Programming Assignment

Due November 22:

Implement DFS. This is easiest, if you create a class Vertex and a class Graph. The vertex class needs to be hashable, so we need to implement __hash__ and __eq__. The Graph class should be based on adjacency lists. The reason to implement vertices as a class is that it is now easy to adorn a vertex with additional properties such as color and predecessor. You can also generate a field in the Graph class that contains the clock.

Your DFS implementation should incorporate the following code. The dfs function should display the color, discovery time, finishing time, and predecessor of each vertex via a print statement.

```
class Vertex:
""" Hashable class of vertices """
    def __init__(self, id):
        self.id = id
    def __hash__(self):
        return hash(id)
    def __str__(self):
        return str(self.id)
    def __eq__(self, other):
        return self.id == other.id


class Graph:
    def __init__(self, n):
        self.vertices = [Vertex(i) for i in range(n)]
        self.edges = { node : [] for node in self.vertices }

    def __repr__(self):
        result = ''
        for x in self.vertices:
            result += str(x)+': '
            for node in self.edges[x]:
                result += str(node)+', '
            result +='\n'
        return result

    def add_edge(self, x, y):
        self.edges[x].append(y)

    def create_random(n, p):
        result = Graph(n)
        for i in result.vertices:
            for j in result.vertices:
                if i!=j and random.random()<p:
                    result.add_edge(i, j)
        return result

    def dfs(self):
        self.clock = 0
        for node in self.vertices:
            node.disc = -1
            node.fin = -1
            node.color = 'w'
        for vertex in self.vertices:
            if vertex.color=='w':
                … your code here …
```