

7장 프로그래밍 실습_파이썬 코드

[프로그래밍 실습 1]

```
from __future__ import print_function
from sys import stdin
import sys

def printf(str, *args):
    print(str % args, end='')

domain = 2
map_power_set = list()
mat_size = 9

# 집합의 크기를 위한 상수
# 멱집합 원소들을 저장하기 위한 map 구조
# 멱집합에서 부분집합을 만족하는 관계의 크기를 위한 변수

# setout 함수가 주어진 집합을 출력
def setout(A):
    count = 0
    if 1 not in A:
        print("\u2205")
        return
    printf("{")
    for i in range(0, domain):
        if A[i] == 1:
            if count != 0:
                printf(",")
            printf("%2d"%(i+1))
            count = 1

    printf(" }\n")

# printPowerSet 함수가 멱집합을 계산하면서 출력
def printPowerSet(set, set_size):

    pow_set_size = 1 << set_size
    counter = 0
    j = 0
    element = ""
    print("P(A) = { \u2205", end = "")

    map_power_set.append("\u2205")

    for counter in range(1, pow_set_size):
        print(" , { ", end = "")
        for j in range(0, set_size):
            # counter의 (오른쪽부터) j번째 비트가 1이면 set[j]를 출력
            if (counter & (1 << j)) > 0:
                element = element + str(j+1)
                print(j+1, end = ", ")

        print("\b\b }", end = "")
        map_power_set.append(element)
        element = ""
        # 멱집합의 다음 원소를 map에 추가
        # 새로운 다른 원소를 받기 위해 재초기화
        print("}")

# 순서쌍을 출력하는 함수
```

```

def print_pair(i, j):
    if map_power_set[i] == "\u2205" and map_power_set[j] == "\u2205" :
        print("(", map_power_set[i], " ,", map_power_set[j], " ),", end = "", sep = '')
    elif map_power_set[i] == "\u2205" and map_power_set[j] != "\u2205" :
        print("(", map_power_set[i], " , {", end = "", sep = '')
        for c in map_power_set[j]:
            print(c, end = ", ")
        print("\b\b})", end = "")
    else:
        print("{", end = "")
        for c in map_power_set[i]:
            print(c, end = ", ")
        print("\b\b}, {", end = "")
        for c in map_power_set[j]:
            print(c, end = ", ")
        print("\b\b})", end = "")

# init_matrix P(A)에 대한 관계행렬을 adjacency에 대입
def init_matrix(adjacency):
    adjacencymx = [[0 for j in range(len(adjacency[0]))] for i in range(len(adjacency))]

    # P(A)에 대한 관계를 저장하는 관계행렬 adjacencymx의 초기화 및 관계 출력
    print("R = (P(A), \u2286 ) = { ", end = "")
    for i in range(0, len(map_power_set)):
        for j in range(0, len(map_power_set)):
            if i <= j:
                # 부분집합 관계를 표현하기 위한 조건
                # {1}이 {2}의 부분집합이 아니므로 반복을 건너뛴다
                if i == 1 and j == 2:
                    continue
                adjacencymx[i][j] = 1
                print_pair(i, j)
            # 순서쌍 출력

    print("\b\b }")
    return adjacencymx
setA = [1 for _ in range(0, domain)] # 집합 A를 위한 0으로 초기화된 배열

# 관계(부울)행렬을 위한 2차원 배열 정의 및 초기화
adjacencymx = [[0 for j in range(mat_size)] for i in range(mat_size)]

print("\nA =", end = " ")
setout(setA) # 집합 A를 출력
printPowerSet(setA, domain) # 집합 A의 멱집합을 출력
adjacencymx = init_matrix(adjacencymx) # A의 멱집합에서 부분집합을 만족하는 관계행렬을 대입

reflexivity = 1
# 관계행렬에서 반사관계 여부를 확인
for i in range(0, len(map_power_set)):
    # iRi가 없으면 바로 반복문을 멈춘다
    if adjacencymx[i][i] == 0:
        reflexivity = 0
        break

if reflexivity == 1:
    print("\n{ ", end = "")
    for i in range(0, len(map_power_set)):
        print_pair(i, i)
    # 순서쌍 출력

    print("\b\b } \u2286 R이므로 R은 반사관계이다.\n")
else:

```

```

print("R은 반사관계가 아니므로 부분순서관계도 아니다.\n")
sys.exit()      # R은 부분순서관계가 아니므로 프로그램을 멈춤

print("반사관계를 만족하는 R의 부분집합은 a, b \u2208 A, (a, b) \u2208 R이고 (b, a) \u2208 R 일 때 a = b를 만족한다.", end = "\n")
for i in range(0, len(map_power_set)):
    for j in range(0, len(map_power_set)):
        if i < j :      # 반대칭관계를 만족하는 원소 출력 및 이유 해석
            print_pair(i, j)      # 순서쌍 출력
            print("에 대해", end = " ")
            print_pair(i, j)      # 순서쌍 출력
            print("\u2208 R이지만", end = " ")
            print_pair(i, j)      # 순서쌍 출력
            print("\u2209 R이므로 만족한다.", end = "\n")

print("\b\b그러므로 반대칭관계이다.\n")

transitivity = 1
test = 1
# 관계행렬에서 추이관계 여부를 확인
for i in range(0, len(map_power_set)):
    for j in range(0, len(map_power_set)):
        if adjacencymx[i][j] == 1:
            for k in range(0, len(map_power_set)):
                # iRj와 jRk가 있을 때 iRk가 있으면
                if adjacencymx[j][k] == 1:
                    if adjacencymx[i][k] == 1:
                        print_pair(i, j)      # 순서쌍 출력
                        print("\b\u2208 R이고", end = " ")
                        print_pair(j, k)      # 순서쌍 출력
                        print("\b\u2208 R이면", end = " ")
                        print_pair(i, k)      # 순서쌍 출력
                        print("\b\u2208 R이 성립한다.", end = "\n")
                    else:
                        transitivity = 0
                        test = 0
                        break
            if test == 0:
                print("R은 추이관계가 아니므로 부분순서관계도 아니다.\n")
                sys.exit()      # R은 부분순서관계가 아니므로 프로그램을 멈춤

if test == 0:
    break
print("\b\b따라서 추이관계를 만족한다.\n")

# 부분순서관계를 하세 도형으로 출력
print("그러므로 A의 멍집합 P(A)에서 \u2286을 만족하는 관계 R은 부분순서관계이다. 이 관계를 하세 도형으로 변환하면 다음과 같다.\n")

print("
                                {1, 2}      ")
print("
                                /  \      ")
print("
                                /    \      ")
print("
                                {1}    {2}  ")
print("
                                \    /      ")
print("
                                \  /      ")
print("
                                \u2205      ")

```

[프로그래밍 실습 2]

```
from __future__ import print_function
from sys import stdin
import sys

def printf(str, *args):
    print(str % args, end='')

domain = 21      # 집합의 크기를 표현하는 상수
rel_size = 18    # 역집합에서 나누어떨어진다는 것을 만족하는 관계의 크기를 위한 변수

# 순서쌍을 출력하는 함수
def print_pair(i, map_power_set, l, r):
    print(l, map_power_set[i][0], ", ", map_power_set[i][1], r, end = ",", sep = '')

# matrixout 함수가 정방행렬 mx를 출력
def matrixout(mx, size):
    print("\u250c" + "\u2510" * (size - 1) + "\u2510")
    for i in range(0, size):
        print("| ", end = "")
        for j in range(0, size):
            print("%3d" % mx[i][j], end = " ")
        print(" |")
    print("\u2514" + "\u2518" * (size - 1) + "\u2518")

# setout 함수가 주어진 집합을 출력
def setout(A):
    count = 0
    if 1 not in A:
        print("\u2205")
        return
    printf("{")
    for i in range(0, domain):
        if A[i] == 1:
            if count != 0:
                printf(", ")
            printf("%d"%(i))
            count = 1

    printf("}")

setB = [0 for _ in range(0,domain)] # 집합 B를 위한 배열 선언
relationD = [] # 관계 D를 위한 이차원 배열 선언
lub_list = [] # 각 원소에 대한 첫 단계의 상계를 저장하는 리스트 구조
glb_list = [] # 각 원소에 대한 첫 단계의 하계를 저장하는 리스트 구조
count = 0 # 집합 B가 가질 원소의 개수를 위한 변수
relation_number = 0 # 격자가 가질 관계의 원소 개수를 위한 변수

# 20보다 작은 자연수에 대해 i|20을 만족하는 원소를 찾음
for i in range(1, domain):
    if 20 % i == 0: # i|20이면
        setB[i] = 1
        count = count + 1

#setB[19] = 1 # 마지막으로 20도 집합 B에 추가
setA = [1 for _ in range(0,count)] # 원소들을 직접 저장하기 위한 배열
index = 0
```

```

for i in range(1, domain):
    if setB[i] == 1:
        setA[index] = i
        index = index + 1

# 집합 D에 대한 관계행렬 선언
adjacencymx = [[0 for j in range(count)] for i in range(count)]
adjacencymx_lattice = [[0 for j in range(count)] for i in range(count)]
print("\nD\u2082\u2080 = ", end = " ")
setout(setB) # 계산된 집합 A를 출력

# 집합 D의 원소에 대해 i|j를 만족하는 쌍을 찾음
for i in range(0, count):
    for j in range(i, count):
        if i <= j:
            relationD.append([setA[i],setA[j]])
            relation_number = relation_number + 1
        if setA[j] % setA[i] == 0: # i|j이면
            adjacencymx[i][j] = 1
            adjacencymx_lattice[i][j] = 1

print("이고 부분순서관계 R = (D\u2082\u2080, ≤) = { ")
for i in range(0, relation_number):
    if relationD[i][1] % relationD[i][0] == 0:
        print_pair(i, relationD, '(', ')')

print("\b}이다. 하세 도형은 다음과 같다.\n")
print("          20  ")
print("        /  \  ")
print("       4   10  ")
print("      \ /  |  ")
print("       2   5  ")
print("        \ /  ")
print("       1   \n")
print("\u201c집합 D\u2099의 임의의 두 원소에 대한 상한과 하한은 다음과 같다.\n")

# 관계행렬에 존재하는 추이관계와 반사관계를 만족하는 원소들을 제거함
for i in range(0, count):
    adjacencymx_lattice[i][i] = 0
    for j in range(0, count):
        if adjacencymx_lattice[i][j] == 1:
            for k in range(0, count):
                # iRj가 있을 때 jRk와 iRk가 있으면
                if adjacencymx_lattice[j][k] == 1 and adjacencymx_lattice[i][k] == 1 and j !=
k:
                    adjacencymx_lattice[i][k] = 0 # 추이관계인 원소 iRk를 제거

flag_break = False
for i in range(0, relation_number):
    first = setA.index(relationD[i][0]) # 관계의 첫 번째 원소의 인덱스를 대입
    second = setA.index(relationD[i][1]) # 관계의 두 번째 원소의 인덱스를 대입
    if first > second: # 첫 번째 원소가 두 번째 원소보다 크면 다음 반복으로 건너뛸
continue
    if first == second: # 관계의 두 원소가 같으면 상한도 하한도 같음
        print_pair(i, relationD, '{', '}')
        print("\b의 상한은 ", relationD[i][0], ", 하한은 ", relationD[i][1], "이다.", end="",
sep = '')
    else:

```

```

# 해당 이중 반복문으로 관계에 대한 상한을 찾고 출력
for j in range(first + 1, count):
    for k in range(second, count):
        # 관계의 원소들의 상계 중에 상한과 같은 것을 확인
        if j == k and adjacencymx[first][j] == 1 and adjacencymx[second][k] == 1:
            # 관계와 상한을 출력
            print_pair(i, relationD, '{', '}')
            print("\b의 상한은 ", setA[j], end=",", sep = '')
            flag_break = True # 상한을 찾은 후 반복문을 나감
            break
    if flag_break == True: # 상한을 찾은 후 바깥쪽 반복문도 나감
        flag_break = False
        break

# 해당 이중 반복문으로 관계에 대한 하한을 찾고 출력
for j in range(first, -1, -1):
    for k in range(second - 1, -1, -1):
        # 관계의 원소들의 하계 중에 하한과 같은 것을 확인
        if j == k and adjacencymx[j][first] == 1 and adjacencymx[k][second] == 1:
            # 관계와 하한을 출력
            print("하한은 ", setA[j], "이다.", end="", sep = '')
            flag_break = True # 하한을 찾은 후 반복문을 나감
            break
    if flag_break == True: # 하한을 찾은 후 바깥쪽 반복문도 나감
        flag_break = False
        break

print()

print("따라서 임의의 두 원소에 대한 상한과 하한이 하나씩 존재하므로 격자이다.\n\n")

```