알고리즘 특강

박성훈

- https://www.noii.kr
- noii@kkokiyo.app

- 경력
  - **現** 꼭코퍼레이션 대표이사
  - **前** 셀러비코리아 대표이사
  - **前** 그루브아이티 대표이사
  - 건국대 강의 교수 재직
  - 건국대 컴퓨터/정보통신공학 박사 수료
  - 건국대 컴퓨터/정보통신공학 석사 졸업

목차

알고리즘

- 알고리즘의 성능

자료구조

- Array
- List
- Stack
- Queue
- Hash table
- Graph
- Tree
- Heap

대표 알고리즘

- Search (Linear, Binary)
- Sort (Selection, Insertion, Bubble, Merge, Quick)
- Tree (DFS, BFS)
- Two pointers
- Greedy
- Dijkstra’s

코딩테스트?

- 프로그래머스
- 실전 문제 풀이

알고리즘

알고리즘

많은 알고리즘 속에서 일상생활을 보내고 있습니다

알고리즘

문제 해결 방법을 정의한 '**일련의 단계적 절차**'이자 어떠한 문제를 해결하기 위한 '**동작들의 모임**'

문제 풀이에 필요한 계산 절차 또는 처리 과정의 순서, 프로그램 명령어 집합

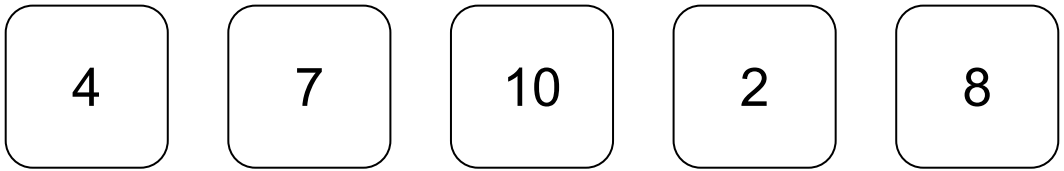
어떤 문제를 해결함에 있어서 특정 동작을 반복하면, 해당 문제가 풀림

동일한 문제에 대해서 일관적인 성능을 보임

예외상황이 많지 않음 (수학적 증명)

알고리즘

다음 수를 오름차순으로 정렬해보세요



알고리즘

1 ~ 100,000,000 사이 임의의 숫자 50,000,000개가 뒤섞여 있습니다

이 숫자들을 오름차순으로 정렬해보세요

알고리즘

1 ~ 100,000,000 사이 임의의 숫자 50,000,000개가 뒤섞여 있습니다

이 숫자들을 오름차순으로 정렬해보세요

- 1 발견, 가장 작은 숫자이니 맨 앞에 놓기
- 10, 100, 1000, … 각 단위별로 나누기
- 카드 두개를 들어 비교하기
- 남아 있는 카드 중 하나를 골라 제자리 찾기
- …

알고리즘 성능

문제를 해결함에 있어서 알고리즘이 **얼마의 리소스(시간, 메모리)를 필요**로 하는지 평가

시간복잡도(Time Complexity)

: 입력 크기에 따른 알고리즘 실행 시간

공간복잡도(Space Complexity)

: 알고리즘이 수행되는 동안 추가로 필요한 메모리 공간의 양

표기법

최상의 상황일 때 측정: 오메가 표기법 (Big-Ω Notation)

평균 성능 측정: 세타 표기법 (Big-θ Notation)

최악의 상황일 때 측정: 빅오 표기법 (**Big-O** Notation)

알고리즘 성능

책 10만권이 꽂혀있는 도서관이 있다.

책은 대출 후 반납 시 빈자리 아무데나 꽂아서 순서 없이 꽂혀있다.

성훈은 “수학의 정석” 책을 찾고자 한다.

찾는 방식은 다음과 같다.

- 임의의 책을 하나 꺼낸다.
- 해당 책이 찾고자 하는 책이 맞는지 확인한다.
- 맞으면 그 책을 들고 나간다.
- 틀리다면 그 책은 바닥에 내려두고 계속한다.
- 이 일련의 과정을 1초에 해결한다.

알고리즘 성능

최고의 상황 - Ω(1)

: 처음 고른 책이 내가 찾는 책 일 때

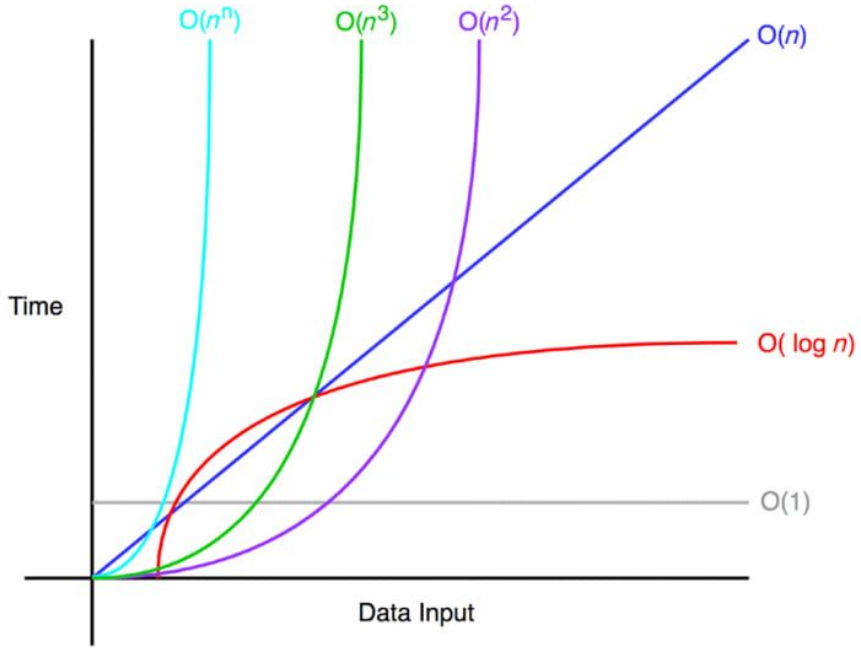
최악의 상황 - O(10만) = O(N)

: 맨 마지막(10만번째)에 고른 책이 내가 찾는 책 일 때

알고리즘 성능

Big-O Complexity 일반적으로 1억초가 걸리면 '계산할 수 없다'고 판단

complexity	1	10	100
O(1)	1	1	1
O(logN)	0	2	5
O(N)	1	10	100
O(NlogN)	0	20	461
$O(N^2)$	1	100	10,000
$O(2^N)$	2	1024	…
O(N!)	1	3,628,800	…



알고리즘 성능

스포츠 리그에서 한 시즌동안 이뤄져야하는 경기의 수를 구하고, 경기 리스트를 출력하려고 한다.

각 팀은 다른 모든 팀과 경기를 2회(홈/어웨이)씩 경기를 한다.

알고리즘 성능

2중 for문 활용

같은 팀 제외, 모두 출력

n번 돌아가는 반복문 중첩 = 실행 횟수 n * n = O(n2)

```python
def algorithm(n):
    count = 0

    for i in range(n):
        for j in range(n):
            if i != j:
                count += 1
                print(f"T{i+1} (home) vs T{j+1} (away)")

    return count
```

알고리즘 성능

2중 for문 활용

같은 팀 제외, 모두 출력

n번 돌아가는 반복문 중첩 = 실행 횟수 n * n = O(n2)

경우의 수 계산

n * (n - 1) = 20 * 19 = 380

상수를 제외하고 표현 $O(n^2)$

```
1   def algorithm(n):
2       count = 0
3
4       for i in range(n): 20번
5           for j in range(n): * 20번 = 400번 반복
6               if i != j:
7                   count += 1
8                   print(f"T{i+1} (home) vs T{j+1} (away)")
9
10      return count
```

자료구조

자료구조

컴퓨터 과학에서 데이터를 **효율적**으로 저장, 관리, 접근하기 위한 체계적인 방식

데이터의 조직화: 데이터 값들의 모임과 그 관계를 논리적으로 정의

연산: 데이터를 조작하기 위한 삽입, 수정, 삭제, 검색 등과 같은 기본 연산을 지원

효율성: 한정된 컴퓨터 메모리 공간에서 데이터를 효과적으로 다루는 것이 핵심

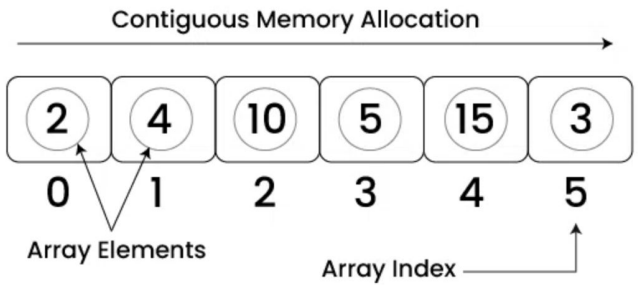
여기에 텍스트 입력

Array

동일한 데이터 타입의 요소들을 연속된 메모리 공간에 저장

선언 시 사이즈 및 데이터 타입 지정

Index를 통하여 각 요소에 접근, 순서 유지



Array

선언 시 필요 메모리 확보

메모리상의 선언된 시작 주소만 기억

각 요소별 접근: 시작 주소 + @

arr[5] = 1200 + 5 * 4 (자료형 크기)= 1220 접근

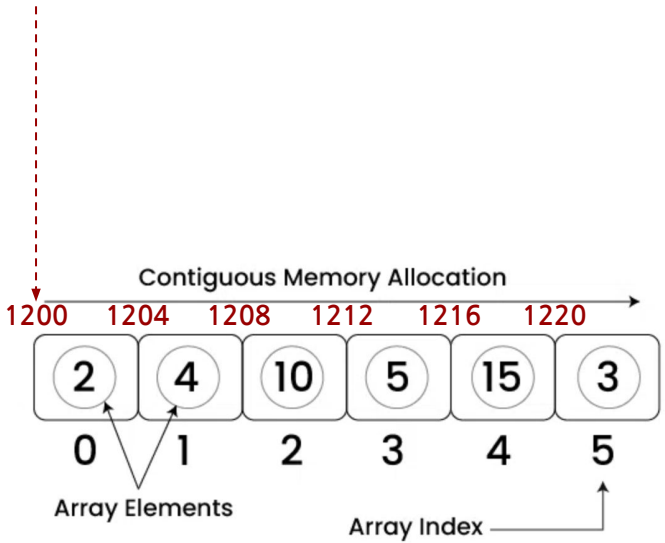
일반적으로 int형은
C언어에서는 4byte,
파이썬에서는 8byte 차지

👍 각 요소별 접근 편리

👎 동일한 자료형 저장 가능

👎 비 효율적 메모리 사용

```
int arr[6] = {2, 4, 10, 5, 15, 3};
```



List (Abstract Data Type)

순서가 있는 데이터 원소들의 집합

특정 위치(Index)를 기준으로 삽입, 삭제, 탐색이 가능한 선형 구조

다양한 자료형 저장 가능

데이터의 중복 허용, 리스트 길이 동적 변경

API

Method	설명
insert(index, item)	index 위치에 item 삽입
delete(index)	index 위치의 원소 삭제
get(index)	index 위치의 원소 반환
update(index, item)	index 위치 값을 item으로 변경
length()	리스트 길이 반환
isEmpty()	리스트가 비어있는지 확인

Array List(Linear List)

(리스트 개념을)

Array를 기반으로 구현

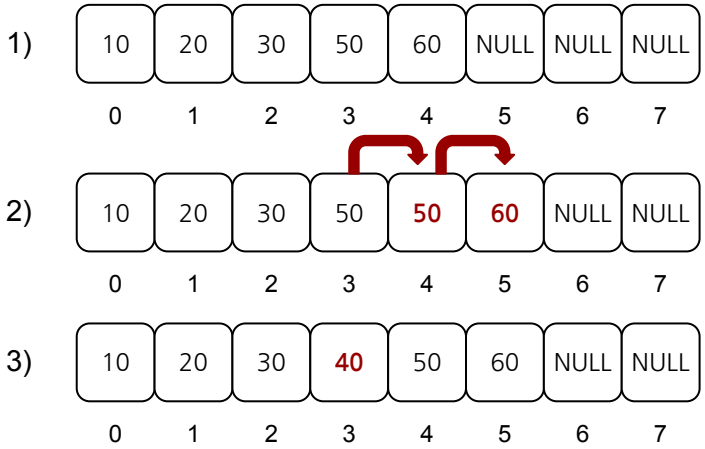
연속된 메모리상에 위치하므로 빠른 접근 가능

삽입, 삭제 시 데이터 이동 필요

가용 범위를 초과한 삽입 시, 추가 작업 필요

⇒ 비효율적

insert(3, 40) 실행 시



insert(3, 40) 실행 시 한 칸 = Array

Linked List 비상 연락망 느낌(다음 사람이 누군지만 알고 있음)

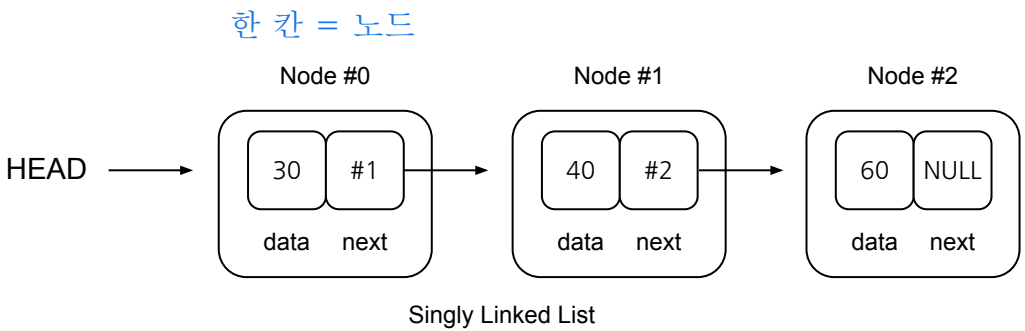
각 노드가 데이터와 다음 노드에 대한 참조로 선언

순차적으로 연결된 선형 자료구조

동적 크기, 메모리 비 연속적

종류

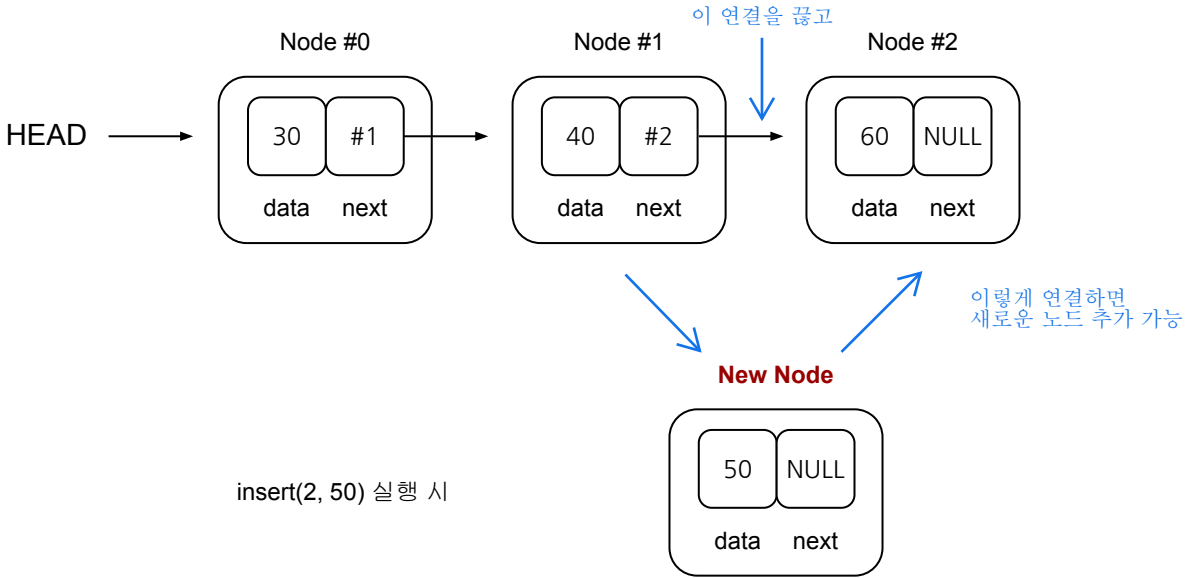
- 단일 연결 리스트(Singly Linked List)
- 이중 연결 리스트(Doubly Linked List)
- 원형 연결 리스트(Circular Linked List)



Linked List

연결 변경만으로 중간 삽입, 삭제 가능

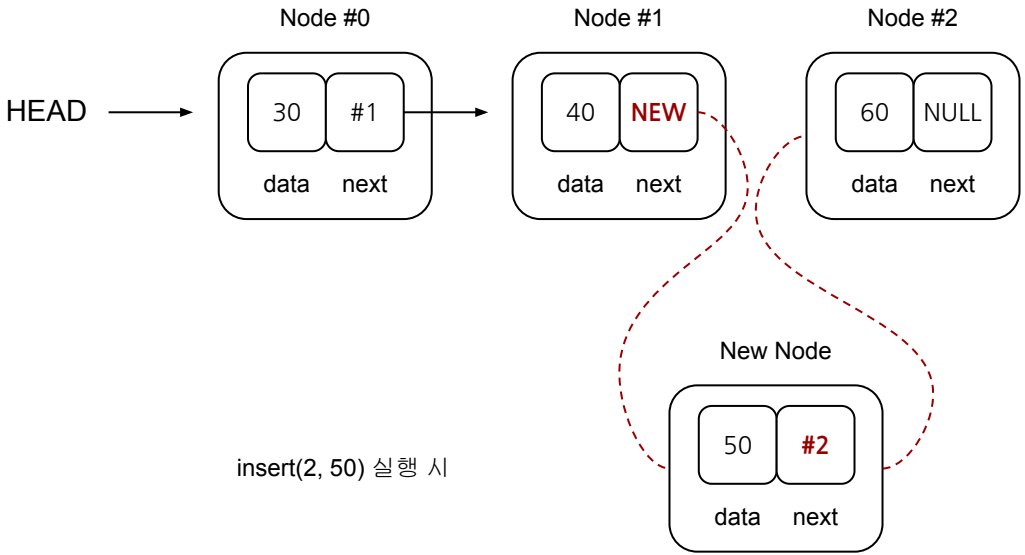
요소 접근 시 순차적으로 접근하여야 함으로 느림



Linked List

연결 변경만으로 중간 삽입, 삭제 가능

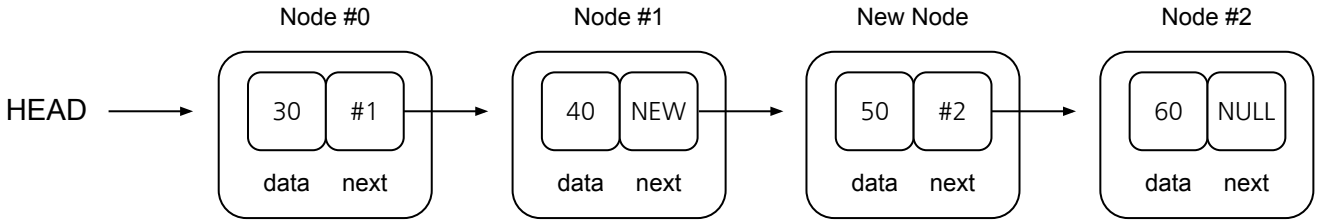
요소 접근 시 순차적으로 접근하여야 함으로 느림



Linked List

연결 변경만으로 중간 삽입, 삭제 가능

요소 접근 시 순차적으로 접근하여야 함으로 느림



Linked List 구현하기

python > 251030 > liked_list.py에 구현해봄
파이썬 자체에 이미 있음

Python에서 다음 API를 지원하는 Linked List 구현

Method	설명
insert(index, item)	index 위치에 item 삽입
delete(index)	index 위치의 원소 삭제
get(index)	index 위치의 원소 반환
update(index, item)	index 위치 값을 item으로 변경
length()	리스트 길이 반환
isEmpty()	리스트가 비어있는지 확인

Linked List 구현하기

Node Class 선언

- data: 실제 데이터 저장
- next: 뒤에 따르는 node 저장

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

Linked List 구현하기

LinkedList Class 선언

- head: Linked List의 첫 시작 Node

```
class LinkedList:
    def __init__(self):
        self.head = None
```

Linked List 구현하기

LinkedList Class 내 API 구현

- isEmpty
  : head가 None인지 확인하여 전달
-

```
def isEmpty(self):
    return self.head is None
```

Linked List 구현하기

LinkedList Class 내 API 구현

- length
  : next가 있으면 계속 따라가면서 count 증가

```
def length(self):
    count = 0
    current = self.head
    while current:
        count += 1
        current = current.next
    return count
```

Linked List 구현하기

LinkedList Class 내 API 구현

- print_list
  : next가 있으면 계속 따라가면서 출력

```
def print_list(self):
    current = self.head
    while current:
        print(current.data, end=' -> ')
        current = current.next
    print('None')
```

Linked List 구현하기

LinkedList Class 내 API 구현

- insert
  : 새로운 Node를 만들어
  기존 연결을 끊고, 중간에 삽입

```
def insert(self, index, item):
    new_node = Node(item)
    if index == 0: # index == 0 head에 삽입
        new_node.next = self.head
        self.head = new_node
        return

    prev = self.head # 해당 index까지 이동
    for _ in range(index - 1):
        if prev is None:
            raise IndexError("Index out of range")
        prev = prev.next

    # 새 Node의 Next는 Prev Node의 Next
    new_node.next = prev.next
    # Prev Node의 Next는 새 Node
    prev.next = new_node
```

Linked List 구현하기

LinkedList Class 내 API 구현

- delete
  : 해당 위치 삭제 후, 앞뒤 연결

```
def delete(self, index):
    # List가 비어있는지 확인
    if self.head is None:
        raise IndexError("List is empty")

    # 헤드를 삭제할 경우
    if index == 0:
        self.head = self.head.next
        return

    # 해당 index가 정상적인지 확인
    prev = self.head
    for _ in range(index - 1):
        if prev.next is None:
            raise IndexError("Index out of range")
        prev = prev.next

    if prev.next is None:
        raise IndexError("Index out of range")

    # 삭제 후, 앞 - 뒤 연결
    prev.next = prev.next.next
```

Linked List 구현하기

LinkedList Class 내 API 구현

- get
  : 해당 index 위치의 data 반환

```
def get(self, index):
    current = self.head
    for _ in range(index):
        if current is None:
            raise IndexError("Index out of range")
        current = current.next

    if current is None:
        raise IndexError("Index out of range")

    return current.data
```

Linked List 구현하기

LinkedList Class 내 API 구현

- update
  : 해당 index 위치의 data 변경

```
def update(self, index, item):
    current = self.head
    for _ in range(index):
        if current is None:
            raise IndexError("Index out of range")
        current = current.next

    if current is None:
        raise IndexError("Index out of range")

    current.data = item
```

Linked List 구현하기

Linked List 구현 확인

```
>>> from linkedlist import LinkedList
>>> ll = LinkedList()
>>> ll.isEmpty()
True
>>> ll.insert(0, 'A')
>>> ll.isEmpty()
False
>>> ll.insert(0, 1000)
>>> ll.print_list()
1000 -> A -> None
>>> ll.insert(1, '*')
>>> ll.print_list()
1000 -> * -> A -> None
>>> ll.delete(0)
>>> ll.print_list()
* -> A -> None
>>> ll.get(1)
'A'
>>> ll.length()
2
```

Stack

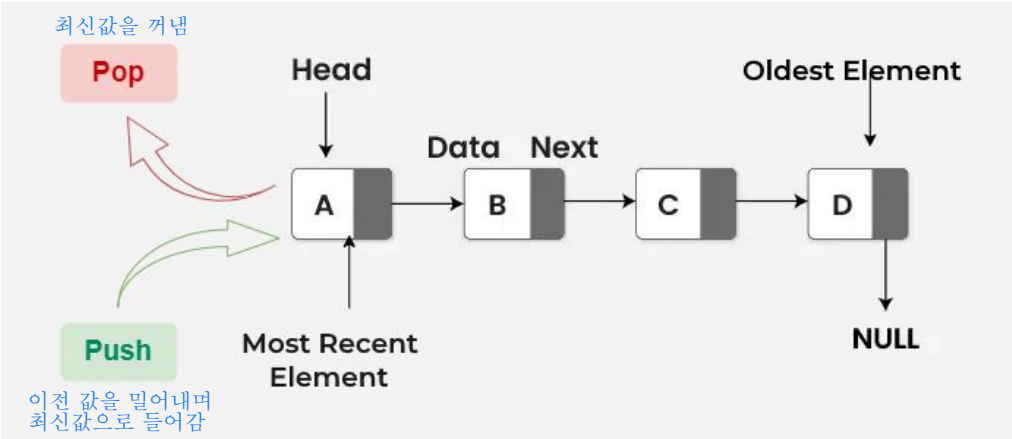
선형 자료구조로써 후입선출(LIFO - Last In, First Out) 방식으로 구현된 자료구조

사용 예시

- 웹브라우저 히스토리(뒤로가기)
- 실행 취소 (Undo)

API

Method	설명
push(item)	Stack 최 상단에 항목 추가
pop()	Stack 최 상단 제거 및 반환
peek() or top()	Stack 최 상단 반환 (제거 X)
isEmpty()	Stack이 비었는지 여부 반환
size() or length()	Stack의 쌓인 항목 수 반환



Queue

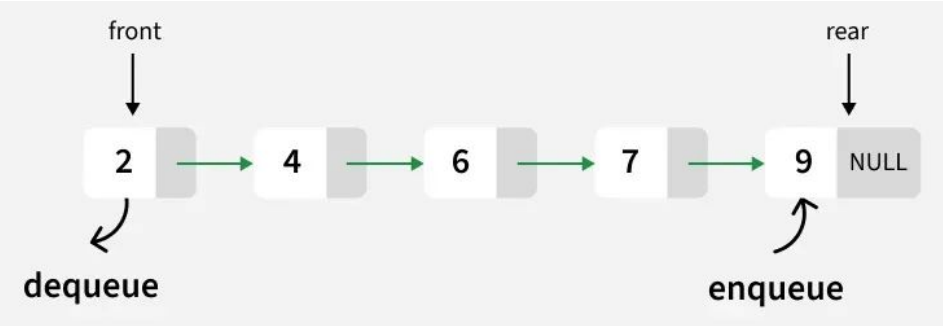
선형 자료구조로써, 선입선출(FIFO - First In, First Out)방식으로 구현된 자료구조

사용 예시

- 사이트 접속 대기
- 주문 리스트

API

Method	설명
enqueue(item)	Queue 뒤(rear)에 항목 추가
dequeue()	Queue 앞 요소 제거 및 반환
peek() or front()	Queue 앞 요소 반환 (제거 X)
isEmpty()	Queue 비었는지 여부 반환
size() or length()	Queue의 쌓인 항목 수 반환



= Python의 딕셔너리와 유사

Hash table

키(key)를 해시 함수(hash function)를 통해 배열의 인덱스(index)로 변환하여 데이터를 저장

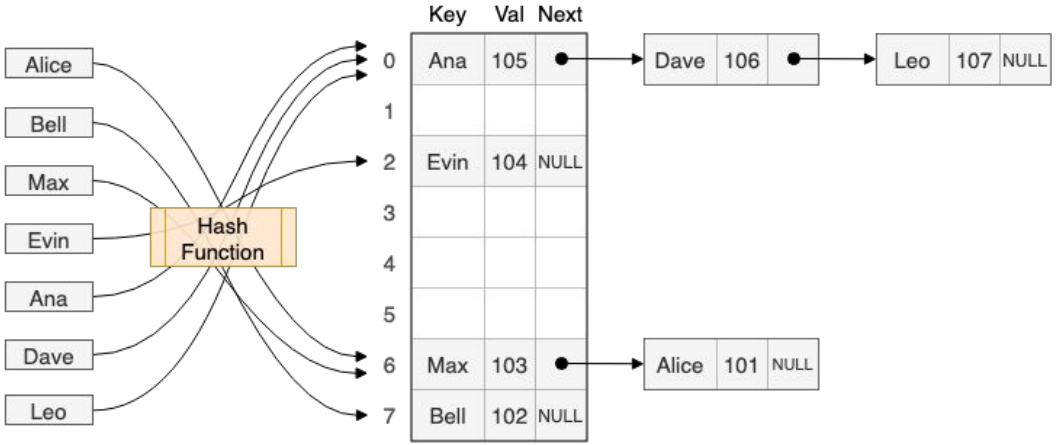
검색, 삭제, 삽입에 탁월한 성능

해시 충돌(Hash collision) 문제 해결 필요

- 서로 다른 키에 대해 동일한 인덱스 값 반환 문제

API

Method	설명
insert(key, value)	데이터를 해시 테이블에 삽입
delete(key)	키에 해당하는 항목 삭제
get(key)	키에 해당하는 항목 조회



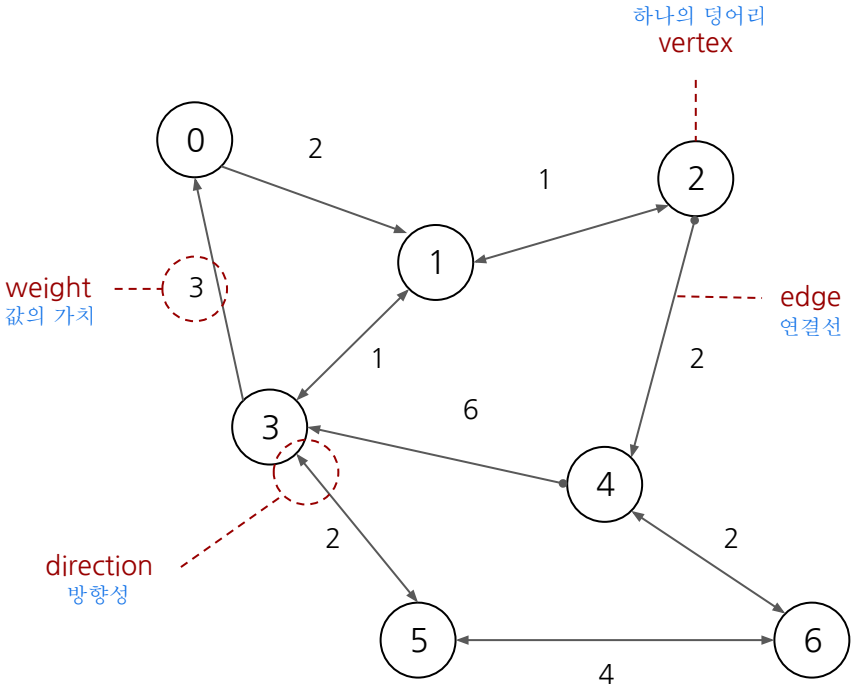
Graph

데이터간 연결 관계를 나타내는 **비선형** 자료구조

용어 정리

vertex	데이터를 저장하고 있는 위치
adjacent vertex	특정 vertex와 직접 연결된 vertex
edge	vertex 간 연결 선
direction	edge의 방향성
weight	edge를 지날 때 비용

예) 내비게이션에서 목적지 도착까지 최적의 경로를 찾을 때 사용



Tree

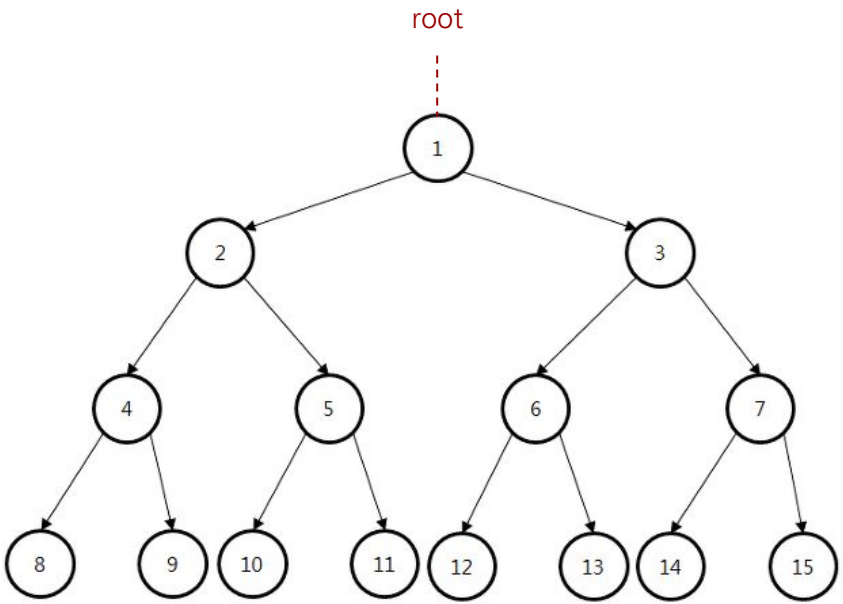
Graph의 한 종류로 노드(Node)들이 부모-자식 관계를 갖는 계층적 비선형 자료구조

노드 개수 N개일 때, 간선은 항상 N - 1개

사이클 없음

모든 노드는 단 하나의 부모를 가짐 (루트 제외)

순차적인 관계성(부모-자식)을 저장할 때 사용
예) 파일 경로



'파이썬 Heap 구현' 검색하면 다양한 샘플 코드 볼 수 있음

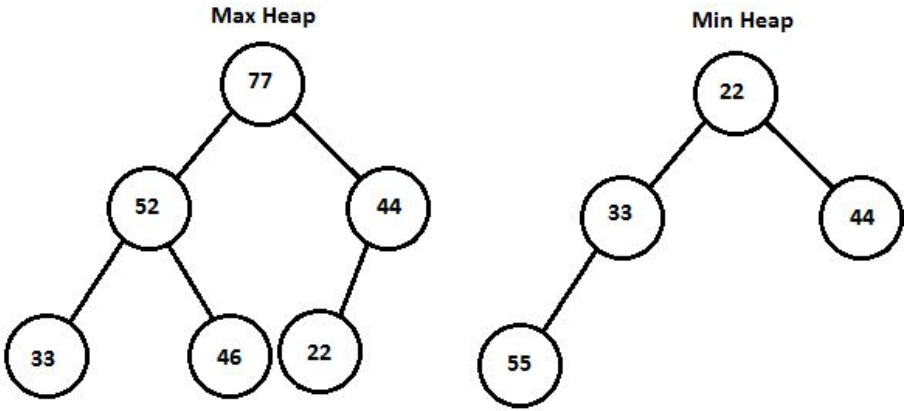
Heap

Heap은 완전 이진 트리(Complete Binary Tree)를 기반으로 한 우선순위 기반의 트리형 자료구조

항상 부모 노드가 자식 노드보다 크거나 작도록 정렬된 구조

완전 이진 트리

- 자식 노드의 수는 항상 2개
- 마지막 레벨을 제외하고는 항상 2개씩
- 노드는 왼쪽에서 오른쪽으로 채워짐



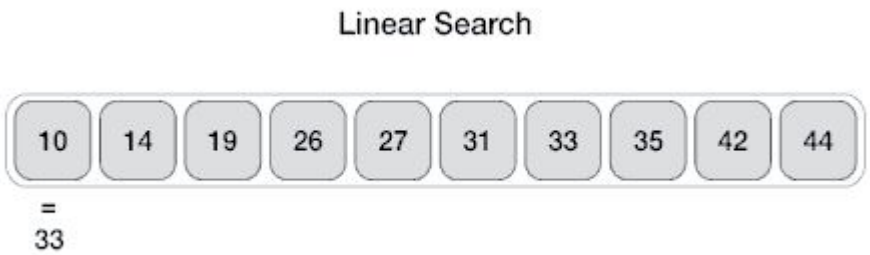
대표 알고리즘

검색 - Linear search

리스트에서 순차적으로 값을 찾는 알고리즘

시간복잡도: O(N)

정렬되어 있지 않은 리스트 또는 데이터가 적을 때 효과적



앞에서부터 순차적으로 조건과 대조하며 원하는 값을 찾아감

python > 251030 > search.py

Linear search 구현

```python
def linear_search(data_list, target):
    # list의 모든 요소를 순서대로 탐색
    for i in range(len(data_list)):
        # 현재 요소가 target과 같은지 확인
        if data_list[i] == target:
            # 같으면 해당 인덱스 반환
            return i
    # 모든 요소를 탐색했는데 target을 찾지 못했으면 -1 반환
    return -1
```

```python
test_list = [5, 2, 8, 1, 9, 4]
target_1 = 9
target_2 = 10
target_3 = 8

print(f"리스트: {test_list}")

# 9를 탐색 (찾음)
index_1 = linear_search(test_list, target_1)
print(f"목표 값 {target_1}의 인덱스: {index_1}")
# 출력: 목표 값 9의 인덱스: 4

# 10을 탐색 (못 찾음)
index_2 = linear_search(test_list, target_2)
print(f"목표 값 {target_2}의 인덱스: {index_2}")
# 출력: 목표 값 10의 인덱스: -1

# 8을 탐색 (찾음)
index_2 = linear_search(test_list, target_2)
print(f"목표 값 {target_2}의 인덱스: {index_2}")
# 출력: 목표 값 8의 인덱스: 2
```

검색 - Binary search

정렬되어 있는 리스트에서 빠르게 값을 찾는 알고리즘

시간복잡도: O(logN)

동작 방식

- 리스트에 중간 위치 값을 탐색
- 찾고자하는 값이면 종료
- 그렇지 않다면, 중간 값과 비교하여 좌/우 구간 중 하나 선택
- 중간 값 탐색부터 반복

Binary search 구현

```python
def binary_search(data_list, target):
    # 탐색 범위의 시작점(low)과 끝점(high)을 설정
    low = 0
    high = len(data_list) - 1

    # low가 high보다 작거나 같을 때까지 반복함
    while low <= high:
        # 중간 지점(mid)을 계산
        mid = (low + high) // 2

        # 1. 중간 지점의 값이 목표 값과 일치하는 경우
        if data_list[mid] == target:
            return mid  # 인덱스를 반환하고 종료

        # 2. 중간 지점의 값이 목표 값보다 작은 경우
        # 목표 값은 중간 지점의 오른쪽(큰 쪽)에 존재
        elif data_list[mid] < target:
            low = mid + 1  # 탐색 범위를 mid의 오른쪽으로 좁힘

        # 3. 중간 지점의 값이 목표 값보다 큰 경우
        # 목표 값은 중간 지점의 왼쪽(작은 쪽)에 존재
        else:
            high = mid - 1 # 탐색 범위를 mid의 왼쪽으로 좁힘

    # 반복문이 종료될 때까지 찾지 못하면 -1을 반환
    return -1
```

```python
# 이진 탐색은 반드시 '정렬된' 리스트를 사용
my_list = [1, 4, 8, 9, 11, 15, 20]
target_1 = 11
target_2 = 10

# 11을 탐색 (찾음)
index_1 = binary_search_iterative(my_list, target_1)
print(f"리스트: {my_list}")
print(f"목표 값 {target_1}의 인덱스: {index_1}")
# 출력: 목표 값 11의 인덱스: 4

# 10을 탐색 (못 찾음)
index_2 = binary_search_iterative(my_list, target_2)
print(f"목표 값 {target_2}의 인덱스: {index_2}")
# 출력: 목표 값 10의 인덱스: -1
```

Linear search vs Binary search

```python
import time
from linear_search import linear_search
from binary_search import binary_search

big_list = []
for i in range(100000000):
    big_list.append(i)

target = 99999998

start_time = time.time()
result = binary_search(big_list, target)
end_time = time.time()
binary_result = end_time - start_time
print("Binary result: ", result)

start_time = time.time()
result = linear_search(big_list, target)
end_time = time.time()
linear_result = end_time - start_time
print("Linear result: ", result)

print("Binary search time: ", binary_result)
print("Linear search time: ", linear_result)

print("Binary < Linear: ", binary_result < linear_result)
```

1억건 실행 결과 비교

Binary result: 99999998

Linear result: 99999998

Binary search time: 3.3140182495117l9e-05

Linear search time: 1.630479097366333

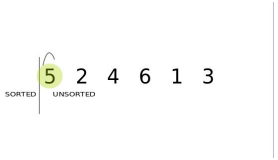
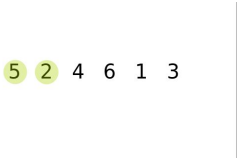
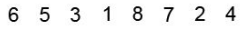
Binary < Linear: True

시간 차이 많이 남

⇒ 데이터 정렬이 중요

정렬

정렬되어있지 않은 요소 중 하나를 뽑아 올바른 위치에 정렬

Selection Sort	Insertion Sort	Bubble Sort	Merge Sort	Quick Sort
해당 위치에 올 요소 찾아서 가져오기 시간: O(N^2) 공간: O(1)	해당 요소의 위치를 찾아서 삽입 시간: O(N^2) 공간: O(1)	인접한 두개씩 계속 비교해 나가기 시간: O(N^2) 공간: O(1)	부분을 나눠서 정렬, 값 합치기 시간: O(NlogN) 공간: O(N)	Pivot을 기준으로 좌우로 분할하여 정리 시간: O(NlogN) 공간: O(NlogN)
				

Selection sort 구현

5 3 4 1 2

```python
def selection_sort(data_list, descending=False):
    n = len(data_list)

    # 1. 리스트의 길이만큼 반복 (마지막 요소는 자동으로 정렬되므로 n-1번만 반복)
    for i in range(n - 1):
        # 현재 정렬되지 않은 부분에서 가장 작은 요소의 인덱스를 저장
        target_index = i # 시작 값 = 현재 정렬코자하는 위치 인덱스

        # 2. 정렬되지 않은 나머지 부분(i+1부터 끝까지)에서 최소값 찾기
        for j in range(i + 1, n):
            if descending:
                if data_list[j] > data_list[target_index]:
                    target_index = j
            else:
                if data_list[j] < data_list[target_index]:
                    target_index = j

        # 3. 찾은 최소값(target_index)과 현재 위치(i)의 요소를 교환
        data_list[i], data_list[target_index] = data_list[target_index], data_list[i]

    return data_list
```

Insertion sort 구현


```python
def insertion_sort(data_list, descending=False):
    n = len(data_list)

    # 1. 두 번째 요소 (인덱스 1)부터 시작하여 리스트 끝까지 반복
    for i in range(1, n):
        # 현재 삽입할 요소 (Key)
        key = data_list[i]

        # key를 삽입할 위치를 찾기 위해, 정렬된 앞부분(i-1부터 0까지)을 탐색
        j = i - 1

        # 2. 앞부분의 요소들을 key와 비교하여, key가 들어갈 공간을 만들기
        # 조건:
        #   1) j가 0 이상이어야 하고 (리스트의 시작을 넘어가지 않도록)
        #   2) 현재 정렬된 요소 data_list[j]가 key와 비교
        while j >= 0 and (data_list[j] > key if not descending else data_list[j] < key):
            # 현재 요소를 한 칸 뒤로 밀어냄 (key를 위한 공간 확보)
            data_list[j + 1] = data_list[j]
            j -= 1

        # 3. while 반복문이 끝난 후, j+1 위치가 key가 삽입될 최종 위치
        data_list[j + 1] = key

    return data_list
```

Bubble sort 구현

5 2 4 6 1 3

```python
def bubble_sort(data_list, descending=False):
    n = len(data_list)

    # 1. Outer Loop: 정렬 과정을 n-1번 반복
    # 매 반복마다 가장 큰 요소가 제 위치(배열의 끝)로 '버블링'
    for i in range(n - 1):
        # 교환이 발생했는지 확인하는 플래그 (최적화)
        swapped = False

        # 2. Inner Loop: 인접한 두 요소를 비교하고 교환
        # 이미 정렬이 완료된 끝 부분은 제외하고 비교 (n-1-i)
        for j in range(n - 1 - i):
            # 인접한 요소 비교
            if descending:
                if data_list[j] < data_list[j + 1]:
                    data_list[j], data_list[j + 1] = data_list[j + 1], data_list[j]
                    swapped = True
            else:
                if data_list[j] > data_list[j + 1]:
                    data_list[j], data_list[j + 1] = data_list[j + 1], data_list[j]
                    swapped = True

        # 3. 최적화: Inner Loop에서 한 번도 교환이 일어나지 않았다면,
        # 리스트는 이미 정렬된 상태이므로 반복을 종료
        if not swapped:
            break
```

재귀함수 형태로 많이 구현

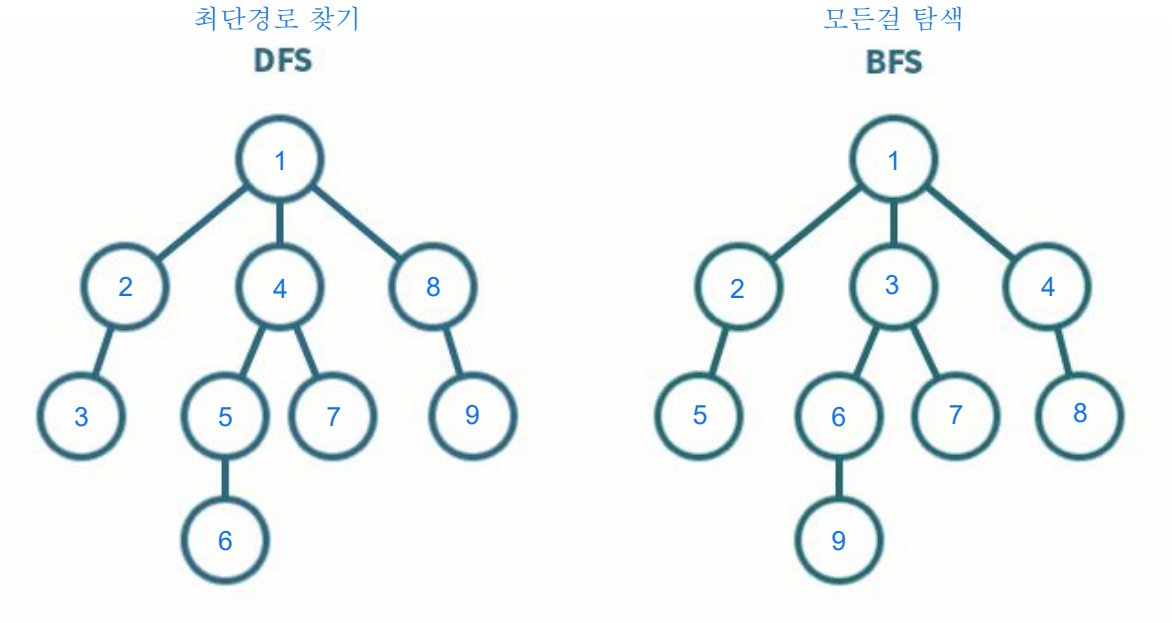
Quick sort 구현

6 5 3 1 8 7 2 4

```python
def quick_sort(data_list, descending=False):
    # 1. 종료 조건 (Base Case): 리스트에 요소가 1개 이하이면 이미 정렬된 것
    if len(data_list) <= 1:
        return data_list

    # 2. 피벗(Pivot) 선택: 리스트의 첫 번째 요소를 피벗으로 사용
    pivot = data_list[0]
    # 리스트의 나머지 요소 (피벗 제외)
    rest_of_list = data_list[1:]

    # 3. 분할 (Partitioning) 조건 변경
    if descending: # 내림차순 정렬
        # - [left]: 피벗보다 크거나 같은 요소들
        # - [right]: 피벗보다 작은 요소들
        left = [x for x in rest_of_list if x >= pivot]
        right = [x for x in rest_of_list if x < pivot]
    else: # 오름차순 정렬
        # - [left]: 피벗보다 작거나 같은 요소들
        # - [right]: 피벗보다 큰 요소들
        left = [x for x in rest_of_list if x <= pivot]
        right = [x for x in rest_of_list if x > pivot]

    # 4. 정복 및 결합 (Conquer & Combine):
    # 재귀 호출 시에도 descending 매개변수를 전달해야 합니다.
    return quick_sort(left, descending) + [pivot] + quick_sort(right, descending)
```

Tree 탐색

DFS(Depth-First Search) & BFS(Breath-First Search) - 그래프 탐색 알고리즘



Tree 탐색

DFS

Stack(또는 재귀 함수) 활용 구현

최적화 문제에 활용

- 최소 이동 거리
- 최단 횟수 찾기
- 레벨 순서대로 탐색할 때

```python
def dfs(graph, v, visited):
    # 현재 노드를 방문 처리
    visited[v] = True
    print(v, end=' ')
    # 현재 노드와 연결된 다른 노드를 재귀적으로 방문
    for i in graph[v]:
        if not visited[i]:
            dfs(graph, i, visited)
```

Tree 탐색

BFS

Queue 활용 구현

전체 탐색할 때 활용

- 모든 경우의 수 탐색 (완전탐색)
- 조합/순열 생성
- 백트래킹 문제

```python
from collections import deque

def bfs(graph, start, visited):
    # 자식 노드를 순차적으로 Dequeue에 적재
    queue = deque([start])
    # 현재 노드를 방문 처리
    visited[start] = True
    # 큐가 빌 때까지 반복
    while queue:
        # 큐에서 하나의 원소를 뽑아 출력
        v = queue.popleft()
        print(v, end=' ')
        # 해당 원소와 연결된, 아직 방문하지 않은 원소들을 큐에 삽입
        for i in graph[v]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True
```

Two pointers

리스트에서 데이터를 처리할 때, 두개의 포인터를 활용하여 문제를 해결하는 알고리즘

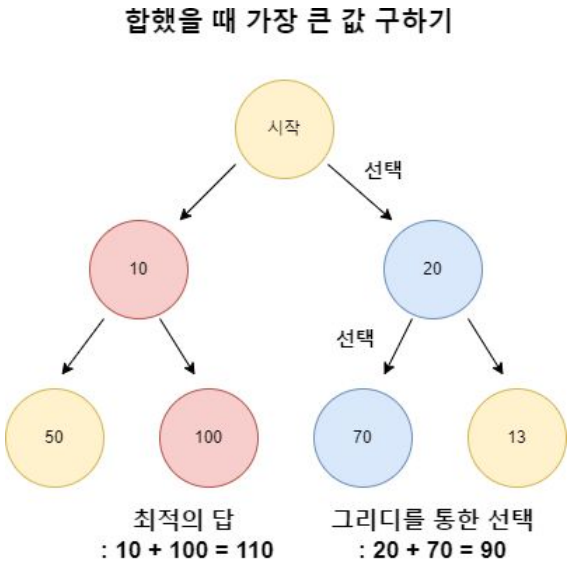
다중 반복문의 비효율적 처리를 효과적으로 처리하도록 하는 방법론

Greedy Algorithm

문제의 부분 최적의 해결법들이 모이면, 해당 문제 전체를 해결하는 정답에 가까운 해답을 얻을 수 있다

잔액을 거슬러 줄 때, 가장 적은 수의 동전으로 거슬러 주는 방법 계산

- 가장 금액이 큰 동전부터 최대한 거슬러 준다
- 남은 잔액만큼 그 다음 동전으로 거슬러 준다

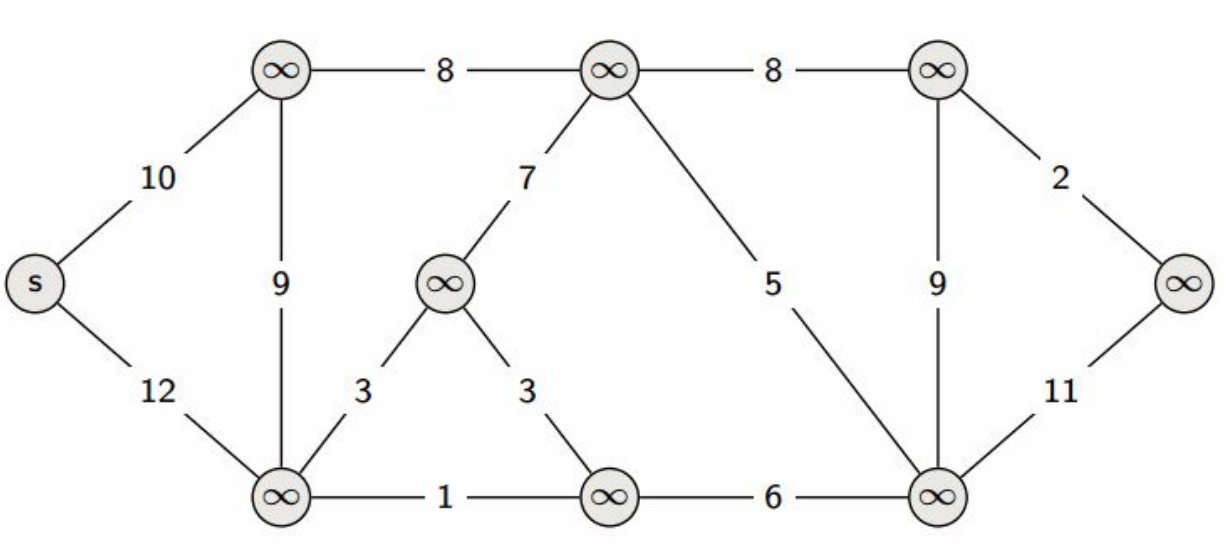


Dijkstra's Algorithm 예) 지하철 노선

제약 조건: weight가 0 이상

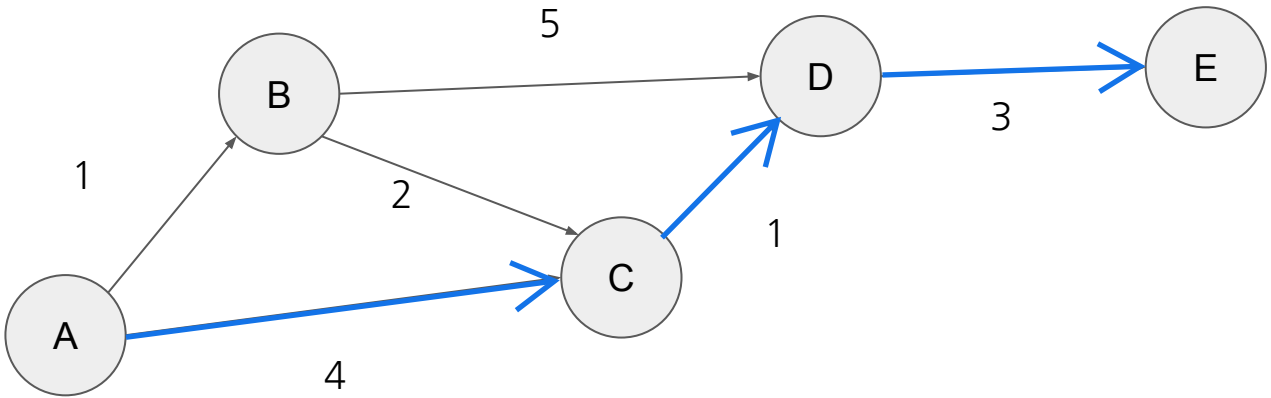
Greedy Algorithm의 한 종류로써 최단경로를 구하는 알고리즘

탐색되지 않은 노드에 현재 갈 수 있는 최단경로가 최적이다 판단



Dijkstra's Algorithm 구현

A -> E 까지 최단경로 구하기



Dijkstra's Algorithm 구현

```python
import heapq

def dijkstra(graph, start_node):
    # 1. 거리 정보를 저장할 딕셔너리 초기화
    # 시작 노드를 제외한 모든 노드의 거리를 무한대로 설정
    distances = {node: float('inf') for node in graph}
    distances[start_node] = 0

    # 2. 우선순위 큐(Min Heap) 초기화
    # (거리, 노드) 형태로 저장하여 거리가 가장 짧은 노드부터 처리
    # 힙의 첫 요소는 (시작 노드까지의 거리, 시작 노드)
    priority_queue = [(0, start_node)]

    # 3. 큐에 요소가 남아있지 않을 때까지 반복
    while priority_queue:
        print("==================")
        # 가장 짧은 노드부터 진행
        current_distance, current_node = heapq.heappop(priority_queue)
        print(current_distance, current_node)

        # 이미 처리된 노드이거나, 현재 꺼낸 거리가 이미 기록된 최단 거리보다 길다면 무시
        if current_distance > distances[current_node]:
            print("continue")
            continue

        # 현재 노드와 연결된 인접 노드들을 확인
        for neighbor, weight in graph[current_node]:
            # 새로운 경로를 계산: (현재 노드까지의 거리 + 현재 노드에서 인접 노드까지의 가중치)
            distance = current_distance + weight

            # 새로운 경로가 기존의 최단 거리보다 짧으면 갱신
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                # 갱신된 거리와 인접 노드를 우선순위 큐에 추가
                heapq.heappush(priority_queue, (distance, neighbor))
                print("push", distance, neighbor)

    return distances
```

Dijkstra's Algorithm 구현

```python
graph = {
    'A': [('B', 1), ('C', 4)],
    'B': [('C', 2), ('D', 5)],
    'C': [('D', 1)],
    'D': [('E', 3)],
    'E': []  # 종착 노드는 비어있는 리스트
}

start_node = 'A'
shortest_distances = dijkstra(graph, start_node)

print(f"시작 노드: {start_node}")
print("최단 거리 결과:")
# 결과: {'A': 0, 'B': 1, 'C': 3, 'D': 4, 'E': 7}
print(shortest_distances)
```

```
==================
0 A
push 1 B
push 4 C
==================
1 B
push 3 C
push 6 D
==================
3 C
push 4 D
==================
4 C
continue
==================
4 D
push 7 E
==================
6 D
continue
==================
7 E
시작 노드 : A
최단 거리 결과 :
{'A': 0, 'B': 1, 'C': 3, 'D': 4, 'E': 7}
```

코딩테스트

코딩테스트

IT 실무 역량을 평가하기 위한 실무 평가

채용 절차 - 서류평가 후 면접 전 보통 진행

실행 방식

- 정기채용: 정해진 시간, 정해진 공간에서 동시에 진행
- 상시채용: 일정 조율 후, 온라인 링크 전달 - 온라인 제출

일반적으로 5~7개 문제 출시

- 1 ~ 2 문제 - 코딩 기본기 확인
- 2 ~ 3 문제 - 지문 이해 및 자료구조, 알고리즘 활용
- 1~ 2 문제 - 회사 연관 실전 문제 (최적화 등)

코딩테스트

주의사항

- 자동 완성 기능 미지원
- Debug mode 활용 불가
- 코드 복사, 붙여넣기 금지 - 코드 작성 과정 녹화됨
- 제시된 입출력 예 외에 상황 고려

코딩테스트

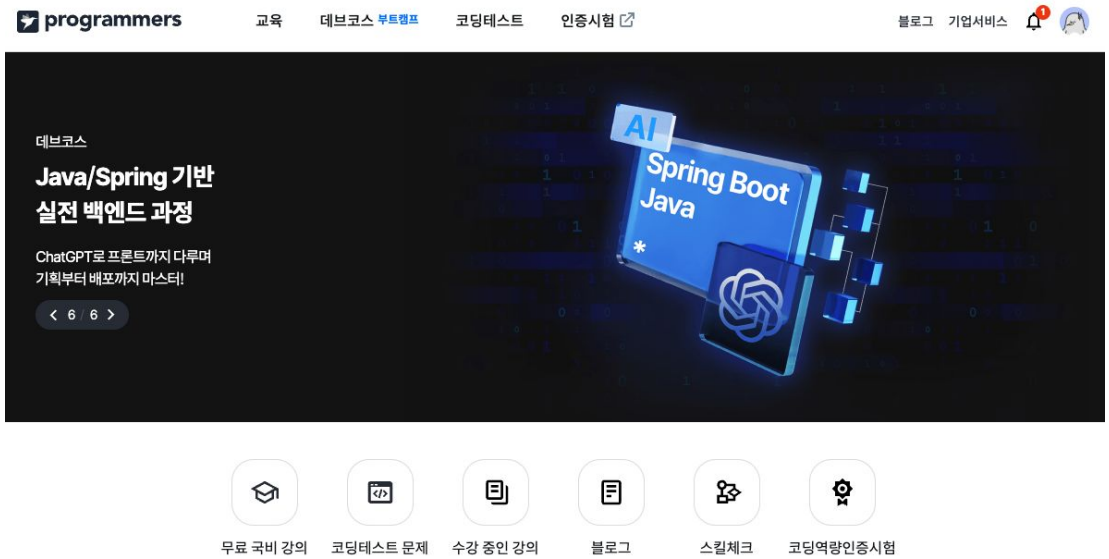
문제 풀기

- 문제를 해결하기에 적합한 방법(알고리즘, 자료구조) 검토
- 해당 방안의 성능 평가
    - Big-O로 계산하여 N의 값의 결과 값이 1억에 인접하면 불가로 판단
- 구현 후 테스트 케이스 확인
    - 문제에 제시된 입출력 값 확인
    - 경계 값, 최대 값에 해당하는 테스트 케이스 확인
- 최적화 방안 고민

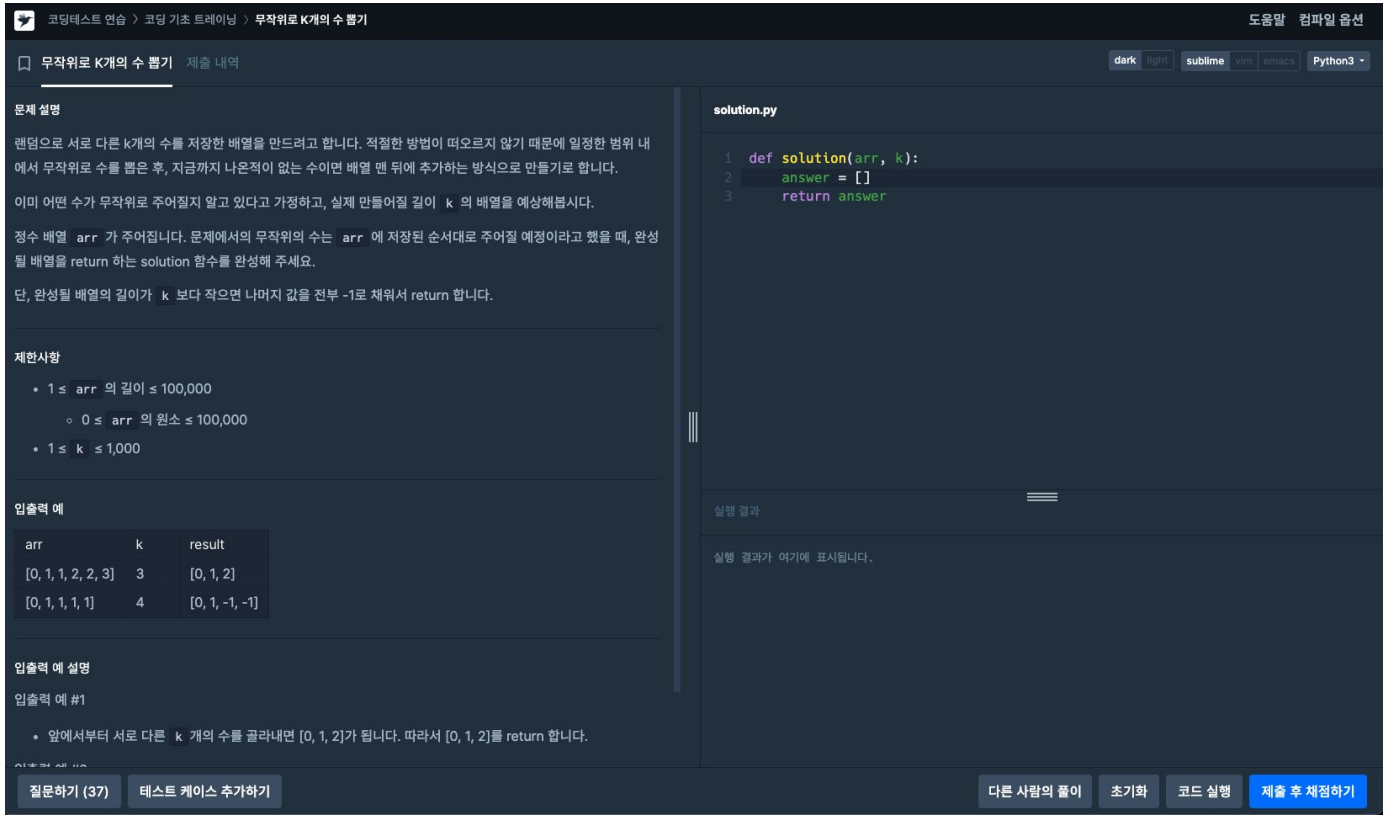
프로그래머스

https://programmers.co.kr/

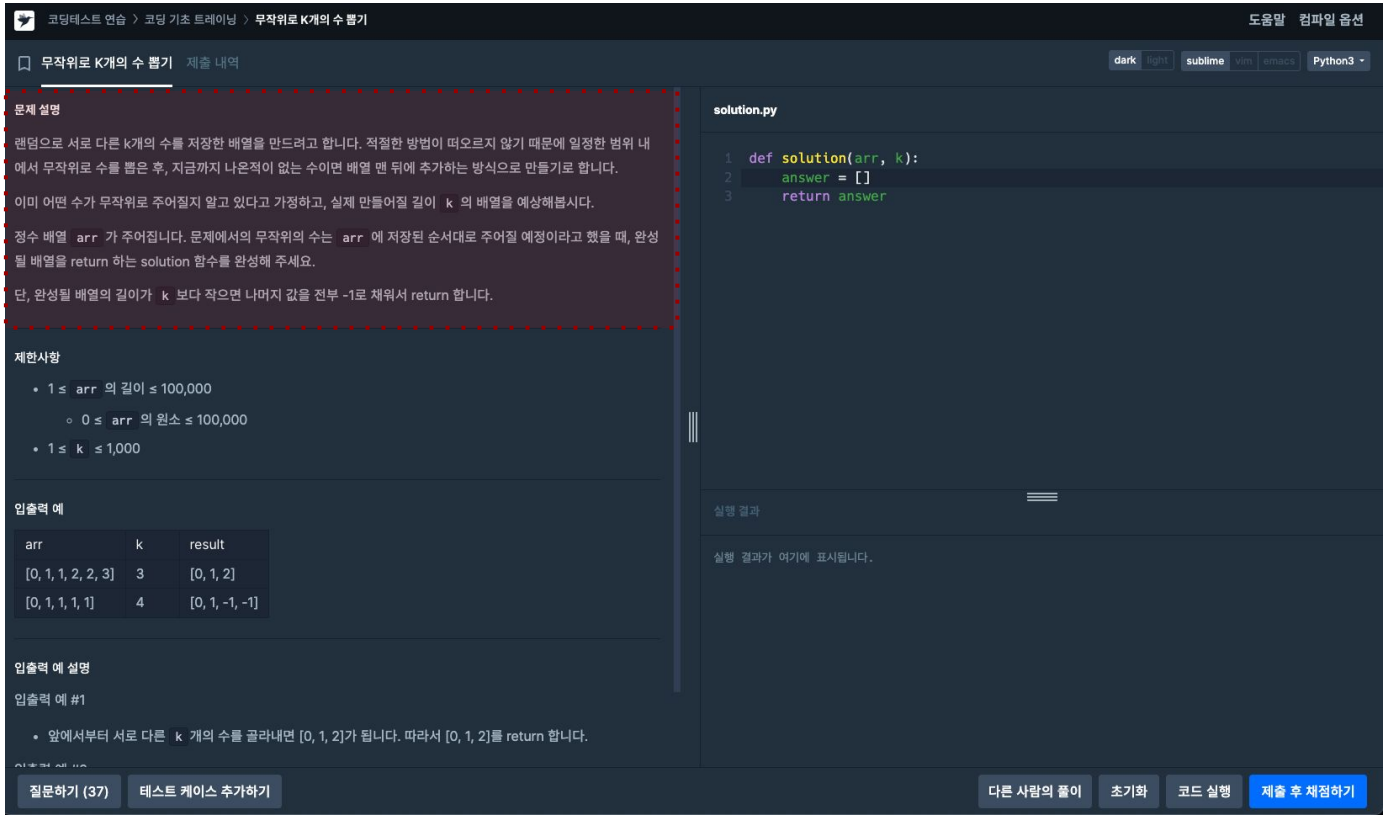
코딩테스트 연습 도움 사이트



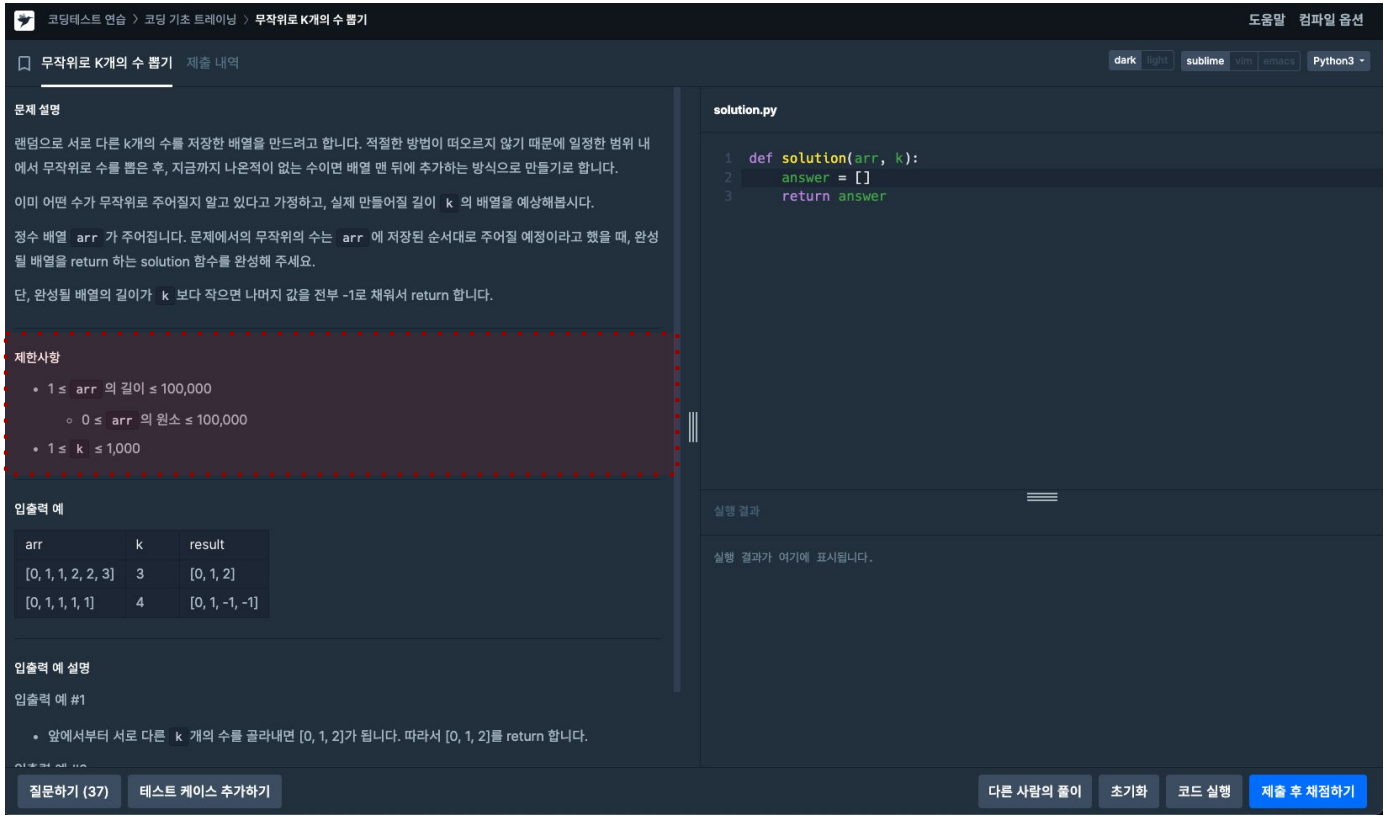
프로그래머스


코딩테스트 연습 〉 코딩 기초 트레이닝 〉 무작위로 K개의 수 뽑기
도움말
컴파일 옵션
무작위로 K개의 수 뽑기
제출 내역
dark light
sublime vim emacs
Python3
문제 설명
랜덤으로 서로 다른 k개의 수를 저장한 배열을 만드려고 합니다. 적절한 방법이 떠오르지 않기 때문에 일정한 범위 내에서 무작위로 수를 뽑은 후, 지금까지 나온적이 없는 수이면 배열 맨 뒤에 추가하는 방식으로 만들기로 합니다.
이미 어떤 수가 무작위로 주어질지 알고 있다고 가정하고, 실제 만들어질 길이 k 의 배열을 예상해봅시다.
정수 배열 arr 가 주어집니다. 문제에서의 무작위의 수는 arr 에 저장된 순서대로 주어질 예정이라고 했을 때, 완성될 배열을 return 하는 solution 함수를 완성해 주세요.
단, 완성될 배열의 길이가 k 보다 작으면 나머지 값을 전부 -1로 채워서 return 합니다.
제한사항
1 ≤ arr 의 길이 ≤ 100,000
0 ≤ arr 의 원소 ≤ 100,000
1 ≤ k ≤ 1,000
입출력 예
arr
k
result
[0, 1, 1, 2, 2, 3]
3
[0, 1, 2]
[0, 1, 1, 1, 1]
4
[0, 1, -1, -1]
입출력 예 설명
입출력 예 #1
앞에서부터 서로 다른 k 개의 수를 골라내면 [0, 1, 2]가 됩니다. 따라서 [0, 1, 2]를 return 합니다.
solution.py
def solution(arr, k):
answer = []
return answer
실행 결과
실행 결과가 여기에 표시됩니다.
질문하기 (37)
테스트 케이스 추가하기
다른 사람의 풀이
초기화
코드 실행
제출 후 채점하기

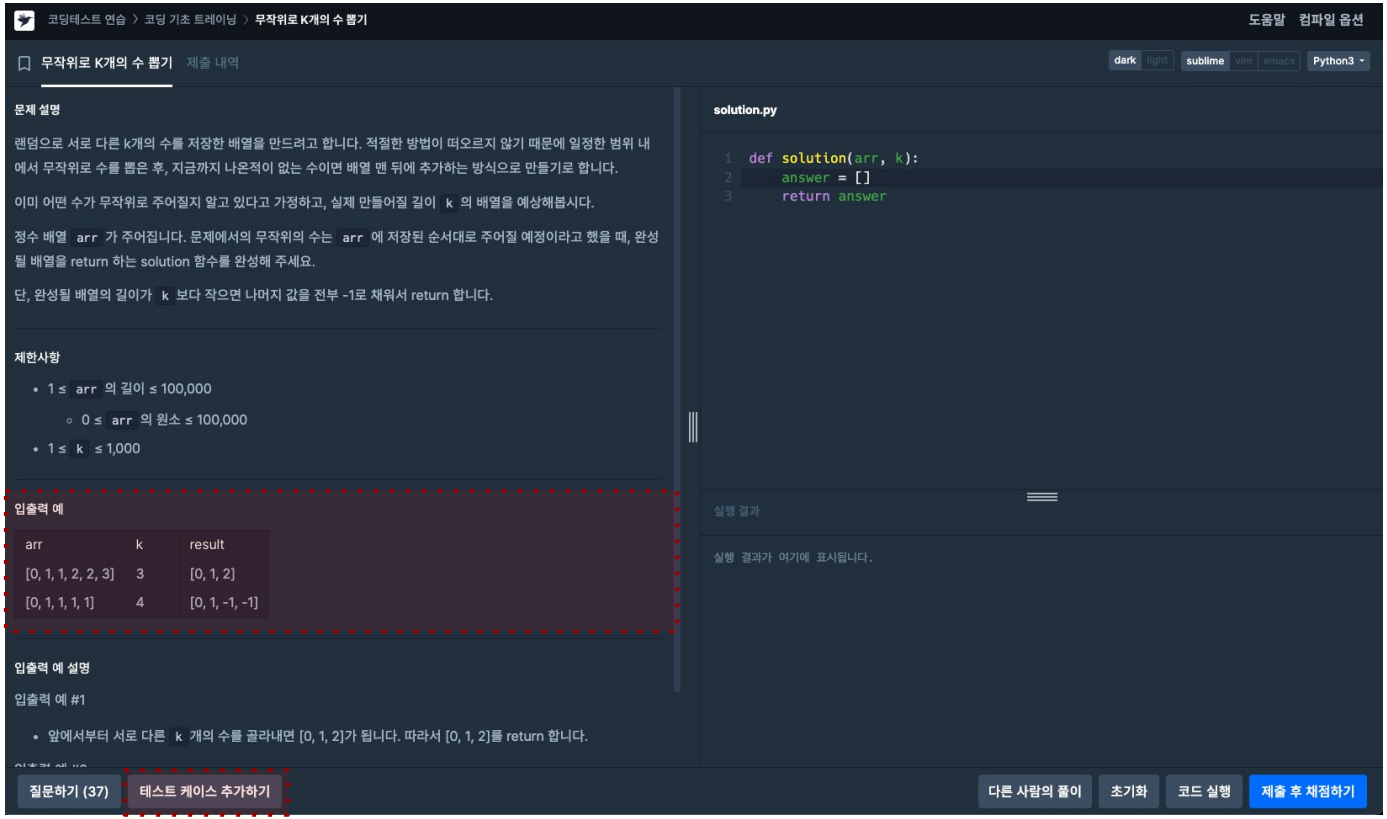
프로그래머스


코딩테스트 연습 〉 코딩 기초 트레이닝 〉 무작위로 K개의 수 뽑기
도움말
컴파일 옵션
무작위로 K개의 수 뽑기
제출 내역
dark light
sublime vim emacs
Python3
문제 설명
랜덤으로 서로 다른 k개의 수를 저장한 배열을 만드려고 합니다. 적절한 방법이 떠오르지 않기 때문에 일정한 범위 내에서 무작위로 수를 뽑은 후, 지금까지 나온적이 없는 수이면 배열 맨 뒤에 추가하는 방식으로 만들기로 합니다.
이미 어떤 수가 무작위로 주어질지 알고 있다고 가정하고, 실제 만들어질 길이 k 의 배열을 예상해봅시다.
정수 배열 arr 가 주어집니다. 문제에서의 무작위의 수는 arr 에 저장된 순서대로 주어질 예정이라고 했을 때, 완성될 배열을 return 하는 solution 함수를 완성해 주세요.
단, 완성될 배열의 길이가 k 보다 작으면 나머지 값을 전부 -1로 채워서 return 합니다.
제한사항
1 ≤ arr 의 길이 ≤ 100,000
0 ≤ arr 의 원소 ≤ 100,000
1 ≤ k ≤ 1,000
입출력 예
arr k result
[0, 1, 1, 2, 2, 3] 3 [0, 1, 2]
[0, 1, 1, 1, 1] 4 [0, 1, -1, -1]
입출력 예 설명
입출력 예 #1
앞에서부터 서로 다른 k 개의 수를 골라내면 [0, 1, 2]가 됩니다. 따라서 [0, 1, 2]를 return 합니다.
solution.py
def solution(arr, k):
answer = []
return answer
실행 결과
실행 결과가 여기에 표시됩니다.
질문하기 (37)
테스트 케이스 추가하기
다른 사람의 풀이
초기화
코드 실행
제출 후 채점하기

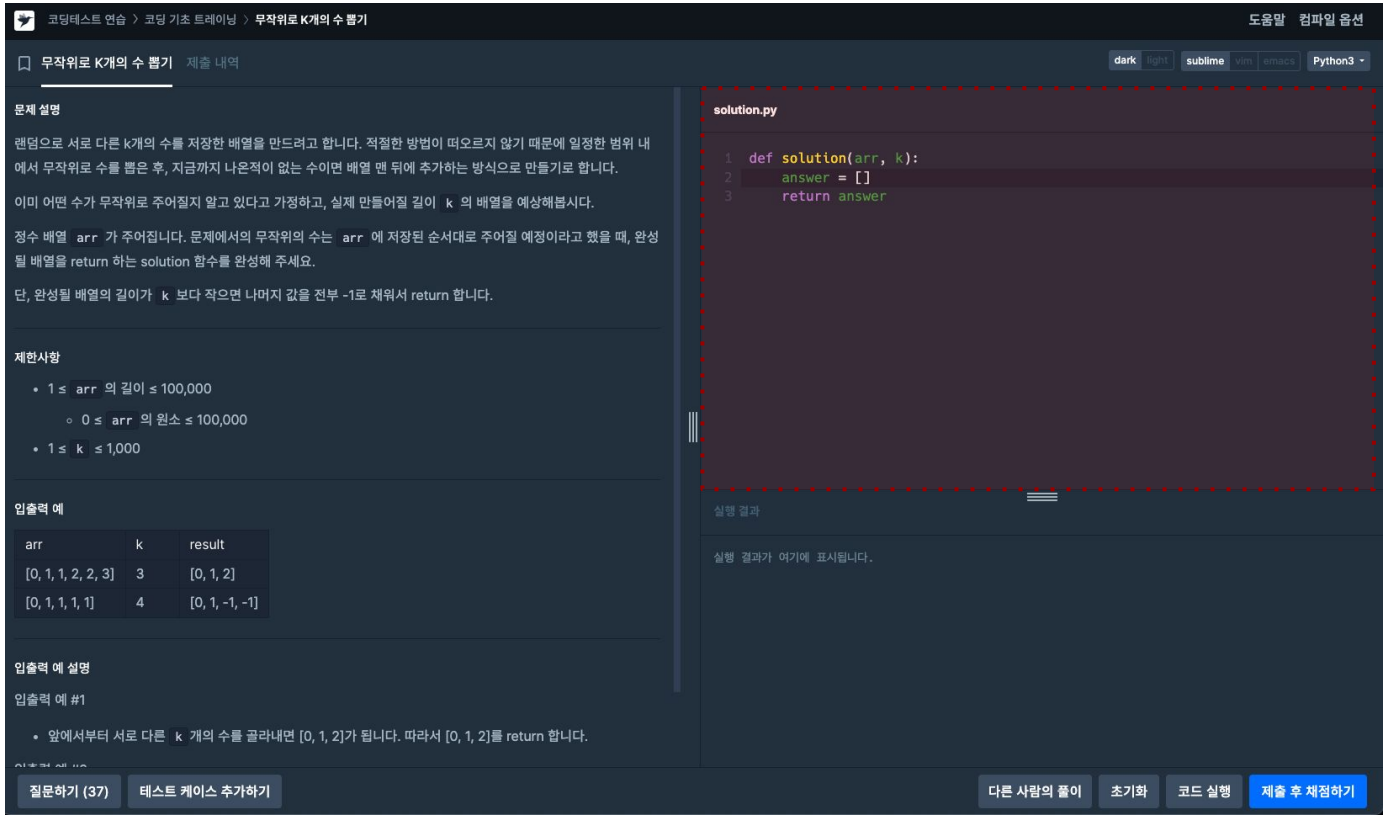
프로그래머스


코딩테스트 연습 〉 코딩 기초 트레이닝 〉 무작위로 K개의 수 뽑기
도움말
컴파일 옵션
무작위로 K개의 수 뽑기
제출 내역
dark light
sublime vim emacs
Python3
문제 설명
랜덤으로 서로 다른 k개의 수를 저장한 배열을 만드려고 합니다. 적절한 방법이 떠오르지 않기 때문에 일정한 범위 내에서 무작위로 수를 뽑은 후, 지금까지 나온적이 없는 수이면 배열 맨 뒤에 추가하는 방식으로 만들기로 합니다.
이미 어떤 수가 무작위로 주어질지 알고 있다고 가정하고, 실제 만들어질 길이 k 의 배열을 예상해봅시다.
정수 배열 arr 가 주어집니다. 문제에서의 무작위의 수는 arr 에 저장된 순서대로 주어질 예정이라고 했을 때, 완성될 배열을 return 하는 solution 함수를 완성해 주세요.
단, 완성될 배열의 길이가 k 보다 작으면 나머지 값을 전부 -1로 채워서 return 합니다.
제한사항
1 ≤ arr 의 길이 ≤ 100,000
0 ≤ arr 의 원소 ≤ 100,000
1 ≤ k ≤ 1,000
입출력 예
arr k result
[0, 1, 1, 2, 2, 3] 3 [0, 1, 2]
[0, 1, 1, 1, 1] 4 [0, 1, -1, -1]
입출력 예 설명
입출력 예 #1
앞에서부터 서로 다른 k 개의 수를 골라내면 [0, 1, 2]가 됩니다. 따라서 [0, 1, 2]를 return 합니다.
solution.py
1 def solution(arr, k):
2 answer = []
3 return answer
실행 결과
실행 결과가 여기에 표시됩니다.
질문하기 (37)
테스트 케이스 추가하기
다른 사람의 풀이
초기화
코드 실행
제출 후 채점하기

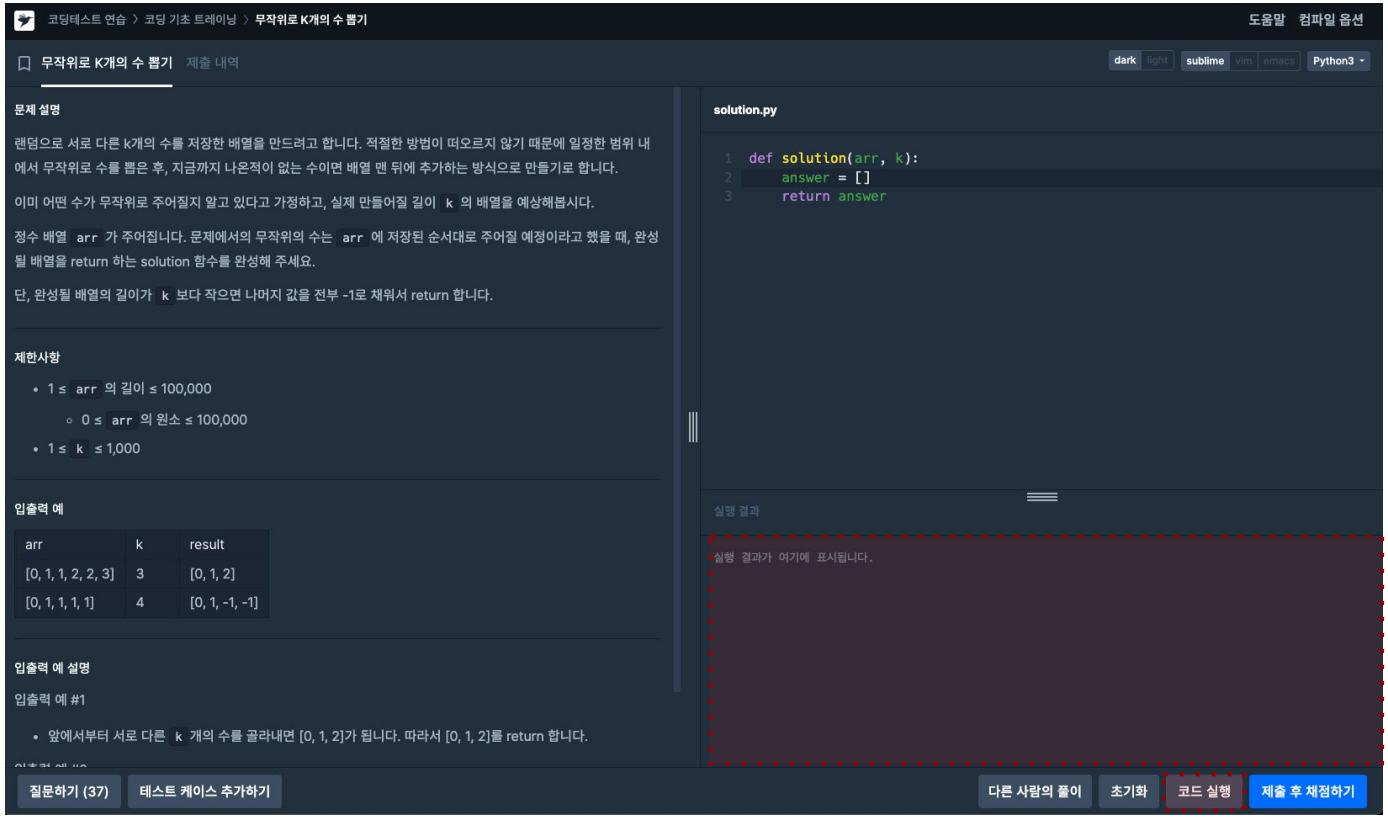
프로그래머스


코딩테스트 연습 〉 코딩 기초 트레이닝 〉 무작위로 K개의 수 뽑기
도움말
컴파일 옵션
무작위로 K개의 수 뽑기
제출 내역
dark light
sublime vim emacs
Python3
문제 설명
랜덤으로 서로 다른 k개의 수를 저장한 배열을 만드려고 합니다. 적절한 방법이 떠오르지 않기 때문에 일정한 범위 내에서 무작위로 수를 뽑은 후, 지금까지 나온적이 없는 수이면 배열 맨 뒤에 추가하는 방식으로 만들기로 합니다.
이미 어떤 수가 무작위로 주어질지 알고 있다고 가정하고, 실제 만들어질 길이 k 의 배열을 예상해봅시다.
정수 배열 arr 가 주어집니다. 문제에서의 무작위의 수는 arr 에 저장된 순서대로 주어질 예정이라고 했을 때, 완성될 배열을 return 하는 solution 함수를 완성해 주세요.
단, 완성될 배열의 길이가 k 보다 작으면 나머지 값을 전부 -1로 채워서 return 합니다.
제한사항
1 ≤ arr 의 길이 ≤ 100,000
0 ≤ arr 의 원소 ≤ 100,000
1 ≤ k ≤ 1,000
입출력 예
arr
k
result
[0, 1, 1, 2, 2, 3]
3
[0, 1, 2]
[0, 1, 1, 1, 1]
4
[0, 1, -1, -1]
입출력 예 설명
입출력 예 #1
앞에서부터 서로 다른 k 개의 수를 골라내면 [0, 1, 2]가 됩니다. 따라서 [0, 1, 2]를 return 합니다.
solution.py
def solution(arr, k):
answer = []
return answer
실행 결과
실행 결과가 여기에 표시됩니다.
질문하기 (37)
테스트 케이스 추가하기
다른 사람의 풀이
초기화
코드 실행
제출 후 채점하기

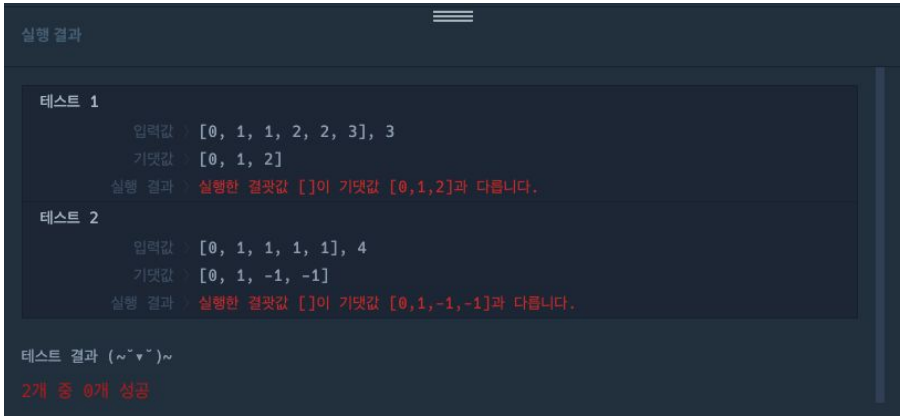
프로그래머스


코딩테스트 연습 〉 코딩 기초 트레이닝 〉 무작위로 K개의 수 뽑기
도움말
컴파일 옵션
무작위로 K개의 수 뽑기
제출 내역
dark light
sublime vim emacs
Python3
문제 설명
랜덤으로 서로 다른 k개의 수를 저장한 배열을 만드려고 합니다. 적절한 방법이 떠오르지 않기 때문에 일정한 범위 내에서 무작위로 수를 뽑은 후, 지금까지 나온적이 없는 수이면 배열 맨 뒤에 추가하는 방식으로 만들기로 합니다.
이미 어떤 수가 무작위로 주어질지 알고 있다고 가정하고, 실제 만들어질 길이 k 의 배열을 예상해봅시다.
정수 배열 arr 가 주어집니다. 문제에서의 무작위의 수는 arr 에 저장된 순서대로 주어질 예정이라고 했을 때, 완성될 배열을 return 하는 solution 함수를 완성해 주세요.
단, 완성될 배열의 길이가 k 보다 작으면 나머지 값을 전부 -1로 채워서 return 합니다.
제한사항
1 ≤ arr 의 길이 ≤ 100,000
0 ≤ arr 의 원소 ≤ 100,000
1 ≤ k ≤ 1,000
입출력 예
arr k result
[0, 1, 1, 2, 2, 3] 3 [0, 1, 2]
[0, 1, 1, 1, 1] 4 [0, 1, -1, -1]
입출력 예 설명
입출력 예 #1
앞에서부터 서로 다른 k 개의 수를 골라내면 [0, 1, 2]가 됩니다. 따라서 [0, 1, 2]를 return 합니다.
solution.py
def solution(arr, k):
answer = []
return answer
실행 결과
실행 결과가 여기에 표시됩니다.
질문하기 (37)
테스트 케이스 추가하기
다른 사람의 풀이
초기화
코드 실행
제출 후 채점하기

프로그래머스


코딩테스트 연습 〉 코딩 기초 트레이닝 〉 무작위로 K개의 수 뽑기
도움말
컴파일 옵션
무작위로 K개의 수 뽑기
제출 내역
dark light
sublime vim emacs
Python3
문제 설명
랜덤으로 서로 다른 k개의 수를 저장한 배열을 만드려고 합니다. 적절한 방법이 떠오르지 않기 때문에 일정한 범위 내에서 무작위로 수를 뽑은 후, 지금까지 나온적이 없는 수이면 배열 맨 뒤에 추가하는 방식으로 만들기로 합니다.
이미 어떤 수가 무작위로 주어질지 알고 있다고 가정하고, 실제 만들어질 길이 k 의 배열을 예상해봅시다.
정수 배열 arr 가 주어집니다. 문제에서의 무작위의 수는 arr 에 저장된 순서대로 주어질 예정이라고 했을 때, 완성될 배열을 return 하는 solution 함수를 완성해 주세요.
단, 완성될 배열의 길이가 k 보다 작으면 나머지 값을 전부 -1로 채워서 return 합니다.
제한사항
1 ≤ arr 의 길이 ≤ 100,000
0 ≤ arr 의 원소 ≤ 100,000
1 ≤ k ≤ 1,000
입출력 예
arr
k
result
[0, 1, 1, 2, 2, 3]
3
[0, 1, 2]
[0, 1, 1, 1, 1]
4
[0, 1, -1, -1]
입출력 예 설명
입출력 예 #1
앞에서부터 서로 다른 k 개의 수를 골라내면 [0, 1, 2]가 됩니다. 따라서 [0, 1, 2]를 return 합니다.
solution.py
def solution(arr, k):
answer = []
return answer
실행 결과
실행 결과가 여기에 표시됩니다.
질문하기 (37)
테스트 케이스 추가하기
다른 사람의 풀이
초기화
코드 실행
제출 후 채점하기

프로그래머스



<개발 확인 과정>

<실제 채점 과정>

문제풀이

무작위로 K개의 수 뽑기

https://school.programmers.co.kr/learn/courses/30/lessons/181858

무작위로 K개의 수 뽑기

```python
def solution(arr, k):
    answer = []
    seen = set()
    # arr 각 요소에 대해서
    for i in arr:
        # 중복 값이 있는지 확인
        if i not in seen:
            answer.append(i)
            seen.add(i)
        # 반환코자하는 길이, k와 같은지 확인
        if len(answer) == k:
            break

    # 부족한 수만큼 -1로 채우기
    while len(answer) < k:
        answer.append(-1)
    return answer
```

set 자료구조는 중복을 허용하지 않음

크기가 작은 부분 문자열

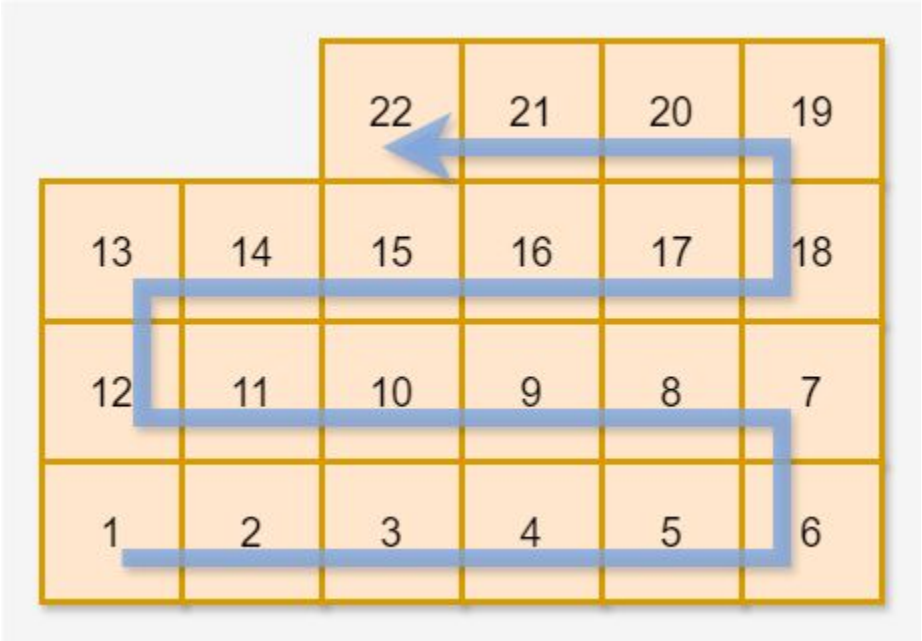
https://school.programmers.co.kr/learn/courses/30/lessons/147355

크기가 작은 부분 문자열

```python
def solution(t, p):
    answer = 0
    for i in range(len(t) - len(p) + 1):
        a = int(t[i : i + len(p)])
        b = int(p)
        if a <= b:
            answer += 1
    return answer
```

택배 상자 꺼내기

https://school.programmers.co.kr/learn/courses/30/lessons/389478



택배 상자 꺼내기

https://school.programmers.co.kr/learn/courses/30/lessons/389478

2차원 배열 구성

		22	21	20	19
13	14	15	16	17	18
12	11	10	9	8	7
1	2	3	4	5	6

택배 상자 꺼내기

https://school.programmers.co.kr/learn/courses/30/lessons/389478

2차원 배열 구성

		22	21	20	19
13	14	15	16	17	18
12	11	10	9	8	7
1	2	3	4	5	6

택배 상자 꺼내기

```python
def solution(n, w, num):
    # 2차원 배열 만들기
    boxes = [[] for _ in range(w)]

    # 박스 번호
    number = 1
    # 방향
    direction = 1
    # 박스 채우기
    while number <= n:
        # 1 = 정방향
        if direction == 1:
            for col in range(0, w, 1):
                boxes[col].append(number)
                number += 1

                if number > n: break
        # -1 = 반대방향
        else:
            for col in range(w - 1, -1, -1):
                boxes[col].append(number)
                number += 1

                if number > n: break

        direction *= -1

    for box in boxes:
        if num in box:
            return len(box) - box.index(num)

    return 0
```

올바른 괄호 (Stack 활용)

https://school.programmers.co.kr/learn/courses/30/lessons/12909

((()))()

올바른 괄호 (Stack 활용)

https://school.programmers.co.kr/learn/courses/30/lessons/12909

((()))()

input: (or)

여는 괄호는 언제든 OK, 닫는 괄호는 앞에 쌍이 있어야지만 OK

올바른 괄호 (Stack 활용)

https://school.programmers.co.kr/learn/courses/30/lessons/12909

((()))()

괄호가 닫히는 시점에 정상적인 괄호인지 판단

올바른 괄호 (Stack 활용)

```python
def solution(s):
    stack = []

    for char in s:
        # 여는 괄호면 언제든 OK
        if char == '(':
            stack.append(char)
        # 닫는 괄호면
        else:
            # 해당 쌍이 없으면 문제
            if not stack:
                return False
            # 완성된 괄호는 지우기
            stack.pop()
    # 짝 맞추기를 끝내도 남아있는게 있다면 문제
    return len(stack) == 0
```

뒤에 있는 큰수 찾기

https://school.programmers.co.kr/learn/courses/30/lessons/154539

제한사항
$4 \leq$ numbers의 길이 $\leq 1{,}000{,}000$
$1 \leq$ numbers[i] $\leq 1{,}000{,}000$

뒤에 있는 큰수 찾기

```python
def solution(numbers):
    # -1로 초기화
    answer = [-1] * len(numbers)

    # 각 요소별 판단
    for i in range(len(numbers) - 1):
        # 해당 요소 뒤에 값들 판단
        for j in range(i + 1, len(numbers)):
            #  가장 가까운 뒤 값 확인
            if numbers[j] > numbers[i]:
                answer[i] = numbers[j]
                break

    return answer
```

뒤에 있는 큰수 찾기

$O(N^2)$ = 100만 * 100만
시간복잡도 초과

```python
def solution(numbers):
    # -1로 초기화
    answer = [-1] * len(numbers)

    # 각 요소별 판단
    for i in range(len(numbers) - 1):
        # 해당 요소 뒤에 값들 판단
        for j in range(i + 1, len(numbers)):
            #  가장 가까운 뒤 값 확인
            if numbers[j] > numbers[i]:
                answer[i] = numbers[j]
                break

    return answer
```

뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

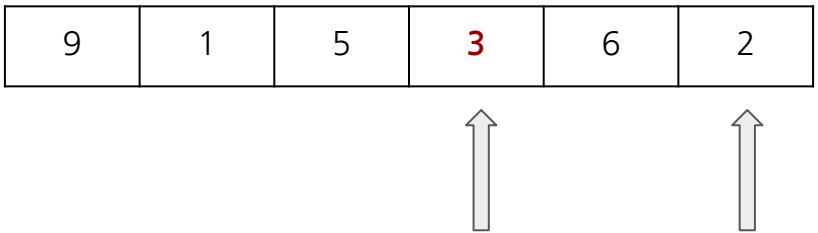
맨 뒤 값부터 판단 시작
맨 뒤 값은 항상 -1

9	1	5	3	6	**2**

뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

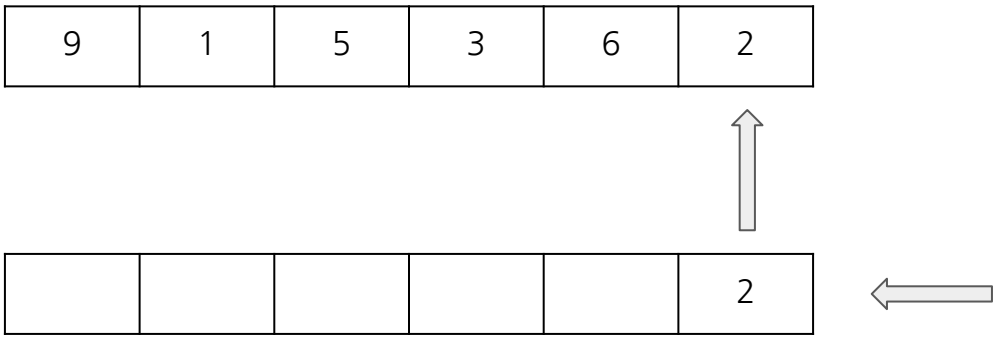
현재 Index값 이 그 뒤 값보다 클 경우,
그 뒤 값은 의미가 없음



뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

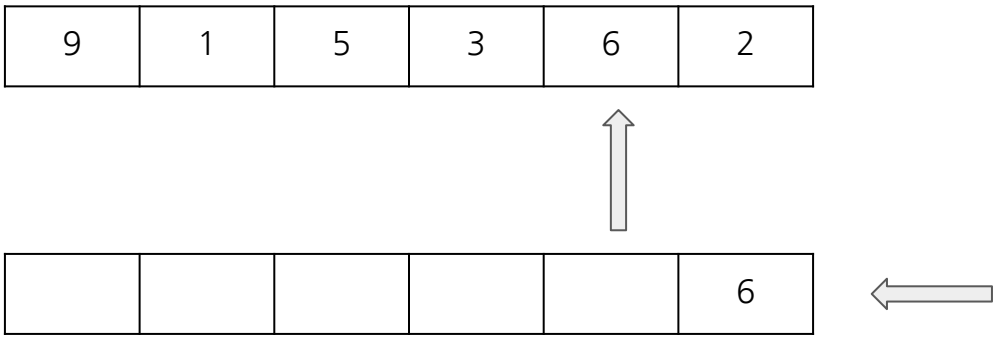
뒤에서 오면서 작은 값들은 지운다



뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

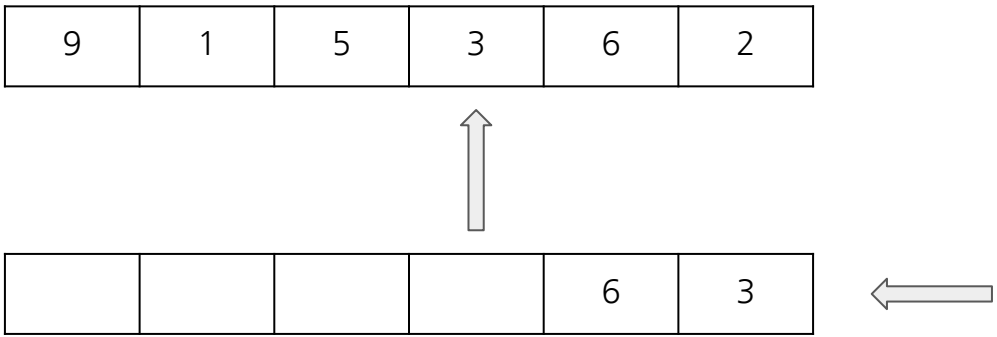
뒤에서 오면서 작은 값들은 지운다



뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

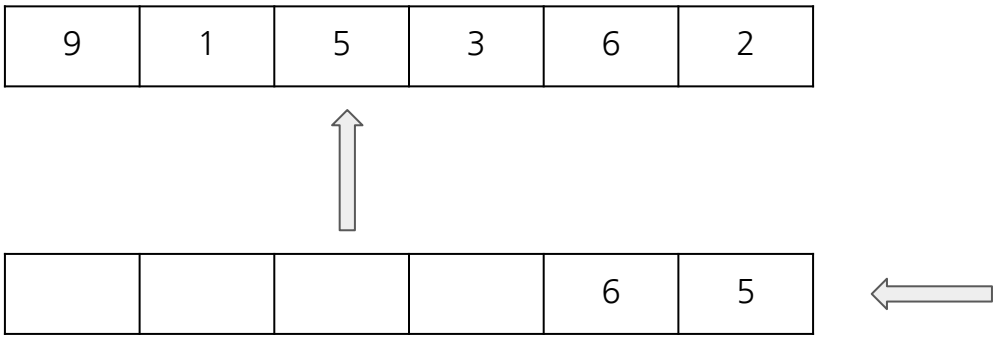
뒤에서 오면서 작은 값들은 지운다



뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

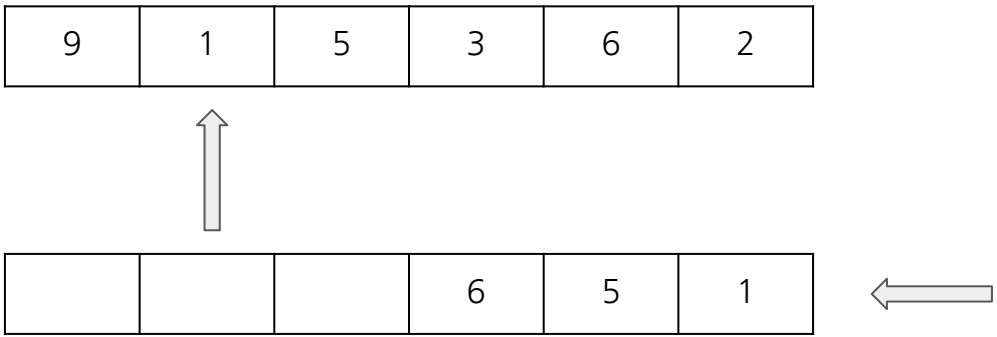
뒤에서 오면서 작은 값들은 지운다



뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

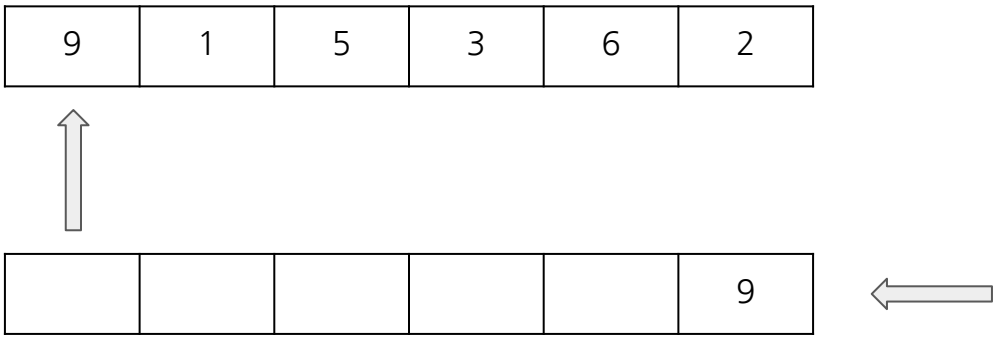
뒤에서 오면서 작은 값들은 지운다



뒤에 있는 큰수 찾기

각 요소별로 뒤에 있는 가장 가까운 값 하나만 찾으면 된다

뒤에서 오면서 작은 값들은 지운다

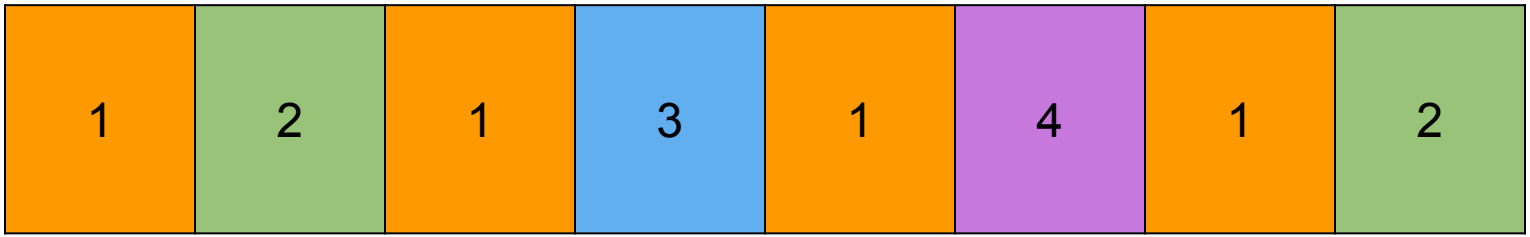


뒤에 있는 큰수 찾기

```python
def solution(numbers):
    stack = []
    result = [-1] * len(numbers)

    for i in range(len(numbers) - 1, -1, -1):
        # 스택에서 현재보다 작거나 같은 수는 제거
        while stack and stack[-1] <= numbers[i]:
            stack.pop()

        # 남아있는 스택 top이 뒷 큰 수
        if stack:
            result[i] = stack[-1]

        # 현재 값을 스택에 push
        stack.append(numbers[i])

    return result
```

롤케이크 자르기

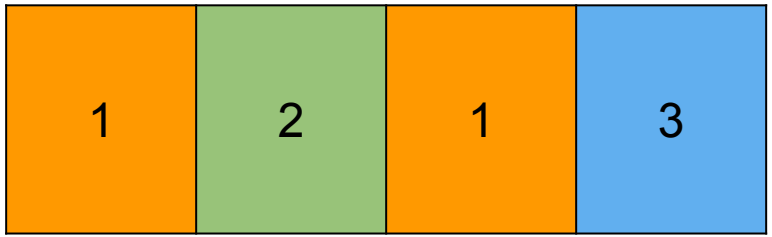
https://school.programmers.co.kr/learn/courses/30/lessons/132265

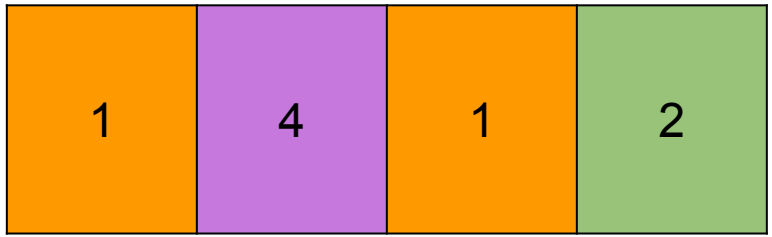


롤케이크 자르기

https://school.programmers.co.kr/learn/courses/30/lessons/132265

왼편과 오른편의 종류 수가 같은지 비교





롤케이크 자르기

```python
from collections import Counter
def solution(topping):
    answer = 0
    # 좌측 부분
    left = set()
    # 우측 부분 - 초기에 우측에 전부 있는걸로 가정
    right = Counter(topping)

    # 우측에서 한칸씩 넘어오며 자르는 경우 계산
    for t in topping:
        left.add(t)             # 좌측에 t 추가
        right[t] -= 1           # 우측에서 t 제거
        if right[t] == 0:
            del right[t]        # 0이면 삭제

        # 양쪽이 같을 경우 카운트 증가
        if len(left) == len(right):
            answer += 1

    return answer
```

소수 찾기

https://school.programmers.co.kr/learn/courses/30/lessons/42839

제한사항

- numbers는 길이 1 이상 7 이하인 문자열입니다.
- numbers는 0~9까지 숫자만으로 이루어져 있습니다.
- "013"은 0, 1, 3 숫자가 적힌 종이 조각이 흩어져있다는 의미입니다.

입출력 예

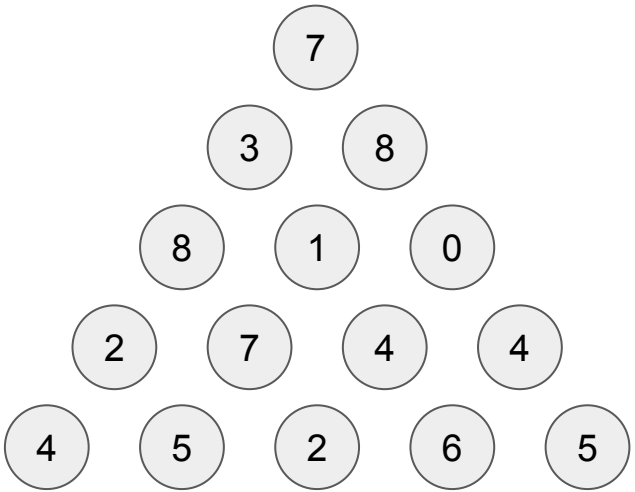
numbers	return
"17"	3
"011"	2

소수 찾기

```python
from itertools import permutations

# 소수 판단 함수
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

def solution(numbers):
    # 중복 제거를 위한 set 활용
    number_set = set()

    # 1자리부터 n자리까지 순열 생성
    for i in range(1, len(numbers)+1):
        for perm in permutations(numbers, i):
            # 각 조합 숫자로 변환
            num = int(''.join(perm))
            number_set.add(num)

    # 소수만 필터링
    count = 0
    for num in number_set:
        if is_prime(num):
            count += 1

    return count
```

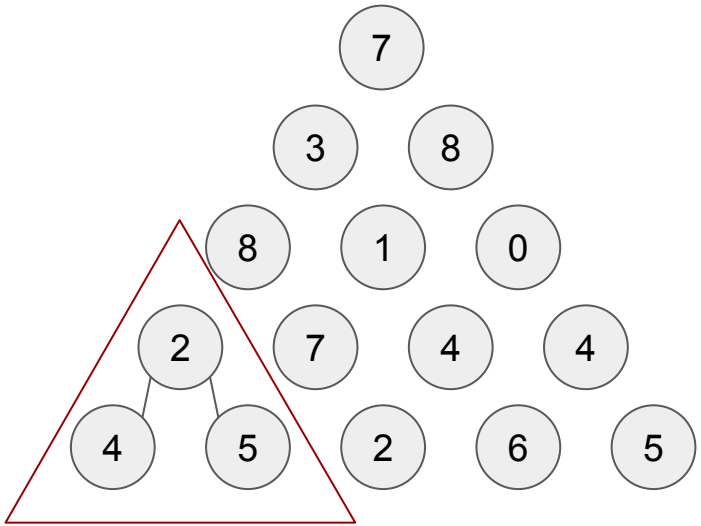
정수 삼각형

https://school.programmers.co.kr/learn/courses/30/lessons/43105



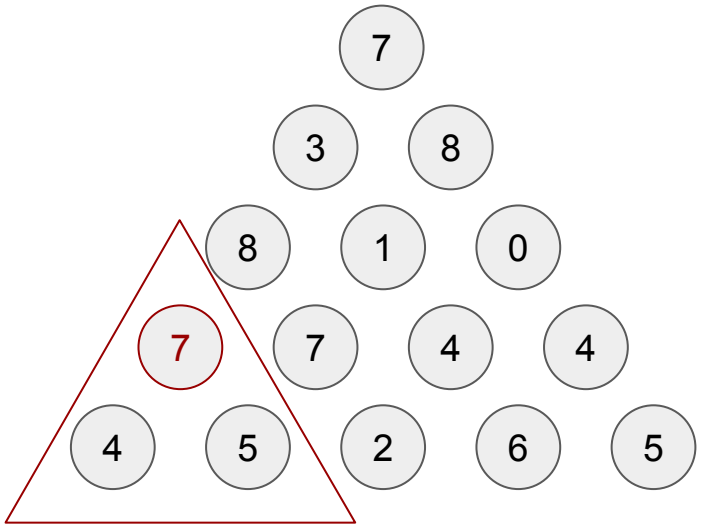
정수 삼각형

https://school.programmers.co.kr/learn/courses/30/lessons/43105



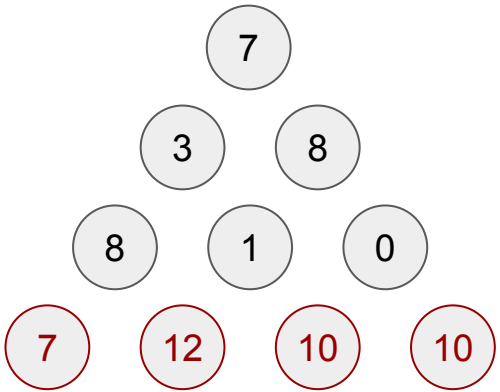
정수 삼각형

https://school.programmers.co.kr/learn/courses/30/lessons/43105



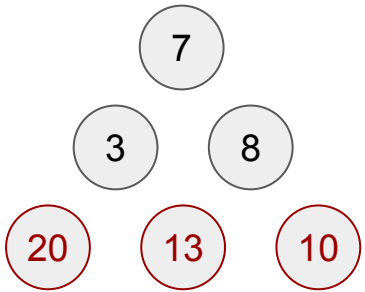
정수 삼각형

https://school.programmers.co.kr/learn/courses/30/lessons/43105



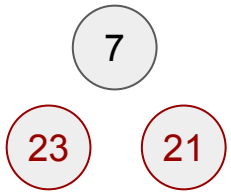
정수 삼각형

https://school.programmers.co.kr/learn/courses/30/lessons/43105



정수 삼각형

https://school.programmers.co.kr/learn/courses/30/lessons/43105



정수 삼각형

https://school.programmers.co.kr/learn/courses/30/lessons/43105

```python
def solution(triangle):
    # 삼각형의 아래 2번째 줄부터
    for i in range(len(triangle) - 2, -1, -1):
        # 해당 줄의 각 요소마다
        for j in range(len(triangle[i])):
            # 아래 두 칸 중 큰 값을 선택해서 현재 값에 더함
            triangle[i][j] += max(triangle[i+1][j], triangle[i+1][j+1])

    # 맨 위까지 도달하면 [0][0]이 최대 합
    return triangle[0][0]
```

큰수 만들기

https://school.programmers.co.kr/learn/courses/30/lessons/42883

큰수 만들기

```python
def solution(number, k):
    stack = []

    for num in number:
        # 지금 만난 숫자가 기존 숫자보다 크다면
        while stack and k > 0 and stack[-1] < num:
            stack.pop()
            k -= 1
        stack.append(num)

    # 아직 제거하지 못한 k가 남았다면 뒤에서 자르기
    if k > 0:
        stack = stack[:-k]

    return ''.join(stack)
```

가장 큰 수

https://school.programmers.co.kr/learn/courses/30/lessons/42746

가장 큰 수

```python
def solution(numbers):
    # 문자열로 변환
    numbers = list(map(str, numbers))

    # 정렬: x+y > y+x 인 경우 x가 앞에 오게
    # x*3은 자리수 맞춤을 위해 사용 (최대 1000 -> 4자리)
    numbers.sort(key=lambda x: x*3, reverse=True)

    answer = ''.join(numbers)

    # '0000...'인 경우는 '0'으로 리턴
    return '0' if answer[0] == '0' else answer
```

다리를 지나는 트럭

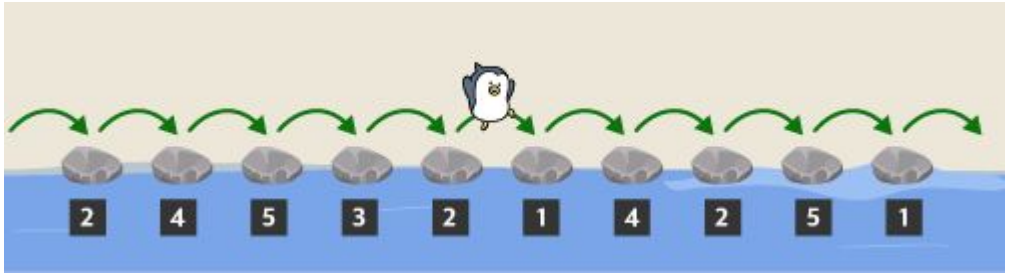
https://school.programmers.co.kr/learn/courses/30/lessons/42583

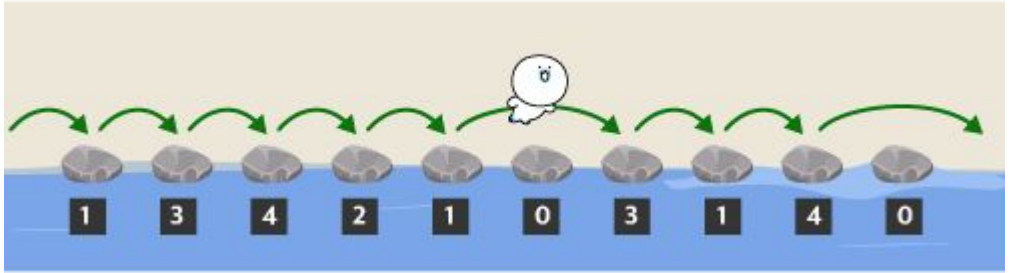
다리를 지나는 트럭

```python
from collections import deque

def solution(bridge_length, weight, truck_weights):
    time = 0
    # 다리 위 상태를 길이만큼 0으로 초기화
    bridge = deque([0] * bridge_length)
    total_weight = 0
    truck_weights = deque(truck_weights)

    while bridge:
        time += 1
        # 트럭 하나가 다리에서 빠져나옴
        out = bridge.popleft()
        total_weight -= out

        if truck_weights:
            if total_weight + truck_weights[0] <= weight:
                # 다음 트럭이 다리에 올라감
                next_truck = truck_weights.popleft()
                bridge.append(next_truck)
                total_weight += next_truck
            else:
                # 다음 트럭이 올라갈 수 없으면 빈 공간 추가
                bridge.append(0)

    return time
```

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062





카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

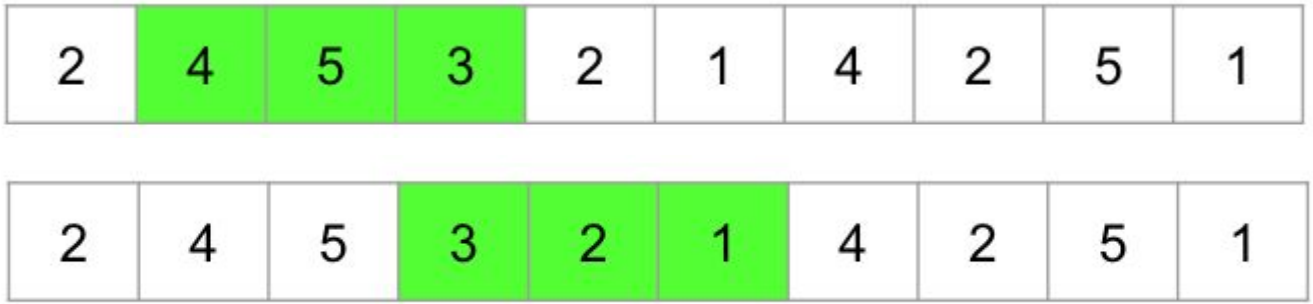
연속으로 건널 수 없는 k개 구간 찾기

최대 값 = 5
5번 지나가면 해당 구간이 모두 0, 0, 0으로 건널 수 없음

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

연속으로 건널 수 없는 k개 구간 찾기



옆으로 한칸씩 이동하면서 해당 구간의 최대 값들 기록

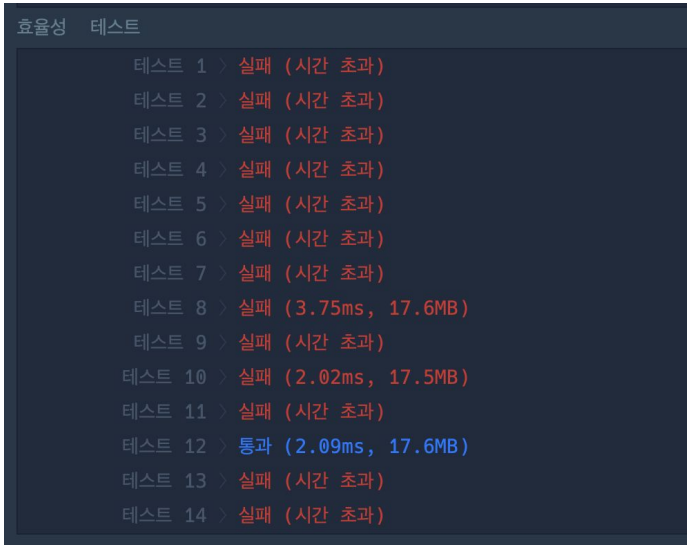
카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

```python
def solution(stones, k):
    windows = [max(stones[:k])]
    # 한칸씩 옆으로가며 k칸 내 최대 값 확인
    for i in range(k, len(stones) - k+1):
        num = max(stones[i:k+i])
        windows.append(num)
    return min(windows)
```

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062




```python
def solution(stones, k):
    windows = [max(stones[:k])]
    # 한칸씩 옆으로가며 k칸 내 최대 값 확인
    for i in range(k, len(stones) - k+1):
        num = max(stones[i:k+i])
        windows.append(num)
    return min(windows)
```

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

```python
def solution(stones, k):
    windows = [max(stones[:k])]
    # 한칸씩 옆으로가며 k칸 내 최대 값 확인
    for i in range(k, len(stones) - k+1):
        num = max(stones[i:k+i])
        windows.append(num)
    return min(windows)
```

stones의 길이 n만큼 수행

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

```python
def solution(stones, k):
    windows = [max(stones[:k])]
    # 한칸씩 옆으로가며 k칸 내 최대 값 확인
    for i in range(k, len(stones) - k+1):
        num = max(stones[i:k+i])
        windows.append(num)
    return min(windows)
```

k의 길이 내에서 최대 값 찾기
= k 수 만큼 반복 수행

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

```python
def solution(stones, k):
    windows = [max(stones[:k])]
    # 한칸씩 옆으로가며 k칸 내 최대 값 확인
    for i in range(k, len(stones) - k+1):
        num = max(stones[i:k+i])
        windows.append(num)
    return min(windows)
```

$O(n*k) = O(n^2)$
$= 200,000 * 200,000$

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = INF

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = INF

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = INF

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = 5

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = 5

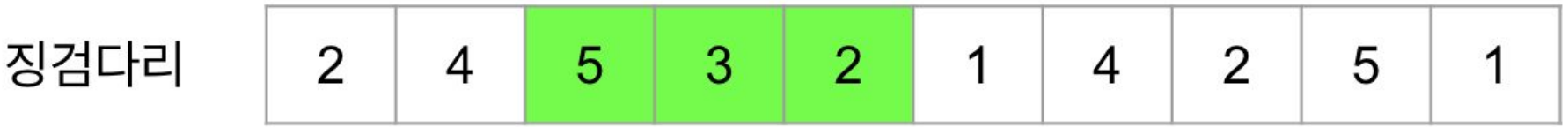
카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = 5

카카오 징검다리 건너기

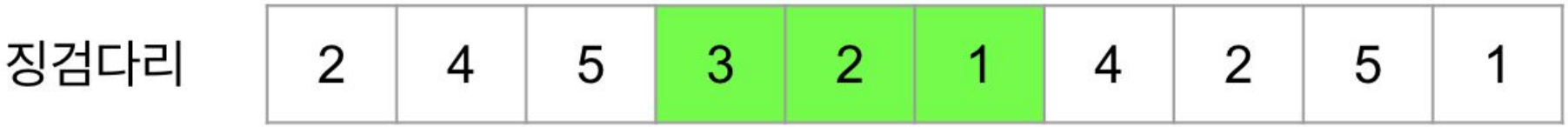
https://school.programmers.co.kr/learn/courses/30/lessons/64062



지나갈 수 있는 캐릭터 수 = 5

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062



지나갈 수 있는 캐릭터 수 = 3

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = 3

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = 3

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

지나갈 수 있는 캐릭터 수 = 3

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

dequeue 자료구조를 활용한 불필요한 탐색 제거

```python
from collections import deque

def solution(stones, k):
    dq = deque()
    result = float('inf')

    for i in range(len(stones)):
        if dq and dq[0] <= i - k:
            dq.popleft()

        while dq and stones[dq[-1]] <= stones[i]:
            dq.pop()

        dq.append(i)

        if i >= k - 1:
            result = min(result, stones[dq[0]])

    return result
```

카카오 징검다리 건너기

https://school.programmers.co.kr/learn/courses/30/lessons/64062

이진 탐색을 활용한 최적의 값 찾기
https://tech.kakao.com/posts/381

```python
def solution(stones, k):
    answer = 0
    left = 1
    right = max(stones)

    while left <= right:
        count = 0
        mid = (left+right) // 2
        for s in stones:
            if s - mid <= 0:
                count += 1
            else:
                count = 0

            if count == k:
                break

        if count < k:
            left = mid +1
        else:
            right = mid -1
            answer = mid

    return answer
```

고생하셨습니다 :)