

Programiz
Python Online Compiler

main.py

Share

Run

Output

Clear

```
1 def graph_coloring(adj_list):
2     colors = {}
3     result = 0
4     for node in sorted(adj_list, key=lambda x: len(adj_list[x]),
5         reverse=True):
6         neighbor_colors = set(colors.get(nei) for nei in
7             adj_list[node])
8         colors[node] = next(color for color in range(len
9             (adj_list)) if color not in neighbor_colors)
10        result = max(result, colors[node] + 1)
11    return result
12 adj_list = {
13     0: [1, 3, 2],
14     1: [0, 2],
15     2: [1, 0, 3],
16     3: [2, 0]
17 }
18 max_regions_colored = graph_coloring(adj_list)
19 print("Maximum number of regions colored:", max_regions_colored)
```

Maximum number of regions colored: 3

=== Code Execution Successful ===

Programiz
Python Online Compiler

main.py

Share

Run

Output

Clear

```
1 def graph_coloring(edges, n, k):
2     graph = {}
3     for edge in edges:
4         if edge[0] not in graph:
5             graph[edge[0]] = set()
6         if edge[1] not in graph:
7             graph[edge[1]] = set()
8         graph[edge[0]].add(edge[1])
9         graph[edge[1]].add(edge[0])
10
11    colors = {}
12    for vertex in graph:
13        neighbor_colors = set()
14        for neighbor in graph[vertex]:
15            if neighbor in colors:
16                neighbor_colors.add(colors[neighbor])
17        for color in range(k):
18            if color not in neighbor_colors:
19                colors[vertex] = color
20                break
21
22    return len(set(colors.values()))
23 edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
24 n = 4
25 k = 3
26 max_regions_colored = graph_coloring(edges, n, k)
27 print("Maximum number of regions colored:", max_regions_colored)
```

Maximum number of regions colored: 3

=== Code Execution Successful ===

Programiz

Python Online Compiler

main.py

Share

Run

Output

Clear

```
1 def has_hamiltonian_cycle(edges, n):
2     graph = {i: set() for i in range(n)}
3     for edge in edges:
4         graph[edge[0]].add(edge[1])
5         graph[edge[1]].add(edge[0])
6     def dfs(node, visited, count):
7         visited.add(node)
8         if count == n:
9             return len(graph[node]) == 1 and 0 in graph[node]
10        for neighbor in graph[node]:
11            if neighbor not in visited:
12                if dfs(neighbor, visited, count + 1):
13                    return True
14        visited.remove(node)
15        return False
16
17    for node in range(n):
18        if dfs(node, set(), 1):
19            return True
20    return False
21 edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2), (2, 4), (4, 0)]
22 n = 5
23 print(has_hamiltonian_cycle(edges, n))
```

False

=== Code Execution Successful ===

Programiz

Python Online Compiler

main.py

Share

Run

Output

Clear

```
1 def has_hamiltonian_cycle(edges, n):
2     graph = {i: set() for i in range(n)}
3     for edge in edges:
4         graph[edge[0]].add(edge[1])
5         graph[edge[1]].add(edge[0])
6     def dfs(node, visited, count):
7         visited.add(node)
8         if count == n:
9             return len(graph[node]) == 1 and 0 in graph[node]
10        for neighbor in graph[node]:
11            if neighbor not in visited:
12                if dfs(neighbor, visited, count + 1):
13                    return True
14        visited.remove(node)
15        return False
16
17    for node in range(n):
18        if dfs(node, set(), 1):
19            return True
20    return False
21 edges = [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)]
22 n = 4
23 print(has_hamiltonian_cycle(edges, n))
```

False

=== Code Execution Successful ===

Programiz
Python Online Compiler

main.py

Share

Run

```
1 def subsets(S):
2     S = sorted(set(S))
3     result = []
4     def backtrack(start, path):
5         result.append(path)
6         for i in range(start, len(S)):
7             backtrack(i + 1, path + [S[i]])
8     backtrack(0, [])
9     return result
10 A = [1, 2, 3]
11 print(subsets(A))
12
```

Output

Clear

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

=== Code Execution Successful ===

Programiz
Python Online Compiler

main.py

Share

Run

```
1 from itertools import combinations
2 def subsets(nums):
3     res = []
4     for i in range(len(nums)+1):
5         res += [list(comb) for comb in combinations(nums, i)]
6     return res
7 nums = [1, 2, 3]
8 result = subsets(nums)
9 print(result)
10
```

Output

Clear

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

=== Code Execution Successful ===

Programiz
Python Online Compiler

main.py

Share

Run

```
1 def is_subset(a, b):
2     return all(a.count(char) >= b.count(char) for char in set(b))
3 def universal_strings(words1, words2):
4     return [word for word in words1 if all(is_subset(word, w)
5         for w in words2)]
6 words1 = ["amazon", "apple", "facebook", "google", "leetcode"]
7 words2 = ["e", "o"]
8 print(universal_strings(words1, words2))
9 words2 = ["l", "e"]
10 print(universal_strings(words1, words2))
11
```

Output

Clear

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

=== Code Execution Successful ===