HW 11. Introduction to algorithms

This is eleventh homework for CS 566.

Task 1. Solve the problem "Path With Maximum Probability" from https://leetcode.com/problems/path-with-maximum-probability/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)

```
import sys
from typing import List
import heapq

class Solution:
    def maxProbability(self, n: int, edges: List[List[int]], succProb: List[flo
        graph, prob = dict(), dict()
        for i, (u, v) in enumerate(edges):
            graph.setdefault(u, []).append(v)
            graph.setdefault(v, []).append(u)
            prob[u, v] = prob[v, u] = succProb[i]
        #Dijkstra
        h = [(-1, start)]
        seen = set()
        while h:
            p, n = heapq.heappop(h)
            if n == end: return -p
            seen.add(n)
            for nn in graph.get(n, []):
                if nn in seen: continue
                heapq.heappush(h, (p * prob.get((n, nn), 0), nn))
        return 0
```

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
n = 3
edges = [[0,1],[1,2],[0,2]]
succProb = [0.5,0.5,0.2]
start = 0
end = 2
```

```
actual = Solution().maxProbability(n, edges, succProb, start, end)
expected = 0.25
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+m), where n is edges and m is vertices

Time Analysis: O(nlog(m)), where n is edges and m is vertices

Task 2. Solve the problem "Course Schedule II" from https://leetcode.com/problems/course-schedule-ii/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)

```
from collections import defaultdict, deque
class Solution:

    def findOrder(self, numCourses, prerequisites):
        graph = defaultdict(list)
        res = []
        def DFS(node):
            if visited[node] == -1:
                return False
            if visited[node] == 1:
                return True
            visited[node] = -1
            for x in graph[node]:
                if not DFS(x):
                    return False
            visited[node] = 1
            res.append(node)
            return True

        for pair in prerequisites:
            graph[pair[0]].append(pair[1])
        visited = [0 for x in range(numCourses)]
        for x in range(numCourses):
            if not DFS(x):
                return []
        return res
```

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

```
OK
```

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

Memory Analysis: O(n+m), where n is edges and m is vertices

Time Analysis: O(n+m), where n is edges and m is vertices

Theoretical Problem (5 points)

Let G = (V, E) be an undirected weighted graph, and let T be a minimum spanning tree of G. Given a subset of vertices V' ⊆ V:

- Let T' be the subgraph of T induced by V'
- Let G' be the subgraph of G induced by V'

Prove: If T' is connected, then T' is a minimum spanning tree of G'.

Definitions:

- A spanning tree of a graph is a tree that includes all vertices and a subset of edges.
- A minimum spanning tree (MST) is a spanning tree whose sum of edge weights is minimal.
- An induced subgraph contains all edges from the original graph whose endpoints are both in the vertex subset.

Use latex or write the proof on the list of paper and embed it as screenshot to the colab.

Since T' is the subgraph of T induced by V', it means it includes all the vertices of V' and means there is a path in T' between any two vertices in V', thus it forms a spanning tree of G', the subgraph of G induced by V'.

For any other spanning tree T' of G', we apply the cut property of MSTs. Any edge not in T' that connects vertices in V' must have a weight that is at least as large as the maximum edge weight in T'. Since T is an MST of G, it contains the minimum weight edges.

Because T' has a weight less than or equal to any other spanning tree T" of G', we conclude that T' is a minimum spanning tree of G'.