[SWEA]4874

0224

Fourth라는 컴퓨터 언어는 스택 연산을 기반으로 하고 있어 후위 표기법을 사용한다.

Fourth에서는 동작은 다음과 같다.
숫자는 스택에 넣는다.
연산자는 만나면 스택이 숫자 두개를 꺼내 더하고 결과를 다시 스택에 넣는다.
‘.’은 스택에서 숫자를 꺼내 출력한다.

Fourth 코드의 연산 결과를 출력하는 프로그램을 만드시오. 만약 형식이 잘못되어 연상이 불가능한 경우 ‘error’를 출력한다.

[입력]
첫 줄에 테스트 케이스 개수가 주어진다. 다음 줄부터 테스트케이스 별로 정수와 연산자가 256자 이내의 연산코드가 주어진다. 피 연산자와 연산자는 여백으로 구분되어 있으며, 코드는’.’로 끝난다. 나눗셈의 경우 항상 나누어 떨어진다.

ex) words = ‘4 2 / .’
↳ 순회

조건 i) 피연산자면 Stack에 저장
조건 ii) 연산자면 Stack[-2][-1] 이용해서 수식계산
↳ 저장된 피연산자가 부족하다면, error 출력!
조건 iii) ‘.’가 순회되었을 때 Stack의 길이가 1이 아니면 error!

```
for i in words:
    if 피연산자 ⟶ Stack 저장
    elif 연산자 ⟶ 연산결과 Stack에 저장 or error!
    else ⟶ Stack 출력 or error!
```

[SWEA]4875

N x N 크기의 미로에서 출발지에서 목적지에 도착하는 경로가 존재하는지 확인하는 프로그램을 작성하시오. 도착할 수 있으면, 1 아니면 0을 출력한다. 주어진 미로 밖으로는 나갈 수 없다.

[입력]
첫 줄에는 테스트 케이스 개수가 주어진다.
다음 줄부터 테스트 케이스의 별로 미로의 크기 N과 N개의 줄에 걸쳐 미로의 통로와 벽에 대한 정보가 주어진다.
0은 통로, 1은 벽, 2는 출발, 3은 도착이다. (5<= N <= 100)

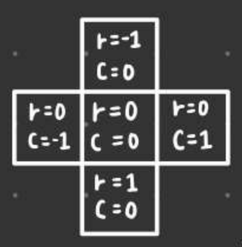
예시) N=5

1	3	1	0	1
1	0	1	0	1
1	0	1	0	1
1	0	1	0	1
1	0	0	2	1

(반복)
(1) 방문했는 지 확인하는 2차원 배열 생성

F/T	F/T	F/T	F/T	F/T
F/T	F/T	F/T	F/T	F/T
F/T	F/T	F/T	F/T	F/T
F/T	F/T	F/T	F/T	F/T
F/T	F/T	F/T	F/T	F/T

(2) 현재좌표의 상하좌우확인 (반복)



단, 미로 범위 내!

조건
(3) 확인 후, 근처좌표가 통로이고 방문하지 않은 곳이면 이동
(4) 확인 후, 근처좌표가 통로이고 방문하지 않은 곳이 없으면 이전 경로로 이동!
조건 i) 근처좌표가 통로이고 방문하지 않은 곳을 찾을 때까지 반복
조건 ii) 이전경로가 없으면 반복 중단

[SWEA]4880

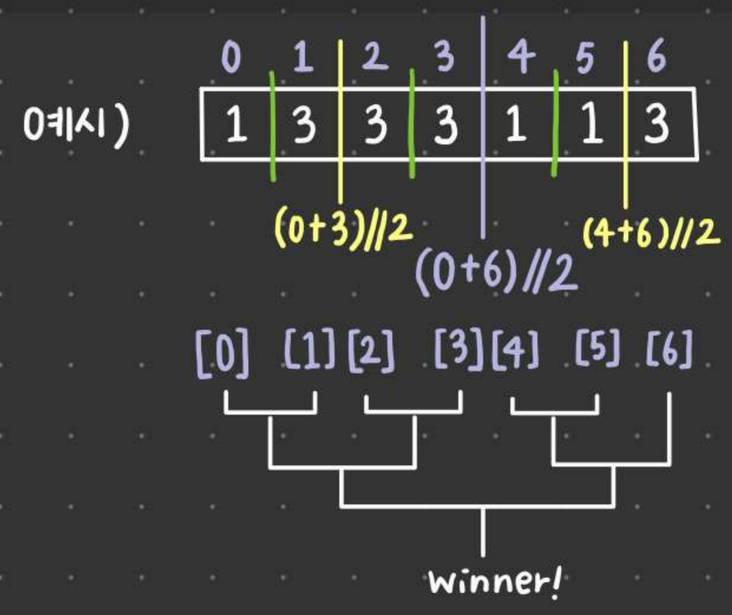
0224

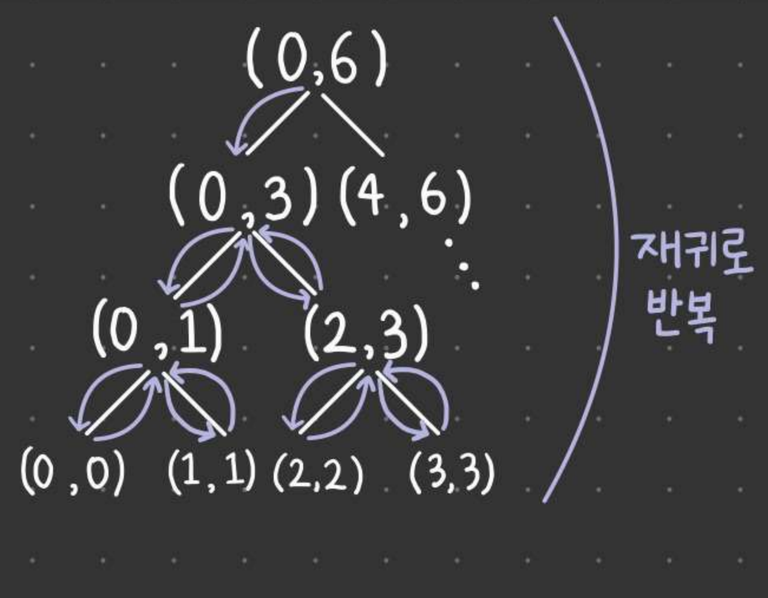
사다리 게임이 지겨워진 알고리즘 반 학생들이 새로운 게임을 만들었다. 가위바위보가 그려진 카드를 이용해 토너먼트로 한 명을 뽑는 것이다. 게임 룰은 다음과 같다.

1번부터 N번까지 N명의 학생이 N장의 카드를 나눠 갖는다. 전체를 두 개의 그룹으로 나누고, 그룹의 승자끼리 카드를 비교해서 이긴 사람이 최종 승자가 된다. 그룹의 승자는 그룹 내부를 다시 두그룹으로 나눠 뽑는데, i번부터 j번까지 속한 그룹은 파이썬 연산으로 다음처럼 두개로 나눈다. | i ~ (i+j)//2 | (i+j)//2+1 ~ j|

두 그룹이 각각 1명이 되면 양 쪽의 카드를 비교해 승자를 가리고, 다시 더 큰 그룹의 승자를 뽑는 방식이다.
다음은 4명이 카드를 비교하는 경우로, 숫자 1은 가위, 2는 바위, 3은 보를 나타낸다. 만약 같은 카드인 경우 편의상 번호가 작은 쪽을 승자로 하고, 처음 선택한 카드는 바꾸지 않는다. N명의 학생들이 카드를 골랐을 때 1등을 찾는 프로그램을 만드시오.

[입력]
첫 줄에는 테스트 케이스 개수 T가 주어진다. 다음 줄부터는 테스트 케이스 별로 인원수 N과 다음 줄에 N명이 고른 카드가 번호 순으로 주어진다. (4 <= N <= 100) 카드의 숫자는 각각 1은 가위, 2는 바위, 3은 보를 나타낸다.



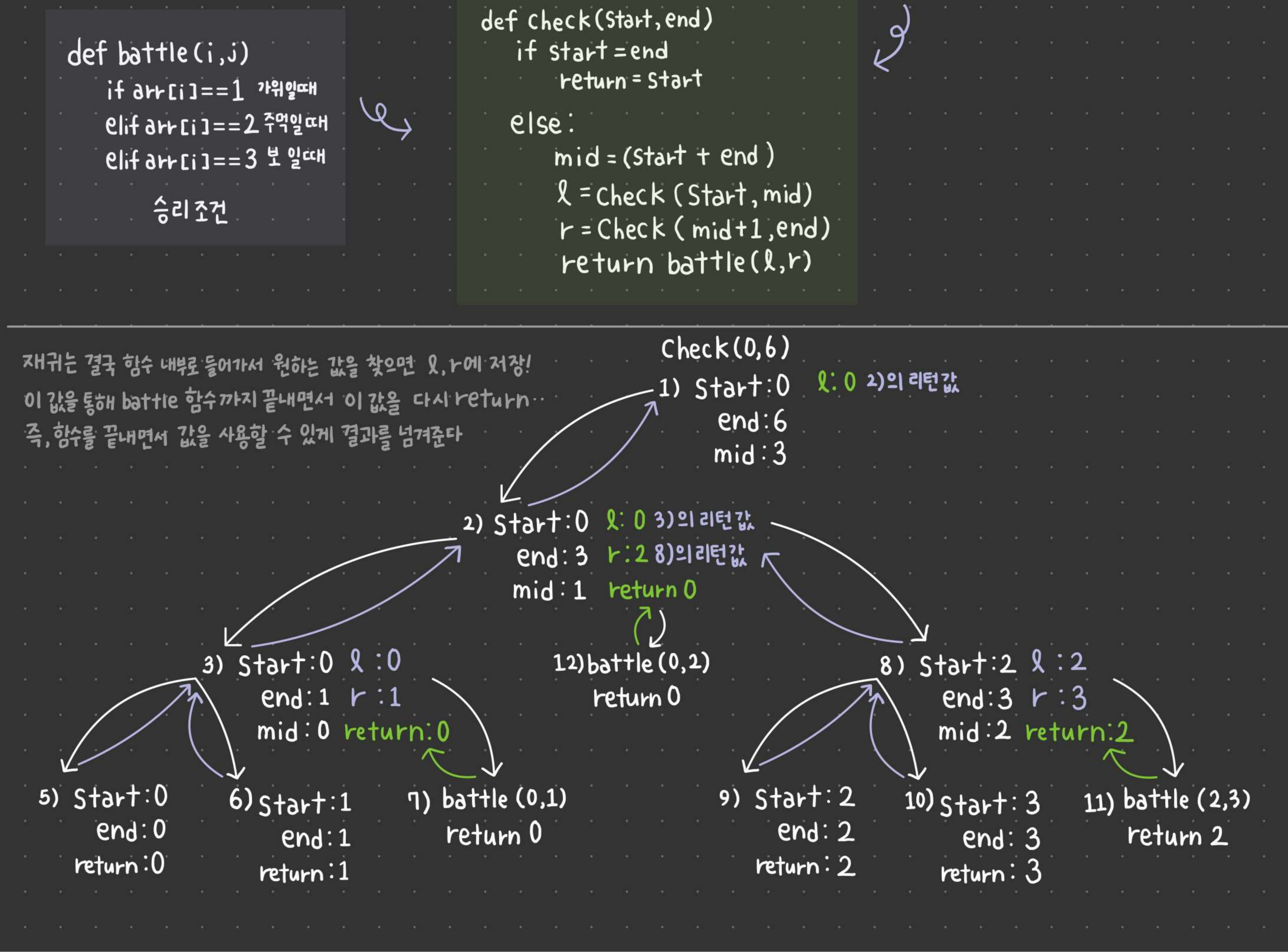



```
def check(start, end)
    if start = end
        종료조건!
    else:
        mid = (start + end)
        l = check(start, mid)
        r = check(mid+1, end)
```

```
def battle(i,j)
    if arr[i]==1 가위일때
    elif arr[i]==2 주먹일때
    elif arr[i]==3 보 일때
        승리조건
```

```
def check(start, end)
    if start = end
        return = start
    else:
        mid = (start + end)
        l = check(start, mid)
        r = check(mid+1, end)
        return battle(l,r)
```

재귀는 결국 함수 내부로 들어가서 원하는 값을 찾으면 l, r에 저장!
이 값을 통해 battle 함수까지 끝내면서 이 값을 다시 return..
즉, 함수를 끝내면서 값을 사용할 수 있게 결과를 넘겨준다



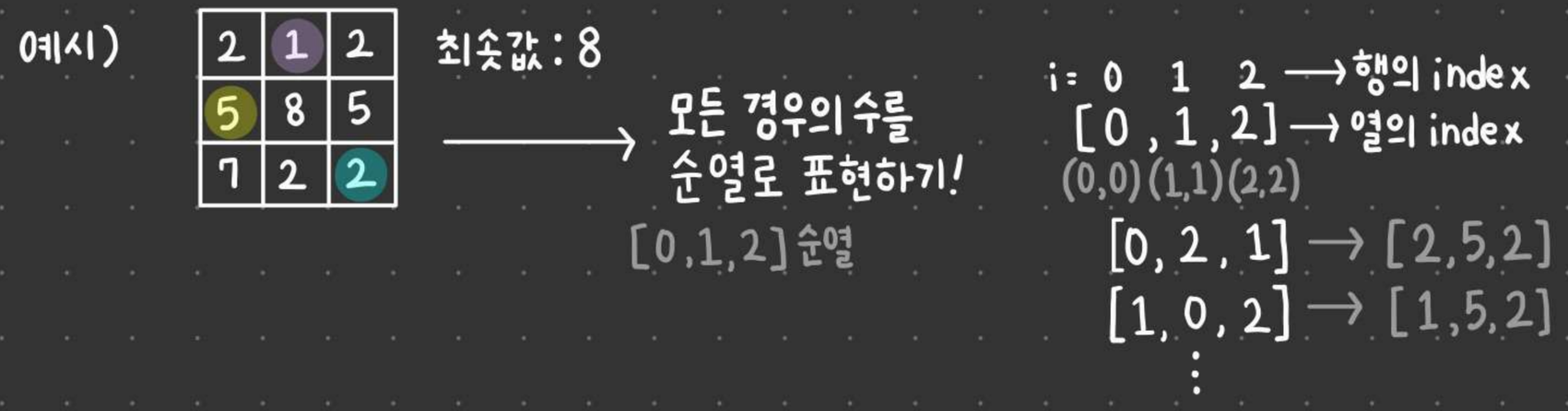
[SWEA]4881

0224

N x N 배열에 숫자가 들어있다. 한 줄에서 하나씩 N개의 숫자를 골라 합이 최소가 되도록 하려고 한다.
단, 세로로 같은 줄에서 두 개 이상의 숫자를 고를 수 없다.

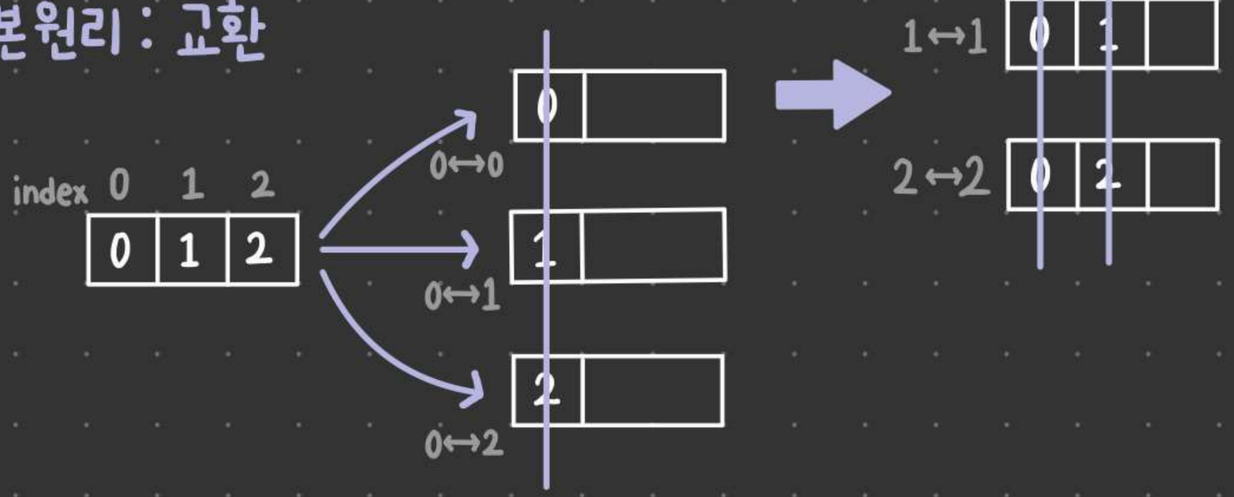
조건에 맞게 숫자를 골랐을 때의 최소 합을 출력하는 프로그램을 만드시오.

[입력]
첫 줄에 테스트 케이스 개수가 주어진다. 다음 줄부터 테스트 케이스의 첫 줄에 숫자 N이 주어지고,
이후 N개씩 N줄에 걸쳐 10보다 작은 자연수가 주어진다. (3<=N<=10)



(1) 순열 구현 (반복문&재귀)

기본원리 : 교환



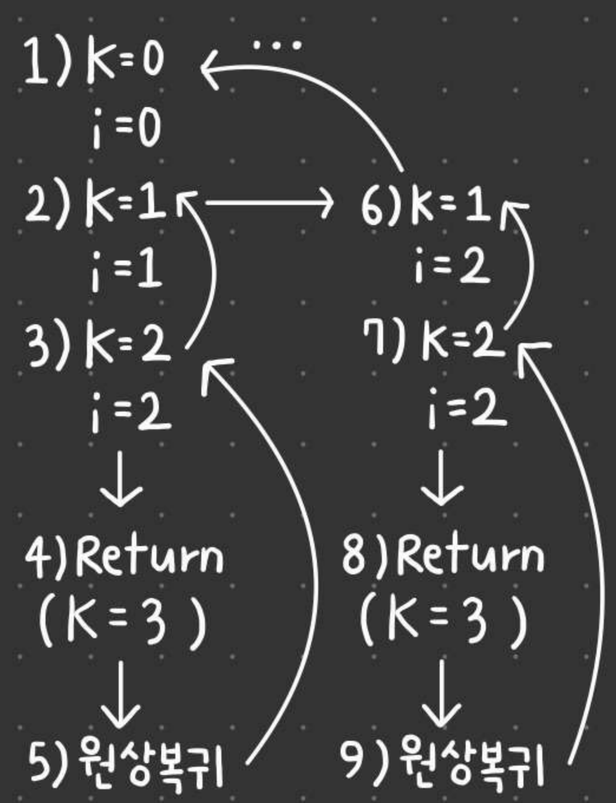

```python
arr = [0, 1, 2]
N = len(arr)
for i in range(0, N):
    arr[0] , arr[i] = arr[i] , arr[0]
    #================================
    for j in range(1, N):
        arr[1], arr[j] = arr[j], arr[1]
        print(arr)                      # [0, 1, 2] [0, 2, 1] [1, 0, 2] [1, 2, 0] [2, 1, 0] [2, 0, 1]
        arr[1], arr[j] = arr[j], arr[1] # 원상 복귀
    #================================
    arr[0], arr[i] = arr[i], arr[0]     # 원상 복귀
```

재귀

```python
arr = [0, 1, 2]
N = len(arr)

def perm(k, N):
    if k == N:
        print(arr)                          # [0, 1, 2] [0, 2, 1] [1, 0, 2] [1, 2, 0] [2, 1, 0] [2, 0, 1]
    else:
        for i in range(k, N):
            arr[k], arr[i] = arr[i], arr[k]
            perm(k + 1, N)
            arr[k], arr[i] = arr[i], arr[k]   # 원상 복귀

perm(0, N)
```



(2) 문제에 적용

0224

```python
arr = [[2, 1, 2],
       [5, 8, 5],
       [7, 2, 2]]

N =3
cols = [i for i in range(N)]

def perm(k):
    if k == N:
        print(cols, end= ' ')
        S = 0
        for i in range(N):  # i는 행 값, cols[i]는 열 값
            S += arr[i][cols[i]]
        print(S)

    else:
        for i in range(k, N):
            cols[k], cols[i] = cols[i], cols[k]
            perm(k + 1)
            cols[k], cols[i] = cols[i], cols[k]   # 원상 복귀

perm(0)
```

출력

```
[0, 1, 2] 12
[0, 2, 1] 9
[1, 0, 2] 8
[1, 2, 0] 13
[2, 1, 0] 17
[2, 0, 1] 9
```

```python
arr = [[2, 1, 2],
       [5, 8, 5],
       [7, 2, 2]]

N =3
cols = [i for i in range(N)]

def perm(k):
    if k == N:
        global ans
        S = 0
        for i in range(N):  # i는 행 값, cols[i]는 열 값
            S += arr[i][cols[i]]
        if ans > S:
            ans = S

    else:
        for i in range(k, N):
            cols[k], cols[i] = cols[i], cols[k]
            perm(k + 1)
            cols[k], cols[i] = cols[i], cols[k]   # 원상 복귀

ans = 9* 10
perm(0)
print(ans)    # 8
```

(3) 가지치기

- 마지막에 다 더한거나 하나씩 누적해서 더한 것은 같다
  - 첫번째 위치가 결정되었을 때 결정된 값을 구한다.
  - 두 번째 위치가 결정되었을 때 결정된 값을 구한다.
  - ⋮
- 누적합을 기존의 최솟값과 비교
  - 누적합이 기존의 최솟값보다크면, 더 이상 계산하지 않고 Return

```python
arr = [[2, 1, 2],
       [5, 8, 5],
       [7, 2, 2]]

N =3
cols = [i for i in range(N)]
def perm(k, cur_sum): # cur_sum : 지금까지 선택한 요소들의 합
    global ans
    if cur_sum >= ans:     # 누적합이 이미 최솟값보다 크다면, 리턴
        return
    if k == N:
        if ans > cur_sum:
            ans = cur_sum
    else:
        for i in range(k, N):
            cols[k], cols[i] = cols[i], cols[k]
            perm(k + 1, cur_sum + arr[k][cols[k]])   # arr[k][cols[k]]
            cols[k], cols[i] = cols[i], cols[k]

ans = 9* 10   # 최솟 값 (기본 설정을 최댓값으로 설정)
perm(0, 0)
print(ans)    # 8
```

K번행 cols[k]번 열

ex) K=1 i=2

0	1	2	→	0	2	1

1번행 2번열을 의미!
즉, 5를 가르킨다