main.py

Share Run

```python
def color_map(edges, n):
    adj_list = [[] for _ in range(n)]
    for u, v in edges:
        adj_list[u].append(v)
        adj_list[v].append(u)
    colors = [-1] * n
    for u in range(n):
        neighbor_colors = {colors[v] for v in adj_list[u] if colors[v]
            != -1}
        colors[u] = next(color for color in range(n) if color not in
            neighbor_colors)
    turn, your_count = 0, 0
    for u in range(n):
        if turn == 0:
            your_count += 1
        turn = (turn + 1) % 3
    return colors, your_count
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
colors, max_your_regions = color_map(edges, n)
print("Coloring of vertices:", colors)
print("Maximum regions you can color:", max_your_regions)

```

Output

```
Coloring of vertices: [0, 1, 2, 1]
Maximum regions you can color: 2

=== Code Execution Successful ===
```

main.py

Share Run

```python
def color_map(edges, n, k):
    adj = [[] for _ in range(n)]
    for u, v in edges:
        adj[u].append(v)
        adj[v].append(u)
    colors = [-1] * n
    for u in range(n):
        used = {colors[v] for v in adj[u] if colors[v] != -1}
        colors[u] = next(c for c in range(k) if c not in used)
    your_turns = sum(1 for i in range(n) if i % 3 == 0)
    return colors, your_turns
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n, k = 4, 3
print(color_map(edges, n, k))

```

Output

```
([0, 1, 2, 1], 2)

=== Code Execution Successful ===
```

main.py

Share Run

Output

```python
from collections import defaultdict
def has_hamiltonian_cycle(edges, n):
    def dfs(v, depth):
        if depth == n:
            return start in graph[v]
        visited[v] = True
        for neighbor in graph[v]:
            if not visited[neighbor] and dfs(neighbor, depth + 1):
                return True
        visited[v] = False
        return False
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)
    start, visited = 0, [False] * n
    return dfs(start, 1)
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2), (2, 4), (4, 0)]
n = 5
print(has_hamiltonian_cycle(edges, n))
```

```
False

=== Code Execution Successful ===
```

main.py

Share Run

Output

```python
from collections import defaultdict
def has_hamiltonian_cycle(edges, n):
    def dfs(v, depth):
        if depth == n:
            return start in graph[v]
        visited[v] = True
        for neighbor in graph[v]:
            if not visited[neighbor] and dfs(neighbor, depth + 1):
                return True
        visited[v] = False
        return False
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)
    start, visited = 0, [False] * n
    return dfs(start, 1)
edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
n = 4
print(has_hamiltonian_cycle(edges, n))  # Output: True

```

```
True

=== Code Execution Successful ===
```

main.py

Share Run

```python
def subsets(S):
    def backtrack(start, path):
        subsets_list.append(path[:])
        for i in range(start, len(S)):
            if i > start and S[i] == S[i - 1]:
                continue
            path.append(S[i])
            backtrack(i + 1, path)
            path.pop()
    S.sort()
    subsets_list = []
    backtrack(0, [])
    return subsets_list
A = [1, 2, 3]
print(subsets(A))

```

Output

```
[[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]]

=== Code Execution Successful ===
```

main.py

Share Run

```python
def subsets(nums):
    result = [[]]
    for num in nums:
        result += [curr + [num] for curr in result]
    return result
def subsets_with_element(nums, element):
    all_subsets = subsets(nums)
    return [subset for subset in all_subsets if element in subset]
E = [2, 3, 4, 5]
x = 3
print("Subsets containing 3:", subsets_with_element(E, x))
nums1 = [1, 2, 3]
nums2 = [0]
print("Power set of [1, 2, 3]:", subsets(nums1))
print("Power set of [0]:", subsets(nums2))

```

Output Clear

```
Subsets containing 3: [[3], [2, 3], [3, 4], [2, 3, 4], [3, 5], [2, 3, 5], [3
    , 4, 5], [2, 3, 4, 5]]
Power set of [1, 2, 3]: [[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2,
    3]]
Power set of [0]: [[], [0]]

=== Code Execution Successful ===
```

main.py

Share Run

Output Clear

```python
from collections import Counter
def word_subsets(words1, words2):
    max_count = Counter()
    for b in words2:
        count_b = Counter(b)
        for char, freq in count_b.items():
            max_count[char] = max(max_count[char], freq)
    universal_words = []
    for a in words1:
        count_a = Counter(a)
        if all(count_a[char] >= freq for char, freq in max_count.items
            ()):
            universal_words.append(a)
    return universal_words
words1 = ["amazon","apple","facebook","google","leetcode"]
words2 = ["e","o"]
print(word_subsets(words1, words2))
words2 = ["l","e"]
print(word_subsets(words1, words2))

```

```
['facebook', 'google', 'leetcode']
['apple', 'google', 'leetcode']

=== Code Execution Successful ===
```