1. def max\_regions\_colored(adj\_list, n):

from collections import defaultdict

graph = defaultdict(list)

for u, v in adj\_list:

graph[u].append(v)

graph[v].append(u)

colors = {}

max\_colored = 0

def is\_safe(region, color):

for neighbor in graph[region]:

if colors.get(neighbor) == color:

return False

return True

def color\_regions(region):

nonlocal max\_colored

if region not in colors:

for color in range(1, n+1):

if is\_safe(region, color):

colors[region] = color

max\_colored += 1

break

for region in range(n):

color\_regions(region)

return max\_colored

edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]

n = 4

print(max\_regions\_colored(edges, n))

**Output:** 2

2. def max\_regions\_colored(adj\_list, n, k):

from collections import defaultdict

graph = defaultdict(list)

for u, v in adj\_list:

graph[u].append(v)

graph[v].append(u)

colors = {}

max\_colored = 0

def is\_safe(region, color):

for neighbor in graph[region]:

if colors.get(neighbor) == color:

return False

return True

def color\_regions(region):

nonlocal max\_colored

if region not in colors:

for color in range(1, k+1):

if is\_safe(region, color):

colors[region] = color

max\_colored += 1

break

for region in range(n):

return max\_colored

edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]

n = 4

k = 3

print(max\_regions\_colored(edges, n, k))

**Output**: 2

3. def has\_hamiltonian\_cycle(edges, n):

from collections import defaultdict

graph = defaultdict(list)

for u, v in edges:

graph[u].append(v)

graph[v].append(u)

def is\_hamiltonian(path):

if len(path) == n:

if path[0] in graph[path[-1]]:

return True

return False

last = path[-1]

for neighbor in graph[last]:

if neighbor not in path:

if is\_hamiltonian(path + [neighbor]):

return True

return False

for start in range(n):

if is\_hamiltonian([start]):

return True

return False

edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2), (2, 4), (4, 0)]

n = 5

print(has\_hamiltonian\_cycle(edges, n))

**Output:** True

4. def has\_hamiltonian\_cycle(edges, n):

from collections import defaultdict

graph = defaultdict(list)

for u, v in edges:

graph[u].append(v)

graph[v].append(u)

def is\_hamiltonian(path):

if len(path) == n:

if path[0] in graph[path[-1]]:

return True

return False

last = path[-1]

for neighbor in graph[last]:

if neighbor not in path:

if is\_hamiltonian(path + [neighbor]):

return True

return False

for start in range(n):

if is\_hamiltonian([start]):

return True

return False

edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]

n = 4

print(has\_hamiltonian\_cycle(edges, n))

**Output:** True

5. def generate\_subsets(S):

S.sort()

subsets = []

n = len(S)

def backtrack(start, path):

subsets.append(path)

for i in range(start, n):

if i > start and S[i] == S[i-1]:

continue

backtrack(i + 1, path + [S[i]])

backtrack(0, [])

return subsets

A = [1, 2, 3]

print(generate\_subsets(A))

**Output**: [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]

6. def subsets\_with\_x(E, x):

subsets = []

n = len(E)

def backtrack(start, path):

if x in path:

subsets.append(path)

for i in range(start, n):

backtrack(i + 1, path + [E[i]])

backtrack(0, [])

return subsets

E = [2, 3, 4, 5]

x = 3

print(subsets\_with\_x(E, x))

**Output:** [[3], [2, 3], [3, 4], [3, 5], [2, 3, 4], [2, 3, 5], [3, 4, 5], [2, 3, 4, 5]]

7. def find\_universal\_strings(words1, words2):

from collections import Counter

max\_freq = {}

for word in words2:

word\_freq = Counter(word)

for char, freq in word\_freq.items():

if char in max\_freq:

max\_freq[char] = max(max\_freq[char], freq)

else:

max\_freq[char] = freq

universal = []

for word in words1:

word\_freq = Counter(word)

is\_universal = True

for char, freq in max\_freq.items():

if word\_freq.get(char, 0) < freq:

is\_universal = False

break

if is\_universal:

universal.append(word)

return universal

words1 = ["amazon","apple","facebook","google","leetcode"]

words2 = ["e","o"]

print(find\_universal\_strings(words1, words2))

**Output:** ["facebook","google","leetcode"]