

main.py



Run

Output

```
1 def color_map(edges, n):
2     adj_list = [[] for _ in range(n)]
3     for u, v in edges:
4         adj_list[u].append(v)
5         adj_list[v].append(u)
6     colors = [-1] * n
7     for u in range(n):
8         neighbor_colors = {colors[v] for v in adj_list[u] if colors[v]
9                             != -1}
10        colors[u] = next(color for color in range(n) if color not in
11                           neighbor_colors)
12    turn, your_count = 0, 0
13    for u in range(n):
14        if turn == 0:
15            your_count += 1
16        turn = (turn + 1) % 3
17    return colors, your_count
18
19 edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
20 n = 4
21 colors, max_your_regions = color_map(edges, n)
22 print("Coloring of vertices:", colors)
23 print("Maximum regions you can color:", max_your_regions)
```

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

=== Code Execution Successful ===

main.py



Share

Run

Output

```
1 def color_map(edges, n, k):
2     adj = [[] for _ in range(n)]
3     for u, v in edges:
4         adj[u].append(v)
5         adj[v].append(u)
6     colors = [-1] * n
7     for u in range(n):
8         used = {colors[v] for v in adj[u] if colors[v] != -1}
9         colors[u] = next(c for c in range(k) if c not in used)
10    your_turns = sum(1 for i in range(n) if i % 3 == 0)
11    return colors, your_turns
12 edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
13 n, k = 4, 3
14 print(color_map(edges, n, k))
15
```

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

=== Code Execution Successful ===

main.py



Run

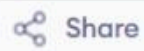
Output

```
1 from collections import defaultdict
2 def has_hamiltonian_cycle(edges, n):
3     def dfs(v, depth):
4         if depth == n:
5             return start in graph[v]
6         visited[v] = True
7         for neighbor in graph[v]:
8             if not visited[neighbor] and dfs(neighbor, depth + 1):
9                 return True
10        visited[v] = False
11        return False
12    graph = defaultdict(list)
13    for u, v in edges:
14        graph[u].append(v)
15        graph[v].append(u)
16    start, visited = 0, [False] * n
17    return dfs(start, 1)
18 edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2), (2, 4), (4, 0)]
19 n = 5
20 print(has_hamiltonian_cycle(edges, n))
```

False

=== Code Execution Successful ===

main.py



Run

Output

```
1 from collections import defaultdict
2 def has_hamiltonian_cycle(edges, n):
3     def dfs(v, depth):
4         if depth == n:
5             return start in graph[v]
6         visited[v] = True
7         for neighbor in graph[v]:
8             if not visited[neighbor] and dfs(neighbor, depth + 1):
9                 return True
10        visited[v] = False
11        return False
12    graph = defaultdict(list)
13    for u, v in edges:
14        graph[u].append(v)
15        graph[v].append(u)
16    start, visited = 0, [False] * n
17    return dfs(start, 1)
18 edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
19 n = 4
20 print(has_hamiltonian_cycle(edges, n)) # Output: True
21
```

True

=== Code Execution Successful ===

main.py



Share

Run

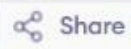
Output

```
1 def subsets(S):
2     def backtrack(start, path):
3         subsets_list.append(path[:])
4         for i in range(start, len(S)):
5             if i > start and S[i] == S[i - 1]:
6                 continue
7             path.append(S[i])
8             backtrack(i + 1, path)
9             path.pop()
10    S.sort()
11    subsets_list = []
12    backtrack(0, [])
13    return subsets_list
14 A = [1, 2, 3]
15 print(subsets(A))
16
```

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

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


main.py



Run

```
1- def subsets(nums):
2     result = [[]]
3-     for num in nums:
4         result += [curr + [num] for curr in result]
5     return result
6- def subsets_with_element(nums, element):
7     all_subsets = subsets(nums)
8     return [subset for subset in all_subsets if element in subset]
9 E = [2, 3, 4, 5]
10 x = 3
11 print("Subsets containing 3:", subsets_with_element(E, x))
12 nums1 = [1, 2, 3]
13 nums2 = [0]
14 print("Power set of [1, 2, 3]:", subsets(nums1))
15 print("Power set of [0]:", subsets(nums2))
16
```

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

```
1 from collections import Counter
2 def word_subsets(words1, words2):
3     max_count = Counter()
4     for b in words2:
5         count_b = Counter(b)
6         for char, freq in count_b.items():
7             max_count[char] = max(max_count[char], freq)
8     universal_words = []
9     for a in words1:
10        count_a = Counter(a)
11        if all(count_a[char] >= freq for char, freq in max_count.items
12              ()):
13            universal_words.append(a)
14    return universal_words
15 words1 = ["amazon","apple","facebook","google","leetcode"]
16 words2 = ["e","o"]
17 print(word_subsets(words1, words2))
18 print(word_subsets(words1, words2))
19
```

['facebook', 'google', 'leetcode']

['apple', 'google', 'leetcode']

=== Code Execution Successful ===