

1. 파이썬 기본

배열조작

```
board = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]

di = [(-1, -1), (-1, 0), (-1, 1),
      (0, -1), (0, 0), (0, 1),
      (1, -1), (1, 0), (1, 1)]
```

배열회전

```
# 90도 한줄
rotated = list(zip(*reversed(arr)))

# 90도
one = [[0] * 3 for _ in range(3)]
for y in range(N):
    for x in range(N):
        one[x][N-1-y] = arr[y][x]

# 180도
two = [[0] * 3 for _ in range(3)]
for y in range(N):
    for x in range(N):
        two[N-1-y][N-1-x] = arr[y][x]

# 270도
three = [[0] * 3 for _ in range(3)]
for y in range(N):
    for x in range(N):
        three[N-1-x][y] = arr[y][x]

# 전치행렬
four = [[0] * 3 for _ in range(3)]
for y in range(N):
    for x in range(N):
        four[x][y] = arr[y][x]
```

정렬

```

a = [(1, 2), (0, 1), (5, 1), (5, 2), (3, 0)]
b = sorted(a)
# [(0, 1), (1, 2), (3, 0), (5, 1), (5, 2)]
c = sorted(a, key=lambda x: x[0])
# [(0, 1), (1, 2), (3, 0), (5, 1), (5, 2)]

d = sorted(a, key=lambda x: x[1])
# [(3, 0), (0, 1), (5, 1), (1, 2), (5, 2)]

# 첫번째 오름차순, 두번째 내림차순
e = [(1, 3), (0, 3), (1, 4), (1, 5), (0, 1), (2, 4)]
f = sorted(e, key=lambda x: (x[0], -x[1]))
# [(0, 3), (0, 1), (1, 5), (1, 4), (1, 3), (2, 4)]

```

딕셔너리

```

a = {'1': 1, '2': 2, '3': 3}
# 삽입
a['4'] = 4
# {'1': 1, '2': 2, '3': 3, '4': 4}
print(a)
# 삭제
a.pop('1')
del(a['2'])
# {'3': 3, '4': 4}
print(a)
# 합치기
b = {'5': 5, '6': 6}
a.update(b)
# {'3': 3, '4': 4, '5': 5, '6': 6}
print(a)

```

set

```

a = {1, 2}
# 삽입
a.add(3)
# {1, 2, 3}
print(a)
# 업데이트
b = {7, 8}
a.update([4, 5, 6])
a.update(b)
# {1, 2, 3, 4, 5, 6, 7, 8}
print(a)
# 삭제
a.remove(1)
# {2, 3, 4, 5, 6, 7, 8}
print(a)

# 합집합
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7}
c = a | b
d = a.union(b)

```

```

# {1, 2, 3, 4, 5, 6, 7}
print(c)
print(d)

# 교집합
c = a & b
d = a.intersection(b)
# {3, 4, 5}
print(c)
print(d)

# 차집합
c = a - b
d = b - a
e = a.difference(b)
# {1, 2}
print(c)
# {6, 7}
print(d)
# {1, 2}
print(e)

# 대칭 차집합
c = a ^ b
d = a.symmetric_difference(b)
# {1, 2, 6, 7}
print(c)
# {1, 2, 6, 7}
print(d)

```

deque

```

from collections import deque

Q = deque('love')
print(Q)
# deque(['l', 'o', 'v', 'e'])

# 1. 스택 구현: append(), pop()
Q.append('m')
# deque(['l', 'o', 'v', 'e', 'm'])
print(Q)
# deque(['l', 'o', 'v', 'e'])
print(Q.pop())
print(Q)

# 2. 큐 구현 : appendleft(), pop(), append(), popleft()
Q.appendleft('I')
# 왼쪽에서 'I'입력
# deque(['I', 'l', 'o', 'v', 'e'])
print(Q)
# 오른쪽에서 'e'출력
print(deque(['I', 'l', 'o', 'v']))
print(Q.pop())

# 3. extend(), extendleft()

```

```

# 오른쪽으로 'y','o','u' 확장
Q.extend('you')
# deque(['l', 'o', 'v', 'e', 'y', 'o', 'u'])
print(Q)

# 왼쪽으로 'I' 확장
Q.extendleft('I')
# deque(['I', 'l', 'o', 'v', 'e', 'y', 'o', 'u'])
print(Q)

# 4. 리스트처럼 사용: insert(), remove()
Q[2] = 'n'
# 인덱스를 이용한 항목 수정 'v' => 'n'
# deque(['l', 'o', 'n', 'e'])
print(Q)

Q = deque('love')
# 첫번째 항목에 'k'를 추가
Q.insert(0, 'k')
# deque(['k', 'l', 'o', 'v', 'e'])
print(Q)

# 100번째 항목(없으니까 가장 큰 쪽에)에 'k' 추가
Q.insert(100, 'k')
# deque(['k', 'l', 'o', 'v', 'e', 'k'])
print(Q)

# 'k'항목 삭제
Q.remove('k')
# 같은 항목이 있을때 지우면 왼쪽부터 삭제
# deque(['l', 'o', 'v', 'e', 'k'])
print(Q)

# 오른쪽에 있는 'k'삭제
Q.remove('k')
# deque(['l', 'o', 'v', 'e'])
print(Q)

# 5. 좌우 반전 reverse
# deque(['l', 'o', 'v', 'e'])
Q.reverse()
# deque(['e', 'v', 'o', 'l'])
print(Q)

# 6. 회전 rotate
Q = deque([1, 2, 3, 4, 5])
Q.rotate(1)
# deque([5, 1, 2, 3, 4])
print(Q)
Q.rotate(-1)
print(Q)

```

2. 정규 표현식

```
re.match("Hello","Hello, world!")
# Hello
re.match("Python","Hello, world!")
# None
re.search("^Hello","Hello,world")
# Hello
```

- `^`: 문자열이 맨 앞에 오는지
- `$`: 문자열이 맨 뒤에 오는지

```
re.search("^Hello","Hello,world")
# Hello
re.search("^Hello","hi,Hello,world")
# None
re.search("world$","Hello, world")
# world
```

- `|`: 문자열이 하나라도 포함되는지

```
re.match("hello|world","hello")
# hello
```

- `*`: 문자(숫자)가 0개 이상인지
- `+`: 문자(숫자)가 1개 이상인지

```
re.match('[0-9]+','1234')
# 1234
re.match('[0-9]*','1234')
# 1234
re.match('[0-9]*','abcd')
# None
re.match('a*b','b')
# b
re.match('a+b','b')
# None
re.match('a*b','aab')
# aab
re.match('a+b','aab')
# aab
```

- `?`: 문자가 0개 또는 1개인지
- `.`: 문자가 1개인지

```
re.match('H?', 'H')
# H?
re.match('H?', 'Hi')
# H?
re.match('H. ', 'Hi ')
# H.
```

- `문자{개수}`: "문자"가 "개수"만큼 있는지
- `문자열{개수}`: "문자열"이 "개수"만큼 있는지

- `[0-9]{개수}`: "숫자"기 "개수"만큼 있는지

```
re.match('h{3}', 'hhhello')
# hhh
re.match('(hello){3}', 'hellohellohello')
# hellohellohello
re.match('[0-9]{3}-[0-9]{3}-[0-9]{4}', '010-101-0101')
# 010-101-0101
```

- `a-z`: 소문자
- `A-Z`: 대문자
- `가-힣`: 한글

```
re.match('[a-zA-Z0-9]+', 'Hello1234')
# Hello1234
re.match('[A-Z0-9]+', 'hello')
# None
re.match('[가-힣]+', '홍길동')
# 홍길동
```

- `[^범위]*`
- `[^범위]+`

```
re.search("[^A-Z]*", 'hello')
# hello
re.search("[^A-Z]+", 'hello')
# hello
```

- `[범위]*$`
- `[범위]*+`

```
re.search("[0-9]+$", 'Hello1234')
# 1234
```

- `\특수문자`: 특수 문자 판단
- `\d`: 모든 숫자
- `\D`: 숫자가 아닌 모든 문자
- `\w`: 영문 대소문자, 숫자, 밑줄 문자
- `\D`: 영문 대소문자, 숫자, 밑줄 문자가 아닌 모든 문자
- `\s`: 공백, `\t`, `\n`, `\r`, `\f`, `\v` 을 포함
- `\S`: 공백을 제외하고 `\t`, `\n`, `\r`, `\f`, `\v`만 포함

```
re.search('\*+', "1 ** 2")
# **
re.search('\d+', '1234')
# 1234
re.search('\D+', '1234')
# None
re.search('\D+', 'Hello')
# Hello
re.search('\w+', 'Hello_1234')
# Hello_1234
re.search('[a-zA-Z0-9 ]+', "Hello 1234")
```

```
# Hello 1234
re.search('[a-zA-Z0-9\s]+', "Hello 1234")
# Hello 1234
```

- (정규 표현식) (정규 표현식)
- 매치객체.group(숫자) : 그룹에 해당하는 문자열(숫자)를 가져옴
- 매치객체.groups() : 그룹에 해당하는 문자열(숫자)을 튜플로 반환
- (?P<이름>정규표현식) -> 매치객체.group('그룹이름') : 그룹에 이름을 지은 뒤 반환

```
r1 = re.match('([0-9]+) ([0-9]+)', '10 123')
print(r1.group(1))
# 10
print(r1.group(2))
# 123
print(r1.group())
# 10 123
print(r1.group(0))
# 10 123
print(r1.groups())
# ('10', '123')
r1 = re.match('(P<func>[a-zA-Z_][a-zA-Z0-9_]+)((P<arg>\w+)\)', 'print(1234)')
print(r1.group('func'))
# print
print(r1.group('arg'))
# 1234
```

- re.findall('패턴', '문자열')

```
re.findall('[0-9]+', '1 2 Fizz 4 Buzz Fizz 7 8')
# ['1', '2', '4', '7', '8']
```

- re.sub('패턴', '바꿀 문자열', '문자열', 바꿀 횟수)
- re.sub('패턴', 교체함수, '문자열', 바꿀 횟수)

```
re.sub('apple|orange', 'fruit', 'apple box orange tree')
# fruit box fruit tree
re.sub('[0-9]+', lambda m: str(int(m.group()) * 10), '1 2 Fizz 4 Buzz Fizz 7 8')
# 10 20 Fizz 40 Buzz Fizz 70 80
```

3. 그래프

DFS

```
def DFS(graph, v, visited):
    visited[v] = True
    print(v, end=" ")
    for i in graph[v]:
        if not visited[i]:
            DFS(graph, i, visited)

graph = [[], [2, 3, 8], [1, 7], [1, 4, 5],
         [3, 5], [3, 4], [7], [2, 6, 8], [1, 7]]
visited = [False] * 9

DFS(graph, 1, visited)
```

BFS

```
from collections import deque

def BFS(graph, start, visited):
    queue = deque([start])
    visited[start] = True
    while queue:
        v = queue.popleft()
        print(v, end=" ")
        for i in graph[v]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True

graph = [[], [2, 3, 8], [1, 7], [1, 4, 5],
         [3, 5], [3, 4], [7], [2, 6, 8], [1, 7]]
```

위상 정렬(DAG)

```
from collections import deque

N, M = map(int, input().split())
graph = [[] for _ in range(N+1)]
check = [0 for _ in range(N+1)]
for i in range(M):
    a, b = map(int, input().split())
    graph[a].append(b)
    check[a] += 1
Q = deque()
for i in range(1, N+1):
    if check[i] == 0:
        Q.append(i)
while Q:
    u = Q.popleft()
    for v in graph[u]:
        check[v] -= 1
        if check[v] == 0:
            Q.append(v)
    print(u, end=" ")
```


다익스트라

```
from collections import defaultdict

V, E = map(int, input().split())
start = int(input())
graph = defaultdict(list)
for _ in range(E):
    a, b, c = map(int, input().split())
    graph[a].append((b, c))
dist = defaultdict(int)
Q = [(0, start)]
while Q:
    time, node = heappop(Q)
    if node not in dist:
        dist[node] = time
        for v, w in graph[node]:
            alt = time + w
            heappush(Q, (alt, v))
print(dist)
print(graph)
```

유니온 파인드

```
def find(parent, x):
    if parent[x] != x:
        parent[x] = find(parent, parent[x])
    return parent[x]

def union(parent, a, b):
    a = find(parent, a)
    b = find(parent, b)
    if a > b:
        parent[b] = a
    else:
        parent[a] = b
```

크루스칼

```
V, E = map(int, input().split())
parent = [i for i in range(V+1)]

edges = []
for _ in range(E):
    A, B, C = map(int, input().split())
    edges.append((C, A, B))
edges.sort()
result = 0

for C, A, B in edges:
    if find(parent, A) != find(parent, B):
        union(parent, A, B)
        result += C
```

```
        result += C
    print(result)
```

프림

```
from collections import deque
import heapq
V, E = map(int, input().split())
graph = [[] for _ in range(V+1)]
visited = [False] * (V+1)

for _ in range(E):
    a, b, c = map(int, input().split())
    graph[a].append((c, b))
    graph[b].append((c, a))

heap = []
visited[1] = True
result = 0
cnt = 1
for a in graph[1]:
    heapq.heappush(heap, a)
while heap:
    cost, to = heapq.heappop(heap)
    if not visited[to]:
        visited[to] = True
        cnt += 1
        result += cost
        for u in graph[to]:
            heapq.heappush(heap, u)
    if cnt == V:
        break
print(result)
```

플로이드

```
import sys
INF = sys.maxsize
N, M = map(int, input().split())
graph = [[INF]*(N+1) for _ in range(N+1)]
for _ in range(M):
    a, b, c = map(int, input().split())
    graph[a][b] = c

for y in range(1, N+1):
    for x in range(1, N+1):
        if y == x:
            graph[y][x] = 0

for z in range(1, N+1):
    for y in range(1, N+1):
        for x in range(1, N+1):
            graph[y][x] = min(graph[y][x], graph[y][z] + graph[z][x])

for y in range(1, N+1):
```

```
print(graph[y][1:])
```

4. DP

DP(LIS)

```
N = int(input())
S = [0] + list(map(int, input().split()))
DP = [0] * (N+1)
DP[1] = 1
for i in range(2, N+1):
    for j in range(1, i):
        if S[i] > S[j]:
            DP[i] = max(DP[j], DP[i])
    DP[i] += 1
print(max(DP))
```

TOP DOWN

```
import sys
sys.setrecursionlimit(2000*2000)
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    if DP[n] != -1:
        return DP[n]
    DP[n] = fibonacci(n-1) + fibonacci(n-2)
    return DP[n]
n = int(input())
DP = [-1] * (n+1)
fibonacci(n)
print(DP[n])
```

BOTTOM UP

```
def fibonacci(n):
    DP[0] = 0
    DP[1] = 1
    for i in range(2, n+1):
        DP[i] = DP[i-1] + DP[i-2]
n = int(input())
DP = [-1] * (n+1)
fibonacci(n)
print(DP[n])
```

KMP

```

def LPS(pat, lps):
    leng = 0
    i = 1
    while i < len(pat):
        if pat[i] == pat[leng]:
            leng += 1
            lps[i] = leng
            i += 1
        else:
            if leng != 0:
                leng = lps[leng-1]
            else:
                lps[i] = 0
                i += 1

def KMP(pat, txt):
    M = len(pat)
    N = len(txt)
    lps = [0]*M
    LPS(pat, lps)
    i = 0 # index for txt[]
    j = 0 # index for pat[]
    while i < N:
        if txt[i] == pat[j]:
            i += 1
            j += 1
        elif txt[i] != pat[j]:
            if j != 0:
                j = lps[j-1]
            else:
                i += 1
        if j == M:
            print("Found pattern at index " + str(i-j))
            j = lps[j-1]

txt = 'ABXABABXAB'
pat = 'ABXAB'
KMP(pat, txt)

```

트라이

```

from collections import defaultdict

class TrieNode:
    def __init__(self):
        self.word = False
        self.children = defaultdict(TrieNode)

    def __repr__(self):
        return f'TrieNode({self.word}:{self.children.items()})'

class Trie:
    def __init__(self):
        self.root = TrieNode()

```

```

def insert(self, word):
    node = self.root
    for char in word:
        node = node.children[char]
    node.word = True

def search(self, word):
    node = self.root
    for char in word:
        if char not in node.children:
            return False
        node = node.children[char]
    return node.word

trie = Trie()
trie.insert('apple')
trie.insert('appeal')
print(trie.search('apple'))

```

5. 순열 조합

순열

```

def PERM(arr, r):
    result = []

    def perm(k, choice, used):
        if k == r:
            result.append(choice[:])
            return
        for i in range(len(arr)):
            if used & (1 << i):
                continue
            choice.append(arr[i])
            perm(k+1, choice, used | (1 << i))
            choice.pop()

    perm(0, [], 0)
    return result

result = PERM('ABC', 2)

```

```

def PERM(arr, r):
    result = []
    def perm(arr, r):
        for i in range(len(arr)):
            if r == 1:
                yield [arr[i]]
            else:
                for next in perm(arr[:i] + arr[i+1:], r-1):
                    yield [arr[i]] + next
    return result

```

```

for i in perm(arr, r):
    result.append(i)
return result

```

```
result = PERM('ABCDE', 2)
```

조합

```

def COMB(arr, r):
    result = []

    def comb(k, chosen, start):
        if k == r:
            result.append(chosen[:])
            return
        for i in range(start, len(arr)):
            chosen.append(arr[i])
            comb(k+1, chosen, i+1)
            chosen.pop()
    comb(0, [], 0)
    return result

```

```
result = COMB('ABCDE', 2)
```

```

def COMB(arr, r):
    result = []
    def comb(arr, r):
        for i in range(len(arr)):
            if r == 1:
                yield [arr[i]]
            else:
                for next in comb(arr[i+1:], r-1):
                    yield [arr[i]] + next
    for i in comb(arr, r):
        result.append(i)
    return result

```

```
result = COMB('ABCDE', 2)
```

부분집합

```

def SUBSET(nums):
    result = []
    def subset(index, path):
        result.append(path)
        for i in range(index, len(nums)):
            subset(i+1, path+[nums[i]])
    subset(0, [])
    return result

```

```
result = SUBSET([1, 2, 3])
```

6. 트리

```
class TreeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

    def __repr__(self):
        return 'TreeNode({})'.format(self.val)

def deserialize(string):
    if string == '{}':
        return None
    nodes = [None if val == 'null' else TreeNode(int(val))
              for val in string.strip('{}').split(',')]
    return nodes

deserialize('[1,2,3,null,null,4,null,null,5]')
deserialize(
    '[2,1,3,0,7,9,1,2,null,1,0,null,null,8,8,null,null,null,null,7]')
```

순회

```
class Node:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

root = Node("F",
            Node("B",
                Node("A"),
                Node("D",
                    Node("C"),
                    Node("E"))
                ),
            Node("G",
                None,
                Node("I", Node("H"))
            )
        )

# 전위순회
def preorder(node):
    if node is None:
        return
    print(node.val, end=" ")
    preorder(node.left)
    preorder(node.right)

# 중위순회
def inorder(node):
    if node is None:
        return
    inorder(node.left)
```

```

        print(node.val, end=" ")
        inorder(node.right)
# 후위 순회
def postorder(node):
    if node is None:
        return
    postorder(node.left)
    postorder(node.right)
    print(node.val, end=" ")
preorder(root)
inorder(root)
postorder(root)

```

LCA

```

from collections import deque
def LCA(u, v):
    if depth[u] < depth[v]:
        temp = u
        u = v
        v = temp
    while depth[u] != depth[v]:
        u = parent[u]
    while u != v:
        u = parent[u]
        v = parent[v]
    return u
N = int(input())
tree = [[] for _ in range(N+1)]
for _ in range(N-1):
    u, v = map(int, input().split())
    tree[u].append(v)
    tree[v].append(u)
depth = [0] * (N+1)
check = [False] * (N+1)
parent = [0] * (N+1)
check[1] = True
depth[1] = 0
Q = deque([1])
while Q:
    u = Q.popleft()
    for v in tree[u]:
        if not check[v]:
            depth[v] = depth[u] + 1
            check[v] = True
            parent[v] = u
            Q.append(v)
M = int(input())
while M:
    u, v = map(int, input().split())
    print(LCA(u, v))
    M -= 1

```


7. 세그먼트 트리/펜윅 트리

세그먼트 트리 구성(최소)

```
from math import ceil, log2

N, M = map(int, input().split())
size = (1 << (ceil(log2(N))+1))
board = [int(input()) for _ in range(N)]
tree = [0] * size

def init(node, start, end):
    if start == end:
        tree[node] = board[start]
    else:
        init(2 * node, start, (start + end) // 2)
        init(2 * node + 1, (start + end) // 2 + 1, end)
        tree[node] = min(tree[node * 2], tree[node * 2 + 1])
init(1, 0, N-1)
print(tree)
```

세그먼트 트리 쿼리(최소)

```
def query(node, start, end, s, e):
    if s > end or e < start:
        return -1
    if s <= start and end <= e:
        return tree[node]
    mid = (start + end) // 2
    left = query(2 * node, start, mid, s, e)
    right = query(2 * node + 1, mid + 1, end, s, e)
    if left == -1:
        return right
    elif right == -1:
        return left
    else:
        return min(left, right)

for _ in range(M):
    start, end = map(int, input().split())
    print(query(1, 0, N-1, start-1, end-1))
```

펜윅트리

```
def update(tree, i, plus):
    while i < len(tree):
        tree[i] += plus
        i += (i & -i)
        print(bin(i & -i))
def sum(tree, i):
    s = 0
    while i > 0:
```

```

        s += tree[i]
        i -= (i & -i)
    return s
n, m, k = map(int, input().split())
tree = [0]*(n+1)
board = [0]
for i in range(1, n+1):
    board.append(int(input()))
    update(tree, i, board[i])
for i in range(0, m+k):
    q, a, b = map(int, input().split())
    if q == 1:
        update(tree, a, b-board[a])
        board[a] = b
    if q == 2:
        print(sum(tree, b) - sum(tree, a-1))

```

8. 수학

1) $(A+B)\%C = ((A\%C) + (B\%C))\%C$

2) $(A*B)\%C = ((A\%C) * (B\%C))\%C$

9. 이분탐색

lower_bound

```

# 1 2 3 4 4 4 5 6 7 8
N, M = map(int, input().split())
nums = list(map(int, input().split()))
start, end = 0, N-1
result = 0
while start < end:
    mid = (start + end) // 2
    temp = nums[mid]
    if temp >= M:
        end = mid
    else:
        start = mid + 1
print(end)

```

upper_bound

```

# 1 2 3 4 4 4 5 6 7 8
N, M = map(int, input().split())
nums = list(map(int, input().split()))
start, end = 0, N-1
while start <= end:
    mid = (start + end) // 2
    temp = nums[mid]
    if temp <= M:
        start = mid + 1
    else:
        end = mid - 1
print(start)
# 6

```

모듈

```

from bisect import bisect_left, bisect_right
# 1 2 3 4 4 4 5 6 7 8
N, M = map(int, input().split())
nums = list(map(int, input().split()))
print(bisect_left(nums, 4))
print(bisect_right(nums, 4))

```