Lab 11. Introduction to algorithms

This is Eleventh Lab for CS 566. This problem was given in lecture.

Task 1. Solve the problem "Number of provinces" from https://leetcode.com/problems/number-of-provinces/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

```
from typing import Optional, List

class Solution:
    def findCircleNum(self, isConnected: List[List[int]]) -> int:
        n = len(isConnected)
        visited = set()

        def dfs(node):
            connections = isConnected[node]
            visited.add(node)
            for city_j in range(n):
                if (city_j not in visited) and (connections[city_j] == 1) and (
                    dfs(city_j)
            return

        num_provinces = 0
        for i in range(n):
            if i not in visited:
                dfs(i)
                num_provinces += 1

        return num_provinces
```

Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

```
#test_case_1
height = [[1,1,0],[1,1,0],[0,0,1]]
expected = 2
actual = Solution().findCircleNum(height)
assert actual==expected, "Mistake in test case 1"

height = [[1,0,0],[0,1,0],[0,0,1]]
expected = 3
```

```
actual = Solution().findCircleNum(height)
assert actual==expected, "Mistake in test case 1"

print("OK")
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Aymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(n^2)

Task 2. Solve the problem "Course Schedule" from https://leetcode.com/problems/course-schedule/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

```
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> boo
        graph = [[] for i in range(numCourses)]
        visited = [0 for i in range(numCourses)]

        def dfs(graph,visited,i):
            if visited[i] == -1:
                return False
            if visited[i] == 1:
                return True
            visited[i] = -1
            for j in graph[i]:
                if not dfs(graph, visited, j):
                    return False
            visited[i] = 1
            return True

        for pair in prerequisites:
            x, y = pair
            graph[x].append(y)
        for i in range(numCourses):
            if not dfs(graph, visited, i):
                return False
        return True
```

```
#test case 1
```

```
numCourses = 2
prerequisites = [[1,0]]
expected = True
actual = Solution().canFinish(numCourses, prerequisites)
assert actual==expected, "Mistake in test case 1"

#test_case_2
numCourses = 2
prerequisites = [[1,0],[0,1]]
expected = False
actual = Solution().canFinish(numCourses, prerequisites)
assert actual==expected, "Mistake in test case 2"

print('OK')
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(m+n)

Time Analysis: O(m+n)

Task 3. Solve the problem "Min Cost To Connect All Points" from https://leetcode.com/problems/min-cost-to-connect-all-points/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

```
class Solution:
    def minCostConnectPoints(self, points: List[List[int]]) -> int:
        n = len(points)
        graph = [[0] * n for i in range(n)]


        for i, (x1, y1) in enumerate(points):
            for j in range(i+1, n):
                x2, y2 = points[j]
                distance = abs(x1 - x2) + abs(y1 - y2)
                graph[i][j] = graph[j][i] = distance

        # print(graph)

        visited = [False] * n
        distances = [float('inf')] * n
        distances[0] = 0

        min_cost = 0

        for i in range(n):
            i = -1
            for j in range(n):
                if not visited[j] and (i == -1 or distances[j] < distances[i]):
                    i = j

            visited[i] = True
            min_cost += distances[i]
            for j in range(n):
                if not visited[j]:
                    distances[j] = min(distances[j], graph[i][j])

        return min_cost
```

```
#test_case_1
prices = [[0,0],[2,2],[3,10],[5,2],[7,0]]
expected = 20
actual = Solution().minCostConnectPoints(prices)
assert actual==expected, "Mistake in test case 1"

prices = [[3,12],[-2,5],[-4,1]]
expected = 18
actual = Solution().minCostConnectPoints(prices)
assert actual==expected, "Mistake in test case 2"

print('OK')
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n^2)

Time Analysis: O(n^2)