1부 코딩 인터뷰

01 코딩 인터뷰

코딩 인터뷰 기술직군 채용을 위한 기술 문제 중심의 개발인터뷰

면접관이 함께함

코딩 테스트 면접관이 없음

코딩 인터뷰를 위한 온라인 테스트 플랫폼 해커랭크 외국

코딜리티 외국

리모트 인터뷰 외국, 결과를 안 알려줌

프로그래머스 국내

별도 자기 플랫폼 삼성

온라인 코딩 테스트의 사전 준비사항 연습장과 필기 도구 구조를 적어가며 풀 것

프로그래밍 언어 기업에서 강제하는 경우도 있음

자신만의 코드 스니펫 자주 쓰이는 동작에 한해 정리

ex) 연결리스트 삭제, 뒤집기

모든 테스트케이스 통과하기 결과를 통해 오류 수정하는 방식은 적용하기 힘듬 🡪 제출 전에 꼼꼼히 확인 후 제출

타임아웃이 발생하는 경우 최적화 필요, 특히 파이썬은

예외처리 상단에 입력값 검증 꼭 넣기 or 알고리즘을 짜임새있게 적기

문제를 잘못 접근할 때 ex) 최대값을 최소값으로 잘못 읽었을 때, BFS을 DFS로 잘못 풀었을 때

문제 당 제한시간을 두고, 제한시간을 넘기면 다음 문제로

풀이시간을 초과했을 때 일단 끝까지 시도해보고, 되면 면접관에게 메일 보내보기

코딩 도구?? 필수라고 생각함

IDE에 부정적인 면접관 내가 IDE없어도 개발이 가능하다는 점을 강조할 것

능력이 없어서가 아닌 “개발의 편의성”을 위해 IDE를 선택했음을 알릴 것

REFL도구 사용하기 REFL 사용자가 입력한 프로그램을 읽고 값을 계산한 다음 출력하는 일을 반복하는 구조

ex) 주피터 노트북

화이트 코딩 인터뷰 오프라인, 면접관 앞에서 코드 설명

온라인 코딩 테스트 사소한 것까지 모두 챙겨서, 실제로 오류없이 구동되는 프로그램 제작하기

화이트 코딩 인터뷰 핵심을 관통하는 알고리즘 중심

면접관에게 질문 가능

02 프로그래밍 언어 선택

수도코드 작동원리, 알고리즘을 형식이 정해져있지 않은 고차원언어로 기술한 것

언어 선호도 1. c++, 2. 파이썬

C++보다 파이썬이 라인 수가 더 작음

주언어: 파이썬, 보조언어: C++로 하자

\* 언어를 회사에서 강제하는 경우도 있음

코딩 테스트에 최적인 프로그래밍 언어 면접관이 쉽게 이해할 수 있는가? 아무도 모르는 얼랭, 리스프같은거 쓰지마라

코딩 플랫폼에서 지원하는 가? C++은 모든 플랫폼은 거의 지원한다, 파이썬은 회사에서 제한하는 경우도 있다.

유연한 언어인가? C++ 엄청 엄격하다 ex) 변수 선언시 자료형 지정 필요

파이썬은 유연함

언어레벨에서 풍부한 기능을 지원하는가? C++은 STL을 지원함

파이썬은 거의 치트키 수준의 라이브러리 제공

2부 파이썬

02 파이썬

파이썬 실행가능한 수도코드

연구 개발의 주력 언어

파이썬에 대한 이해 문법을 제대로 알아야 함

파이썬 문법 인덴트 PEP8에 따라 공백 4칸으로 할 것

파이참을 이용하면 자동으로 맞춰줌

네이밍 컨벤션 각 단어를 \_로 구분 = 스네이크 케이스

타입 힌트 타입을 지정할 수 있음

ex) a: str = “1”

b: int = 1

c: str = 123 #이것도 가능, 강제적으로 지정하는 것은 아님

ex) def fn(a: int) -> bool : #매개변수는 정수를 받고, return값은 true, false임

리스트 컴프리헨션 기존 리스트를 기반으로 새로운 리스트를 만들어 내는 기법

ex) [ n \* 2 for n in range(1, 10+1) if n % 2 == 0 ]

딕셔너리에서도 사용 가능

제너레이터 루프의 반복 동작을 제어할 수 있는 루틴 형태

제너레이터만 생성해두면 필요할 때만 값을 만들어 낼 수 있음 🡪 메모리적으로 이득

리턴 yield 값

값생성 next(제너레이터)

ex) def generator():

yield 1

yield "string"

yield True

>>> g = generator()

>>> next(g)

1

>>> next(g)

'string'

>>> next(g)

True

range() range 클래스 리턴

내부적으로는 매번 다음 숫자를 생성 = 제너레이터

인덱스로 접근 시, 인덱스에 해당하는 값을 바로 생성 🡪 리스트랑 비슷한 느낌

enumerate() 여러가지 자료형을 인덱스를 포함한 enumerate객체로 리턴

인덱스를 매겨야할 필요가 있을 때 편함

>>> a = [1,2,3,2,45,2,5]

>>> for i, v in enumerate(a):

print(i,v)

0 1

1 2

2 3

3 2

4 45

5 2

6 5

나눗셈 연산자 // /은 실수값을 리턴함 ex) 5/3 = 1.666

//은 정수값을 리턴함 ex) 5//3 = 1

%는 나머지를 리턴함 ex) 5%3 = 2

divmod() 몫과 나머지를 동시에 리턴함 ex) >>> divmod(5,3)

(1, 2)

print() sep매개변수 구분자를 ‘\n’에서 사용자 지정으로 바꿈 ex) >>> print(1,2 , sep = ", ")

1, 2

문자열 포맷 출력 {}이 서식지정자, 문자열의 멤버함수 format(출력되는 변수)

ex) >>> print("{}: {}".format(idx+1, fruit))

2: apple

pass 코드의 외부, 골격을 잡아놓을 때, 내부에 써넣는 코드

아무 연산도 하지 않음

locals 로컬 심볼 테이블 딕셔너리를 가져오는 메소드

pprint.pprint() 보기 좋게 줄바꿈 해주면서 출력

로컬에 선언된 모든 변수를 조회

ex)

>>> import pprint

>>> pprint.pprint(locals())

{'\_\_annotations\_\_': {'a': <class 'str'>, 'b': <class 'int'>},

'\_\_builtins\_\_': <module 'builtins' (built-in)>,

'\_\_doc\_\_': None,

'\_\_loader\_\_': <class '\_frozen\_importlib.BuiltinImporter'>,

'\_\_name\_\_': '\_\_main\_\_',

'\_\_package\_\_': None,

'\_\_spec\_\_': None,

'a': ['1', '2', '3', '2', '45', '2', '5'],

'b': '234',

'fruit': 'apple',

'g': <generator object generator at 0x0000020BEE458430>,

'generator': <function generator at 0x0000020BEE47A9D0>,

'i': 6,

'idx': 1,

'pprint': <module 'pprint' from 'C:\\Users\\Lee\\AppData\\Local\\Programs\\Python\\Python39\\lib\\pprint.py'>,

'v': 5}

코딩 스타일 코드를 면접관이 정성적으로 평가함 🡪 코딩 스타일에 신경을 써야함

파이참 같은 IDE를 사용하면 good

변수명과 주석 변수명은 상세히

간단한 주석을 달 것

리스트 컴프리헨션 리스트 컴프리헨션을 너무 남발하면 가독성을 떨어뜨림

표현식이 2개를 넘지말 것

구글 파이썬 스타일 가이드 함수의 매개변수로 가변객체(리스트, 딕셔너리)를 쓰지 말 것

True, False를 판별할 때는 암시적인 방법 활용 ex) if not users:

정수를 처리할 때는 명시적인 방법 활용 ex) if foo == 0:

04 빅오, 자료형

빅오 입력값이 커질 때 알고리즘의 실행시간과 함께 공간복잡도가 어떻게 증가하는 지 분류

빅오 입력값 n이 무한대를 향할 때, lim 함수의 실행시간 추이를 나타냄

최고차항만을 고려

O(1) 입력값이 커도 실행시간 일정

O(log n) 매우 큰 입력값에도 크게 영향을 받지 않음

O(n) 선형시간 알고리즘

O(nlog n) 대부분의 효율좋은 알고리즘

O(n2) 비효율적인 알고리즘

O(2n) 큼

O(n!) 최악

시간과 공간의 트레이드오프 실행시간이 빠른 알고리즘은 공간을 많이 사용하고, 공간을 적게 차지하는 알고리즘은 실행시간이 느림

상한과 세타 빅오 상한을 의미

n0이하의 값은 무시(빅오가 상한보다 클 때)

빅오메가 가장 빨리 실행될 때

빅세타 평균적으로 실행될 때

분할 상환분석 최악의 경우를 여러 번에 걸쳐 골고루 나눠주는 형태로 알고리즘의 시간복잡도 계산

병렬화 병렬연산으로 실행속도 높이기

병렬화가 가능한가는 알고리즘의 우수성을 평가하는 매우 중요한 척도

자료형 객체로 자료형 표현

종류 None(class None)

숫자 실수(class float) 불변

정수형 정수(class int) 불변

불리언(class bool) 불변

집합형 집합(class set) 가변

매핑 딕셔너리(class dict) 가변

시퀀스 불변 문자열(class str) 불변

튜플(class tuple) 불변

바이트(class bytes) 불변

가변 리스트(class list) 가변

원시타입 c, 자바

원시타입은 메모리에 정확하게 타입크기만큼 공간 할당 🡪 매우 빠른 속도

자바에서는 객체로 원시타입을 표현 가능 🡪 문자변환, 시프트 연산, 지원, 공간복잡도 up

객체 파이썬은 객체로만 타입 정의

변수는 불변객체를 가리키는 참조

c++과는 다른 참조 c++ 참조 변수에 값을 할당하면 참조대상 또한 할당되는 값으로 변경

ex) int a = 10;

int &b = a;

b = 7; //a는 7

파이썬 참조변수에 새로운 값을 할당하면 기존의 참조를 없애고 새로운 참조로 변경

ex) a = 10

b = a

b = 7 #a는 10

is와 == ==은 값을 비교하는 연산자

is는 id를 비교하는 연산자

None은 값이 없음 🡪 None은 ==로 비교불가, is로만 비교해야 함 ex) if a is None:

pass

속도 객체로 처리하는 자료형은 원시타입보다 훨씬 느림

05 리스트, 딕셔너리

리스트 순서대로 저장하는 시퀀스

입력 순서 유지, 내부적으로는 동적배열

파이썬 스택, 큐의 연산을 메서드로 사용가능

활용방법 선언 >>> a = list()

>>> a = []

초기값 지정 >>> a = [1,2,3]

요소 추가 끝에 추가 >>> a.append(4)

인덱스 지정 후 추가 >>> a.insert(3,5) #3번째에 5삽입

요소 추출 하나만 >>> a[3]

슬라이싱 >>> a[1:3]

>>> a[:3]

띄엄띄엄 슬라이싱 >>> a[1:4:2]

요소 삭제 인덱스로 삭제 >>> del a[1] #인덱스1 삭제

값으로 삭제 >>> a.remove(3)

맨 뒤에 요소 팝 >>> a.pop()

중간의 요소 팝 >>> a.pop(3)

리스트의 특징 리스트의 내부적 구현 요소에 대한 포인터 목록을 갖고 있는 구조체

C언어적으로 표현하면, 노드에 대한 포인터 목록을 배열 형태로 관리

딕셔너리 순서대로 키/값 구조로 이뤄진 딕셔너리

입력 순서는 유지되나 버전에 따라 안될 수 있음

내부적으로 해시 테이블로 구현

키는 불변객체, 값은 모두 가능

활용방법 선언 >>> a = dict()

>>> a = {}

키로 조회 >>> a['key1']

for 반복문으로 순회 >>> for k, v in a.items():

print(k,v)

키로 삭제 >>> del a['key1']

딕셔너리 모듈 딕셔너리와 관련된 특수한 형태의 컨테이너 자료형

모두 collections의 메서드로 생성 🡪 import collections 필요

defaultdict 객체 존재하지 않는 키를 조회할 경우, 디폴트값을 기준으로 새로운 아이템 생성

>>> a = collections.defaultdict(int)

>>> a['a'] = 5

>>> a['b'] += 1

>>> a

defaultdict(<class 'int'>, {'a': 5, 'b': 1})

Counter 객체 아이템에 대한 개수를 계산해 딕셔너리로 리턴

딕셔너리를 collections.Counter로 래핑한 collections.Counter 클래스 리턴

>>> a = [1,2,3,4,5,5,5,6,6]

>>> b = collections.Counter(a)

>>> b

Counter({5: 3, 6: 2, 1: 1, 2: 1, 3: 1, 4: 1})

빈도가 가장 높은 요소 추출 most\_common 사용

>>> b.most\_common(2)

[(5, 3), (6, 2)]

OrderedDict 객체 3.7 이하의 버전에서는 딕셔너리가 순서대로 동작하지 않음

OrderedDict 순서를 보장하는 딕셔너리

>>> collections.OrderedDict({'banana':3, 'apple':4, 'pear':1})

OrderedDict([('banana', 3), ('apple', 4), ('pear', 1)])

06 문자열 조작

문자열 조작 문자열 변경, 분리

쓰이는 분야 정보처리 분야

통신 시스템 분야

프로그래밍 시스템 분야

01 유효한 펠린드롬 주어진 문자열이 펠린드롬인지 확인하라. 대소문자를 구분하지 않으며, 영문자와 숫자만을 대상으로 한다.

125

처음 내 풀이 class Solution:  
 def isPalindrome(self, s: str) -> bool:  
 str\_list = []  
 for i in range(0, len(s)):  
 ss = s[i].lower()  
 if(ss in "qwertyuiopasdfghjklzxcvbnm1234567890"):  
 str\_list.append(ss)  
 str\_list\_reversed = list(reversed(str\_list))  
 print(str\_list)  
 print(str\_list\_reversed)  
 for i in range(0, len(str\_list)):  
 if(str\_list\_reversed[i] != str\_list[i]):  
 print(str\_list[i], str\_list\_reversed[i])  
 return False  
 return True  
  
lol = Solution()  
print(lol.isPalindrome("123A man, a plan, a canal: Panama321"))

풀이1 리스트로 변환

class Solution:  
 def isPalindrome(self, s: str) -> bool:  
 strs = []  
 for char in s:  
 if char.isalnum():  
 strs.append(char.lower())  
 while len(strs) > 1:  
 if strs.pop(0) != strs.pop():  
 return False  
 return True

풀이2 데크 자료형을 이용한 최적화

풀이1보다 빠름

import collections  
   
 class Solution:  
 def isPalindrome(self, s: str) -> bool:  
 strs: Deque = collections.deque()  
 for char in s:  
 if char.isalnum():  
 strs.append(char.lower())  
 while len(strs) > 1:  
 if strs.popleft() != strs.pop():  
 return False  
 return True  
   
 lol = Solution()  
 print(lol.isPalindrome("123A man, a plan, a canal: Panama321"))

풀이3 슬라이싱 사용

내부적으로 슬라이싱은 C로 구현 🡪 엄청나게 빠름

문자열조작 시, 슬라이싱으로 처리하는 것이 제일 빠름

import re  
   
 class Solution:  
 def isPalindrome(self, s: str) -> bool:  
 s = s.lower()  
 s = re.sub('[^a-z0-9]', '', s)  
 return s == s[::-1]

02 문자열 뒤집기 문자열을 뒤집는 함수를 작성하라. 입력값은 문자 배열이며, 리턴 없이 리스트 내부를 직접 조작해라

처음 내 풀이 class Solution:  
 def reverseString(self, s: List[str]) -> None:  
 *"""*  *Do not return anything, modify s in-place instead.*  *"""* s.reverse()

풀이1 투 포인터를 이용한 스왑

class Solution:  
 def reverseString(self, s: List[str]) -> None:  
 *"""*  *Do not return anything, modify s in-place instead.*  *"""* left, right = 0, len(s)-1  
 while left < right:  
 s[left], s[right] = s[right], s[left]  
 left+=1  
 right-=1

풀이2 파이썬다운 방식

그냥 reverse함수를 이용

class Solution:  
 def reverseString(self, s: list[str]) -> None:  
 *"""*  *Do not return anything, modify s in-place instead.*  *"""* s.reverse()

03 로그파일 재정렬 로그를 재정렬하라. 기준은 다음과 같다.

1. 로그의 가장 앞 부분은 식별자다.
2. 문자로 구성된 로그가 숫자로그보다 앞에 온다
3. 식별자는 순서에 영향을 끼치지 않지만, 문자가 동일할 경우 식별자 순으로 한다.
4. 숫자 로그는 입력 순서대로 한다.

처음 풀이 못품

풀이1 문자로그, 숫자로그를 모으는 리스트를 각각 선언 🡪 후에 +로 합침

class Solution:  
 def reorderLogFiles(self, logs: list[str]) -> list[str]:  
 letters, digits = [], []  
 for log in logs:  
 if log.split()[1]. isdigit():  
 digits.append(log)  
 else:  
 letters.append(log)  
   
 letters.sort(key = lambda x: (x.split()[1:], x.split()[0]))  
 return letters + digits

04 가장 흔한 단어 금지된 단어를 제외한 가장 흔하게 등장하는 단어를 출력하라. 대소문자 구분을 하지 않음, 구두점 또한 무시한다.

내 풀이 import collections  
 import re  
   
 class Solution:  
 def mostCommonWord(self, paragraph: str, banned: list[str]) -> str:  
 paragraph = paragraph.lower()  
 paragraph\_re = re.sub("[^\w]", " ", paragraph)  
 paragraph\_list = paragraph\_re.split(" ")  
 counter = dict(collections.Counter(paragraph\_list))  
   
 for i in banned:  
 if i in counter:  
 del counter[i]  
   
 if " " in counter:  
 del counter[" "]  
   
 if "" in counter:  
 del counter[""]  
   
 print(counter)  
 max\_value = 0  
 max\_key = 0  
 for k, v in counter.items():  
 if(v > max\_value):  
 max\_value = v  
 max\_key = k  
   
 return max\_key

풀이1 리스트 컴프리헨션, Counter 객체 사용

데이터 클렌징 입력값에 대한 전처리 작업

주로 re 모듈로 진행

import collections  
 import re

class Solution:  
 def mostCommonWord(self, paragraph: str, banned: list[str]) -> str:  
 words = [word for word in re.sub(r"[^\w]", " ", paragraph).lower().split()  
 if word not in banned]  
   
 counts = collections.Counter(words)  
 return counts.most\_common(1)[0][0]

05 그룹 애너그램 문자열 배열을 받아 애너그램 단위로 그룹핑해라

내 풀이 class Solution:  
 def groupAnagrams(self, strs: list[str]) -> list[list[str]]:  
 dic = {}  
 for word in strs:  
 key = str("".join(sorted(word)))  
 if key not in dic.keys():  
 dic[key] = [word]  
 else:  
 dic[key].append(word)  
 return list(dic.values())

풀이1 정렬하여 딕셔너리 추가

애너그램 관계인 단어들을 정렬 🡪 같은 값을 가짐 🡪 키로 이용

defaultdict()를 사용 키 값이 없을 때를 고려

비교구문을 쓰지 않아도 됨

여러가지 정렬 방법 sorted() 파이썬에서는 sorted()라는 정렬함수 제공

key 지정 정렬을 위한 키, 함수를 별도로 지정 가능

ex) c = ['ccc', 'aaaa', 'd', 'bb']  
 sorted(c, key = len)

ex) a = ['cde', 'cfc', 'abc']  
 def fn(s):  
 return s[0], s[-1]  
 scorted(a, key=fn)

리턴 정렬된 리스트

리스트.sort() 리스트 자체를 제 자리 정렬

리턴 none

06 가장 긴 팰린드롬 부분 문자열 가장 긴 팰린드롬 문자열을 출력하라

5

풀이1 중앙을 중심으로 확장하는 풀이

투 포인터가 중앙을 중심으로 확장하는 풀이

class Solution:  
 def longestPalindrome(self, s: str) -> str:  
 def expand(left: int, right:int) ->str:  
 while left >= 0 and right < len(s) and s[left] == s[right]:  
 left -=1  
 right += 1  
 return s[left + 1:right]  
   
 if len(s) < 2 or s == s[::-1]:  
 return s  
   
 result = ''  
 for i in range(len(s)-1):  
 result = max(result, expand(i, i+1), expand(i, i+2), key = len)  
 return result

유니코드와 UTF-8 유니코드 파이썬3에서 문자열을 처리하는 방식

영문자를 표현할 때 메모리 낭비가 심함

UTF-8 인코딩 방식

맨 앞 비트를 확인해 몇 바이트인지 결정 🡪 메모리 절약

3부 선형 자료구조

07 배열

배열 값 또는 변수 엘리먼트의 집합으로 구성된 구조

하나 이상의 인덱스 또는 키로 식별

자료구조의 구분 메모리 공간 기반의 연속방식 – 배열

포인터 기반의 연결 방식 – 연결리스트

동적 배열 자동으로 크기를 리사이징

ex) std::vector

재할당 비율 자동으로 크기를 늘려가는 비율

07 두 수의 합 덧셈하여 타겟을 만들 수 있는 배열의 두 숫자 인덱스를 리턴해라

내 풀이 class Solution:  
 def twoSum(self, nums: list[int], target: int) -> list[int]:  
 res = []  
 for i in range(len(nums)):  
 for j in range(i+1, len(nums)):  
 if nums[i] + nums[j] == target:  
 res.append(i)  
 res.append(j)  
 return res

풀이1 브루트 포스로 계산

최적화할 수 있는 여러 방법들이 있음

너무 느림

풀이2 in을 이용한 탐색

타겟에서 첫 번째 값을 뺀 값 target – n이 존재하는지 탐색

class Solution:  
 def twoSum(self, nums: list[int], target: int) -> list[int]:  
 for i, n in enumerate(nums):  
 complement = target - n  
 if complement in nums[i+1:]:  
 return [nums.index(n), nums[i+1:].index(complement)+i+1]

풀이3 첫 번째 수를 뺀 결과 키 조회

타겟에서 첫 번째 수를 빼면 두 번째 수를 바로 알아낼 수 있음 🡪 두 번째 수를 키로 하고 기존 인덱스를 값으로 하는 딕셔너리 생성 🡪 바로 정답을 찾음

class Solution:  
 def twoSum(self, nums: list[int], target: int) -> list[int]:  
 nums\_map = {}  
 for i, num in enumerate(nums):  
 nums\_map[num] = i  
   
 for i, num in enumerate(nums):  
 if target - num in nums\_map and i != nums\_map[target-num]:  
 return [i, nums\_map[target - num]]

풀이4 풀이3에서 조회구조 개선

class Solution:  
 def twoSum(self, nums: list[int], target: int) -> list[int]:  
 nums\_map = {}  
 for i, num in enumerate(nums):  
 if target - num in nums\_map:  
 return [nums\_map[target-num], i]  
 nums\_map[num] = i

풀이5 투 포인터 이용

왼쪽 포인터, 오른쪽 포인터의 합이 타겟보다 크면 오른쪽 포인터를 왼쪽으로, 작다면 왼쪽 포인터를 오른쪽으로 옮기면서 값을 조정

불가능 nums가 정렬된 상태가 아님 🡪 정렬하면 인덱스가 틀어짐 🡪 풀이불가

이 방법으로 푼다면 상당히 고생

08 빗물 트래핑 내 풀이 모르겠음…

풀이1 투 포인터를 최대로 이동

막대의 역할 왼쪽과 오른쪽을 가르는 장벽역할

최대 높이의 막대까지 각각 (좌우 기둥 최대 높이 – 현재 높이의 차이)를 물의 양에 더함 🡪 포인터 이동

class Solution:  
 def trap(self, height: list[int]) -> int:  
 if not height:  
 return 0  
   
 res = 0  
 left, right = 0, len(height)-1  
 left\_max, right\_max = height[left], height[right]  
 while left < right:  
 left\_max, right\_max = max(left\_max, height[left]), max(right\_max, height[right])  
   
 if left\_max <= right\_max:  
 res += left\_max - height[left]  
 left += 1  
 else:  
 res += right\_max - height[right]  
 right -= 1  
 return res

풀이2 스택 쌓기

변곡점을 기준으로 격차 만큼 물 높이를 채움

높이를 스택으로 채워나가다가 변곡점을 만날때마다 스택에서 하나씩 꺼내면서 이전과의 차이만큼 물높이를 채움 = 현재 높이가 이전 높이보다 높을 때

class Solution:  
 def trap(self, height:list[int]) -> int:  
 stack = []  
 volume = 0  
   
 for i in range(len(height)):  
 while stack and height[i] > height[stack[-1]]:  
 top = stack.pop()  
   
 if not len(stack):  
 break  
   
 distance = i - stack[-1] -1  
 waters = min(height[i], height[stack[-1]]) - height[top]  
 volume += distance \* waters  
 stack.append(i)  
 return volume

09 세 수의 합 배열을 입력받아 합으로 0을 만들 수 있는 3개의 엘리먼트를 출력하라

내 풀이 from itertools import combinations  
  
 class Solution:  
 def threeSum(self, nums: list[int]) -> list[list[int]]:  
 list\_com = list(combinations(nums, 3))  
 set\_com = set( tuple(sorted(x)) for x in list\_com )  
   
 res = []  
   
 for i in set\_com:  
 if sum(i) == 0:  
 res.append(list(i))  
 return res

시간초과

풀이1 브루트포스로 계산

먼저 리스트를 정렬 🡪 중복된 값 검사를 쉽게하기 위함

class Solution:  
 def threeSum(self, nums: list[int]) -> list[list[int]]:  
 results = []  
 nums.sort()  
   
 for i in range(len(nums)-2):  
 if i > 0 and nums[i] == nums[i-1]:  
 continue  
 for j in range(i+1, len(nums)-1):  
 if j > i+1 and nums[j] == nums[j-1]:  
 continue  
 for k in range(j+1, len(nums)):  
 if k > j+1 and nums[k] == nums[k-1]:  
 continue  
 if nums[i] + nums[j] + nums[k] == 0:  
 results.append([nums[i], nums[j], nums[k]])  
   
 return results

시간 초과 때문에 이 풀이는 쓰지 못함

풀이2 투 포인터로 합 계산

중복된 값인 경우 continue로 스킵

i의 다음지점과 마지막지점을 left와 right로 설정 🡪 간격을 좁혀가며 sum 계산

left이동 sum이 0보다 작음

right이동 sum이 0보다 큼

sum이 0이면 추가 🡪 중복된 값 검사

10 배열 파티션1 n개의 페어를 이용한 min(a, b)의 합으로 만들 수 있는 가장 큰 수를 출력하라

내 풀이 class Solution:  
 def Array\_Partition(self, nums: list[int]) -> int:  
 nums.sort()  
 start = len(nums) // 2  
 return sum(nums[start:-1])

풀이1 오름차순 풀이

min을 합산했을 때 최대를 만들어야 함 🡪 min이 되도록 커야함

정렬한 상태에서 앞에서부터 오름차순으로 짝을 지음

class Solution:  
 def Array\_Partition(self, nums: list[int]) -> int:  
 sum = 0  
 pair = []  
 nums.sort()  
   
 for n in nums:  
 pair.append(n)  
 if len(pair) == 2:  
 sum += min(pair)  
 pair = []  
 return sum

풀이2 짝수 번째 값 계산

min을 구하지 않고 정렬을 한번 더 이용 🡪 항상 짝수번째 값에 위치

class Solution:  
 def Array\_Partition(self, nums: list[int]) -> int:  
 sum = 0  
 nums.sort()  
 for i, n in enumerate(nums):  
 if i% 2 == 0:  
 sum+=n  
 return sum

풀이3 파이썬다운 방식

슬라이싱 이용

class Solution:  
 def Array\_Partition(self, nums: list[int]) -> int:  
 return sum(sorted(nums)[::2])

11 자신을 제외한 배열의 곱 배열을 입력받아 output[i]가 자신을 제외한 나머지 모든 요소의 곱셈 결과가 되도록 출력하라

처음 풀이 import copy  
 import math  
   
 class Solution:  
 def productExceptSelf(self, nums: list[int]) -> list[int]:  
 nums\_copy = copy.deepcopy(nums)  
 output = []  
 for i in nums:  
 nums\_copy.remove(i)  
 output.append(math.prod(nums\_copy))  
 nums\_copy.append(i)  
 return output

풀이1 자기 자신을 제외한 왼쪽 곱셈 결과 \* 오른쪽 곱셈 결과

class Solution:  
 def productExceptSelf(self, nums: list[int]) -> list[int]:  
 output = []  
 p = 1  
 for i in range(len(nums)):  
 output.append(p)  
 p = p \* nums[i]  
 p = 1  
 for i in range(len(nums)-1, -1, -1):  
 output[i] = output[i] \* p  
 p = p \* nums[i]  
 return output

12 주식을 사고팔기 가장 좋은 시점 한번의 거래로 낼 수 있는 최대 이익을 산출해라

처음 풀이 import heapq  
import sys  
  
  
class Solution:  
 def maxProfit(self, prices: list[int]) -> int:  
 profit = 0  
 profit\_list = [sys.maxsize]  
  
 for i in range(1, len(prices)):  
  
 if profit + (prices[i] - prices[i - 1]) > 0:  
 heapq.heappush(profit\_list, -profit)  
 profit += (prices[i] - prices[i - 1])  
  
 else:  
 heapq.heappush(profit\_list, -profit)  
 if i != len(prices) - 1:  
 profit = 0  
  
 return max(profit, -heapq.nsmallest(1, profit\_list)[0])

풀이1 브루트 포스로 계산

class Solution:  
 def maxProfit(self, prices: list[int]) -> int:  
 max\_price = 0  
   
 for i, price in enumerate(prices):  
 for j in range(i, len(prices)):  
 max\_price = max(prices[j] - price, max\_price)  
   
 return max\_price

풀이2 저점과 현재값의 차이 계산

저점 계속 찾으면서, 저점 이후 포인터와의 가격차이 계산 🡪 최대값 교체

import sys  
   
   
 class Solution:  
 def maxProfit(self, prices: list[int]) -> int:  
 profit = 0  
 min\_price = sys.maxsize #시스템상의 최대값으로 설정  
   
 for price in prices:  
 min\_price = min(min\_price, price)  
 profit = max(profit, price-min\_price)  
   
 return profit

08 연결리스트

연결리스트 선형 자료구조

노드 삽입, 삭제 간편

탐색 O(n)

시작, 끝에 추가∙삭제 O(1)

13 팰린드롬 연결리스트 연결리스트가 팰린드롬 구조인지 판별하라

처음 풀이 class Solution:  
 def isPalindrome(self, head: [ListNode]) -> bool:  
 res = []  
 p = head  
 while p != None:  
 res.append(p.val)  
 p = p.next  
  
 return res == res[::-1]

풀이1 리스트 변환

리스트 변환 🡪 리스트에서 요소를 추출해가면서 비교

class Solution:  
 def isPalindrome(self, head: [ListNode]) -> bool:  
 q: list = []  
  
 if not head:  
 return True  
  
 node = head  
  
 while node is not None:  
 q.append(node.val)  
 node = node.next  
  
 while len(q) > 1:  
 if q.pop(0) != q.pop():  
 return False  
 return True

풀이2 데크를 이용한 최적화

리스트.pop() O(n)의 시간복잡도 🡪 매우 느림 🡪 데크 이용

import collections  
  
  
class Solution:  
 def isPalindrome(self, head: [ListNode]) -> bool:  
 q: deque = collections.deque()  
  
 if not head:  
 return True  
  
 node = head  
  
 while node is not None:  
 q.append(node.val)  
 node = node.next  
  
 while len(q) > 1:  
 if q.popleft() != q.pop():  
 return False  
 return True

풀이3 고 언어를 이용한 구현

데크를 모두 다 구현해야함 🡪 고 언어는 쓰지마

풀이4 런너를 이용한 우아한 풀이

런너 연결리스트를 순회할 때 2개의 포인터를 동시에 사용하는 기법

병합지점, 중간위치, 길이를 판별 시 사용

빠른 런너 대개 2칸 씩 이동

느린 런너 대개 1칸 씩 이동 🡪 빠른런너가 끝에 도달하면, 느린런너는 정확히 중간지점

느린런너를 중간에 위치시키면서 새로운 연결리스트의 헤드에 요소 추가

중간부터 끝까지 느린런너를 이동시키면서, 역순으로 만든 연결리스트의 요소와 일치하는 지 비교

14 두 정렬리스트의 병합 정렬되어 있는 두 연결리스트를 합쳐라

내 풀이 class Solution:  
 def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:  
 if list1 == None and list2 == None:  
 return None  
 elif list1 == None and list2 != None:  
 return list2  
 elif list1 != None and list2 ==None:  
 return list1  
  
 nlist = ListNode()  
 nlist\_p = nlist  
  
 while (list1 != None) and (list2 != None):  
 if list1.val > list2.val:  
 nlist\_p.val = list2.val  
 list2 = list2.next  
 else:  
 nlist\_p.val = list1.val  
 list1 = list1.next  
 if (list1 != None) and (list2 != None):  
 nlist\_p.next = ListNode()  
 nlist\_p = nlist\_p.next  
  
 if list1 == None:  
 nlist\_p.next = list2  
  
 if list2 == None:  
 nlist\_p.next = list1  
  
 return nlist

풀이1 재귀구조로 연결

첫 번째부터 비교하면서 리턴

연결리스트 2개를 하나로 합침 🡪 스왑이용

class Solution:  
 def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) -> Optional[ListNode]:  
 if (not list1) or (list2 and list1.val > list2.val):  
 list1, list2 = list2, list1  
 if list1:  
 list1.next = self.mergeTwoLists(list1.next, list2)  
 return list1

15 역순 연결 리스트 연결리스트를 뒤집어라

내 풀이 import collections  
   
 class Solution:  
 def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 if head == None:  
 return head  
   
 dq = collections.deque()  
   
 while head != None:  
 dq.appendleft(head.val)  
 head = head.next  
   
 res = ListNode()  
 res\_p = res  
 while len(dq) != 0:  
 res\_p.val = dq.popleft()  
 if len(dq) != 0:  
 res\_p.next = ListNode()  
 res\_p = res\_p.next  
   
 return res

풀이1 재귀구조로 뒤집기

다음 노드와 현재노드를 파라미터로 해서 재귀호출

class Solution:  
 def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 def reverse(node: ListNode, prev: ListNode = None):  
 if not node:  
 return prev  
 next, node.next = node.next, prev  
 return reverse(next, node)  
 return reverse(head)

풀이2 반복구조로 뒤집기

class Solution:  
 def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 node, prev = head, None  
   
 while node:  
 next, node.next = node.next, prev  
 prev, node = node, next  
   
 return prev

16 두 수의 덧셈 역순으로 저장된 연결리스트의 숫자를 더하라

내 풀이 class Solution:  
 def addTwoNumbers(self, l1, l2):  
 d1 = collections.deque()  
 d2 = collections.deque()  
   
 if l1 == None:  
 return l2  
   
 if l2 == None:  
 return l1  
   
 while l1 != None:  
 d1.appendleft(str(l1.val))  
 l1 = l1.next  
   
 while l2 != None:  
 d2.appendleft(str(l2.val))  
 l2 = l2.next  
   
 str1 = "".join(d1)  
 str2 = "".join(d2)  
   
 res = (str(eval(str1+"+"+str2)))  
 l3 = ListNode()  
 l3\_p = l3  
   
 for i in range(len(res)-1, 0, -1):  
 l3\_p.val = int(res[i])  
 if i != 0 :  
 l3\_p.next = ListNode()  
 l3\_p = l3\_p.next  
 l3\_p.val = int(res[0])  
 return l3

풀이1 자료형 변환

연결리스트 🡪 문자열 🡪 숫자 🡪 계산 🡪 연결리스트

class Solution:  
 def addTwoNumbers(self, l1, l2):  
 def reverseList(self, head):  
 node, prev = head, None  
 while node:  
 next, node.next = node.next, prev  
 prev, node = node, next  
 return prev  
   
 def toList(self, node):  
 list = []  
 while node:  
 list.append(node.val)  
 node = node.next  
 return list  
   
 def toReverseLinkedList(self, result):  
 prev = None  
 for r in result:  
 node = ListNode(r)  
 node.next = prev  
 prev = node  
 return node  
   
 def addTwoNumbers(self, l1, l2):  
 a = self.toList(self.reverseList(l1))  
 b = self.toList(self.reverseList(l2))  
   
 resultStr = int(''.join(str(e) for e in a) + int(''.join(str(e) for e in b)))  
 return self.toReverseLinkedList(str(resultStr))

풀이2 전가산기 구현

divmod()를 통해 자리올림수 처리

class Solution:  
 def addTwoNumbers(self, l1, l2):  
 root = head = ListNode(0)  
   
 carry = 0  
   
 while l1 or l2 or carry:  
 sum = 0  
 if l1:  
 sum += l1.val  
 l1 = l1.next  
 if l2:  
 sum += l2.val  
 l2 = l2.next  
   
 carry, val = divmod(sum + carry, 10)  
 head.next = ListNode(val)  
 head = head.next  
   
 return root.next

17 페어의 노드 스왑 연결리스트를 입력받아 페어 단위로 스왑하라

처음 풀이 class Solution:  
 def swapPairs(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 if head == None:  
 return None  
  
 p = head  
 while p != None and p.next != None:  
 p.val, p.next.val = p.next.val, p.val  
 p = p.next.next  
  
 return head

풀이1 값만 교환

처음 풀이와 동일

현실적이지 않음 🡪 연결리스트가 여러가지 구조체로 이뤄졌을 수도 있기 때문

class Solution:  
 def swapPairs(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 cur = head  
  
 while cur and cur.next:  
 cur.val, cur.next.val = cur.next.val, cur.val  
 cur = cur.next.next  
  
 return head

풀이2 반복구조로 스왑

next를 변경

head가 변경됨 🡪 root라는 별도 변수 필요

class Solution:  
 def swapPairs(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 root = prev = ListNode(None)  
 prev.next = head  
  
 while head and head.next:  
 b = head.next  
 head.next = b.next  
 b.next = head  
  
 prev.next = b  
  
 head = head.next  
 prev = prev.next.next  
  
 return root.next

풀이3 재귀구조로 스왑

기본적으로는 풀이2와 동일

class Solution:  
 def swapPairs(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 if head and head.next:  
 p = head.next  
  
 head.next = self.swapPairs(p.next)  
  
 p.next = head  
 return p  
 return head

18 홀짝연결리스트 연결리스트를 홀수 노드 다음에 짝수노드가 오도록 재구성해라

공간복잡도 O(1), 시간복잡도O(n)에 풀이하라

처음 풀이 class Solution:  
 def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 if head == None:  
 return None  
  
 p1 = head  
 p2\_root = p2 = head.next  
  
 while p2 and p2.next:  
 p1.next = p2.next  
 p1 = p1.next  
 p2.next = p1.next  
 p2 = p2.next  
  
 p1.next = p2\_root  
  
 return head

None이 항상 짝수 노드리스트에 오게함

풀이1 반복구조로 홀짝노드 처리

처음 풀이와 동일

class Solution:  
 def oddEvenList(self, head: Optional[ListNode]) -> Optional[ListNode]:  
 if head is None:  
 return None  
   
 odd = head  
 even = head.next  
 even\_head = head.next  
   
 while even and even.next:  
 odd.next, even.next = odd.next.next, even.next.next  
 odd, even = odd.next, even.next  
   
 odd.next = even\_head  
 return head

19 역순 연결 리스트2 인덱스 m에서 n까지를 역순으로 만들어라. 인덱스 m은 1부터 시작한다.

처음 풀이 class Solution:  
 def reverseBetween(self, head: Optional[ListNode], left: int, right: int) -> Optional[ListNode]:  
 if left == right:  
 return head  
 root = ListNode()  
 root.next = head  
  
 p1 = root  
 right\_p = head  
 node\_stack = []  
  
 for i in range(1, left):  
 p1 = p1.next  
 left\_p = p1.next  
  
 for i in range(1, right):  
 right\_p = right\_p.next  
 p2 = right\_p.next  
  
 new\_right\_p = left\_p  
  
 while left\_p != right\_p:  
 node\_stack.append(left\_p)  
 left\_p = left\_p.next  
  
 new\_left\_p = right\_p  
  
 while node\_stack:  
 node = node\_stack.pop()  
 right\_p.next = node  
 right\_p = right\_p.next  
  
 p1.next = new\_left\_p  
 new\_right\_p.next = p2  
  
 return root.next

풀이1 반복구조로 노드 뒤집기

root라는 노드를 하나 만들어 head전에 위치시킴 = root.next는 head

class Solution:  
 def reverseBetween(self, head: Optional[ListNode], left: int, right: int) -> Optional[ListNode]:  
 if not head or left == right:  
 return head  
  
 root = start = ListNode(None)  
 root.next = head  
  
 for \_ in range(right - 1):  
 start = start.next  
 end = start.next  
  
 for \_ in range(right - left):  
 tmp, start.next, end.next = start.next, end.next, end.next.next  
 start.next.next = tmp  
  
 return root.next

안 풀림…

책이 틀렸낭

09 스택, 큐

매우 고전적인 자료구조

스택 후입선출

큐 선입선출

리스트대신 데크를 사용할 것 🡪 성능차이 막심

스택 연산 push() 요소를 컬렉션에 추가

pop() 가장 최근에 push된 요소 제거

콜스택 서브루틴에 대한 정보 제공

스택 버퍼 오버플로우 꽉 찬 스택에 요소를 삽입하고자 할 때 발생하는 에러

연결리스트를 통한 스택 ADT 구현 class Node:  
 def \_\_init\_\_(self, item, next):  
 self.item = item  
 self.next = next  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.last = None  
   
 def push(self, item):  
 self.last = Node(item, self.last)  
   
 def pop(self):  
 item = self.last.item  
 self.last = self.last.next  
 return item

20 유효한 괄호 괄호로 된 입력값이 올바른지 판별하라

처음 풀이 import collections  
  
class Solution:  
 def isValid(self, s: str) -> bool:  
 dq = collections.deque()  
  
 if s == "" :  
 return True  
  
 for c in s:  
 if c in "[{(":  
 dq.append(c)  
  
 else:  
 if len(dq) == 0:  
 return False  
 if ( c == "}" and dq.pop() != "{"):  
 return False  
 elif (c == "]" and dq.pop() != "["):  
 return False  
 elif (c == ")" and dq.pop() != "("):  
 return False  
  
  
 return len(dq) == 0

풀이2 스택 일치 여부 판별

{[( 🡪 스택 푸시

)}] 🡪 스택 팝 일치 진행

불일치 return False

class Solution:  
 def isValid(self, s: str) -> bool:  
 stack = []  
 table = {  
 ')':'(', '}':'{', ']':'['  
 }  
  
 for char in s:  
 if char not in table:  
 stack.append(char)  
 elif not stack or table[char] != stack.pop():  
 return False  
  
 return len(stack) == 0

21 중복 문자 제거 “중복된 문자”를 없애 사전식 순서로 나열하라

처음 풀이 모르겠음…

풀이1 재귀를 이용한 분리

class Solution:  
 def removeDuplicateLetters(self, s: str) -> str:  
 for char in sorted(set(s)):  
 suffix = s[s.index(char):]  
 if set(s) == set(suffix):  
 return char + self.removeDuplicateLetters(suffix.replace(char, ''))  
 return ''

풀이2 스택을 이용한 문자제거

class Solution:  
 def removeDuplicateLetters(self, s: str) -> str:  
 counter, seen, stack = collections.Counter(s), set(), []  
  
 for char in s:  
 counter[char] -= 1  
 if char in seen:  
 continue  
 while stack and char < stack[-1] and counter[stack[-1]] > 0:  
 seen.remove(stack.pop())  
 stack.append(char)  
 seen.add(char)  
  
 return ''.join(stack)

22 일일온도 매일의 화씨 온도 리스트 T를 입력받아서, 더 따뜻한 날씨를 위해서는 며칠을 더 기다려야 하는 지를 출력하라

처음풀이 모르겠음…

풀이1 스택값 비교

빗물트래핑문제와 유사

현재 인덱스를 스택에 쌓아두다가, 상승하는 시점에서 현재온도와 스택에 쌓아둔 인덱스 지점의 온도차이 비교

class Solution:  
 def dailyTemperatures(self, temperatures: list[int]) -> list[int]:  
 answer = [0] \* len(temperatures)  
 stack = []  
   
 for i, cur in enumerate(temperatures):  
 while stack and cur > temperatures[stack[-1]]:  
 last = stack.pop()  
 answer[last] = i - last  
 stack.append(i)  
   
 return answer

큐 선입선출

데크, 우선순위 큐, BFS, 캐시 구현에 사용

되도록이면 데크 사용

23 큐를 이용한 스택 구현 큐를 이용해 스택을 구현하라

처음풀이 import collections  
   
   
 class MyStack:  
 def \_\_init\_\_(self):  
 self.dq = collections.deque()  
   
 def push(self, x: int) -> None:  
 self.dq.append(x)  
   
   
 def pop(self) -> int:  
 return self.dq.pop()  
   
 def top(self) -> int:  
 return self.dq[-1]  
   
 def empty(self) -> bool:  
 return 0 == len(self.dq)

풀이1 push() 할 때 큐를 이용해 재정렬

import collections  
   
   
 class MyStack:  
 def \_\_init\_\_(self):  
 self.dq = collections.deque()  
   
 def push(self, x: int) -> None:  
 self.dq.append(x)  
 for \_ in range(len(self.dq)-1):  
 self.dq.append(self.dq.popleft())  
   
 def pop(self) -> int:  
 return self.dq.popleft()  
   
 def top(self) -> int:  
 return self.dq[0]  
   
 def empty(self) -> bool:  
 return 0 == len(self.dq)

24 큐를 이용한 스택 구현 스택을 이용해 큐 연산을 지원하는 큐를 구현하라

처음풀이 import collections  
   
   
 class MyQueue:  
 def \_\_init\_\_(self):  
 self.dq = collections.deque()  
   
 def push(self, x: int) -> None:  
 self.dq.append(x)  
   
 def pop(self) -> int:  
 return self.dq.popleft()  
   
 def peek(self) -> int:  
 return self.dq[0]  
   
 def empty(self) -> bool:  
 return len(self.dq) == 0

풀이1 스택2개 사용

class MyQueue:  
 def \_\_init\_\_(self):  
 self.input = []  
 self.output = []  
  
 def push(self, x: int) -> None:  
 self.input.append(x)  
   
 def pop(self) -> int:  
 self.peek()  
 return self.output.pop()  
   
   
 def peek(self) -> int:  
 if not self.output:  
 while self.input:  
 self.output.append(self.input.pop())  
 return self.output[-1]  
   
 def empty(self) -> bool:  
 return self.input == [] and self.output == []

25 원형 큐 디자인 원형 큐를 디자인하라

처음풀이 import collections  
   
   
 class MyCircularQueue:  
 def \_\_init\_\_(self, k: int):  
 self.dq = collections.deque()  
 self.size = k  
   
 def enQueue(self, value: int) -> bool:  
 if len(self.dq) == self.size:  
 return False  
 self.dq.append(value)  
 return True  
   
 def deQueue(self) -> bool:  
 if len(self.dq) == 0:  
 return False  
 self.dq.popleft()  
 return True  
   
 def Front(self) -> int:  
 if len(self.dq) == 0:  
 return -1  
 return self.dq[0]  
   
 def Rear(self) -> int:  
 if len(self.dq) == 0:  
 return -1  
 return self.dq[-1]  
   
 def isEmpty(self) -> bool:  
 if len(self.dq) == 0:  
 return True  
 return False   
   
 def isFull(self) -> bool:  
 if len(self.dq) == self.size:  
 return True  
 return False

풀이1 배열을 이용한 풀이

원형 큐 = 환형 큐 = 링 버퍼

class MyCircularQueue:  
 def \_\_init\_\_(self, k: int):  
 self.q = [None] \* k  
 self.maxlen = k  
 self.p1 = 0  
 self.p2 = 0  
   
 def enQueue(self, value: int) -> bool:  
 if self.q[self.p2] is None:  
 self.q[self.p2] = value  
 self.p2 = (self.p2 + 1) % self.maxlen  
 return True  
 else:  
 return False  
   
 def deQueue(self) -> bool:  
 if self.q[self.p1] is None:  
 return False  
 else:  
 self.q[self.p1] = None  
 self.p1 = (self.p1+ 1) % self.maxlen  
 return True  
   
 def Front(self) -> int:  
 return -1 if self.q[self.p1] is None else self.q[self.p1]  
   
 def Rear(self) -> int:  
 return -1 if self.q[self.p2-1] is None else self.q[self.p2-1]  
   
 def isEmpty(self) -> bool:  
 return self.p1 == self.p2 and self.q[self.p1] is None  
   
 def isFull(self) -> bool:  
 return self.p1 == self.p2 and self.q[self.p1] is not None

10 데크, 우선순위 큐

데크 더블 엔디드 큐의 줄임말

양쪽 끝을 모두 추출할 수 있음

이중 연결리스트로 구현 가능

26 원형 데크 디자인 원형 데크를 디자인하라

처음 풀이 import collections  
   
   
 class MyCircularDeque:  
 def \_\_init\_\_(self, k: int):  
 self.dq = collections.deque()  
 self.size = k  
   
 def insertFront(self, value: int) -> bool:  
 if len(self.dq) == self.size:  
 return False  
 else:  
 self.dq.appendleft(value)  
 return True  
   
 def insertLast(self, value: int) -> bool:  
 if len(self.dq) == self.size:  
 return False  
 else:  
 self.dq.append(value)  
 return True  
   
 def deleteFront(self) -> bool:  
 if len(self.dq) == 0:  
 return False  
 else:  
 self.dq.popleft()  
 return True  
   
 def deleteLast(self) -> bool:  
 if len(self.dq) == 0:  
 return False  
 else:  
 self.dq.pop()  
 return True  
   
 def getFront(self) -> int:  
 if len(self.dq) == 0:  
 return -1  
 else:  
 return self.dq[0]  
   
 def getRear(self) -> int:  
 if len(self.dq) == 0:  
 return -1  
 else:  
 return self.dq[-1]  
   
 def isEmpty(self) -> bool:  
 if len(self.dq) == 0:  
 return True  
 else:  
 return False  
   
 def isFull(self) -> bool:  
 if len(self.dq) == self.size:  
 return True  
 else:  
 return False

풀이1 이중 연결 리스트를 이용한 데크 구현

class MyCircularDeque:  
 def \_\_init\_\_(self, k: int):  
 self.head, self.tail = ListNode(None)  
 self.k, self.len = k, 0  
 self.head.right, self.tail.left = self.tail, self.head  
   
 def \_add(self, node, new):  
 n = node.right  
 node.right = new  
 new.left, new.right = node, n  
 n.left = new  
   
 def \_del(self, node):  
 n = node.right.right  
 node.right = n  
 n.left = node  
   
 def insertFront(self, value: int) -> bool:  
 if self.len == self.k:  
 return False  
 self.len += 1  
 self.\_add(self.head, ListNode(value))  
 return True  
   
 def insertLast(self, value: int) -> bool:  
 if self.len == self.k  
 return False  
 self.len += 1  
 self.\_add(self.head, ListNode(value))  
 return True  
   
 def deleteFront(self) -> bool:  
 if self.len == 0:  
 return False  
 self.len -= 1  
 self.\_del(self.head)  
 return True  
   
 def deleteLast(self) -> bool:  
 if self.len == 0:  
 return False  
 self.len -=1  
 self.\_del(self.tail.left.left)  
 return True  
   
 def getFront(self) -> int:  
 return self.head.right.val if self.len else -1  
   
 def getRear(self) -> int:  
 return self tail.left.val if self.len else -1  
  
 def isEmpty(self) -> bool:  
 return self.len == 0  
   
 def isFull(self) -> bool:  
 return self.len == self.k

우선순위 큐 특정 조건에 따라 우선순위가 가장 높은 요소가 추출되는 자료형

최단 경로, 힙 자료구조와 연관이 있음

27 K개 정렬 리스트 병합 k개의 정렬된 리스트를 1개의 정렬된 리스트로 병합하라

처음풀이 모름

풀이1 우선순위 큐를 이용한 리스트 병합

heapq 모듈 사용

import heapq  
   
 class Solution:  
 def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:  
   
 root = result = ListNode(None)  
 heap = []  
   
 for i in range(len(lists)):  
 if lists[i]:  
 heapq.heappush(heap, (lists[i].val, i, lists[i]))  
   
 while heap:  
 node = heapq.heappop(heap)  
 idx = node[1]  
 result.next = node[2]  
   
 result = result.next  
 if result.next:  
 heapq.heappush(heap, (result.next.val, idx, result.next))  
 return root.next

11 해시 테이블

해시테이블 = 해시맵

대부분의 연산이 O(1)

해시 임의크기 입력값을 고정크기 출력값으로 매핑하는데 사용할 수 있는 함수

해싱 해싱함수를 사용하는 것

정보를 빠르게 저장하고, 검색하기 우해 사용하는 중요한 기법

성능좋은 해싱함수의 특징 1. 해시 함수 값 충돌의 최소화

2. 쉽고 빠른 연산

3. 해시 테이블 전체에 해시 값이 균일하게 분포

4. 사용할 키의 모든 정보를 이용하여 해싱

5. 해시 테이블 사용 효율이 높을 것

생일 문제 충돌 발생이 생각보다 흔함

비둘기집 원리 해시함수에서 충돌이 일어날 수 밖에 없음

로드 팩터 해시테이블에 저장된 데이터 개수 n을 버킷의 개수 k로 나눈 것

로드팩터 비율에 따라 해시함수 재작성, 해시테이블 크기 조정 결정

충돌이 발생할 경우 처리 방법 개별체이닝 충돌 발생 시, 충돌난 아이템들로 연결리스트로 만듬

1. 키의 해시 값 계산
2. 해시 값을 이용해 배열의 인덱스를 구함
3. 같은 인덱스가 있다면 연결리스트로 연결

오픈 어드레싱 충돌 시 탐사를 통해 빈 공간을 찾아나서는 방법 탐사방법-선형탐사

전체 슬롯의 개수 이상은 저장 불가

클러스터링(테이블에 연속된 데이터 그룹이 생기는 현상) 발생 가능

28 해시맵 디자인 해시맵을 디자인하라

처음풀이 class MyHashMap:  
 def \_\_init\_\_(self):  
 self.hashmap = {}  
  
 def put(self, key: int, value: int) -> None:  
 self.hashmap[key] = value  
  
 def get(self, key: int) -> int:  
 x = self.hashmap.get(key)  
 if x == None:  
 return -1  
 else:  
 return x  
  
 def remove(self, key: int) -> None:  
 if key not in self.hashmap.keys():  
 return -1  
 else:  
 self.hashmap.pop(key)

풀이1 개별 체이닝 방식을 이용한 해시 테이블 구현

import collections  
  
  
class ListNode:  
 def \_\_init\_\_(self, key = None, value = None):  
 self.key = key  
 self.value = value  
 self.next = None  
  
class MyHashMap:  
 def \_\_init\_\_(self):  
 self.size = 1000  
 self.table = collections.defaultdict(ListNode)  
  
 def put(self, key: int, value: int) -> None:  
 index = key % self.size  
 if self.table[index].value is None:  
 self.table[index] = ListNode(key, value)  
 return  
  
 p=self.table[index]  
 while p:  
 if p.key == key:  
 p.value = value  
 return  
 if p.next is None:  
 break  
 p = p.next  
 p.next = ListNode(key, value)  
  
 def get(self, key: int) -> int:  
 index = key % self.size  
 if self.table[index].value is None:  
 return -1  
  
 p = self.table[index]  
 while p:  
 if p.key == key:  
 return p.value  
 p = p.next  
 return -1  
  
 def remove(self, key: int) -> None:  
 index = key % self.size  
 if self.table[index].value is None:  
 return  
  
 p = self.table[index]  
 if p.key == key:  
 self.table[index] = ListNode() if p.next is None else p.next  
 return  
  
 prev = p  
 while p:  
 if p.key == key:  
 prev.next = p.next  
 return  
 prev, p = p, p.next

29 보석과 돌 J는 보석이며, S는 갖고 있는 돌이다. S에는 보석이 몇 개나 있을까? 대소문자는 구분한다

처음풀이 import collections  
  
class Solution:  
 def numJewelsInStones(self, jewels: str, stones: str) -> int:  
 counter = collections.Counter(stones)  
 res = 0  
 for i in jewels:  
 res += counter[i]  
 return res

풀이1 해시테이블을 이용한 풀이

S의 개수를 헤아리고, J의 각 요소를 키로하는 각 개수를 합산해 풀이

class Solution:  
 def numJewelsInStones(self, jewels: str, stones: str) -> int:  
 freqs = {}  
 count = 0  
  
 for char in stones:  
 if char not in freqs:  
 freqs[char] = 1  
 else:  
 freqs[char] += 1  
  
 for char in jewels:  
 if char in freqs:  
 count += freqs[char]  
 return count

풀이2 defaultdict를 이용한 비교 생략

풀이1에 defaultdict를 이용해 키가 존재하지 않을 때를 제외

class Solution:  
 def numJewelsInStones(self, jewels: str, stones: str) -> int:  
 freqs = collections.defaultdict(int)  
 count = 0  
  
 for char in stones:  
 freqs[char] += 1  
  
 for char in jewels:  
 count += freqs[char]  
  
 return count

풀이3 Counter로 계산 생략

import collections  
  
class Solution:  
 def numJewelsInStones(self, jewels: str, stones: str) -> int:  
 freqs = collections.Counter(stones)  
 count = 0  
  
 for char in jewels:  
 count += freqs[char]  
  
 return count

풀이4 파이썬다운 방식

리스트 컴프리헨션 사용

class Solution:  
 def numJewelsInStones(self, jewels: str, stones: str) -> int:  
 return sum(s in jewels for s in stones)

30 중복문자없는 가장 긴 부분 문자열 중복문자가 없는 가장 긴 부분문자열의 길이를 리턴하라

처음풀이 시간초과

import collections  
  
class Solution:  
 def lengthOfLongestSubstring(self, s: str) -> int:  
 res = 0  
 for i in range(0, len(s)):  
 for j in range(i, len(s)):  
 counter = collections.Counter(s[i:j+1])  
 if counter.most\_common()[0][1] >= 2:  
 break  
 else:  
 res = max(res, len(s[i:j+1]))  
 return res

풀이1 슬라이딩 윈도우와 투 포인터로 사이즈 조절

class Solution:  
 def lengthOfLongestSubstring(self, s: str) -> int:  
 used = {}  
 max\_length = start = 0  
 for index, char in enumerate(s):  
 if char in used and start <= used[char]:  
 start = used[char ] + 1  
 else:  
 max\_length = max(max\_length, index-start +1)  
 used[char] = index  
 return max\_length

31 상위 k 빈도 요소 상위 k번 이상 등장하는 요소를 추출해라 = 빈도 기준으로 상위 k번째까지의 요소를 추출해라

처음풀이 import collections  
  
class Solution:  
 def topKFrequent(self, nums: list[int], k: int) -> list[int]:  
 collection = collections.Counter(nums)  
 dict\_k = collection.most\_common(k)  
 res = []  
 for i in dict\_k:  
 res.append(i[0])  
 return res

풀이1 Counter를 이용한 음수 순 추출

요소를 키로, 빈도를 값으로 하는 해시테이블 제작 🡪 우선순위 큐를 통해 상위 K번만큼 추출

최소힙 파이썬에서는 최소힙만을 지원 🡪 최대힙으로 전환하려면 값에 -1을 곱해줘야함

import collections  
import heapq  
  
class Solution:  
 def topKFrequent(self, nums: list[int], k: int) -> list[int]:  
 freqs = collections.Counter(nums)  
 freqs\_heap = []  
  
 for f in freqs:  
 heapq.heappush(freqs\_heap, (-freqs[f], f))  
  
 topk = list()  
 for \_ in range(k):  
 topk.append(heapq.heappop(freqs\_heap)[1])  
  
 return topk

풀이2 파이썬다운 방식

most\_common() 사용

import collections  
  
class Solution:  
 def topKFrequent(self, nums: list[int], k: int) -> list[int]:  
 return list(zip(\*collections.Counter(nums).most\_common(k)))[0]

zip함수 2개 이상의 시퀀스를 짧은 시퀀스의 길이 기준으로 일대일 대응하는 새로운 튜플시퀀스를 만듬

ex) zip([a,b,c,d], [1,2,3,4,5]) 🡺 [(1,a)(2,b)(3,c)(4,d)]

\* 시퀀스 언패킹 연산자 시퀀스를 풀어해치는 연산자

일반적으로 쓸 때

ex) \*[(1, 2)(a, b)] 🡪 (1,2), (a, b)

시퀀스 패킹 연산자 시퀀스를 만드는 연산자

함수의 매개변수앞에 붙을 때

ex) f(\*param)

\*\* 키 값 언패킹 연산자 딕셔너리를 풀어해치는 연산자

4부 비선형 자료구조

12 그래프

그래프 객체의 일부 쌍들이 연관되어 있는 객체 집합구조

오일러 경로 정점

간선

모든 정점이 짝수 개의 차수를 갖는다면 모든 다리를 한 번씩만 건너서 도달하는 것이 성립

오일러 경로 = 한붓 그리기

모든 간선을 한 번씩만 방문하는 유한그래프

해밀턴 경로 각 정점을 한 번씩 방문하는 무향, 유향 그래프 경로

오일러 경로 간선 기준

해밀턴 경로 정점 기준

NP문제 = 최적알고리즘이 없음

그래프 순회 DFS 깊이 우선 탐색

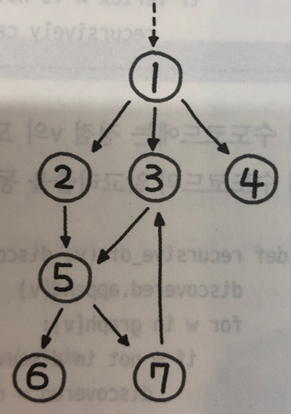
큐로 구현

BFS 너비 우선 탐색

DFS에 비해 더 많이 쓰임

스택, 재귀로 구현

예를 통한 설명 { 1:[2,3,4], 2:[5], 3:[5], 4:[], 5:[6,7], 6:[], 7:[3] }



DFS 구현 일반적으로 스택으로 구현

재귀를 이용하면 좀 더 간단하게 구현

재귀를 이용한 구현 def recursivev\_dfs(v, discovered = []):  
 discovered.append(v)  
 for w in graph[v]:  
 if w not in discovered:  
 discovered = recursivev\_dfs(w,discovered)  
 return discovered

스택을 이용한 구현 스택1에 모든 인접 간선 추가 🡪 도착점을 스택2에 추가

def iterative\_dfs(start\_v):  
 discoverd = []  
 stack = [start\_v]  
 while stack:  
 v = stack.pop()  
 if v not in discoverd:  
 discoverd.append(v)  
 for w in graph[v]:  
 stack.append(w)  
 return discoverd

재귀를 이용한 구현, 스택을 이용한 구현의 방문순서가 다름

BFS 구현 최단경로를 찾는 다익스트라 알고리즘에 유용하게 쓰임

큐를 이용한 반복 구조로 구현 def iterative\_bfs(start\_v):  
 discovered = [start\_v]  
 queue = [start\_v]  
 while queue:  
 v = queue.pop(0)  
 for w in graph[v]:  
 if w not in discovered:  
 discovered.append(w)  
 queue.append(w)  
 return discovered

재귀로 구현 불가

백트래킹 해결책에 대한 후보를 구축해 나아가다 가능성이 없다고 판단되는 즉시, 후보를 포기(백트랙)해 정답을 찾아가는 범용적인 알고리즘

가고 되돌아오고를 반복

운이 좋은 경우 시행착오를 덜 거치고 목적지에 도착, 최악의 경우 모든 경우를 다 거치고 도착(브루트포스)

제약충족문제 수많은 제약 조건을 충족하는 상태를 찾아내는 수학문제

백트래킹 기법이 필수적으로 사용됨

32 섬의 개수 1을 육지로 0을 물로 가정한 2D그리드 맵이 주어졌을 때, 섬의 개수를 계산하라 = 연결되어 있는 1의 덩어리 개수를 구하라

처음풀이 모르겠음…

풀이1 DFS로 그래프 탐색

입력값을 동서남북이 모두 연결된 그래프로 가정

방문한 정점은 0으로 변경 🡪 다시 방문할 일이 없게 함

class Solution:  
 def numIslands(self, grid: list[list[str]]) -> int:  
 def dfs(i, j):  
 if i < 0 or i >= len(grid) or j < 0 or j >= len(grid[0]) or grid[i][j] != '1':  
 return  
  
 grid[i][j] = 0  
 dfs(i+1, j)  
 dfs(i-1, j)  
 dfs(i, j+1)  
 dfs(i, j-1)  
  
 count = 0  
 for i in range(len(grid)):  
 for j in range(len(grid[0])):  
 if grid[i][j] == '1':  
 dfs(i, j)  
 count += 1  
 return count

33 전화 번호 문자 조합 2에서 9까지 숫자가 주어졌을 때 전화번호로 조합가능한 모든 문자를 출력하라

처음풀이 모르겠음

풀이1 모든 조합 탐색

class Solution:  
 def letterCombinations(self, digits: str) -> list[str]:  
 def dfs(index, path):  
 if len(path) == len(digits):  
 result.append(path)  
 return  
 for i in range(index, len(digits)):  
 for j in dic[digits[i]]:  
 dfs(i+1 ,path+j)  
  
 if not digits:  
 return []  
  
 dic = {"2":"abc", "3":"def", "4":"ghi", "5":"jkl", "6":"mno", "7":"pqrs", "8":"tuv", '9':"wxyz"}  
 result = []  
 dfs(0,"")  
  
 return result

34 순열 서로 다른 정수를 입력받아 가능한 모든 순열을 리턴하라

처음풀이 모르겠음

풀이1 DFS를 활용한 순열 생성

class Solution:  
 def permute(self, nums: list[int]) -> list[list[int]]:  
 results = []  
 prev\_elements = []  
  
 def dfs(elements):  
 if len(elements) == 0:  
 results.append(prev\_elements[:])  
  
 for e in elements:  
 next\_elements = elements[:]  
 next\_elements.remove(e)  
  
 prev\_elements.append(e)  
 dfs(next\_elements)  
 prev\_elements.pop()  
  
 dfs(nums)  
 return results

풀이2 itertools 모듈 사용

class Solution:  
 def permute(self, nums: list[int]) -> list[list[int]]:  
 return list(itertools.permutations(nums))

35 조합 전체 수 n을 입력받아 k의 조합을 리턴하라

처음풀이 class Solution(object):  
 def combine(self, n, k):  
 *"""* ***:type*** *n: int* ***:type*** *k: int* ***:rtype****: List[List[int]]  
 """* nums = list(range(1, n+1))  
 pick\_num = []  
 res = []  
  
 def dfs(element):  
 if len(pick\_num) == k:  
 res.append(pick\_num[:])  
  
 else:  
 for i in range(0, len(element)):  
 next\_num = element[i+1::]  
 pick\_num.append(element[i])  
 dfs(next\_num)  
 pick\_num.pop()  
  
 dfs(nums)  
 return res  
  
  
sol = Solution()  
print(sol.combine(4,2))

풀이1 DFS로 k개 조합 생성

class Solution:  
 def combine(self, n: int, k: int) -> list[list[int]]:  
 results = []  
   
 def dfs(elements, start, k):  
 if k == 0:  
 results.append(elements[:])  
 return  
   
 for i in range(start, n+1):  
 elements.append(i)  
 dfs(elements, i+1, k-1)  
 elements.pop()  
   
 dfs([], 1, k)  
 return results

풀이2 itertools 모듈 사용

import itertools  
  
  
class Solution:  
 def combine(self, n: int, k: int) -> list[list[int]]:  
 return list(itertools.combinations(range(1, n+1), k))

36 조합의 합 숫자 조합 candidates를 조합하여 합이 target이 되는 원소를 나열하라. 각 원소는 중복으로 나열 가능하다

처음 풀이 class Solution(object):  
 def combinationSum(self, candidates, target):  
 res = []  
  
 def dfs(pick\_num, element, target\_changed):  
 if target\_changed == 0:  
 res.append(pick\_num[:])  
 return  
 elif target\_changed < 0:  
 return  
 else:  
 for i in range(0, len(element)):  
 if target\_changed >= element[i]:  
 pick\_num.append(element[i])  
 dfs(pick\_num, element[i::], target\_changed-element[i])  
 pick\_num.pop()  
 dfs([],candidates, target)  
 return res

풀이1 DFS로 중복 조합 그래프 탐색

class Solution:  
 def combinationSum(self, candidates: list[int], target: int) -> list[list[int]]:  
 result = []  
   
 def dfs(csum, index, path):  
 if csum < 0:  
 return  
 if csum == 0:  
 result.append(path)  
 return  
   
 for i in range(index, len(candidates)):  
 dfs(csum - candidates[i], i, path + [candidates[i]])  
   
 dfs(target, 0, [])  
 return result

37 부분집합 모든 부분 집합을 리턴하라

처음 풀이 모르겟음

풀이1 트리의 모든 DFS 결과

class Solution:  
 def subsets(self, nums: list[int]) -> list[list[int]]:  
 res = []  
   
 def dfs(index, path):  
 res.append(path)  
 for i in range(index, len(nums)):  
 dfs(i+1, path+[nums[i]])  
   
 dfs(0, [])  
 return res

38 일정 재구성 [from, to]로 구성된 항공권 목록을 이용해 JFK에서 출발하는 여행일정을 구성하라, 여러 일정이 있는 경우 사전 어휘순으로 방문한다

처음풀이 모르겠음..

풀이1 DFS로 일정 그래프 구성

import collections  
   
   
 class Solution:  
 def findItinerary(self, tickets: list[list[str]]) -> list[str]:  
 graph = collections.defaultdict(list)  
 for a, b in sorted(tickets):  
 graph[a].append(b)  
   
 route = []  
 def dfs(a):  
 while graph[a]:  
 dfs(graph[a].pop(0))  
 route.append(a)  
   
 dfs('JFK')  
 return route[::-1]

풀이2 스택연산으로 큐연산 최적화

import collections  
   
   
 class Solution:  
 def findItinerary(self, tickets: list[list[str]]) -> list[str]:  
 graph = collections.defaultdict(list)  
 for a, b in sorted(tickets, reverse=True):  
 graph[a].append(b)  
   
 route = []  
 def dfs(a):  
 while graph[a]:  
 dfs(graph[a].pop())  
 route.append(a)  
   
 dfs('JFK')  
 return route[::-1]

풀이3 일정그래프 반복

반복풀이보다 재귀에 더 높은 점수

import collections  
   
   
 class Solution:  
 def findItinerary(self, tickets: list[list[str]]) -> list[str]:  
 graph = collections.defaultdict(list)  
 for a, b in sorted(tickets):  
 graph[a].append(b)  
   
 route, stack = [], ['JFK']  
 while stack:  
 while graph[stack[-1]]:  
 stack.append(graph[stack[-1]].pop(0))  
 route.append(stack.pop())  
   
 return route[::-1]

39 코스 스케줄 0을 완료하기 위해서 1을 끝내야 한다는 것을 [0,1]쌍으로 표현하는 n개의 코스가 있다. 코스 개수 n과 이 쌍들을 입력으로 받았을 때 모든 코스가 완료가능한지 판별하라

처음풀이 모르겠음

풀이1 DFS로 순환 구조 판별

import collections  
   
 class Solution:  
 def canFinish(self, numCourses: int, prerequisites: list[list[int]]) -> bool:  
 graph = collections.defaultdict(list)  
 for x, y in prerequisites:  
 graph[x].append(y)  
   
 traced = set()  
   
 def dfs(i):  
 if i in traced:  
 return False   
 traced.add(i)  
   
 for y in graph[i]:   
 if not dfs(y):  
 return False  
   
 traced.remove(i)  
 return True  
   
 for x in list(graph):  
 if not dfs(x):  
 return False  
   
 return True

풀이2 가지치기를 이용한 최적화

import collections  
   
 class Solution:  
 def canFinish(self, numCourses: int, prerequisites: list[list[int]]) -> bool:  
 graph = collections.defaultdict(list)  
 for x, y in prerequisites:  
 graph[x].append(y)  
   
 traced = set()  
 visited = set()  
   
 def dfs(i):  
 if i in traced:  
 return False  
 if i in visited:  
 return True  
   
 traced.add(i)  
 for y in graph[i]:  
 if not dfs(y):  
 return False  
   
 traced.remove(i)  
 visited.add(i)  
   
 return True  
   
 for x in list(graph):  
 if not dfs(x):  
 return False  
   
 return True

13 최단 경로 문제

최단 경로 문제 각 간선의 가중치 합이 최소가 되는 두 정점 사이의 경로 찾기

다익스트라 알고리즘 항상 노드 주변의 최단 경로만을 택함

그리디 알고리즘

BFS사용

40 네트워크 딜레이 타임 k부터 출발해 모든 노드가 신호를 받을 수 있는 시간을 계산하라. 불가능할 경우 -1를 리턴한다. 입력값 (u, v, w)는 각각 출발지, 도착지, 소요시간으로 구성된다

전체 노드의 개수는 N으로 입력받는다

처음풀이 모르겠음..

풀이1 다익스트라 알고리즘 구현

두 가지 상황 판별 1. 모든 노드가 신호를 받는 데 걸리는 시간

2. 모든 노드에 도달할 수 있는지 여부

import collections  
 import heapq  
   
   
 class Solution:  
 def networkDelayTime(self, times: list[list[int]], n: int, k: int) -> int:  
 graph = collections.defaultdict(list)  
 for u, v, w in times:  
 graph[u].append((v,w))  
   
 Q = [(0,k)]  
 dist = collections.defaultdict(int)  
   
 while Q:  
 time, node = heapq.heappop(Q)  
 if node not in dist:  
 dist[node] = time  
 for v, w in graph[node]:  
 alt = time + w  
 heapq.heappush(Q, (alt ,v))  
   
 if len(dist) == n:  
 return max(dist.values())  
 return -1

41 k경유지 내 가장 저렴한 항공권 시작점에서 도착점까지의 가장 저렴한 가격을 계산하되, K개 경유지 이내에 도착하는 가격을 리턴하라. 경로가 존재하지 않을 경우 -1을 리턴한다.

처음풀이 모르겟음…

풀이1 다익스트라 알고리즘 응용

import collections  
 import heapq  
   
   
 class Solution:  
 def findCheapestPrice(self, n: int, flights: list[list[int]], src: int, dst: int, k: int) -> int:  
 graph = collections.defaultdict(list)  
   
 for u, v, w in flights:  
 graph[u].append((v,w))  
   
 Q = [(0, src, k)]  
   
 while Q:  
 price, node, k = heapq.heappop(Q)  
 if node == dst:  
 return price  
 if k >= 0:  
 for v, w in graph[node]:  
 alt = price + w  
 heapq.heappush(Q, (alt, v, k-1))  
 return -1

14 트리

트리 계층형 트리 구조를 시뮬레이션하는 추상 자료형

루트값과 부모-자식 관계로 구성

서로 연결된 노드의 집합

트리의 속성 재귀로 정의된 자기참조 자료구조

자식노드 = 서브트리의 노드

간선은 항상 단 방향

트리의 각 명칭 루트 -----간선----- 자식노드

차수 자식노드의 개수

크기 자신을 포함한 모든 자식노드의 개수

높이 현재 위치부터 리프까지의 거리

깊이 루트에서부터 현재 노드까지의 거리

레벨 루트노드, 0부터 리프노드까지 1씩 증가

그래프 vs 트리 그래프 순환 구조를 가짐

트리 순환구조를 갖지 않는 그래프

* 트리는 순환구조를 갖지 않는 그래프입니다

이진트리 m-ary 트리 m개 이하의 자식을 갖는 트리

다진트리

이진트리 2개 이하의 자시을 갖는 트리

2-ary 트리

정 이진트리 모든 노드가 0개 또는 2개의 자식을 가짐

완전 이진 트리 마지막 레벨을 제외하고 모든 레벨이 완전히 채워져있는 트리

포화 이진 트리 모든 노드가 자식을 2개씩 갖는 트리

42 이진 트리의 최대 깊이 이진 트리의 최대 깊이를 구하라

처음풀이 class Solution(object):  
 def maxDepth(self, root):  
 if root == None:  
 return 0  
 return max(self.maxDepth(root.left)+1, self.maxDepth(root.right)+1)

풀이1 반복구조로 BFS 풀이

import collections  
  
  
class Solution(object):  
 def maxDepth(self, root):  
 if root is None:  
 return 0  
  
 queue = collections.deque([root])  
 depth = 0  
  
 while queue:  
 depth += 1  
 for \_ in range(len(queue)):  
 cur\_root = queue.popleft()  
 if cur\_root.left:  
 queue.append(cur\_root.left)  
 if cur\_root.right:  
 queue.append(cur\_root.right)  
  
 return depth

43 이진 트리의 직경 이진트리에서 두 노드 간 가장 긴 경로의 길이를 출력하라

처음풀이 모르겠음…

풀이1 상태값 누적 트리 DFS

class Solution(object):  
 longest = 0  
 def diameterOfBinaryTree(self, root):  
 def dfs(node):  
 if not node:  
 return -1  
 left = dfs(node.left)  
 right = dfs(node.right)  
  
 self.longest = max(self.longest, left + right + 2)  
 return max(left, right) + 1  
 dfs(root)  
 return self.longest

43 이진 트리의 직경 동일한 값을 지닌 가장 긴 경로를 찾아라

처음풀이 모르겠음…

풀이1 상태값 거리 계산 DFS

class Solution(object):  
 result = 0  
 def longestUnivaluePath(self, root):  
 def dfs(node):  
 if node is None:  
 return 0  
 left = dfs(node.left)  
 right = dfs(node.right)  
  
 if node.left and node.left.val == node.val:  
 left += 1  
 else:  
 left = 0  
 if node.right and node.right.val == node.val:  
 right += 1  
 else:  
 right = 0  
  
 self.result = max(self.result, left + right)  
 return max(left, right)  
  
 dfs(root)  
 return self.result

45 이진 트리 반전 중앙을 기준으로 이진트리를 반전시키기

처음풀이 class Solution(object):  
 def invertTree(self, root):  
 def dfs(root):  
 if root is None:  
 return  
 root.left, root.right = root.right, root.left  
 dfs(root.left)  
 dfs(root.right)  
 return root  
  
 return dfs(root)

풀이1 파이썬다운 방식

class Solution(object):  
 def invertTree(self, root):  
 if root:  
 root.left, root.right = self.invertTree(root.right), self.invertTree(root.left)  
 return root  
 return None

풀이2 반복구조로 BFS

import collections  
  
class Solution(object):  
 def invertTree(self, root):  
 queue = collections.deque([root])  
  
 while queue:  
 node = queue.popleft()  
 if node:  
 node.left, node.right = node.right, node.left  
 queue.append(node.left)  
 queue.append(node.right)  
 return root

풀이3 반복 구조로 DFS

import collections  
  
class Solution(object):  
 def invertTree(self, root):  
 stack = collections.deque([root])  
  
 while stack:  
 node = stack.pop()  
 if node:  
 node.left, node.right = node.right, node.left  
 stack.append(node.left)  
 stack.append(node.right)  
 return root

풀이4 반복 구조로 DFS 후위 순회

import collections  
  
class Solution(object):  
 def invertTree(self, root):  
 stack = collections.deque([root])  
  
 while stack:  
 node = stack.pop()  
 if node:  
 stack.append(node.left)  
 stack.append(node.right)  
  
 node.left, node.right = node.right, node.left  
 return root

46 두 이진 트리 병합 두 이진 트리를 병합하라. 중복되는 노드는 값을 합산한다.

처음풀이 class Solution:  
 def mergeTrees(self, root1: Optional[TreeNode], root2: Optional[TreeNode]) -> Optional[TreeNode]:  
 def dfs(node1, node2):  
 if node1 is None and node2 is None:  
 return None  
 if node1 is None and node2 is not None:  
 node2.left = dfs(None, node2.left)  
 node2.right = dfs(None, node2.right)  
 return node2  
 elif node1 is not None and node2 is None:  
 node2 = TreeNode(node1.val)  
 node2.left = dfs(node1.left, None)  
 node2.right = dfs(node1.right, None)  
 return node2  
 else:  
 node2.left = dfs(node1.left, node2.left)  
 node2.right = dfs(node1.right, node2.right)  
 node2.val = node2.val + node1.val  
 return node2  
   
 root2 = dfs(root1, root2)  
 return root2  
 풀이1 재귀 탐색

class Solution:  
 def mergeTrees(self, root1: Optional[TreeNode], root2: Optional[TreeNode]) -> Optional[TreeNode]:  
 if root1 and root2:  
 node = TreeNode(root1.val + root2.val)  
 node.left = self.mergeTrees(root1.left, root2.left)  
 node.right = self.mergeTrees(root1.right, root2.right)  
 return node  
 else:  
 return root1 or root2

47 이진 트리 직렬화 & 역직렬화 이진트리를 배열로 직렬화하고 반대로 역직렬화하는 기능을 구현하라.

처음풀이

풀이1 import collections  
   
   
 class Codec:  
 def serialize(self, root):  
 queue = collections.deque([root])  
 result = ['#']  
   
 while queue:  
 node = queue.popleft()  
 if node:  
 queue.append(node.left)  
 queue.append(node.right)  
 result.append(str(node.val))  
 else:  
 result.append('#')  
   
 def deserialize(self, data):  
 *"""Decodes your encoded data to tree.*  ***:type*** *data: str* ***:rtype****: TreeNode*  *"""* if data == '# #':  
 return None  
   
 nodes = data.split()  
   
 root = TreeNode(int(nodes[1]))  
 queue = collections.dequeu([root])  
 index = 2  
 while queue:  
 node = queue.popleft()  
 if node[index] is not '#':  
 node.left = TreeNode(int(nodes[index]))  
 queue.append(node.left)  
 index += 1  
   
 if nodes[index] is not '#':  
 node.right = TreeNode(int(nodes[index]))  
 queue.append(node.right)  
 index += 1  
 return root

48 균형 이진 트리 이진트리가 높이 균형인지 판단하라

처음풀이 import collections  
   
   
 class Solution:  
 def isBalanced(self, root: Optional[TreeNode]) -> bool:  
 def getheight(node):  
 if node is None:  
 return 0  
 else:  
 return max(getheight(node.left), getheight(node.right)) + 1  
   
 dq = collections.deque([root])  
 while len(dq) != 0:  
 node = dq.popleft()  
 if node is None:  
 continue  
 dq.append(node.left)  
 dq.append(node.right)  
 diff = getheight(node.left) - getheight(node.right)   
 if diff < -1 or diff > 1:  
 return False  
 return True

풀이1 재귀구조로 높이 차이 계산

class Solution:  
 def isBalanced(self, root: Optional[TreeNode]) -> bool:  
 def check(root):  
 if not root:  
 return 0  
   
 left = check(root.left)  
 right = check(root.right)  
   
 if left == -1 or right == -1or abs(left-right) > 1:  
 return -1  
 return max(left, right) + 1  
   
 return check(root) != -1

49 최소 높이 트리 노드 개수와 무방향 그래프를 입력받아 트리가 최소 높이가 되는 루트의 목록을 리턴해라

처음풀이 모르겠음…

풀이1 단계별 리프 노드 제거

import collections  
  
  
class Solution(object):  
 def findMinHeightTrees(self, n, edges):  
 if n <= 1:  
 return [0]  
  
 graph = collections.defaultdict(list)  
 for i, j in edges:  
 graph[i].append(j)  
 graph[j].append(i)  
  
 leaves = []  
 for i in range(n+1):  
 if len(graph[i]) == 1:  
 leaves.append(i)  
  
 while n > 2:  
 n -= len(leaves)  
 new\_leaves = []  
 for leaf in leaves:  
 neighbor = graph[leaf].pop()  
 graph[neighbor].remove(leaf)  
  
 if len(graph[neighbor]) == 1:  
 new\_leaves.append(neighbor)  
 leaves = new\_leaves  
  
 return leaves

이진 탐색 트리 노드의 왼쪽 서브트리는 노드보다 작은 값으로 이뤄짐

노드의 오른쪽 서브트리는 노드보다 큰 값으로 이뤄짐

탐색시 시간복잡도 O(log n)

균형이 깨진 최악의 경우, 시간복잡도 O(n)

자가 균형 이진 탐색 트리 삽입, 삭제 시 자동으로 높이(균형)을 작게 유지하는 노드 기반의 이진 탐색 트리

ex) AVL트리, 레드-블랙 트리

50 정렬된 배열의 이진탐색트리 변환 오름차순로 정렬된 배열을 높이 균형 이진 탐색 트리로 변환해라

처음풀이 class Solution(object):  
 def sortedArrayToBST(self, nums):  
  
 def dfs(li):  
 if len(li) == 0:  
 return None  
 node = TreeNode(li[len(li)//2], dfs(li[0:len(li)//2]), dfs(li[len(li)//2+1: len(li)]))  
 return node  
  
 return dfs(nums)

풀이1 이진검색 결과로 트리 구성

class Solution(object):  
 def sortedArrayToBST(self, nums):  
 if not nums:  
 return None  
  
 mid = len(nums)//2  
 node = TreeNode(nums[mid])  
 node.left = self.sortedArrayToBST(nums[:mid])  
 node.right = self.sortedArrayToBST(nums[mid+1:])  
  
 return node

51 이진탐색트리를 더 큰 수 합계 트리로 BST의 각 노드를 현재값보다 더 큰 값을 가진 모든 노드의 합으로 만들어라

처음풀이 class Solution(object):  
 ac = 0  
 def bstToGst(self, root):  
 def dfs(node):  
 if node is None:  
 return  
 dfs(node.right)  
 num = node.val  
 node.val += self.ac  
 self.ac += num  
 dfs(node.left)  
 return node  
  
 return dfs(root)

풀이1 중위순회로 노드 값 누적

class Solution(object):  
 val = 0  
 def bstToGst(self, root):  
 if root:  
 self.bstToGst(root.right)  
 self.val += root.val  
 root.val = self.val  
 self.bstToGst(root.left)  
 return root

52 이진탐색트리 합의 범위 이진탐색트리가 주어졌을 때, L이상 R이하의 값을 가진 노드의 합을 구하라

처음풀이 class Solution(object):  
 val = 0  
 def rangeSumBST(self, root, low, high):  
 if root is None:  
 return   
 if root.val < low:  
 self.rangeSumBST(root.right, low, high)  
 elif root.val > high:  
 self.rangeSumBST(root.left, low, high)  
 else:  
 self.val += root.val  
 self.rangeSumBST(root.right, low, high)  
 self.rangeSumBST(root.left, low, high)  
 return self.val

풀이1 재귀구조 DFS로 브루트포스 탐색

class Solution(object):  
 val = 0  
 def rangeSumBST(self, root, low, high):  
 if not root:  
 return 0  
 return (root.val if low <= root.val <=high else 0) + self.rangeSumBST(root.left, low, high) + self.rangeSumBST(root.right, low, high)

풀이2 DFS 가지치기로 필요한 노드 탐색

class Solution(object):  
 val = 0  
 def rangeSumBST(self, root, low, high):  
 def dfs(node):  
 if not node:  
 return 0  
 if node.val < low:  
 return dfs(node.right)  
 elif node.val > high:  
 return dfs(node.left)  
 return node.val + dfs(node.left) + dfs(node.right)  
  
 return dfs(root)

풀이3 반복 구조 DFS로 필요한 노드 탐색

class Solution(object):  
 val = 0  
 def rangeSumBST(self, root, low, high):  
 stack, sum = [root], 0  
 while stack:  
 node = stack.pop()  
 if node:  
 if node.val > low:  
 stack.append(node.left)  
 if node.val < high:  
 stack.append(node.right)  
 if low <= node.val <= high:  
 sum += node.val  
 return sum

풀이4 반복구조 DFS로 필요한 노드 탐색

class Solution(object):  
 val = 0  
 def rangeSumBST(self, root, low, high):  
 stack, sum = [root], 0  
 while stack:  
 node = stack.pop(0)  
 if node:  
 if node.val > low:  
 stack.append(node.left)  
 if node.val < high:  
 stack.append(node.right)  
 if low <= node.val <= high:  
 sum += node.val  
 return sum

53 이진탐색트리 노드 간 최소거리 두 노드 간 값의 차이가 가장 작은 노드의 값의 차이를 출력하라

처음풀이 import sys  
  
class Solution(object):  
 diffval = sys.maxsize  
 def minDiffInBST(self, root):  
 if root is None:  
 return  
 self.minDiffInBST(root.left)  
 tmp = root.left  
 while tmp is not None:  
 if self.diffval > abs(root.val-tmp.val):  
 self.diffval = abs(root.val-tmp.val)  
 tmp = tmp.right  
 tmp = root.right  
 while tmp is not None:  
 if self.diffval > abs(root.val-tmp.val):  
 self.diffval = abs(root.val-tmp.val)  
 tmp = tmp.left  
 self.minDiffInBST(root.right)  
  
 return self.diffval

풀이1 재귀 구조로 중위 순회

import sys  
  
class Solution(object):  
 prev = -sys.maxsize  
 result = sys.maxsize  
 def minDiffInBST(self, root):  
 if root.left:  
 self.minDiffInBST(root.left)  
  
 self.result = min(self.result, root.val - self.prev)  
 self.prev = root.val  
  
 if root.right:  
 self.minDiffInBST(root.right)  
  
 return self.result

풀이2 반복 구조로 중위 순회

import sys  
  
class Solution(object):  
 def minDiffInBST(self, root):  
 prev = -sys.maxsize  
 result = sys.maxsize  
  
 stack = []  
 node = root  
  
 while stack or node:  
 while node:  
 stack.append(node)  
 node = node.left  
  
 node = stack.pop()  
 result = min(result, node.val - prev)  
 prev = node.val  
  
 node = node.right  
  
 return result

트리순회 그래프 순회의 한 형태로 트리에서 각 노드를 정확히 한번 방문하는 과정

전위순회 N 🡪 L 🡪 R

중위순회 L 🡪 N 🡪 R

후위순회 L 🡪 R 🡪 N

54 전위, 중위순회 결과로 이진트리 구축 트리의 전위, 중위 순회 결과를 입력으로 받아 이진트리룰 구축해라

처음풀이 class Solution:  
 def buildTree(self, preorder: list[int], inorder: list[int]) -> Optional[TreeNode]:  
 if len(preorder) == 0 or len(inorder) == 0:  
 return None  
 root = TreeNode(preorder[0], None, None)  
 root\_index = inorder.index(preorder[0])  
 preorder.pop(0)  
 root.left = self.buildTree(preorder, inorder[:root\_index])  
 root.right = self.buildTree(preorder, inorder[root\_index+1:])  
 return root

풀이1 전위순회 결과로 중위순회 분할정복

class Solution:  
 def buildTree(self, preorder: list[int], inorder: list[int]) -> Optional[TreeNode]:  
 if inorder:  
 index = inorder.index(preorder.pop(0))  
   
 node = TreeNode(inorder[index])  
 node.left = self.buildTree(preorder, inorder[0:index])  
 node.right = self.buildTree(preorder, inorder[index+1:])  
   
 return node

15 힙

힙 부모가 항상 자식보다 작거나 같은(최소힙) 거의 완전한 트리

heapq가 최소힙으로 구현되어 있음

힙은 정렬되어 있는 구조가 아님 부모, 자식관의 관계만 정의

힙은 배열로 표현 0번 인덱스는 사용하지 않음

힙의 용도 힙 정렬

프림 알고리즘 최소 신장 트리 구현

중앙의 근사값을 빠르게 구함

힙 연산 삽입 1. 요소를 가장 하위 레벨의 최대한 왼쪽으로 삽입한다. = 배열의 경우, 가장 마지막에 삽입한다

2. 부모값과 비교해 값이 더 작은 경우 위치를 변경한다 (부모값은 길이의 //2)

3. 계속해서 부모값과 비교해 위치를 변경한다.

추출 삽입의 반대

1. 인덱스 1의 요소, 루트노드 추출
2. 가장 마지막의 요소를 루트노드로 올림
3. 루트노드와 자식노드와 비교해 자식보다 크면 내려감

부모노드 구하기 부모노드 버림((i-1)/2)

왼쪽 자식노드 2i

오른쪽 자식노드 2i+1

구현 class BinaryHeap(object):  
 def \_\_init\_\_(self):  
 self.items = [None]  
 def \_\_len\_\_(self):  
 return len(self.items) -1  
   
 def \_percolate\_up(self):  
 i = len(self)  
 parent = i//2  
 while parent > 0:  
 if self.items[i] < self.items[parent]:  
 self.items[parent], self.items[i] = self.items[i], self.items[parent]  
 i = parent  
 parent = i //2  
   
 def insert(self, k):  
 self.items.append(k)  
 self.\_percolate\_up()  
   
 def \_percolate\_down(self, idx):  
 left = idx \* 2  
 right = idx \* 2 + 1  
 smallest = idx  
   
 if left <= len(self) and self.items[left] < self.items[smallest]:  
 smallest = left  
   
 if right <= len(self) and self.items[right] < self.items[smallest]:  
 smallest = right  
   
 if smallest != idx:  
 self.items[idx], self.items[smallest] = self.items[smallest], self.items[idx]  
 self.\_percolate\_down(smallest)  
   
 def extract(self):  
 extracted = self.items[1]  
 self.items[1]= self.items[len(self)]  
 self.items.pop()  
 self.\_percolate\_down(1)  
 return extracted

55 배열의 k번째 큰 요소 정렬되지 않은 배열에서 k번째 큰 요소를 추출하라

처음풀이1 class Solution:  
 def findKthLargest(self, nums: list[int], k: int) -> int:  
 nums.sort(reverse=True)  
 return nums[k-1]

풀이1 heapq 모듈 이용

import heapq  
   
   
 class Solution:  
 def findKthLargest(self, nums: list[int], k: int) -> int:  
 heap = list()  
   
 for n in nums:  
 heapq.heappush(heap, -n)  
   
 for \_ in range(1, k):  
 heapq.heappop(heap)  
   
 return -heapq.heappop(heap)

풀이2 heapq 모듈의 heapify 이용

import heapq  
   
   
 class Solution:  
 def findKthLargest(self, nums: list[int], k: int) -> int:  
 heapq.heapify(nums)  
   
 for \_ in range(len(nums)-k):  
 heapq.heappop(nums)  
   
 return heapq.heappop(nums)

풀이3 heapq 모듈의 nlargest 이용

import heapq  
   
   
 class Solution:  
 def findKthLargest(self, nums: list[int], k: int) -> int:  
 return heapq.nlargest(k, nums)[-1]

풀이4 정렬을 이용한 풀이

class Solution:  
 def findKthLargest(self, nums: list[int], k: int) -> int:  
 return sorted(nums, reverse = True)[k-1]

16 트라이

트라이 검색 트리의 일종

일반적으로 키가 문자열인 동적, 배열 또는 연관 배열을 저장하는 데 사용하는 정렬된 트리 자료구조

다진트리

56 트라이 구현 트라이의 insert, search, startsWith 메소드를 구현하라

처음풀이 모르겠음…

풀이1 딕셔너리를 이용한 간결한 트라이 구현

import collections  
   
 class TrieNode:  
 def \_\_init\_\_(self):  
 self.word = False  
 self.children = collections.defaultdict(TrieNode)  
   
   
 class Trie:  
 def \_\_init\_\_(self):  
 self.root = TrieNode()  
   
 def insert(self, word: str) -> None:  
 node = self.root  
 for char in word:  
 node = node.children[char]  
 node.word = True  
   
   
 def search(self, word: str) -> bool:  
 node = self.root  
 for char in word:  
 if char not in node.children:  
 return False  
 node = node.children[char]  
 return node.word  
   
 def startsWith(self, prefix: str) -> bool:  
 node = self.root  
 for char in prefix:  
 if char not in node.children:  
 return False  
 node = node.children[char]  
 return True

57 펠린드롬 페어 단어리스트에서 word[i] + word[j]가 펠린드롬이 되는 모든 인덱스 조합(i,j)를 구하라

처음풀이 모르겠음…

풀이1 팰린드롬을 브루트포스로 계산

class Solution(object):  
 def palindromePairs(self, words):  
 def is\_palindrome(word):  
 return word == word[::-1]  
  
 output = []  
 for i, word1 in enumerate(words):  
 for j,word2 in enumerate(words):  
 if i == j:  
 continue  
 if is\_palindrome(word1 + word2):  
 output.append([i, j])  
 return output

풀이2 트라이 구현

import collections  
  
  
class TrieNode:  
 def \_\_init\_\_(self):  
 self.children = collections.defaultdict(TrieNode)  
 self.word\_id = -1  
 self.palindrome\_word\_ids = []  
  
  
class Trie:  
 def \_\_init\_\_(self):  
 self.root = TrieNode()  
  
 @staticmethod  
 def is\_palindrome(word):  
 return word[::] == word[::-1]  
  
 def insert(self, index, word):  
 node = self.root  
 for i, char in enumerate(reversed(word)):  
 if self.is\_palindrome(word[0:len(word) - i]):  
 node.palindrome\_word\_ids.append(index)  
 node = node.children[char]  
 node.word\_id = index  
  
 def search(self, index, word):  
 result = []  
 node = self.root  
  
 while word:  
 if node.word\_id >= 0:  
 if self.is\_palindrome(word):  
 result.append([index, node.word\_id])  
 if not word[0] in node.children:  
 return result  
 node = node.children[word[0]]  
 word = word[1:]  
  
 if node.word\_id >= 0 and node.word\_id != index:  
 result.append([index, node.word\_id])  
  
 for palindrome\_word\_id in node.palindrome\_word\_ids:  
 result.append([index, palindrome\_word\_id])  
  
 return result  
  
  
class Solution(object):  
 def palindromePairs(self, words):  
 tries = Trie()  
  
 for i, word in enumerate(words):  
 tries.insert(i, word)  
  
 results = []  
 for i, word in enumerate(words):  
 results.extend(tries.search(i, word))  
  
 return results

5부 알고리즘

17 정렬

정렬 목록의 요소를 특정 순서대로 넣는 알고리즘

방식 숫자식 순서, 사전식 순서

버블 정렬 내가 늘상 하는 정렬

def bubblesort(a):  
 for i in range(1, len(a)):  
 for j in range(i, len(a)-1):  
 if a[j] > a[j+1]:  
 a[j], a[j+1] = a[j+1], a[j]

병합 정렬 분할 정복

최선과 최악 모두 O(nlog n)

안정정렬

더 이상 쪼갤 수 없을 때까지 계속해서 분할한 후, 분할이 끝나면 정렬하면서 정복

퀵 정렬 피벗을 기준으로 좌우로 나눔 🡪 파티션 교환 정렬이라고도 불림

피벗보다 작으면 왼쪽, 피벗보다 크면 오른쪽

파티션을 나누고 각각 재귀호출

슈도코드 def QuickSort(a, lo, hi)  
 def partition(lo, hi):  
 pivot = a[hi]  
 left = lo  
 for right in range(lo, hi):  
 if a[right] < pivot:  
 a[left], a[right] = a[right], a[left]  
 left += 1  
 a[left], a[hi] = a[hi], a[left]  
 return left  
  
 if lo < hi:  
 pivot = partition(a, lo, hi)  
 QuickSort(a, lo, pivot - 1)  
 QuickSort(a, pivot, hi)

최악 O(n2)

안정 정렬 vs 불안정 정렬 안정정렬 중복된 값을 입력순서와 동일하게 정렬

58 리스트 정렬 연결리스트를 O(nlog n)에 정렬하라

처음풀이 class Solution(object):  
 node\_list = []  
 def sortList(self, head):  
 p = head  
  
 while p is not None:  
 self.node\_list.append(p.val)  
 p = p.next  
  
 self.node\_list.sort()  
  
 p = head  
 while p is not None:  
 p.val = self.node\_list.pop(0)  
 p = p.next  
  
 return head

풀이1 병합정렬

# Definition for singly-linked list.  
# class ListNode(object):  
# def \_\_init\_\_(self, val=0, next=None):  
# self.val = val  
# self.next = next  
class Solution(object):  
 def mergeTwoLists(self, l1, l2):  
 if l1 and l2:  
 if l1.val > l2.val:  
 l1, l2 = l2, l1  
 l1.next = self.mergeTwoLists(l1.next, l2)  
 return l1 or l2  
  
 def sortList(self, head):  
 if not (head and head.next):  
 return head  
  
 half, slow, fast = None, head, head  
 while fast and fast.next:  
 half, slow, fast = slow, slow.next, fast.next.next  
 half.next = None  
  
 l1 = self.sortList(head)  
 l2 = self.sortList(slow)  
  
 return self.mergeTwoLists(l1, l2)

풀이2 퀵 정렬

파이썬으로는 타임아웃 발생, C++로는 동일 알고리즘으로도 정상 풀이

풀이3 내장함수를 이용하는 실용적인 방법

class Solution(object):  
 def sortList(self, head):  
 p = head  
 lst = []  
  
 while p:  
 lst.append(p.val)  
 p = p.next  
  
 lst.sort()  
  
 p = head  
 for i in range(len(lst)):  
 p.val = lst[i]  
 p = p.next  
 return head

59 구간 병합 겹치는 구간을 병합하라

처음풀이 class Solution(object):  
 def merge(self, intervals):  
 *"""* ***:type*** *intervals: List[List[int]]* ***:rtype****: List[List[int]]  
 """* intervals = set([tuple(x) for x in intervals])  
 intervals = list([list(x) for x in intervals])  
 intervals.sort(key=lambda x: x[0])  
 print(intervals)  
 i = 0  
  
 while len(intervals) != 1 and i != len(intervals):  
 if i != len(intervals)-1:  
 if intervals[i][1] in range(intervals[i+1][0], intervals[i+1][1]+1):  
 intervals[i+1][0] = intervals[i][0]  
 intervals.pop(i)  
 if not (len(intervals) != 1 and i != len(intervals)):  
 break  
 if i != 0:  
 if intervals[i][0] in range(intervals[i - 1][0], intervals[i - 1][1] + 1) and intervals[i][0] in range(  
 intervals[i - 1][0], intervals[i - 1][1] + 1):  
 intervals[i-1][1] = max(intervals[i][1], intervals[i-1][1])  
 intervals.pop(i)  
 else:  
 i = i + 1  
 continue  
 if not (len(intervals) != 1 and i != len(intervals)):  
 break  
 if i == 0:  
 i = i+1  
  
 return intervals

풀이1 정렬하여 병합

class Solution(object):  
 def merge(self, intervals):  
 merged = []  
 for i in sorted(intervals, key=lambda x: x[0]):  
 if merged and i[0] <= merged[-1][1]:  
 merged[-1][1] = max(merged[-1][1], i[1])  
 else:  
 merged += [i]  
 return merged

60 삽입 정렬 연결리스트를 삽입 정렬로 연결하라

처음풀이 모르겠음

풀이1 삽입정렬

class Solution(object):  
 def insertionSortList(self, head):  
 cur = parent = ListNode(None)  
 while head:  
 while cur.next and cur.next.val < head.val:  
 cur = cur.next  
  
 cur.next, head.next, head = head, cur.next, head.next  
 cur = parent  
 return cur.next

풀이2 삽입정렬의 비교 조건 개선

class Solution(object):  
 def insertionSortList(self, head):  
 cur = parent = ListNode(0)  
 while head:  
 while cur.next and cur.next.val < head.val:  
 cur = cur.next  
  
 cur.next, head.next, head = head, cur.next, head.next  
  
 if head and cur.val > head.val:  
 cur = parent  
 return parent.next

필요한 경우에만, cur가 돌아감

61 가장 큰 수 항목들을 조합하여 만들수 있는 가장 큰 수를 출력하라

처음풀이 모르겠음

풀이1 삽입정렬

class Solution:  
 @staticmethod  
 def to\_swap(n1, n2):  
 return str(n1) + str(n2) < str(n2) + str(n1)  
 def largestNumber(self, nums):  
 i = 1  
 while i < len(nums):  
 j = i  
 while j > 0 and self.to\_swap(nums[j - 1], nums[j]):  
 nums[j], nums[j - 1] = nums[j - 1], nums[j]  
 j -= 1  
 i += 1  
 return str(int(''.join(map(str, nums))))

62 유효한 애너그램 t가 s의 애너그램인지 판별하라

처음풀이 class Solution:  
 def isAnagram(self, s: str, t: str) -> bool:  
 return sorted(s) == sorted(t)

풀이1 정렬을 이용한 비교

처음풀이와 동일

63 색 정렬 빨간색을 0, 흰색을 1, 파란색을 2라 할 때 순서대로 인접하는 제자리 정렬을 수행하라

처음풀이 모르겠음

풀이1 네덜란드 국기 문제를 응용한 풀이

class Solution:  
 def sortColors(self, nums: list[int]) -> None:  
 red, white, blue = 0, 0, len(nums)  
 while white < blue:  
 if nums[white] < 1:  
 nums[red], nums[white] = nums[white], nums[red]  
 white += 1  
 red += 1  
 elif nums[white] > 1:  
 blue -= 1  
 nums[white], nums[blue] = nums[blue], nums[white]  
 else:  
 white += 1

64 원점에 K번째로 가까운 점 평면 상에 point 목록이 있을 때, 원점(0, 0)에서 k번 가까운 점 목록을 순서대로 출력하라. 평면상의 두점의 거리는 유클리드 거리로 한다

처음풀이 class Solution:  
 def kClosest(self, points: list[list[int]], k: int) -> list[list[int]]:  
 distance = []  
 res = []  
   
  
 for i in points:  
 distance.append(sum([x\*\*2 for x in i]))  
   
 distance = enumerate(distance)  
 distance = sorted(distance, key=lambda x: x[1])   
 for i in range(k):  
 res.append(points[distance[i][0]])  
   
 return res

풀이1 유클리드 거리의 우선순위 큐 순서

import heapq  
   
 class Solution:  
 def kClosest(self, points: list[list[int]], k: int) -> list[list[int]]:  
 heap = []  
   
 for (x, y) in points:  
 dist = x \*\* 2 + y \*\* 2  
 heapq.heappush(heap, (dist, x, y))  
   
 result = []  
 for \_ in range(k):  
 (dist, x, y) = heapq.heappop(heap)  
 result.append((x, y))  
 return result

18 이진 검색

이진 검색 정렬된 배열에서 타겟을 찾는 검색 알고리즘

시간복잡도 O(log n)

ex) 1~100에서 77찾기 1. left:0 right:100

2. left:50 right: 100

3. left:75 right:100

4. left:75 right:87

…

65 이진 검색 정렬된 nums를 입력박아 이진 검색으로 target에 해당하는 인덱스를 찾아라

처음풀이 class Solution:  
 def search(self, nums: list[int], target: int) -> int:  
 left = 0  
 right = len(nums)-1  
   
 while left <= right:  
 mid = (right + left) // 2  
 if nums[mid] < target:  
 left = mid+1  
 elif nums[mid] > target:  
 right = mid-1  
 else:  
 return mid  
   
 return -1

풀이1 재귀풀이

class Solution:  
 def search(self, nums: list[int], target: int) -> int:  
 def binary\_search(left, right):  
 if left <= right:  
 mid = (left + right) // 2  
   
 if nums[mid] < target:  
 return binary\_search(mid + 1, right)  
 elif nums[mid] > target:  
 return binary\_search(left, mid - 1)  
 else:  
 return mid  
 else:  
 return -1  
   
 return binary\_search(0, len(nums) - 1)

풀이2 반복 풀이

처음풀이와 동일

풀이3 이진 검색 모듈

import bisect  
   
 class Solution:  
 def search(self, nums: list[int], target: int) -> int:  
 index = bisect.bisect\_left(nums, target)  
   
 if index < len(nums) and nums[index] == target:  
 return index  
 else:  
 return -1

풀이4 이진 검색을 사용하지 않는 index 풀이

class Solution:  
 def search(self, nums: list[int], target: int) -> int:  
 try:  
 return nums.index(target)  
 except ValueError:  
 return -1

66 회전 정렬된 배열 검색 특정 피벗을 기준으로 회전하여 정렬된 배열에서 target 값의 인덱스를 출력하라

처음풀이 import bisect  
   
   
 class Solution:  
 def search(self, nums: list[int], target: int) -> int:  
 pivot = nums.index(min(nums))  
 new\_nums = []  
   
 for i in range(pivot, len(nums)):  
 new\_nums.append(nums[i])  
   
 for i in range(pivot):  
 new\_nums.append(nums[i])  
   
 index = bisect.bisect\_left(new\_nums, target)  
 if index < len(new\_nums) and new\_nums[index] == target:  
 return (index + pivot) % len(new\_nums)  
 else:  
 return -1

풀이1 피벗을 기준으로 한 이진 검색

class Solution:  
 def search(self, nums: list[int], target: int) -> int:  
 if not nums:  
 return -1  
   
 left, right= 0, len(nums)-1  
 while left < right:  
 mid = left + (right-left)//2  
   
 if nums[mid] > nums [right]:  
 left = mid + 1  
 else:  
 right = mid  
   
 pivot = left  
   
 left, right = 0, len(nums)-1  
 while left <= right:  
 mid = left + (right-left) //2  
 mid\_pivot = (mid + pivot) % len(nums)  
   
 if nums[mid\_pivot] < target:  
 left = mid + 1  
 elif nums[mid\_pivot] > target:  
 right = mid -1  
 else:  
 return mid\_pivot  
 return -1

67 두 배열의 교집합 두 배열의 교집합을 구하라

처음풀이 class Solution:  
 def intersection(self, nums1: list[int], nums2: list[int]) -> list[int]:  
 return list(set(nums1).intersection(set(nums2)))

풀이1 브루트포스로 계산

class Solution:  
 def intersection(self, nums1: list[int], nums2: list[int]) -> list[int]:  
 result = set()  
 for n1 in nums1:  
 for n2 in nums2:  
 if n1 == n2:  
 result.add(n1)  
 return result

풀이2 이진검색으로 일치여부 판별

import bisect  
   
 class Solution:  
 def intersection(self, nums1: list[int], nums2: list[int]) -> list[int]:  
 result = set()  
 nums2.sort()  
   
 for n1 in nums1:  
 i2 = bisect.bisect\_left(nums2, n1)  
 if len(nums2) >0 and len(nums2) > i2 and n1 == nums2[i2]:  
 result.add(n1)  
   
 return result

풀이3 투 포인터로 일치 여부 판별

양쪽 다 정렬 후 투 포인터 사용

class Solution:  
 def intersection(self, nums1: list[int], nums2: list[int]) -> list[int]:  
 result = set()  
 nums1.sort()  
 nums2.sort()  
   
 i = j = 0  
   
 while i < len(nums1) and j < len(nums2):  
 if nums1[i] > nums2[j]:  
 j += 1  
 elif nums1[i] < nums2[j]:  
 i += 1  
 else:  
 result.add(nums1[i])  
 i += 1  
 j += 1  
   
 return result

68 두 수의 합 정렬된 배열을 받아 덧셈하여 타겟을 만들 수 있는 배열의 두 숫자 인덱스를 리턴하라

배열의 인덱스는 1부터 시작한다

처음풀이 잘 모르겠음

풀이1 투 포인터

class Solution:  
 def twoSum(self, numbers: list[int], target: int) -> list[int]:  
 left, right = 0, len(numbers) - 1  
   
 while left != right:  
 if numbers[left] + numbers[right] < target:  
 left += 1  
 elif numbers[left] + numbers[right] > target:  
 right -= 1  
 else:  
 return left + 1, right + 1

풀이2 이진검색

class Solution:  
 def twoSum(self, numbers: list[int], target: int) -> list[int]:  
 for k, v in enumerate(numbers):  
 left, right = k+1, len(numbers)-1  
 expected = target - v  
   
 while left <= right:  
 mid = left + (right - left) //2  
 if numbers[mid] < expected:  
 left = mid + 1  
 elif numbers[mid] > expected:  
 right = mid -1  
 else:  
 return k+1, mid + 1

풀이3 bisert모듈 + 슬라이싱

import bisect  
   
 class Solution:  
 def twoSum(self, numbers: list[int], target: int) -> list[int]:  
 for k, v in enumerate(numbers):  
 expected = target - v  
 i = bisect.bisect\_left(numbers[k+1:], expected)  
 if i < len(numbers[k+1:]) and numbers[i+k+1] == expected:  
 return k+1, i+k+2

풀이4 bisect모듈 + 슬라이싱 최소화

import bisect  
 class Solution:  
 def twoSum(self, numbers: list[int], target: int) -> list[int]:  
 for k, v in enumerate(numbers):  
 expected = target - v  
 nums = numbers[k+1:]  
 i = bisect.bisect\_left(nums, expected)  
 if i < len(nums) and numbers[i+k+1] == expected:  
 return k+1, i+k+2

풀이5 bisect모듈 + 슬라이싱 제거

import bisect  
   
 class Solution:  
 def twoSum(self, numbers: list[int], target: int) -> list[int]:  
 for k, v in enumerate(numbers):  
 expected = target - v  
 i = bisect.bisect\_left(numbers, expected, k+1)  
 if i < len(numbers) and numbers[i] == expected:  
 return k+1, i+1

69 2D 행렬 검색2 MxN 행렬에서 값을 찾아내는 효율적인 알고리즘을 구현하라

행렬은 왼쪽에서 오른쪽, 위쪽에서 아래 오른차순으로 정렬되어 있다.

처음 풀이 class Solution:  
 def searchMatrix(self, matrix: list[list[int]], target: int) -> bool:  
 for i in range(0, len(matrix)):  
 if(target in matrix[i]):  
 return True  
   
 return False

풀이1 첫 행의 맨 뒤에서 탐색

첫 행의 맨 뒤 요소를 택한다음 비교후, 작으면 왼쪽 ∙ 크면 아래로 이동

class Solution:  
 def searchMatrix(self, matrix: list[list[int]], target: int) -> bool:  
 if not matrix:  
 return False  
   
 row = 0  
 col = len(matrix[0]) - 1  
   
 while row <= len(matrix) -1 and col >= 0:  
 if target == matrix[row][col]:  
 return True  
   
 elif target < matrix[row][col]:  
 col -=1  
   
 elif target > matrix[row][col]:  
 row+=1  
   
 return False

풀이2 파이썬다운 방식

class Solution:  
 def searchMatrix(self, matrix: list[list[int]], target: int) -> bool:  
 return any(target in row for row in matrix)

any()와 all() any() 포함된 값들 중 하나라도 참이 있으면 참 출력

all() 포함된 값들이 모두 참이어야 참 출력

19 비트 조작

부울 연산자 and, or not,

xor and, or, not을 조합해서 구성

ex) x = y = True

( x and not y ) or ( not x and y )

비트 연산자 and &

or |

xor ^

not ~

비트조작퀴즈 + 10진수의 덧셈과 동일

\* 10진수의 곱셈과 동일

>> 오른쪽 시프팅

오른쪽 자리는 없어짐

<< 왼쪽 시프팅

왼쪽 자리에 0을 삽입함

~ ~는 음수로 바꾸고 1을 뺀 연산

즉 1을 0으로, 0을 1로

ex) ~12 = -13(-12 -1)

자릿수를 제한한 비트 연산 xor이용

비트가 모두 1인 MASK와 XOR

2의 보수 2의 보수 숫자 포맷 부호비트 MSB

1이면 음수

NOT 연산 2의 보수에서 1을 뺀 것

70 싱글넘버 딱 하나를 제외하고 모든 엘리먼트는 2개씩 있다. 하나 있는 엘리먼트를 찾아라

처음 풀이 import collections  
   
 class Solution:  
 def singleNumber(self, nums: list[int]) -> int:  
 counter: collections.Counter = collections.Counter(nums)  
 for k, v in dict(counter).items():  
 if v == 1:  
 return k

풀이1 XOR풀이

import collections  
   
 class Solution:  
 def singleNumber(self, nums: list[int]) -> int:  
 result = 0  
 for num in nums:  
 result ^= num  
   
 return result

71 해밍거리 두 정수를 입력받아 몇 비트가 다른지 계산하라

처음 풀이 class Solution(object):  
 def hammingDistance(self, x, y):  
 return bin(x ^ y).count('1')

풀이1 처음풀이와 동일

72 두 정수의 합 두 정수 a, b의 합을 구하라, +∙-는 이용할 수 없다

처음 풀이 모르겠음…

풀이1 전가산기 구현

class Solution:  
 def getSum(self, a: int, b: int) -> int:  
 MASK = 0xffffffff  
 INT\_MAX = 0x7FFFFFFF  
   
 a\_bin = bin(a & MASK)[2:].zfill(32)  
 b\_bin = bin(b & MASK)[2:].zfill(32)  
   
 result = []  
 carry = 0  
 sum = 0  
   
 for i in range(32):  
 A = int(a\_bin[31-i])  
 B = int(b\_bin[31 - i])  
   
 Q1 = A&B  
 Q2 = A^B  
 Q3 = Q2 & carry  
 sum = carry ^ Q2  
 carry = Q1 | Q3  
   
 result.append(str(sum))  
   
 if carry == 1:  
 result.append('1')  
   
 result = int(''.join(result[::-1]),2) & MASK  
   
 if(result > INT\_MAX):  
 result = ~(result ^ MASK)  
   
 return result

풀이2 좀 더 간소한 구현

class Solution:  
 def getSum(self, a: int, b: int) -> int:  
 MASK = 0xffffffff  
 INT\_MAX = 0x7FFFFFFF  
   
 while b != 0:  
 a, b = (a^b)& MASK, ((a & b ) << 1 ) & MASK  
 if a > INT\_MAX:  
 a =~(a^MASK)  
   
 return a

73 UTF-8 검증 입력값이 UTF-8 문자열이 맞는지 검증하라.

처음풀이 모르겠음

풀이1 첫 바이트를 기준으로 한 판별

class Solution:  
 def validUtf8(self, data: list[int]) -> bool:  
   
 def check(size):  
 for i in range(start + 1, start + size + 1):  
 if i >= len(data) or (data[i] >> 6) != 0b10:  
 return False  
 return True  
   
 start = 0  
 while start < len(data):  
 first = data[start]  
 if (first >> 3) == 0b11110 and check(3):  
 start += 4  
 elif (first >> 4) == 0b1110 and check(2):  
 start += 3  
 elif (first >> 5) == 0b110 and check(1):  
 start += 2  
 elif (first >> 7) == 0:  
 start += 1  
 else:  
 return False  
 return True

74 1비트의 개수 부호없는 정수형을 입력받아 1비트의 개수를 출력하라

처음풀이 class Solution:  
 def hammingWeight(self, n: int) -> int:  
 res = 0  
   
 for i in str(bin(n)):  
 if i == '1':  
 res += 1

return res

풀이1 1의 개수 계산

class Solution:  
 def hammingWeight(self, n: int) -> int:  
 return bin(n).count('1')

풀이2 비트 연산

class Solution:  
 def hammingWeight(self, n: int) -> int:  
 count = 0  
   
 while n:  
 n &= n-1  
 count+=1

return count

20 슬라이딩 윈도우

슬라이딩 윈도우 고정 사이즈의 윈도우가 이동하면서 윈도우 내에 있는 데이터를 이용해 문제를 풀이하는 알고리즘

투포인터 주로 정렬된 배열 대상

슬라이딩 윈도우 정렬 관계 X

75 최대 슬라이딩 윈도우 배열 nums가 주어졌을 때 k크기의 슬라이딩 윈도우를 오른쪽 끝까지 이동하면서 최대 슬라이딩 윈도우를 구하라

처음풀이 class Solution:  
 def maxSlidingWindow(self, nums: list[int], k: int) -> list[int]:  
 res = []  
 res.append(max(nums[0:k]))  
   
 for i in range(1, len(nums)-k+1):  
 if res[-1] == nums[i-1]:  
 res.append(max(nums[i:k+i]))  
 elif res[-1] >= nums[i+k-1]:  
 res.append(res[-1])  
 elif res[-1] < nums[i+k-1]:  
 res.append(nums[i+k-1])  
   
 return res

풀이1 브루트포스로 계산  
 class Solution:  
 def maxSlidingWindow(self, nums: list[int], k: int) -> list[int]:  
 if not nums:  
 return nums  
   
 r = []  
 for i in range(len(nums)-k+1):  
 r.append(max(nums[i:i+k]))  
   
 return r

풀이2 큐를 이용한 최적화

import collections  
   
 class Solution:  
 def maxSlidingWindow(self, nums: list[int], k: int) -> list[int]:  
 result = []  
 window = collections.deque()  
   
 current\_max = float('-inf')  
   
 for i, v in enumerate(nums):  
 window.append(v)  
 if i < k-1:  
 continue  
   
 if current\_max == float('-inf'):  
 current\_max = max(window)  
 elif v > current\_max:  
 current\_max = v  
   
 result.append(current\_max)  
   
 if current\_max == window.popleft():  
 current\_max = float('-inf')  
   
 return result

76 부분 문자열이 포함된 최소 윈도우 문자열 S와 T를 입력받아 O(n)에 T의 모든 문자가 포함된 S의 최소 윈도우를 찾아라,

처음풀이 모르겠음

풀이1 모든 윈도우 크기를 브루트포스로 검색

class Solution:  
 def minWindow(self, s: str, t: str) -> str:  
 def contains(s\_substr\_lst:list, t\_lst:list):  
 for t\_elem in t\_lst:  
 if t\_elem in s\_substr\_lst:  
 s\_substr\_lst.remove(t\_elem)  
 else:  
 return False  
   
 if not s or not t:  
 return ''  
   
 window\_size = len(t)  
   
 for size in range(window\_size, len(s)+1):  
 for left in range(len(s)-size+1):  
 s\_substr = s[left:left+size]  
 if contains(list(s\_substr), list(t)):  
 return s\_substr  
   
 return ''  
 풀이2 투 포인터, 슬라이딩 윈도우로 최적화

import collections  
  
 class Solution:  
 def minWindow(self, s: str, t: str) -> str:  
 need = collections.Counter(t)  
 missing = len(t)  
 left = start = end = 0  
   
 for right, char in enumerate(s, 1):  
 missing -= need[char] > 0  
 need[char]-= 1  
   
 if missing == 0:  
 while left < right and need[s[left]] < 0:  
 need[s[left]] += 1  
 left += 1  
   
 if not end or right -left <= end -start:  
 start, end = left, right  
 need[s[left]] += 1  
 missing += 1  
 left += 1  
 return s[start:end]

풀이3 Counter로 좀 더 편리한 풀이

import collections  
   
   
 class Solution:  
 def minWindow(self, s: str, t: str) -> str:  
 t\_count = collections.Counter(t)  
 current\_count = collections.Counter()  
   
 start = float('-inf')  
 end = float('-inf')  
   
 left = 0  
 for right, char in enumerate(s,1):  
 current\_count[char] += 1  
   
 while current\_count & t\_count == t\_count:  
 if right - left < end-start:  
 start, end = left, right  
 current\_count[s[left]] -= 1  
 left += 1  
   
 return s[start:end] if end-start <= len(s) else ''

77 가장 긴 반복 문자 대체 대문자로 구성된 문자열 s가 주어졌을 때 k번만큼의 변경으로 만들 수 있느느, 연속으로 반복된 문자열의 가장 긴 길이를 출력하라

처음풀이 모르겠음

풀이1 투 포인터, 슬라이딩 윈도우, Counter를 모두 이용

import collections  
   
   
 class Solution:  
 def characterReplacement(self, s: str, k: int) -> int:  
 left = right = 0  
   
 counts = collections.Counter()  
 for right in range(1, len(s)+1):  
 counts[s[right-1]] += 1  
   
 max\_char\_n = counts.most\_common(1)[0][1]  
   
 if right - left -max\_char\_n > k:  
 counts[s[left]] -=1  
 left+=1  
   
 return right-left

21 그리디 알고리즘

그리디 알고리즘 눈 앞의 이익만을 쫒는 알고리즘

드물게 최적해 보장

그리디 알고리즘이 잘 작동하는 문제 탐욕 선택 속성을 갖고 있는 최적 부분 구조

탐욕 선택 속성 앞의 선택이 이후 선택에 영향을 주지 않는 것

최적 부분 구조 최적 해결 방법이 부분 문제에 대한 최적 해걀 방법으로 구성되는 경우

대부분의 경우 계산속도가 빠름

다이나믹 프로그래밍과 비교 서로 풀 수 있는 문제의 성격이 다름

접근방식도 다름

다이나믹 프로그래밍 하위 문제에 대한 최적의 솔루션을 찾은 다음, 이 결과들을 결합한 정보에 입각해 전역 최적 솔루션에 대한 선택

그리디 알고리즘 각 단계마다 로컬 최적해를 찾는 문제로 접근해 문제를 더 작게 줄여 나가는 형태

반대 방향으로 접근

배낭문제 배낭에 담을 수 있는 무게의 최댓값이 정해져 있고, 각각 짐의 가치와 무게가 있는 짐들을 배낭에 넣을 때 가치의 합이 최대가 되도록 짐을 고르는 문제

그리디 알고리즘으로 풀이

동전 바꾸기 문제 동전의 액면이 이전 액면의 배수 이상이면 그리디 알고리즘으로 가능

아닌 경우, 다이나믹 알고리즘

가장 큰 합 트리 상의 노드를 계속 더해가다가 마지막에 가장 큰 합이 되는 경로를 찾는 문제

그리디 알고리즘 불가능

78 주식을 사고팔기 가장 좋은 시점2 여러 번의 거래로 낼 수 있는 최대 이익을 산출하라

처음 풀이 class Solution:  
 def maxProfit(self, prices: list[int]) -> int:  
 profit = 0  
   
 for i,v in enumerate(prices):  
 if i == len(prices)-1:  
 break  
 if v < prices[i+1]:  
 profit += (prices[i+1]-v)  
   
 return profit

풀이1 그리디 알고리즘

class Solution:  
 def maxProfit(self, prices: list[int]) -> int:  
 result = 0  
   
 for i in range(len(prices)-1):  
 if prices[i+1] > prices[i]:  
 result += prices[i+1]-prices[i]  
   
 return result

풀이2 파이썬다운 방식

class Solution:  
 def maxProfit(self, prices: list[int]) -> int:  
 return sum(max(prices[i+1]-prices[i], 0) for i in range(len(prices)-1))

79 키에 따른 대기열 재구성 여러명의 사람들이 줄을 서 있다.

각각의 사람은 (h,k)의 두 정수 쌍을 갖는데, h는 그 사람의 키, k는 앞에 줄 서 있는 사람들 중 자신의 키 이상인 사람들의 수를 뜻한다. 이 값이 올바르도록 재배열하라,

처음풀이 모르겠음

풀이1 우선순위 큐 이용

import heapq  
  
class Solution:  
 def reconstructQueue(self, people: list[list[int]]) -> list[list[int]]:  
 heap = []  
  
 for person in people:  
 heapq.heappush(heap, (-person[0], person[1]))  
  
 res = []  
  
 while heap:  
 person = heapq.heappop(heap)  
 res.insert(person[1], [-person[0], person[1]])  
  
 return res

80 테스크 스케줄러 A에서 Z로 표현된 태스크가 있다. 각 간격마다 CPU는 한 번의 태스크만 실행할 수 있다. n번의 간격 내에는 동일한 태스크를 실행할 수 없다.

더 이상 태스크를 실행할 수 없는 경우 idle상태가돠 된다. 모든 태스크를 실행하기 위한 최소 간격을 출력하라

처음풀이 잘 모르겠음

import collections  
  
class Solution:  
 def leastInterval(self, tasks: list[str], n:int) -> int:  
 counter = collections.Counter(tasks)  
 res = 0  
 c = 0  
 print(counter)  
 while True:  
 mc = counter.most\_common(n+1)  
 print(mc)  
 res += c  
 c = 0  
 for i in mc:  
 if i[1] != 0:  
 res += 1  
 counter[i[0]] -= 1  
 else:  
 c+=1  
 if len(mc) < n+1:  
 c += (n+1-len(mc))  
 mc = counter.most\_common(1)  
 if mc[0][1] == 0:  
 return res

풀이1 우선순위 큐 이용

import collections  
  
class Solution:  
 def leastInterval(self, tasks: list[str], n:int) -> int:  
 counter = collections.Counter(tasks)  
 result = 0  
  
 while True:  
 sub\_count = 0  
  
 for task, \_ in counter.most\_common(n+1):  
 sub\_count += 1  
 result += 1  
  
 counter.subtract(task)  
 counter += collections.Counter()  
  
 if not counter:  
 break  
  
 result += n-sub\_count+1  
  
 return result

81 주유소 원형으로 경로가 연결된 주유소가 있다.

각 주유소는 gas[i]만큼의 기름을 갖고 있으며, 다음 주유소로의 이동에, costs[i]가 필요하다. 기름이 부족하면 이동할 수 없다고 할 때, 모든 주유소를 방문할 수 있는 출발점의 인덱스를 출력하라.

출발점이 존재하지 않는 경우 -1을 리턴하며, 출발점은 유일하다.

처음풀이 모르겠음…

풀이1 모두 방문

class Solution:  
 def canCompleteCircuit(self, gas: list[int], cost: list[int]) -> int:  
 for start in range(len(gas)):  
 fuel = 0  
 for i in range(start, len(gas) + start):  
 index = i % len(gas)  
  
 can\_travel = True  
 if gas[index] + fuel < cost[index]:  
 can\_travel = False  
 break  
 else:  
 fuel += gas[index] -cost[index]  
 if can\_travel:  
 return start  
 return -1

풀이2 한 번 방문

class Solution:  
 def canCompleteCircuit(self, gas: list[int], cost: list[int]) -> int:  
 if sum(gas) < sum(cost):  
 return -1  
  
 start , fuel = 0, 0  
 for i in range(len(gas)):  
 if gas[i] + fuel < cost[i]:  
 start = i+1  
 fuel = 0  
 else:  
 fuel += gas[i] - cost[i]  
 return start

82 쿠키부여 아이들에게 1개씩 쿠키를 나눠줘야 한다. 각 아이 child\_i마다 그리드 팩터를 갖고 있으며, 이는 아이가 만족하는 최소 크기의 쿠키를 말한다

각 쿠키는 cookie\_j는 크기를 갖고 있으며, 쿠키가 그리드 팩터 이상이어야 아이가 만족하며 쿠키를 받는다

최대 몇 명의 아이들에게 쿠키를 줄 수 있는 지 출력하라

처음풀이 class Solution:  
 def findContentChildren(self, g: list[int], s: list[int]) -> int:  
   
 if g == [] or s == []:  
 return 0  
   
 g.sort(reverse=True)  
 s.sort(reverse=True)  
 g\_count = 0  
 s\_count = 0  
   
 while g\_count < len(g) and s\_count < len(s):  
 if g[g\_count] > s[s\_count]:  
 g\_count+=1  
 else:  
 s\_count+=1  
 g\_count+=1  
   
 return s\_count

풀이1 그리디 알고리즘

class Solution:  
 def findContentChildren(self, g: list[int], s: list[int]) -> int:  
 g.sort()  
 s.sort()  
   
 child\_i = cookie\_j = 0  
   
 while child\_i < len(g) and cookie\_j < len(s):  
 if s[cookie\_j] >= g[child\_i]:  
 child\_i += 1  
 cookie\_j += 1  
   
 return child\_i

풀이2 이진 탐색

import bisect  
   
 class Solution:  
 def findContentChildren(self, g: list[int], s: list[int]) -> int:  
 g.sort()  
 s.sort()  
   
 result = 0  
 for i in s:  
 index = bisect.bisect\_right(g, i)  
 if index > result:  
 result+=1  
 return result

22 분할 정복

분할 정복 다중 분기 재귀를 기반으로 하는 알고리즘 디자인 패러다임

직접 해결할 수 있을 정도로 간단한 문제가 될 때까지 문제를 재귀적으로 쪼개나간 다음, 그 하위 문제의 결과들을 조합해 원래 문제의 결과로 만들어 냄

ex) 병합정렬

과정 1. 분할: 문제를 동일한 유형의 여러 하위 문제로 나눈다.

2. 정복: 가장 작은 단위의 하위 문제를 해결하여 정복한다.

3. 조합: 하위 문제에 대한 결과를 원래 문제에 대한 결과로 조합한다.

83 과반수 엘리먼트 과반수를 차지하는 엘리먼트를 출력하라

처음풀이 import collections  
  
class Solution:  
 def majorityElement(self, nums: list[int]) -> int:  
 c = collections.Counter(nums)  
 return c.most\_common(1)[0][0]

풀이1 브루트포스로 과반수 비교

class Solution:  
 def majorityElement(self, nums: list[int]) -> int:  
 for num in nums:  
 if nums.count(num) > len(nums) //2:  
 return num

풀이2 다이나믹 프로그래밍

import collections  
  
class Solution:  
 def majorityElement(self, nums: list[int]) -> int:  
 counts = collections.defaultdict(int)  
 for num in nums:  
 if counts[num] == 0:  
 counts[num] = nums.count(num)  
  
 if counts[num] > len(nums) //2:  
 return num

풀이3 분할 정복

class Solution:  
 def majorityElement(self, nums: list[int]) -> int:  
 if not nums:  
 return None  
 if len(nums) == 1:  
 return nums[0]  
  
 half = len(nums)//2  
 a = self.majorityElement(nums[:half])  
 b = self.majorityElement(nums[half:])  
  
 return [b,a][nums.count(a)>half]

풀이4 파이썬다운 방식

class Solution:  
 def majorityElement(self, nums: list[int]) -> int:  
 return sorted(nums)[len(nums)//2]

84 괄호를 삽입하는 여러가지 방법 숫자와 연산자를 입력받아 가능한 모든 조합의 결과를 출력하라

처음풀이 모르겠음

풀이1 분할정복을 이용한 다양한 조합

class Solution:  
 def diffWaysToCompute(self, expression: str) -> list[int]:  
 def compute(left, right, op):  
 results = []  
 for l in left:  
 for r in right:  
 results.append(eval(str(l) + op + str(r)))  
   
 return results  
   
 if expression.isdigit():  
 return [int(expression)]  
   
 results = []  
   
 for index, value in enumerate(expression):  
 if value in "-+\*":  
 left = self.diffWaysToCompute(expression[:index])  
 right = self.diffWaysToCompute(expression[index+1:])  
   
 results.extend(compute(left, right, value))  
 return results

23 다이나믹 프로그래밍

다이나믹 프로그래밍 문제를 각각의 작은 문제로 나누어 해결한 결과를 저장해뒀다가 나중에 큰 문제의 결과와 합하여 풀이하는 알고리즘

최적 부분 구조 문제의 최적 해결 방법이 부분 문제에 대한 최적 해결 방법으로 구성되는 문제

그리디 알고리즘과의 비교 그리디 알고리즘 항상 그 순간에 최적으로 생각되는 것을 선택하면서 풀이해 나가는 것

다이나믹 프로그래밍 “중복”된 하위 문제들의 결과를 저장해뒀다가 풀이해 나감

대부분의 재귀 알고리즘은 최적 부분 구조 문제를 풀 수 있음

다이나믹 프로그래밍 방법론 상향식 = 타뷸레이션

더 작은 하위 문제부터 살펴본 다음 , 작은 문제의 정답을 이용해 큰 문제의 정답을 품

하향식 = 메모이제이션

하위 문제의 정답을 계산했는지 확인해가며 문제를 자연스러운 방식으로 풀어나감

85 피보나치 수 피보나치 수를 구하라

처음 풀이 class Solution:  
 def fib(self, n: int) -> int:  
 n0 = 0  
 n1 = 1  
 n2 = None  
   
 if n == 0:  
 return 0  
 elif n == 1:  
 return 1  
   
 for i in range(n - 1):  
 n2 = n0 + n1  
 n0 = n1  
 n1 = n2  
   
 return n2

풀이1 재귀구조 브루트포스

class Solution:  
 def fib(self, n: int) -> int:  
 if n <= 1:  
 return n  
 return self.fib(n-1) + self.fib(n-2)

풀이2 메모이제이션

다이나믹 프로그래밍의 하향식 풀이

import collections  
   
   
 class Solution:  
 dp = collections.defaultdict(int)  
   
 def fib(self, n: int) -> int:  
 if n <= 1:  
 return n  
   
 if self.dp[n]:  
 return self.dp[n]  
   
 self.dp[n] = self.fib(n-1) + self.fib(n-2)  
 return self.dp[n]

풀이3 타뷸레이션

import collections  
   
   
 class Solution:  
 dp = collections.defaultdict(int)  
   
 def fib(self, n: int) -> int:  
 self.dp[0] = 0  
 self.dp[1] = 1  
   
 for i in range(2, n+1):  
 self.dp[i] = self.dp[i-1] + self.dp[i-2]  
 return self.dp[n]

풀이4 두 변수만을 이용해 공간 절약

class Solution:  
 def fib(self, n: int) -> int:  
 x, y = 0, 1  
 for i in range(0, n):  
 x, y = y, x + y  
 return x

풀이5 행렬

O(log n) 만에 가능

numpy를 이용하므로 풀이하지 않음

0-1 배낭 문제 짐을 쪼갤수 가 없음

86 최대서브배열 합이 최대가 되는 연속 서브 배열을 찾아 합을 리턴해라

처음풀이 모르겠음…

풀이1 메모이제이션

class Solution:  
 maxSubArray(self, nums: list[int]) -> int:  
 for i in range(1, len(nums)):  
 nums[i] += nums[i-1] if nums[i-1] > 0 else 0  
 return max(nums)

풀이2 카데인 알고리즘

import sys  
   
   
 class Solution:  
 def maxSubArray(self, nums: list[int]) -> int:  
 best\_sum = -sys.maxsize  
 current\_sum = 0  
 for num in nums:  
 current\_sum = max(num, current\_sum + num)  
 best\_sum = max(best\_sum, current\_sum)  
   
 return best\_sum

87 계단오르기 당신은 계단을 오르고 있다. 정상에 도달하기 위해 n개의 계단을 올라야 한다. 매번 각각 1,2개의 계단을 오를 수 있다면 정상에 도달하기 위한 방법은 몇가지 경로가 되는가?

처음풀이 import collections  
  
  
class Solution:  
 dp = collections.defaultdict(int)  
  
 def climbStairs(self, n: int) -> int:  
 if n <= 2:  
 return n  
  
 if self.dp[n] == 0:  
 self.dp[n] = self.climbStairs(n-1)+self.climbStairs(n-2)  
  
 return self.dp[n]

풀이1 재귀구조 브루트포스

class Solution:  
 def climbStairs(self, n: int) -> int:  
 if n <= 2:  
 return n  
   
 return self.climbStairs(n-1) + self.climbStairs(n-2)

풀이2 메모이제이션

import collections  
  
  
class Solution:  
 dp = collections.defaultdict(int)  
  
 def climbStairs(self, n: int) -> int:  
 if n <= 2:  
 return n  
  
 if self.dp[n] == 0:  
 self.dp[n] = self.climbStairs(n-1)+self.climbStairs(n-2)  
  
 return self.dp[n]

88 집 도둑 어느 집에서든 돈을 훔칠 수 있지만 바로 옆집은 훔칠 수 없고 한 칸 이상 떨어진 집만 가능하다

각 집에는 훔칠 수 있는 돈의 액수가 입력값으로 표기되어 있다. 훔칠수 있는 가장 큰 금액을 출력하라

처음풀이 모르겠음

풀이1 재귀 구조 브루트포스

class Solution:  
 def rob(self, nums: list[int]) -> int:  
 def \_rob(i):  
 if i < 0:  
 return 0  
   
 return max(\_rob(i-1), \_rob(i-2)+nums[i])  
   
 return \_rob(len(nums)-1)

풀이2 타뷸레이션

import collections  
   
 class Solution:  
 def rob(self, nums: list[int]) -> int:  
 if not nums:  
 return 0  
   
 if len(nums) <= 2:  
 return max(nums)  
   
 dp = collections.OrderedDict()  
 dp[0], dp[1] = nums[0], max(nums[0], nums[1])  
   
 for i in range(2, len(nums)):  
 dp[i] = max(dp[i-1], dp[i-2]+nums[i])  
   
 return dp.popitem()[1]