Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Лабораторна робота №7

Дисципліна: Комп'ютерна дискретна математика

Виконав

Студент групи ПЗПІ-23-3

Харченко Федір Олександрович

Перевірив

Старший викладач каф. ПІ

Терещенко Гліб Юрійович

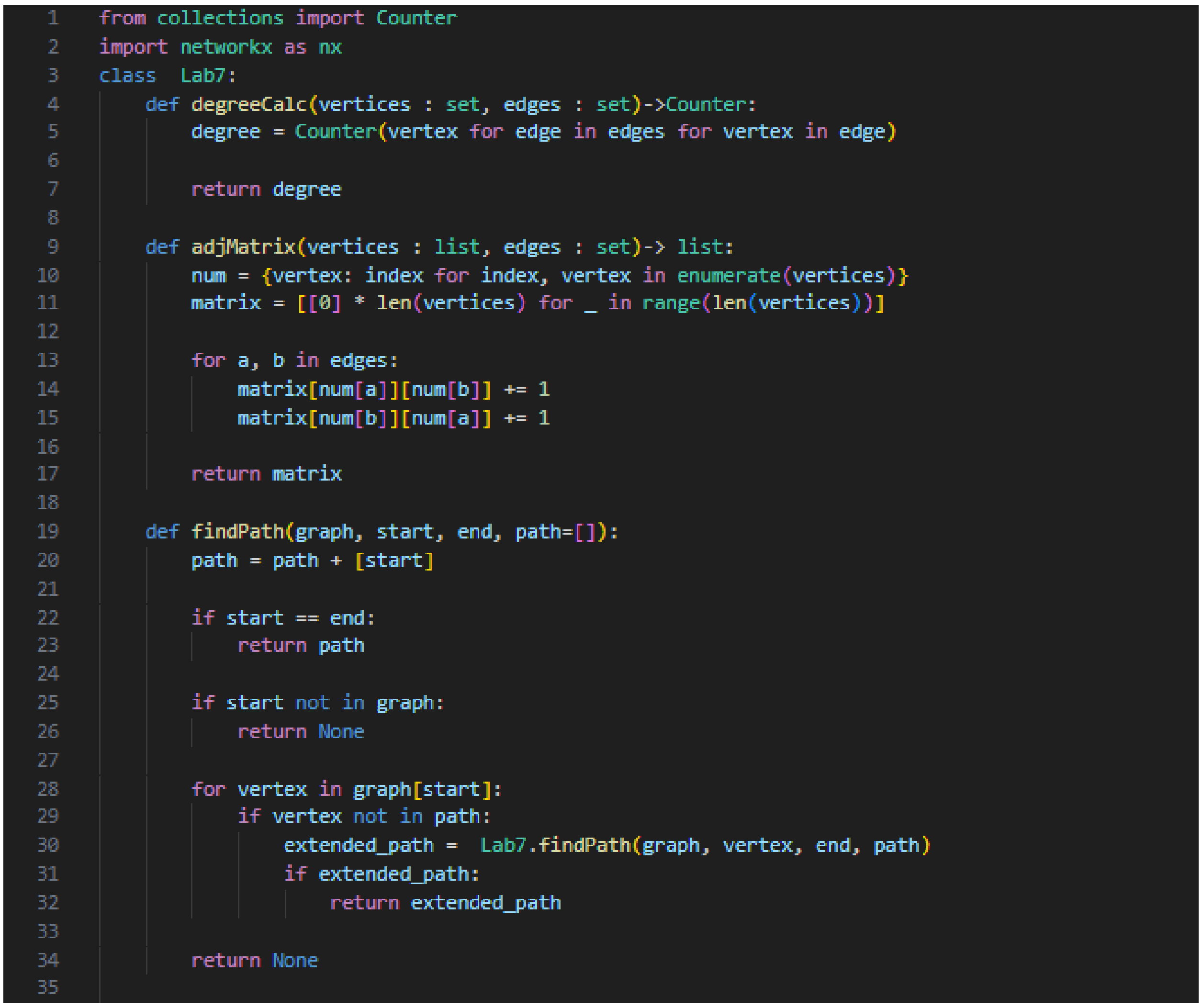
2023

Тема: "Graph Theory"

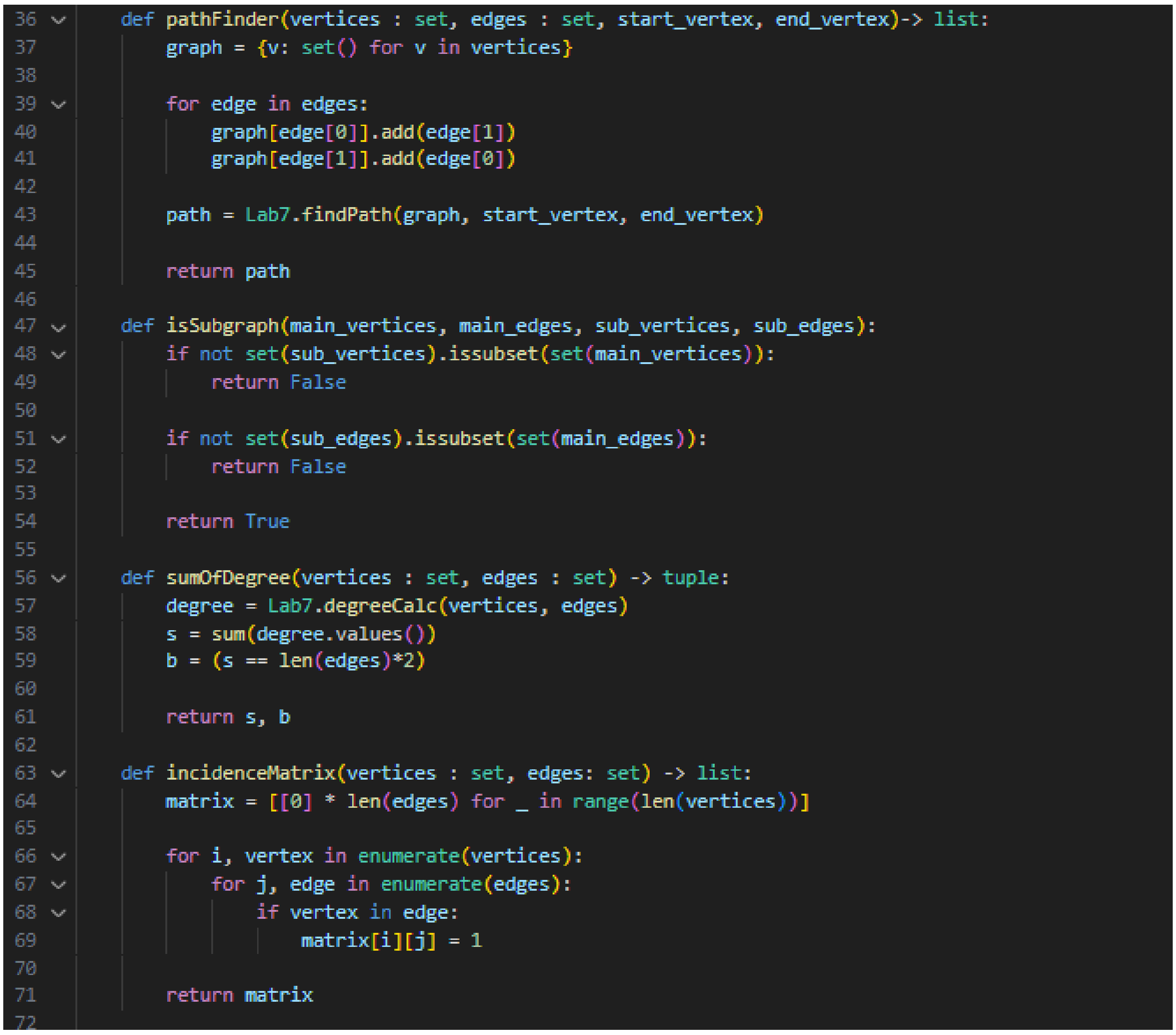
Мета: Understand and implement the graph theory operations

Код програми:

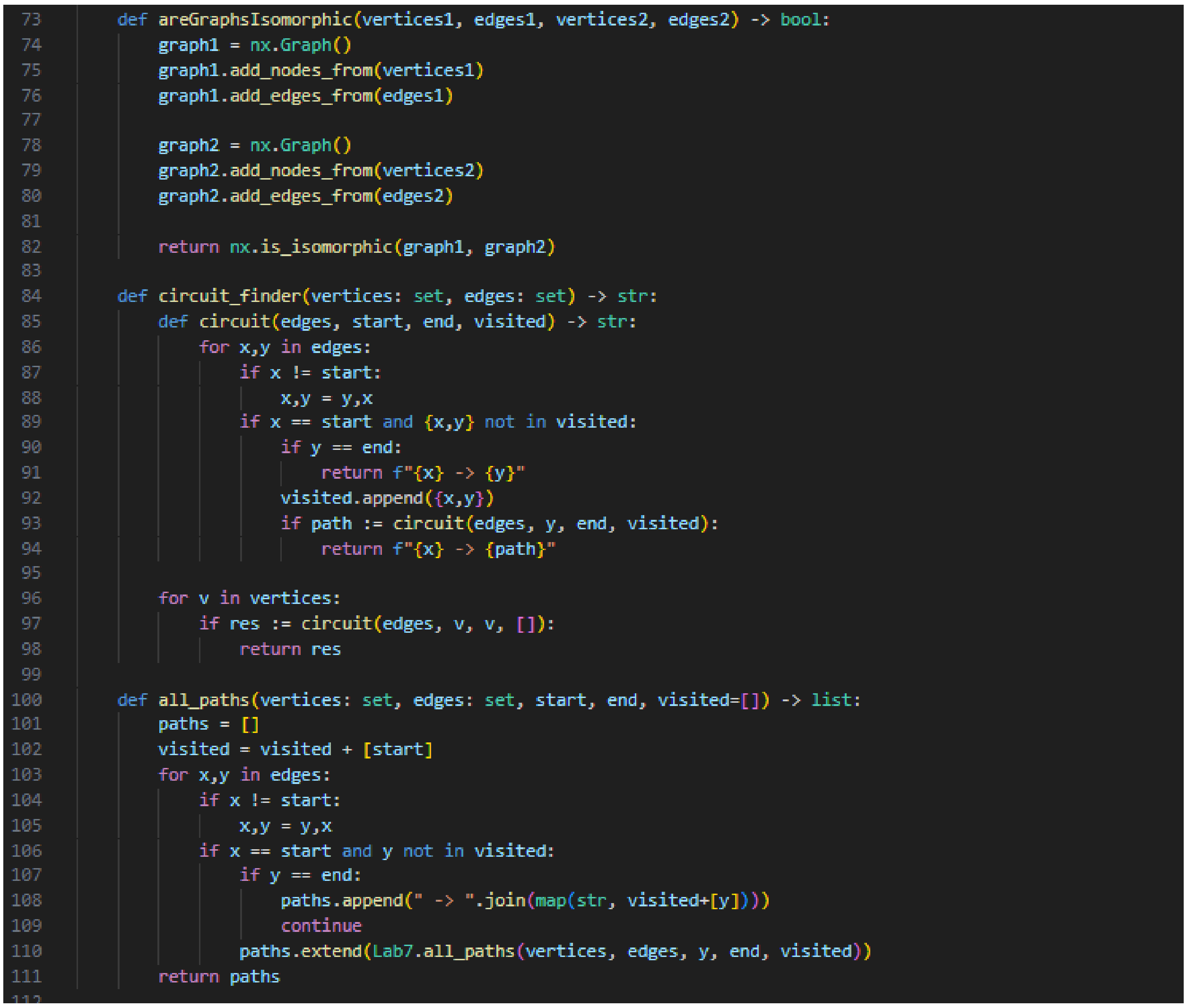
https://github.com/rintaro129/CDM/tree/main/LAB7



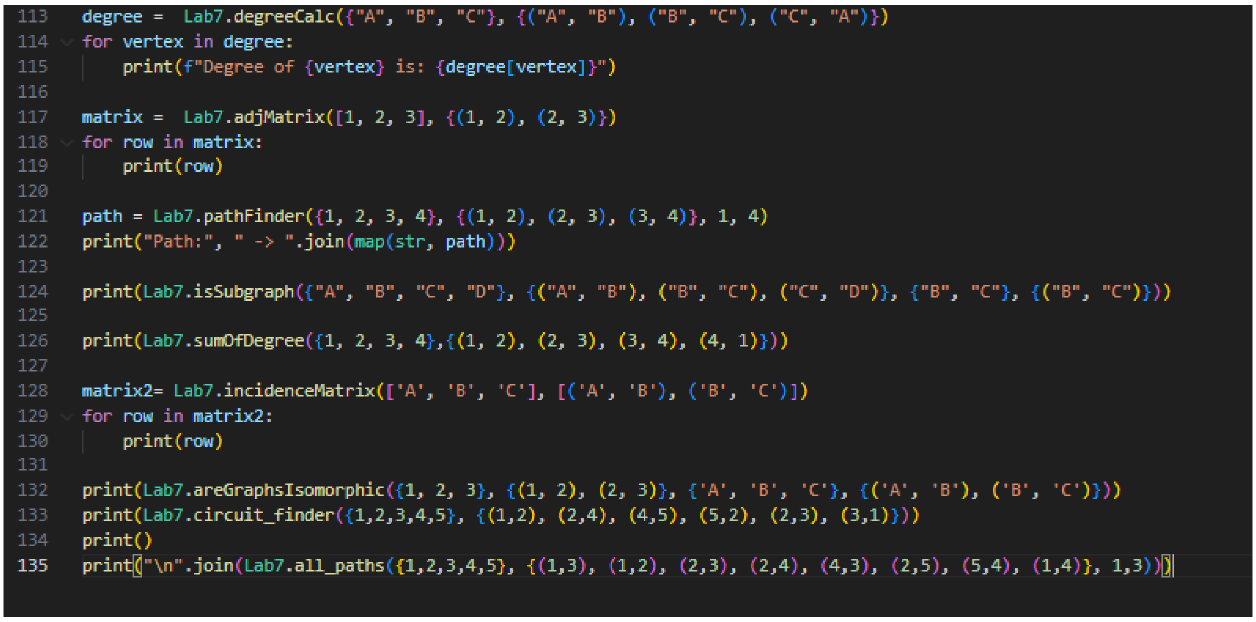
```python
from collections import Counter
import networkx as nx
class  Lab7:
    def degreeCalc(vertices : set, edges : set)->Counter:
        degree = Counter(vertex for edge in edges for vertex in edge)

        return degree

    def adjMatrix(vertices : list, edges : set)-> list:
        num = {vertex: index for index, vertex in enumerate(vertices)}
        matrix = [[0] * len(vertices) for _ in range(len(vertices))]

        for a, b in edges:
            matrix[num[a]][num[b]] += 1
            matrix[num[b]][num[a]] += 1

        return matrix

    def findPath(graph, start, end, path=[]):
        path = path + [start]

        if start == end:
            return path

        if start not in graph:
            return None

        for vertex in graph[start]:
            if vertex not in path:
                extended_path =  Lab7.findPath(graph, vertex, end, path)
                if extended_path:
                    return extended_path

        return None

```



```python
    def pathFinder(vertices : set, edges : set, start_vertex, end_vertex)-> list:
        graph = {v: set() for v in vertices}

        for edge in edges:
            graph[edge[0]].add(edge[1])
            graph[edge[1]].add(edge[0])

        path = Lab7.findPath(graph, start_vertex, end_vertex)

        return path

    def isSubgraph(main_vertices, main_edges, sub_vertices, sub_edges):
        if not set(sub_vertices).issubset(set(main_vertices)):
            return False

        if not set(sub_edges).issubset(set(main_edges)):
            return False

        return True

    def sumOfDegree(vertices : set, edges : set) -> tuple:
        degree = Lab7.degreeCalc(vertices, edges)
        s = sum(degree.values())
        b = (s == len(edges)*2)

        return s, b

    def incidenceMatrix(vertices : set, edges: set) -> list:
        matrix = [[0] * len(edges) for _ in range(len(vertices))]

        for i, vertex in enumerate(vertices):
            for j, edge in enumerate(edges):
                if vertex in edge:
                    matrix[i][j] = 1

        return matrix
```



```python
    def areGraphsIsomorphic(vertices1, edges1, vertices2, edges2) -> bool:
        graph1 = nx.Graph()
        graph1.add_nodes_from(vertices1)
        graph1.add_edges_from(edges1)

        graph2 = nx.Graph()
        graph2.add_nodes_from(vertices2)
        graph2.add_edges_from(edges2)

        return nx.is_isomorphic(graph1, graph2)

    def circuit_finder(vertices: set, edges: set) -> str:
        def circuit(edges, start, end, visited) -> str:
            for x,y in edges:
                if x != start:
                    x,y = y,x
                if x == start and {x,y} not in visited:
                    if y == end:
                        return f"{x} -> {y}"
                    visited.append({x,y})
                    if path := circuit(edges, y, end, visited):
                        return f"{x} -> {path}"

        for v in vertices:
            if res := circuit(edges, v, v, []):
                return res

    def all_paths(vertices: set, edges: set, start, end, visited=[]) -> list:
        paths = []
        visited = visited + [start]
        for x,y in edges:
            if x != start:
                x,y = y,x
            if x == start and y not in visited:
                if y == end:
                    paths.append(" -> ".join(map(str, visited+[y])))
                    continue
                paths.extend(Lab7.all_paths(vertices, edges, y, end, visited))
        return paths
```



```python
degree =  Lab7.degreeCalc({"A", "B", "C"}, {("A", "B"), ("B", "C"), ("C", "A")})
for vertex in degree:
    print(f"Degree of {vertex} is: {degree[vertex]}")

matrix =  Lab7.adjMatrix([1, 2, 3], {(1, 2), (2, 3)})
for row in matrix:
    print(row)

path = Lab7.pathFinder({1, 2, 3, 4}, {(1, 2), (2, 3), (3, 4)}, 1, 4)
print("Path:", " -> ".join(map(str, path)))

print(Lab7.isSubgraph({"A", "B", "C", "D"}, {("A", "B"), ("B", "C"), ("C", "D")}, {"B", "C"}, {("B", "C")}))

print(Lab7.sumOfDegree({1, 2, 3, 4},{(1, 2), (2, 3), (3, 4), (4, 1)}))

matrix2= Lab7.incidenceMatrix(['A', 'B', 'C'], [('A', 'B'), ('B', 'C')])
for row in matrix2:
    print(row)

print(Lab7.areGraphsIsomorphic({1, 2, 3}, {(1, 2), (2, 3)}, {'A', 'B', 'C'}, {('A', 'B'), ('B', 'C')}))
print(Lab7.circuit_finder({1,2,3,4,5}, {(1,2), (2,4), (4,5), (5,2), (2,3), (3,1)}))
print()
print("\n".join(Lab7.all_paths({1,2,3,4,5}, {(1,3), (1,2), (2,3), (2,4), (4,3), (2,5), (5,4), (1,4)}, 1,3)))
```

Висновок: All tasks are implemented successfully.