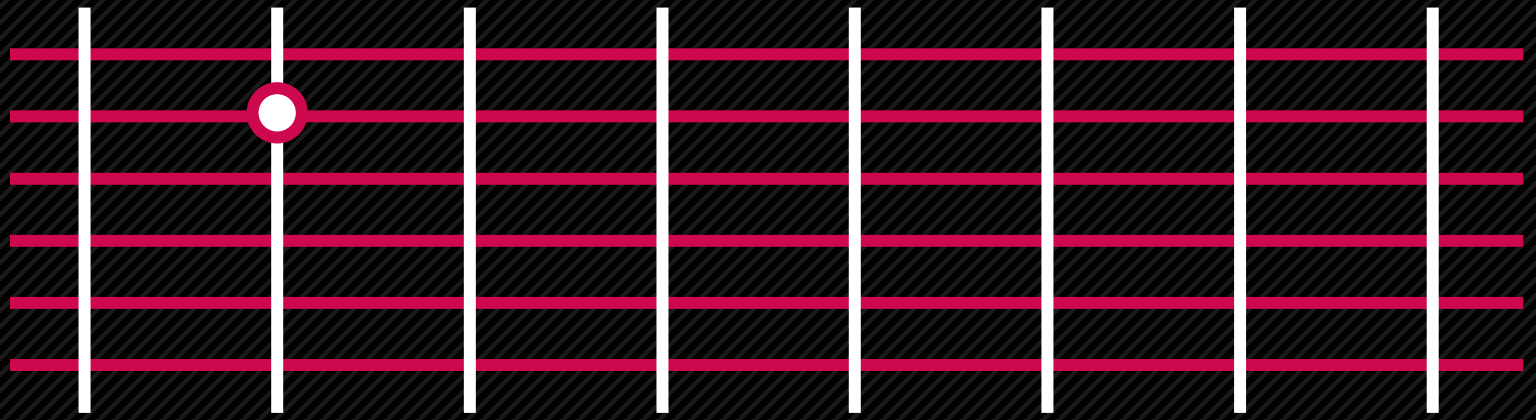
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



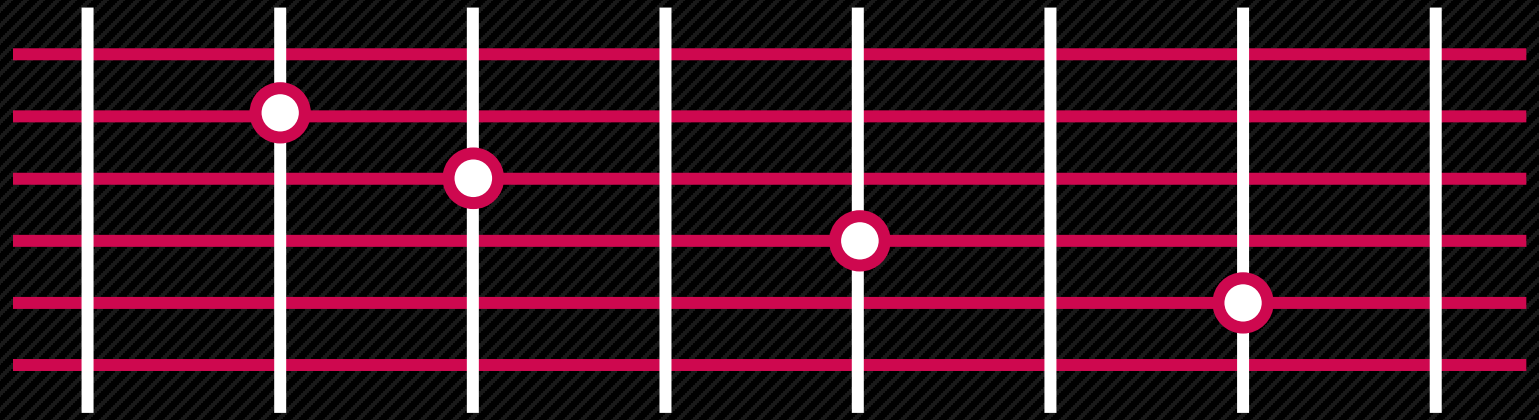
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



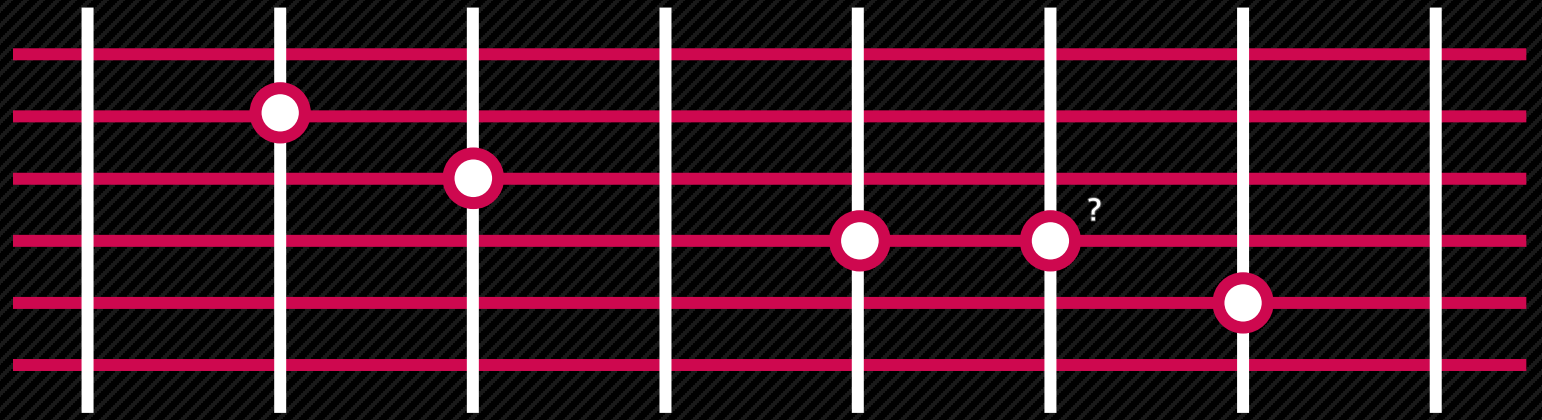
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



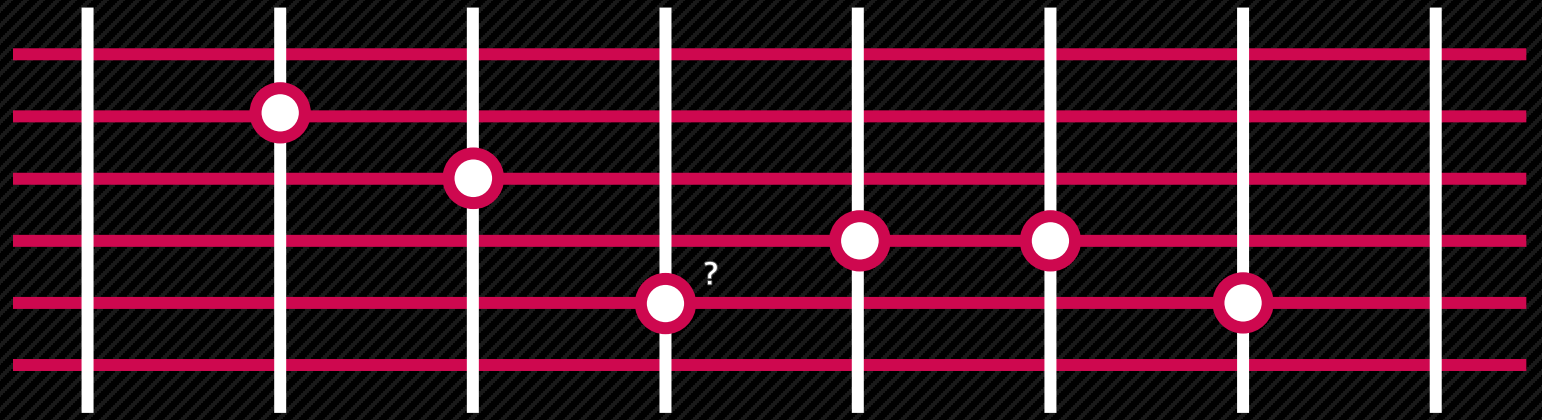
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



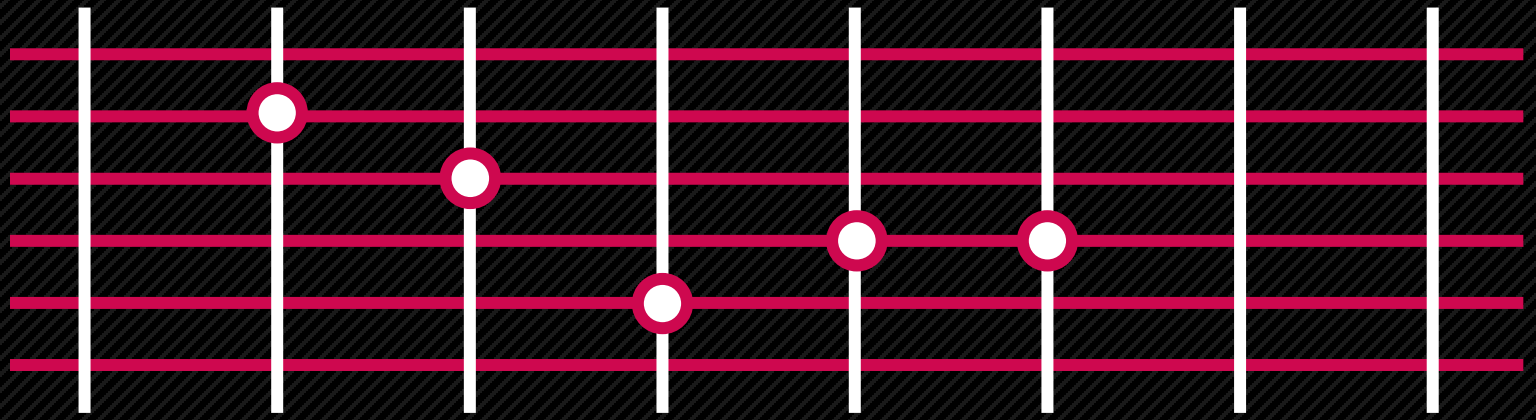
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



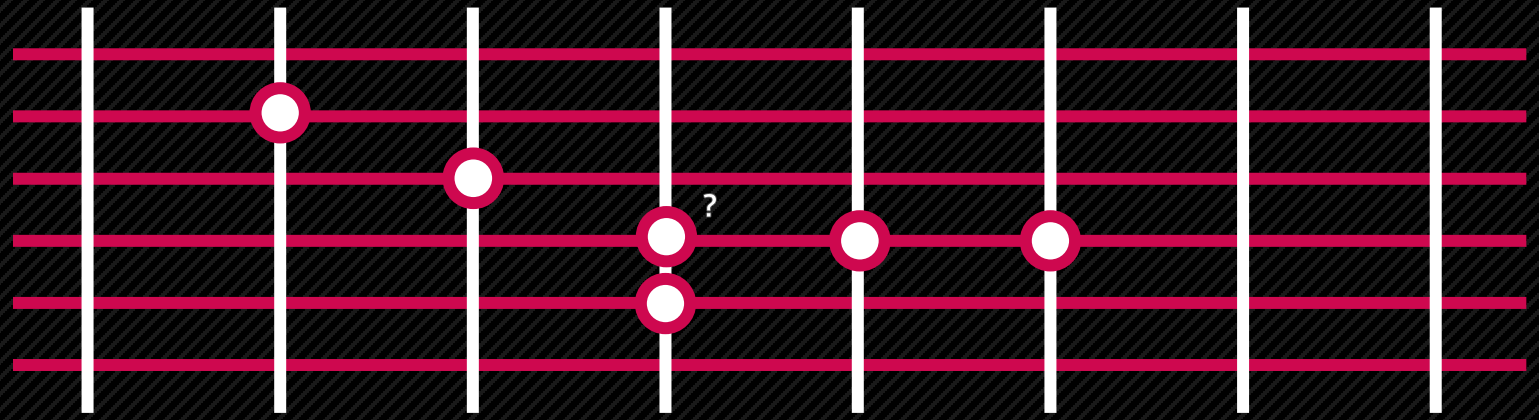
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



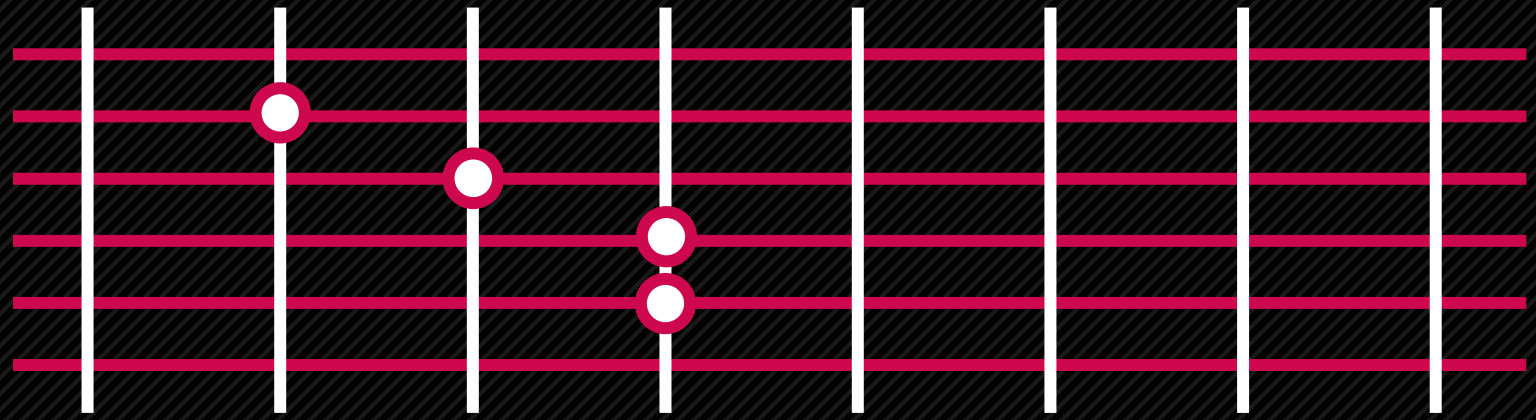
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



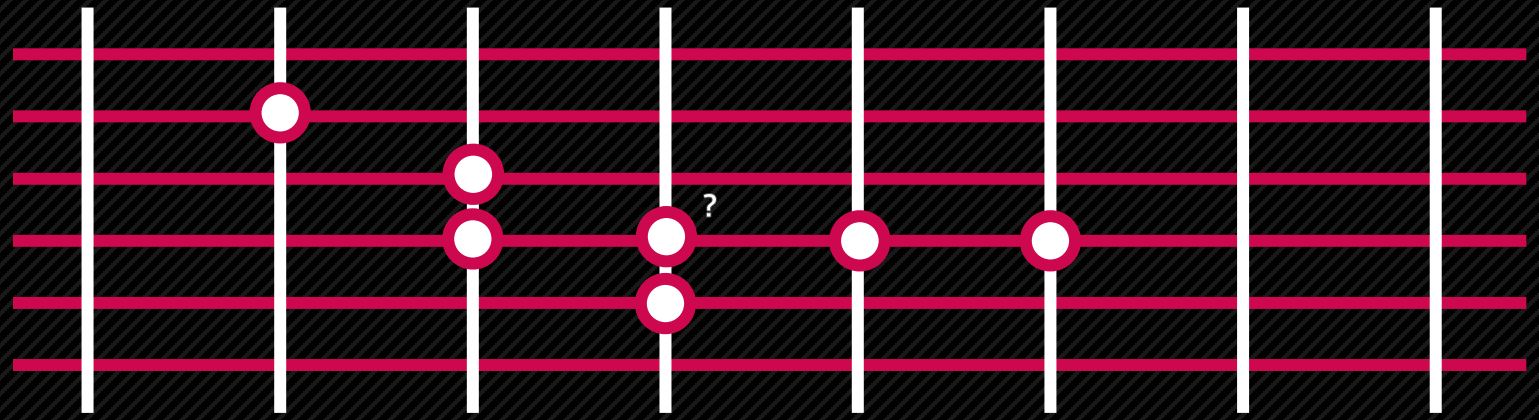
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



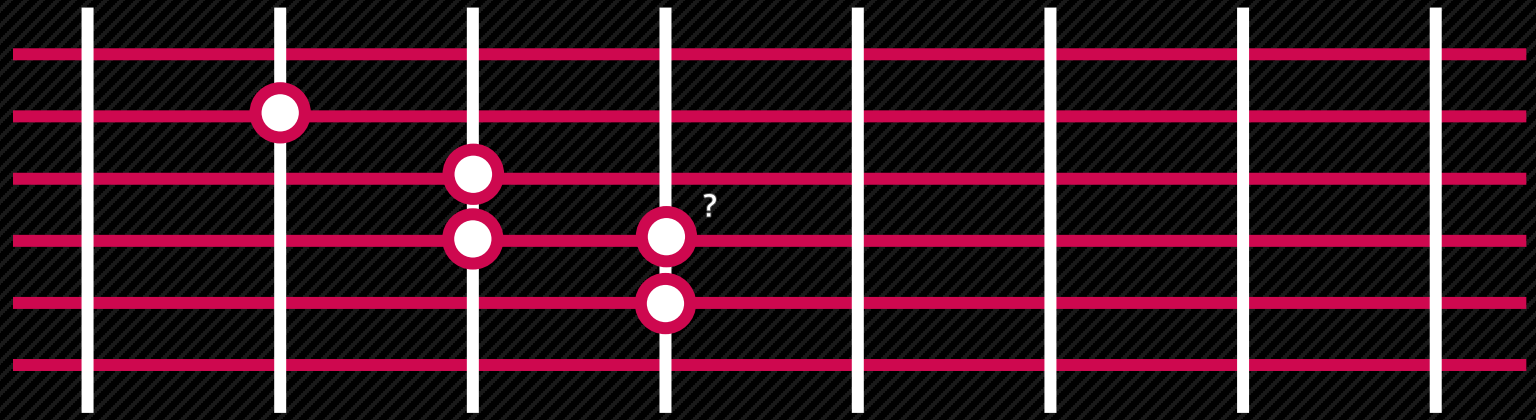
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



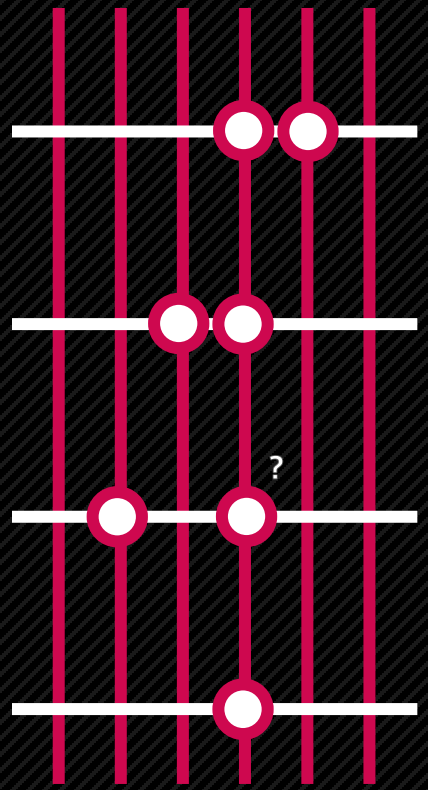
모델링을 해보자.

Silver 2 - 외계인의 기타 연주 (#2841)



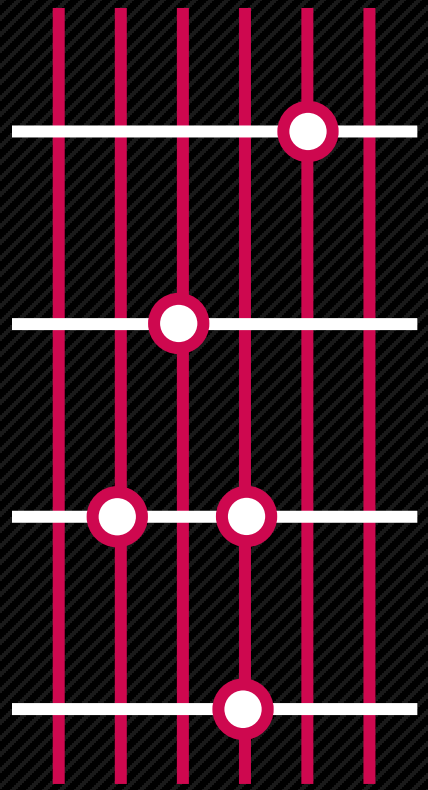
아직도 감이 안 와요?

Silver 2 - 외계인의 기타 연주 (#2841)

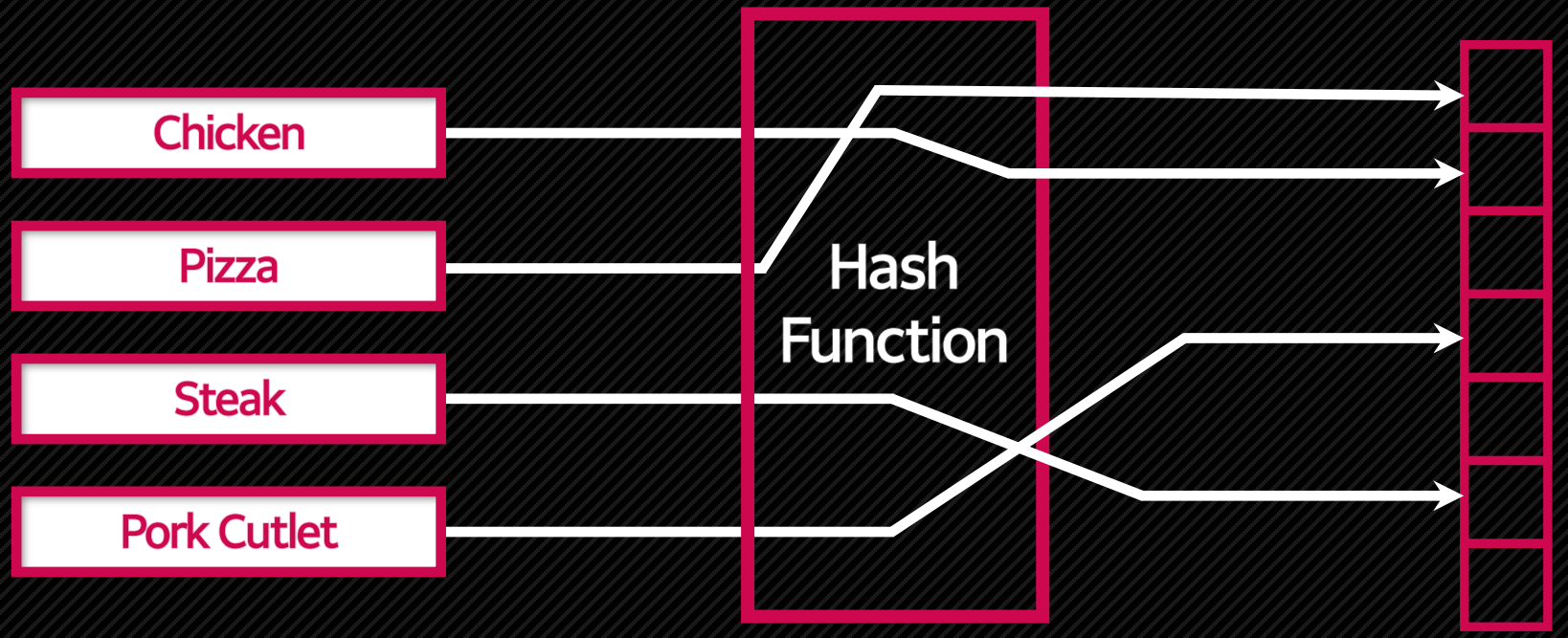


아직도 감이 안 와요?

Silver 2 - 외계인의 기타 연주 (#2841)



해시



Hash

- 임의의 데이터에 대해 고정된 길이의 데이터로 매핑하는 자료구조.
- 이론적으로 삭제 O(1), 삽입 O(1), 검색 O(1)의 시간복잡도를 갖고 있다.
- 그러나, 해시 테이블 내부의 값이 많아지면 해시충돌 현상이 발생해 기본 연산의 시간이 길어진다.

Preclass #2

```python
import random
import time
count = 0
_list = [random.randrange(1, 100000) for i in range(100000)]
start = time.time()
for i in range(100000):
    if i in _list:
        count += 1
print(count)
print(time.time() - start)
```

약 47.74초 소요!

Preclass #2

```python
import random
import time
count = 0
_set = {random.randrange(1, 100000) for i in range(100000)}
start = time.time()
for i in range(100000):
    if i in _set:
        count += 1
print(count)
print(time.time() - start)
```

약 0.012초 소요!

Dict

```
_dict = dict(one = 1, two = 2, three = 3)
_dict = {'one': 1, 'two': 2, 'three': 3}
```

일반적인 딕셔너리 만들기

Dict

```
_number = ['one', 'two', 'three', 'four']
_num = [1, 2, 3, 4]
_dict = dict(zip(_number, _num))

new_dict = {word : number for word, number in _dict.items() if number > 1}
```

이건 어때요?

Zip

2개 이상의 iterable한 데이터를 묶어주는 메소드

```
_number = ['one', 'two', 'three', 'four']
_num = [1, 2, 3, 4]
_zip = (zip(_number, _num))
for i in _zip:
    print(i)
```

```
('one', 1)
('two', 2)
('three', 3)
('four', 4)
```

Dict

딕셔너리 안에 키가 있으면 **값을 출력하고**, 없으면 **값을 넣자.**

```
if 5 in _dict:
    print(_dict[5])
else:
    _dict[5] = "Test"
    print("Test")
```

Dict

딕셔너리 안에 키가 있으면 **값을 출력하고**, 없으면 **값을 넣자.**

```
if 5 in _dict:
    print(_dict[5])
else:
    _dict[5] = "Test"
    print("Test")
```

```
print(_dict.setdefault(5, "Test"))
```

Defaultdict

```python
from collections import defaultdict

int_dict = defaultdict(int)
int_dict["asd"] += 1

print(int_dict)
```

defaultdict

- Dict 클래스의 서브클래스로, 인자로 주어진 객체의 기본값을 딕셔너리의 초깃값으로 지정할 수 있음 .
- 숫자 (int), 리스트 (list), 셋 (set) 등으로 기본값 설정 가능.
- 다만 약간의 성능 저하가 존재함.

Sorted?

```
_number = ['one', 'two', 'three', 'four']
_num = [1, 2, 3, 4]
_dict = dict((zip(_number, _num)))

print(sorted(_dict))
```

출력 값은?

Sorted?

```
_number = ['one', 'two', 'three', 'four']
_num = [1, 2, 3, 4]
_dict = dict((zip(_number, _num)))

print(sorted(_dict))
```

출력 값은?

['four', 'one', 'three', 'two']

Sorted?

```
_number = ['one', 'two', 'three', 'four']
_num = [1, 2, 3, 4]
_dict = dict((zip(_number, _num)))

print(sorted(_dict.items()))
```

출력 값은?

[('four', 4), ('one', 1), ('three', 3), ('two', 2)]

Set

```
_set = set()

_list = [1, 2, 3, 4, 5, 6, 1, 2, 3]
_set = set(_list)

print(_set & {1, 2, 3})
print(_set + {10, 11, 12})

print(sorted(_set))
```

- 기본적인 set의 선언
- 당연히 list로도 선언이 가능하다.
- set은 본래 집합을 위한 자료구조이기 때문에,
  + (합집합), - (차집합), & (교집합) 같은 연산을 지원
- dict와 달리 값의 쌍이 없기 때문에 정렬해도 무방.

Set의 응용

중복된 데이터를 날려버리자!

```
_list = ["test", "chicken", "pizza", "chicken", "hamburger"]
print(list(set(_list)))

# 만약 대소문자를 신경 쓰지 않는다면?
_list = ["Test", "test", "tEst", "TEST"]
print(list(set(map(lambda x : x.lower(), _list))))
```

단순하죠?

Silver 3 - 문자열 집합 (#14425)

요약

- 총 N개의 문자열로 이루어진 집합 S가 주어진다.
- 주어진 M개의 문자열 중 S에 포함된 문자열의 수를 출력하는 프로그램을 작성하시오.

제약조건

- N과 M은 1 <= N, M <= 10,000 이다.
- 문자의 길이는 최대 500이다.

이건 좀 특이한 것 같아요…

Silver 4 - 나는야 포켓몬 마스터 이다솜 (#1620)

요약

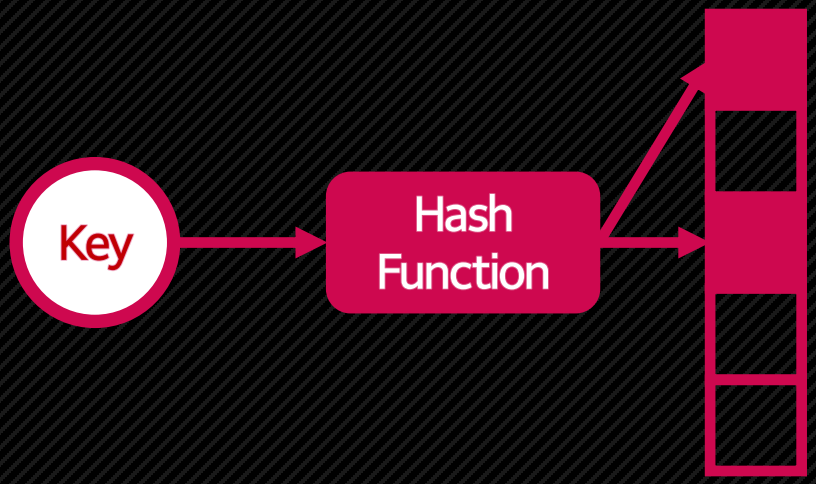
- 포켓몬의 이름이 N개 주어지며, 이후 M개의 줄에 숫자 또는 포켓몬 이름이 주어진다.
- 숫자를 입력 받았을 시 포켓몬의 이름을, 포켓몬의 이름을 입력 받았을 시 번호를 출력하는 프로그램을 작성하시오.

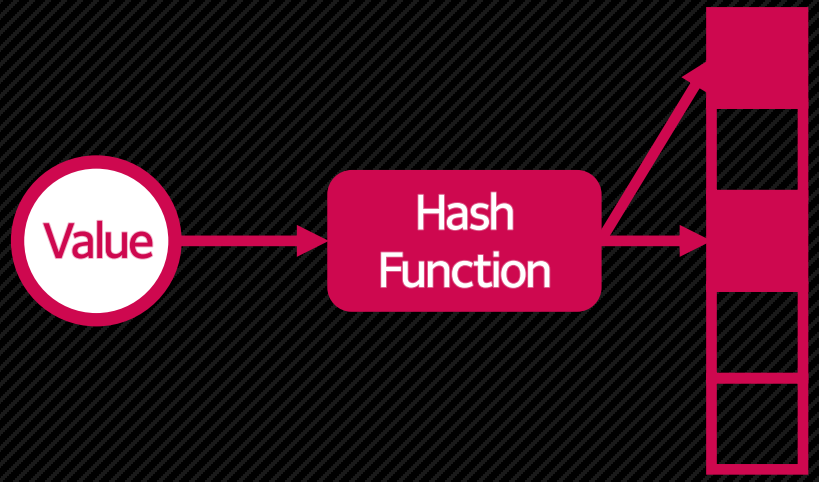
제약조건

- N과 M은 1 <= N, M <= 100,000 이다.
- 포켓몬 이름의 길이는 최대 20이다.

오해하지 말자!

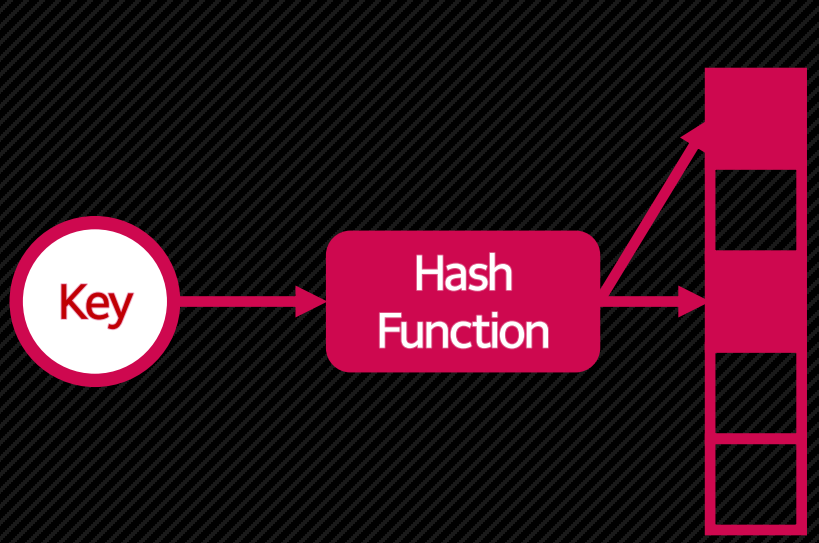
Silver 4 - 나는야 포켓몬 마스터 이다솜 (#1620)

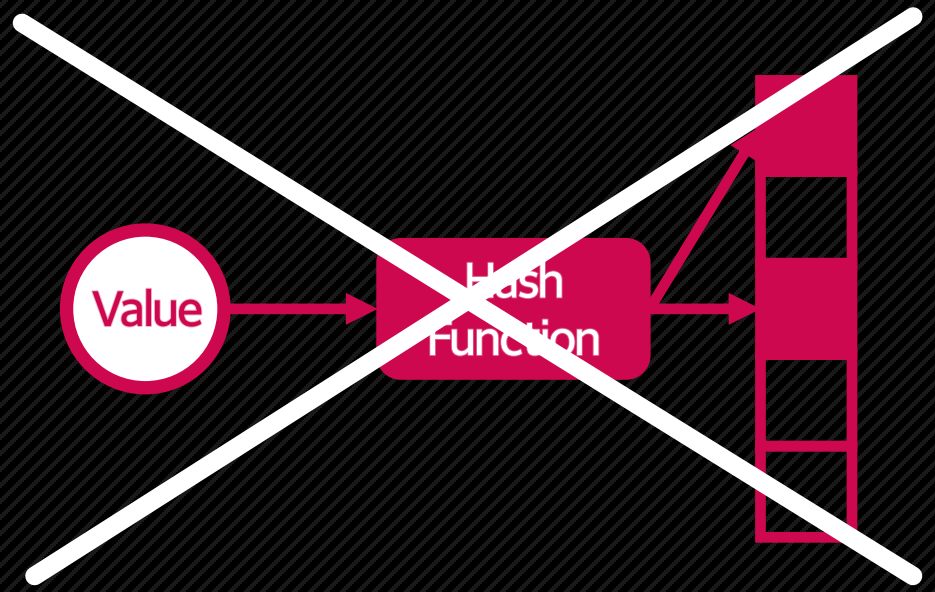




오해하지 말자!

Silver 4 - 나는야 포켓몬 마스터 이다솜 (#1620)





그럼 어떻게 해요?

Silver 4 - 나는야 포켓몬 마스터 이다솜 (#1620)

요약

- 포켓몬의 이름이 N개 주어지며, 이후 M개의 줄에 숫자 또는 포켓몬 이름이 주어진다.
- 숫자를 입력 받았을 시 포켓몬의 이름을, 포켓몬의 이름을 입력 받았을 시 번호를 출력하는 프로그램을 작성하시오.

→ 그냥 딕셔너리를 두 개 만들면 안 되나?

여기까지 풀면 기능은 마스터!

Gold 4 - 생태학 (#4358)

요약

- 나무들이 중복을 포함해서 주어질 때, 각 종이 전체에서 몇 %를 차지하는지 출력하는 프로그램을 작성하시오.

제약조건

- 나무 종은 최대 10,000종 이다.
- 주어지는 나무의 수는 최대 1,000,000그루 이다.

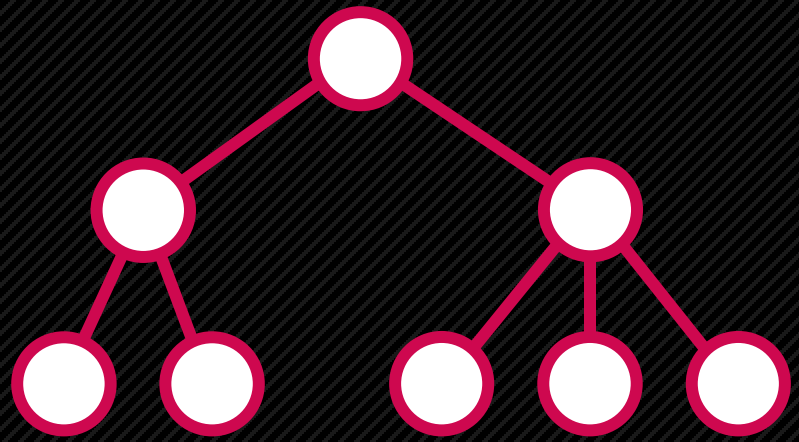
이 친구를 쓰면 쉽죠!

```python
from collections import defaultdict

int_dict = defaultdict(int)
int_dict["asd"] += 1

print(int_dict)
```

defaultdict

- Dict 클래스의 서브클래스로, 인자로 주어진 객체의 기본값을 딕셔너리의 초깃값으로 지정할 수 있음 .
- 숫자 (int), 리스트 (list), 셋 (set) 등으로 기본값 설정 가능.
- 다만 약간의 성능 저하가 존재함.

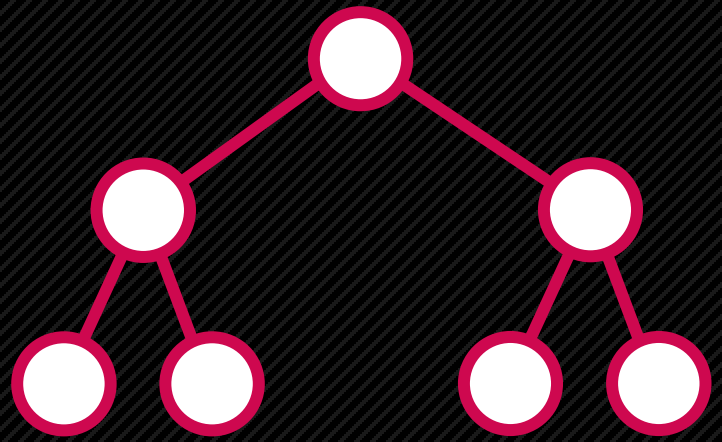
이진 트리



Tree

- 루트 노드와 자식 노드의 **연결관계가 반복적으로 구성된 자료구조.**
- 루트 노드의 **부모는 0개**이며, 자식 노드의 **부모는 1개**이다.
- 루트 노드는 **0개 이상의 자식 노드**를 갖고 있으며, 자식 노드 또한 **0개 이상의 자식 노드**를 가지고 있다.
- 사이클을 갖고 있지 않다.

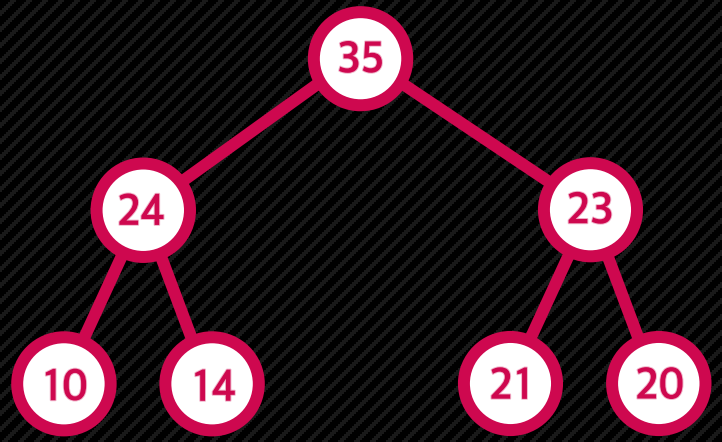
이진 트리



Binary Tree

- 트리에서, 자식 노드의 개수가 0개 이상 2개 이하로 제한된 트리.

힙



Heap

- 완전 이진 트리의 일종으로, 부모의 값이 항상 자식보다 크거나 작아야 함.
- 즉, 루트는 최댓값이거나, 최솟값임이 보장됨.
- 최댓값/최솟값을 O(1)만에 찾을 수 있는 자료구조.

Heap in Python

```python
import heapq

_list = [32, 16, 54, 94, 81, 31]
heapq.heapify(_list)

heapq.heappush(_list, 7)
print(heapq.heappop(_list))

print(heapq.heappushpop(_list, 100))

small_elements = heapq.nsmallest(4, _list)
print(small_elements)

large_elements = heapq.nlargest(4, _list)
print(large_elements)
```

- 사용 전 반드시 import!
- heapq는 리스트 기반 자료구조이며, 기존에 존재하던 list에 heapify를 사용해 배치
- heappush: 값을 heap에 넣음
- heappop: heap에 있는 값 중 최솟값을 뺌
- heappushpop: push하고, pop함.
- nsmallest: heap의 원소 중 최솟값 n개를 리턴
- nlargest: heap의 원소 중 최댓값 n개를 리턴

직접 짜보자!

Silver 1 - 최소 힙 (#1972)

요약

- 최소 힙을 만들어 다음과 같은 일련의 연산을 수행하는 프로그램을 작성하시오.
- 연산은 자연수를 넣거나, 최댓값을 출력하고, 제거하는 연산이다.

제약조건

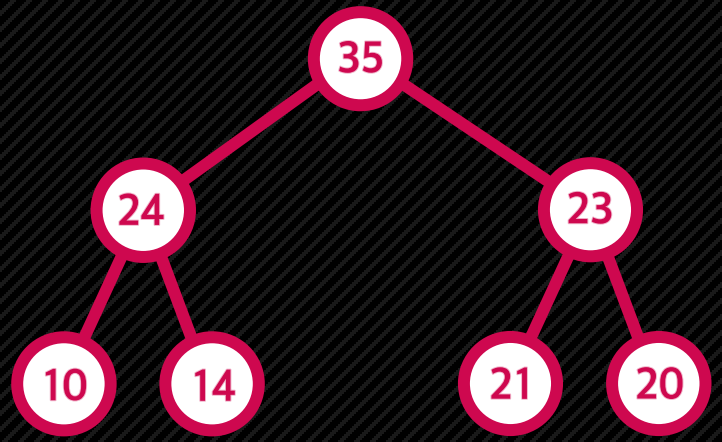
- 총 연산의 수 N은 $1 <= N <= 100,000$ 이다.
- 입력되는 자연수는 2^{31} 미만 이다.

그럼 최대 힙은?

```
_list = [32, 16, 54, 94, 81, 31]
_list = list(map(lambda x : x * -1, _list))
heapq.heapify(_list)
```

- -1을 곱하면 대소 관계가 뒤집힘! → 각 원소에 -1을 곱하자.

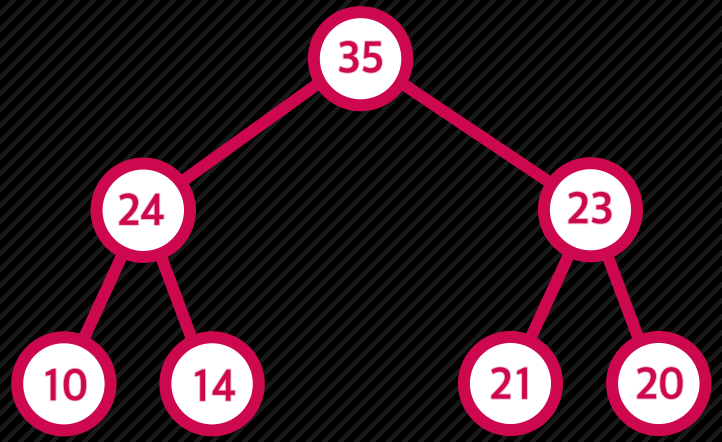
그래서 힙은?



Heap

- 완전 이진 트리의 일종으로, 부모의 값이 항상 자식보다 크거나 작아야 함.
- 즉, 루트는 최댓값이거나, 최솟값임이 보장됨.
- 최댓값/최솟값을 O(1)만에 찾을 수 있는 자료구조.

그래서 힙은?



Heap

- 최댓값/최솟값을 O(1)만에 찾을 수 있는 자료구조.

힙을 활용해보자!

Gold 4 - N번째 큰 수 (#2075)

요약

- N * N 개의 자연수가 있다.
- 자연수가 주어졌을 때, N번째 큰 수를 찾는 프로그램을 작성하시오.

제약조건

- N은 1 <= N <= 1,500 이다.
- 각각의 수의 범위는 $1 <= N_i <= 1,500$ 미만 이다.
- 사용되는 메모리는 12MB 이하여야 한다.

힙을 활용해보자!

Gold 4 - N번째 큰 수

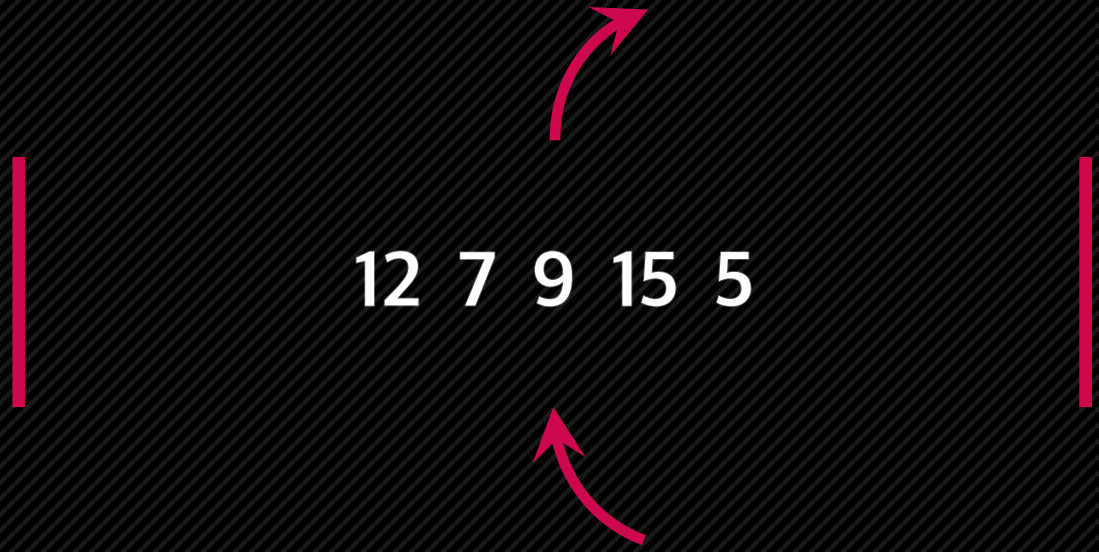
Heap

- 최댓값/최솟값을 O(1)만에 찾을 수 있는 자료구조.

힙을 활용해보자!

Gold 4 - N번째 큰 수

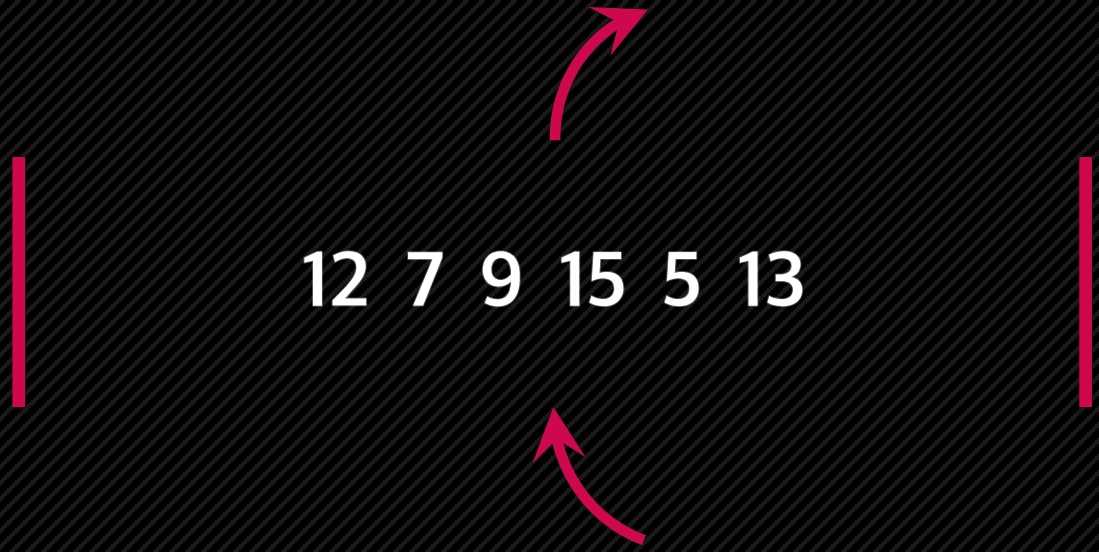
12 7 9 15 5
13 8 11 19 6
21 10 26 31 16
48 14 28 35 25
52 20 32 41 49



힙을 활용해보자!

Gold 4 - N번째 큰 수

12 7 9 15 5
13 8 11 19 6
21 10 26 31 16
48 14 28 35 25
52 20 32 41 49



힙을 활용해보자!

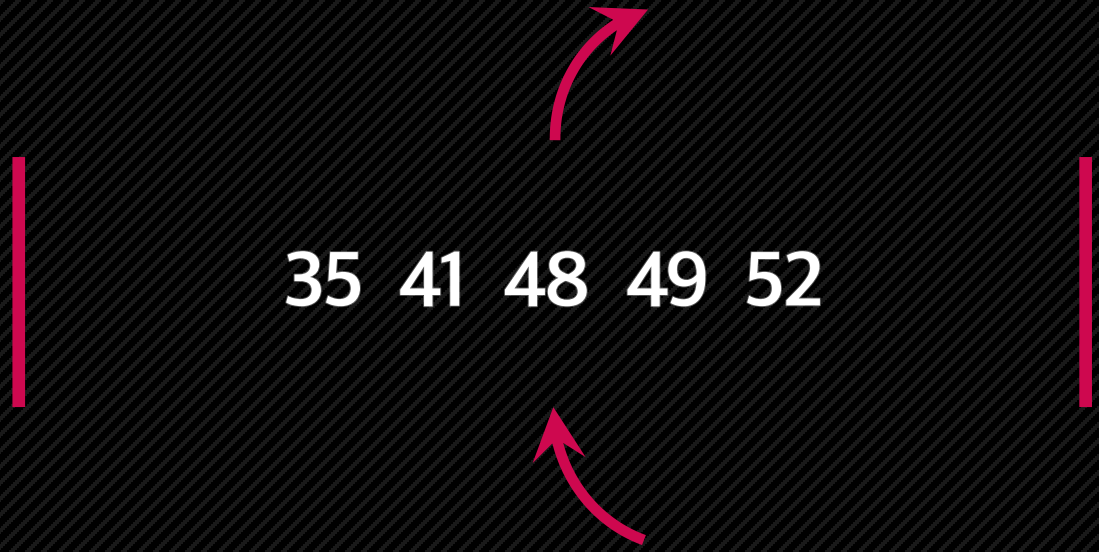
Gold 4 - N번째 큰 수

12 7 9 15 5
13 8 11 19 6
21 10 26 31 16
48 14 28 35 25
52 20 32 41 49

힙을 활용해보자!

Gold 4 - N번째 큰 수

12 7 9 15 5
13 8 11 19 6
21 10 26 31 16
48 14 28 35 25
52 20 32 41 49



최댓값과 최솟값을 활용하는 문제는 힙을 고려해보자!

Level 2 - 더 맵게

요약

- 모든 음식의 스코빌 지수를 K 이상으로 만드려고 한다.
- 스코빌 지수가 K 미만인 경우, 음식을 섞어 스코빌 지수를 올리려고 하는데, 이때 스코빌 지수는 (낮은 스코빌 지수) + (두번째로 낮은 스코빌 지수) * 2 가 된다.
- 모든 음식의 스코빌 지수가 K 이상이 되도록 하기 위해 섞어야 하는 최소 횟수를 구하는 프로그램을 작성하시오.

제약조건

- 음식의 수의 범위는 $2 <= N <= 1,000,000$ 이다.
- 음식의 스코빌 지수의 범위는 $0 <= N_i <= 1,000,000$ 이다.
- K의 범위는 $2 <= K <= 1,000,000,000$ 이다.

이런 건 어떨까?

Gold 3 - 중앙값 구하기 (#2696)

요약

- 홀수 번째 수를 읽을 때 마다, 지금까지 입력 받은 값의 중앙값을 출력하는 프로그램을 작성하시오.

제약조건

- 수열의 크기 M은 $1 <= M <= 9,999$ 이다.
- 각각의 수의 범위는 $1 <= M_i <= 2^{32}-1$ 미만 이다.

이런 건 어떨까?

Gold 3 - 중앙값 구하기

아니!!! 힙은 최솟값과 최댓값만 된다면서요!!!!!

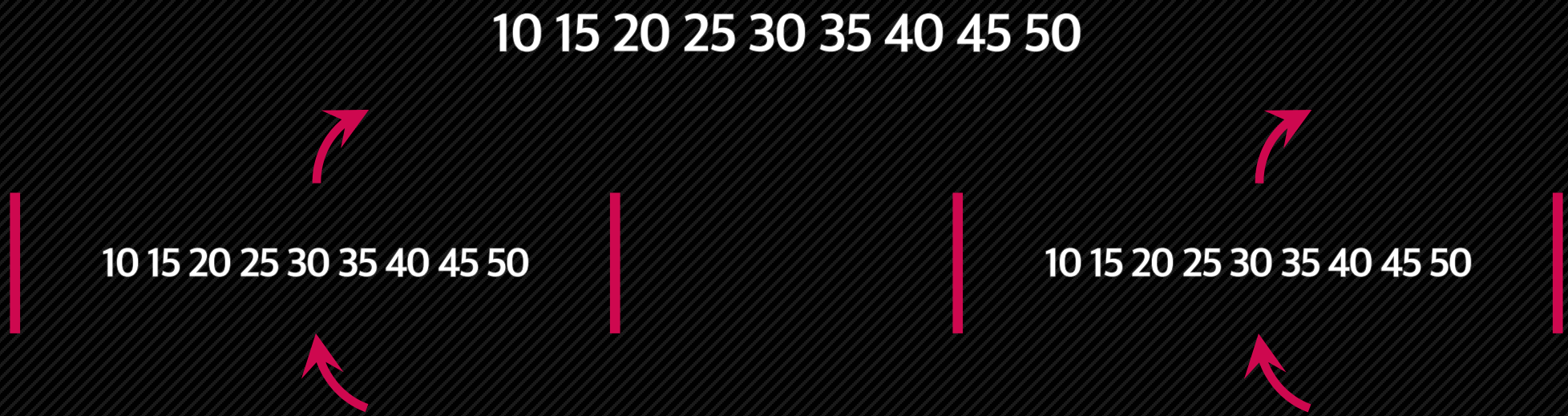
이런 건 어떨까?

Gold 3 - 중앙값 구하기

10 15 20 25 30 35 40 45 50

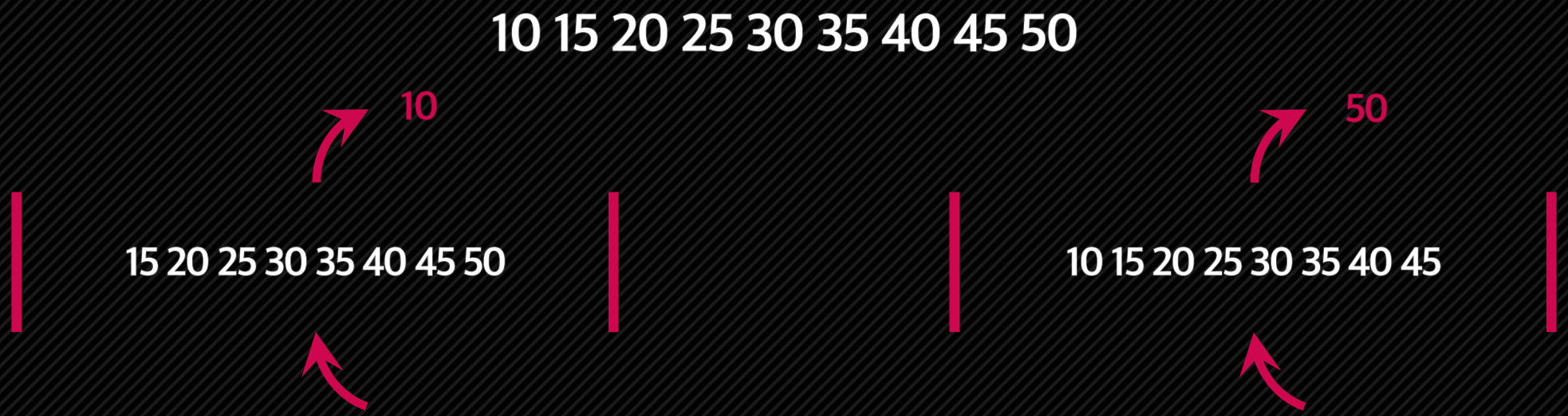
이런 건 어떨까?

Gold 3 - 중앙값 구하기



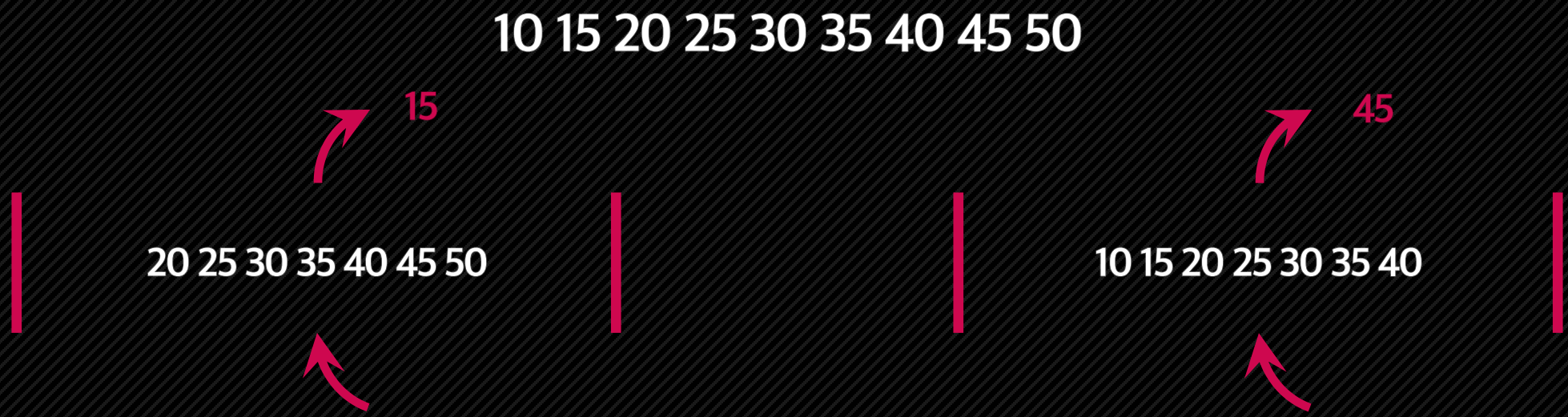
이런 건 어떨까?

Gold 3 - 중앙값 구하기



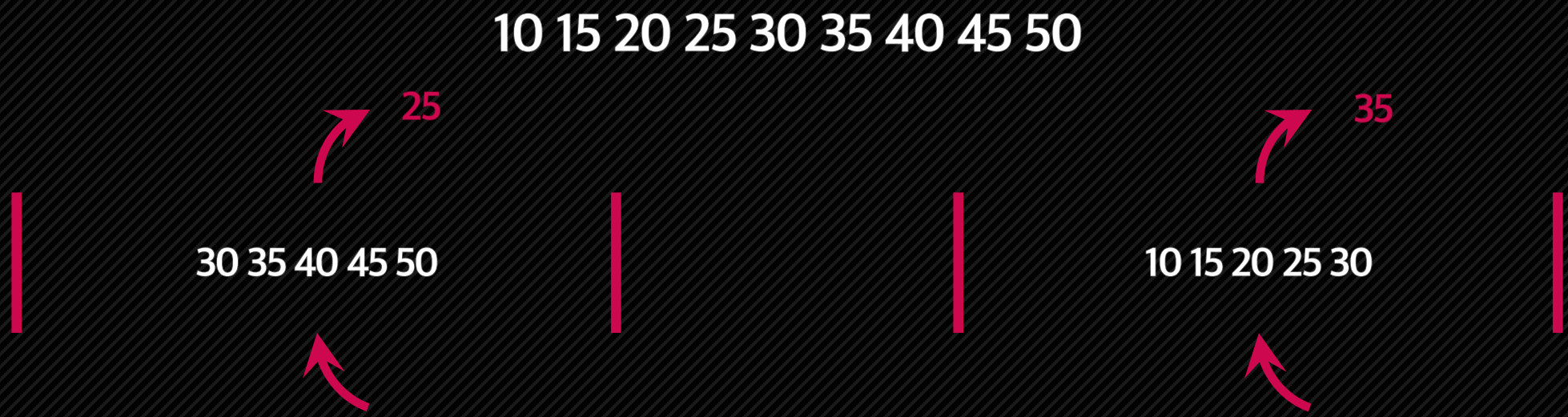
이런 건 어떨까?

Gold 3 - 중앙값 구하기



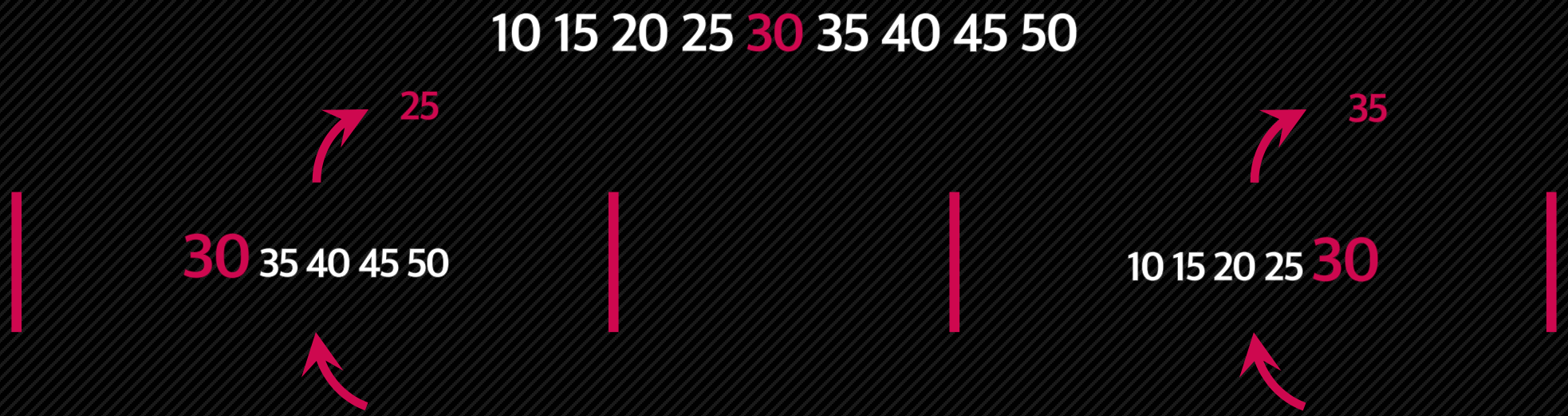
이런 건 어떨까?

Gold 3 - 중앙값 구하기



이런 건 어떨까?

Gold 3 - 중앙값 구하기



“Any Question?”