Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

ЗВІТ

про виконання лабораторного практикуму №1

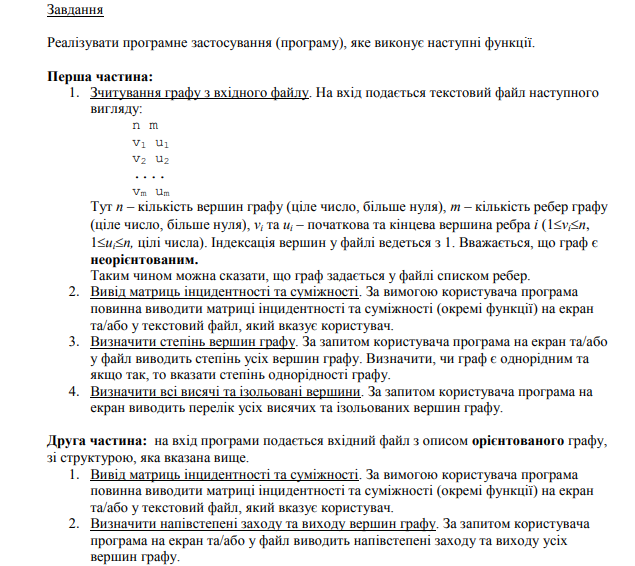
Виконав:

студент 1- го курсу ФІОТ

групи *ІП-91*

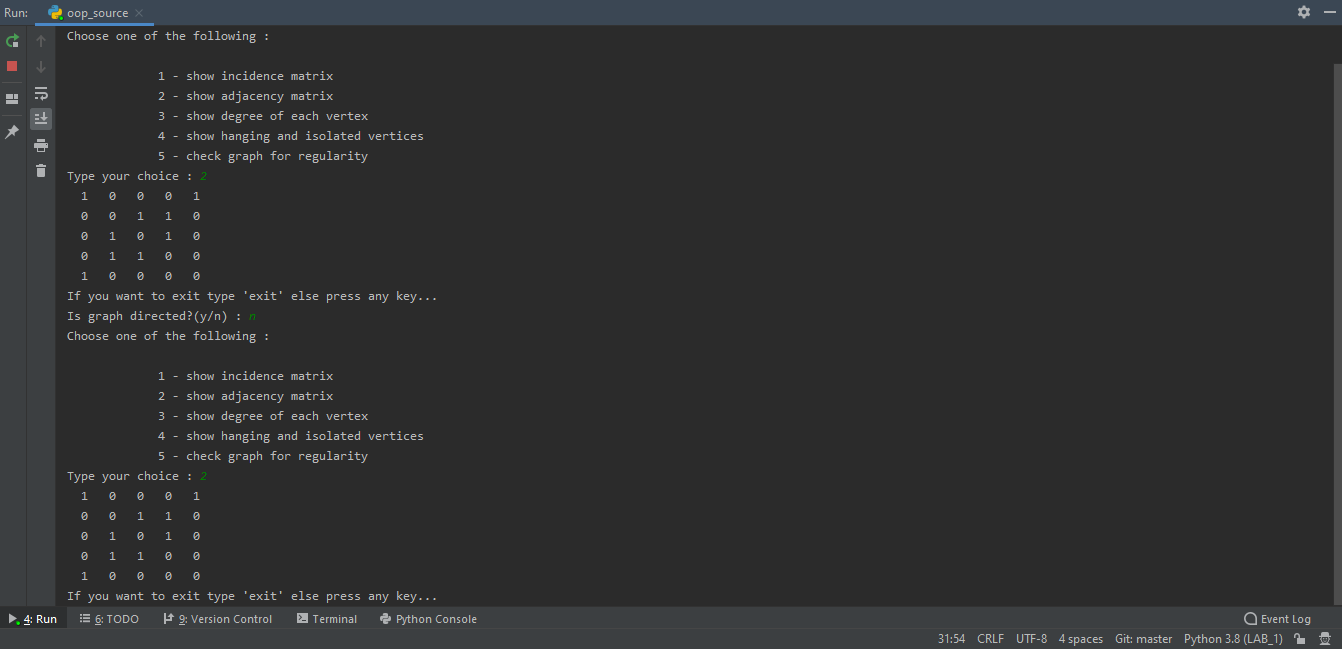
*Кінчур Вадим Вікторович*

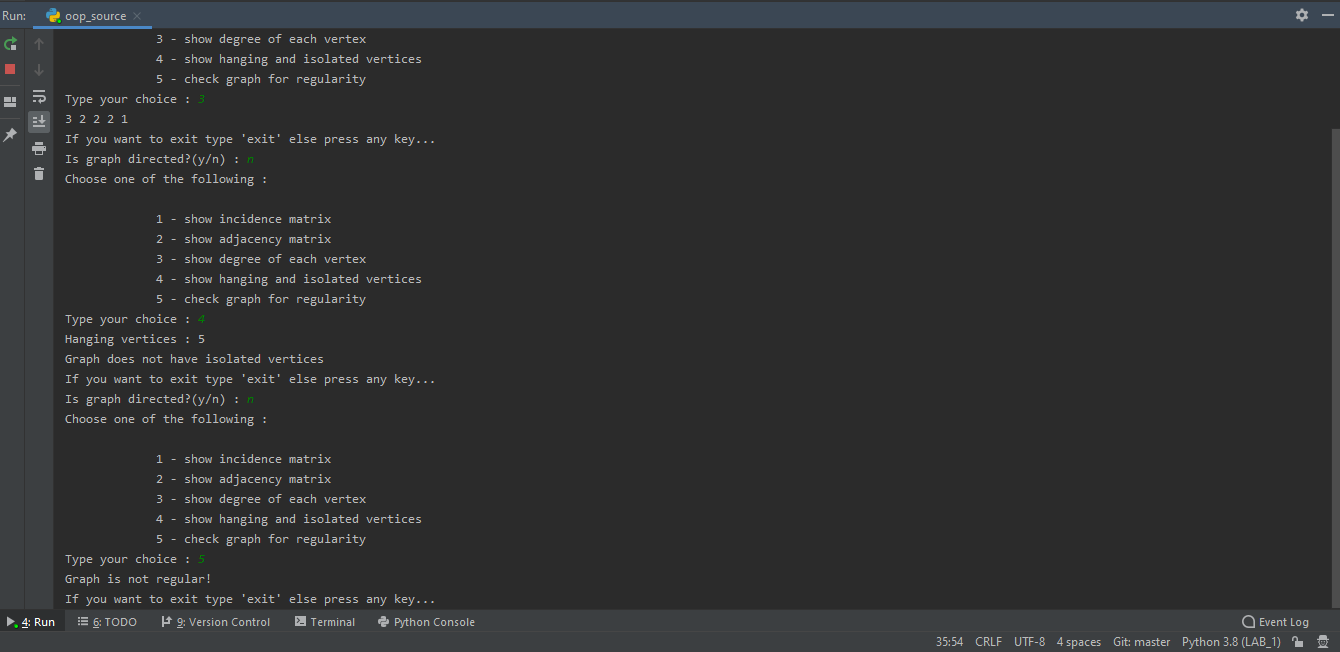
Київ 2020

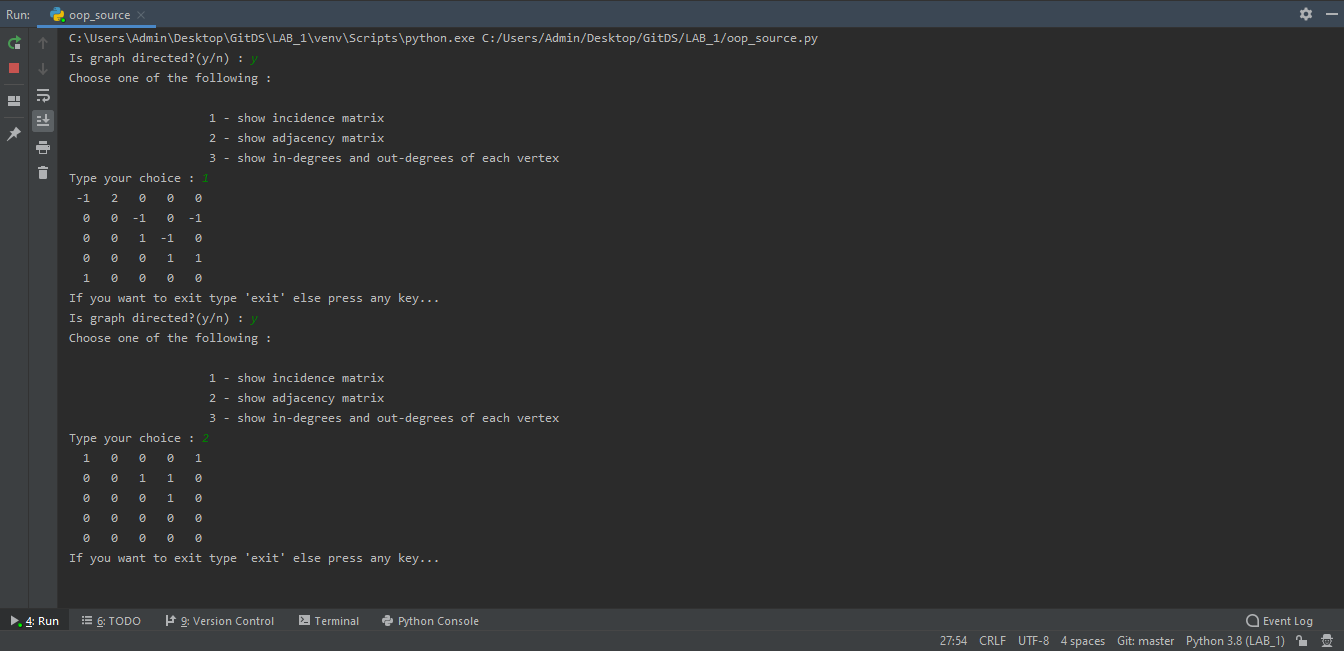
1. Умова лабораторної роботи.
2. Програмний код (Python):

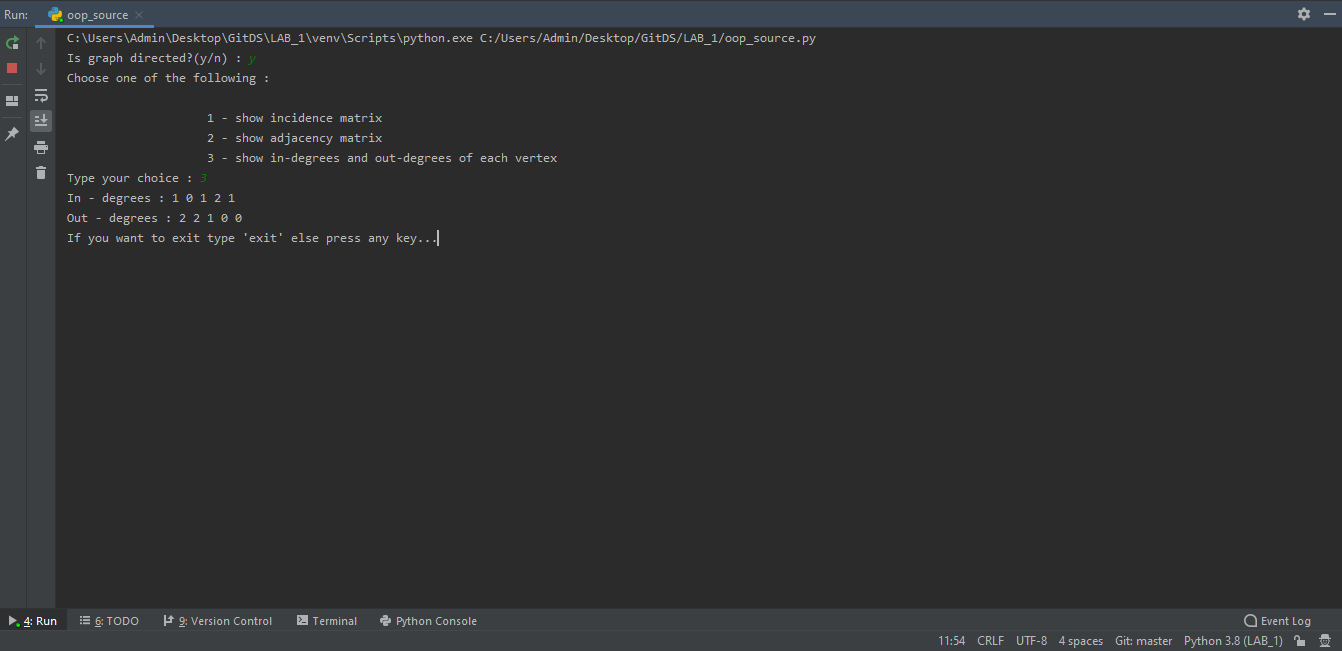
from copy import copy  
  
  
class Graph:  
 def \_\_init\_\_(self, n, m, edges\_list): # Constructor of graph  
 self.\_n = n  
 self.\_m = m  
 self.\_edges\_list = copy(edges\_list)  
  
 def \_\_get\_adjacency\_matrix(self):  
 pass  
  
 def \_\_get\_incidence\_matrix(self):  
 pass  
  
 def \_\_enter\_number(self):  
 pass  
  
 def \_\_show\_menu(self):  
 pass  
  
 def start\_menu(self):  
 pass  
  
 def \_\_work\_with\_number(self, number):  
 pass  
  
  
class NotDirectedGraph(Graph):  
  
 def \_\_get\_vertex\_degrees(self): # Get degree of each vertex  
 incidence\_degree = [0] \* self.\_n  
 for start, finish in self.\_edges\_list:  
 incidence\_degree[start - 1] += 1  
 incidence\_degree[finish - 1] += 1  
 return incidence\_degree  
  
 def \_\_get\_isolated\_vertices(self): # Return isolated vertices  
 vertex\_degree = self.\_\_get\_vertex\_degrees()  
 isolated\_vertices = []  
 for i in range(len(vertex\_degree)):  
 if vertex\_degree[i] == 0:  
 isolated\_vertices.append(i + 1)  
 return isolated\_vertices  
  
 def \_\_get\_hanging\_vertices(self): # Return hanging vertices  
 vertex\_degree = self.\_\_get\_vertex\_degrees()  
 hanging\_vertices = []  
 for i in range(len(vertex\_degree)):  
 if vertex\_degree[i] == 1:  
 hanging\_vertices.append(i + 1)  
 return hanging\_vertices  
  
 def \_\_get\_adjacency\_matrix(self): # Return adjacency matrix  
 adj\_matrix = [[0 for i in range(self.\_n)] for j in range(self.\_n)]  
 for start, finish in self.\_edges\_list:  
 adj\_matrix[start - 1][finish - 1] = 1  
 adj\_matrix[finish - 1][start - 1] = 1  
 return adj\_matrix  
  
 def \_\_get\_incidence\_matrix(self): # Return incidence matrix  
 inc\_matrix = [[0 for i in range(self.\_m)] for j in range(self.\_n)]  
 edge\_number = 0  
 for edge in self.\_edges\_list:  
 start, finish = edge[0] - 1, edge[1] - 1  
 inc\_matrix[start][edge\_number] = 1  
 inc\_matrix[finish][edge\_number] = 1  
 edge\_number += 1  
 return inc\_matrix  
  
 def \_\_is\_regular(self): # Check graph for regulating  
 vertex\_degrees = self.\_\_get\_vertex\_degrees()  
 for value in vertex\_degrees:  
 if value != vertex\_degrees[0]:  
 return 0  
 return 1  
  
 def \_\_get\_regular\_degree(self): # Return regular degree  
 vertex\_degrees = self.\_\_get\_vertex\_degrees()  
 return vertex\_degrees[0]  
  
 def \_\_enter\_number(self): # Enter and check number  
 answer = 0  
 while answer > 5 or answer < 1:  
 answer = input("Type your choice : ")  
 if answer.isdigit():  
 answer = int(answer)  
 else:  
 answer = 0  
 if answer > 5 or answer < 1:  
 print("Invalid input!")  
 return answer  
  
 def \_\_work\_with\_number(self, number): # Process number  
 if number == 1:  
 print\_matrix(self.\_\_get\_incidence\_matrix())  
 elif number == 2:  
 print\_matrix(self.\_\_get\_adjacency\_matrix())  
 elif number == 3:  
 print(\*self.\_\_get\_vertex\_degrees())  
 elif number == 4:  
 hanging\_vertices = self.\_\_get\_hanging\_vertices()  
 isolated\_vertices = self.\_\_get\_isolated\_vertices()  
 if hanging\_vertices:  
 print("Hanging vertices :", \*hanging\_vertices)  
 else:  
 print("Graph does not have hanging vertices!")  
 if isolated\_vertices:  
 print("Isolated vertices :", \*isolated\_vertices)  
 else:  
 print("Graph does not have isolated vertices")  
 elif number == 5:  
 if self.\_\_is\_regular():  
 print("Graph is regular!\nRegular degree is equal to", self.\_\_get\_regular\_degree())  
 else:  
 print("Graph is not regular!")  
  
 def \_\_show\_menu(self): # Show menu  
 print("Choose one of the following : ")  
 print("""  
 1 - show incidence matrix  
 2 - show adjacency matrix  
 3 - show degree of each vertex  
 4 - show hanging and isolated vertices  
 5 - check graph for regularity """)  
  
 def start\_menu(self): # Start work with graph  
 self.\_\_show\_menu()  
 self.\_\_work\_with\_number(self.\_\_enter\_number())  
  
  
class DirectedGraph(Graph):  
  
 def \_\_get\_adjacency\_matrix(self): # Return adjacency matrix  
 adj\_matrix = [[0 for i in range(self.\_n)] for j in range(self.\_n)]  
 for start, finish in self.\_edges\_list:  
 adj\_matrix[start - 1][finish - 1] = 1  
 return adj\_matrix  
  
 def \_\_get\_incidence\_matrix(self): # Return incidence matrix  
 inc\_matrix = [[0] \* len(self.\_edges\_list) for j in range(self.\_n)]  
 edge\_number = 0  
 for edge in self.\_edges\_list:  
 start, finish = edge[0] - 1, edge[1] - 1  
 if start == finish:  
 inc\_matrix[start][edge\_number] = 2  
 else:  
 inc\_matrix[start][edge\_number] = -1  
 inc\_matrix[finish][edge\_number] = 1  
 edge\_number += 1  
 return inc\_matrix  
  
 def \_\_get\_half\_in\_degrees(self): # Get in-degree of each vertex  
 half\_in\_degrees = [0] \* self.\_n  
 for start, finish in self.\_edges\_list:  
 half\_in\_degrees[finish - 1] += 1  
 return half\_in\_degrees  
  
 def \_\_get\_half\_out\_degrees(self): # Get out-degree of each vertex  
 half\_out\_degrees = [0] \* self.\_n  
 for start, finish in self.\_edges\_list:  
 half\_out\_degrees[start - 1] += 1  
 return half\_out\_degrees  
  
 def \_\_show\_menu(self): # Show menu  
 print("Choose one of the following : ")  
 print("""  
 1 - show incidence matrix  
 2 - show adjacency matrix  
 3 - show in-degrees and out-degrees of each vertex """)  
  
 def \_\_enter\_number(self): # Enter and check number  
 answer = 0  
 while answer > 3 or answer < 1:  
 answer = input("Type your choice : ")  
 if answer.isdigit():  
 answer = int(answer)  
 else:  
 answer = 0  
 if answer > 3 or answer < 1:  
 print("Invalid input!")  
 return answer  
  
 def \_\_work\_with\_number(self, number): # Process number  
 if number == 1:  
 print\_matrix(self.\_\_get\_incidence\_matrix())  
 elif number == 2:  
 print\_matrix(self.\_\_get\_adjacency\_matrix())  
 elif number == 3:  
 print("In - degrees :", \*self.\_\_get\_half\_in\_degrees())  
 print("Out - degrees :", \*self.\_\_get\_half\_out\_degrees())  
  
 def start\_menu(self): # Start work with number  
 self.\_\_show\_menu()  
 self.\_\_work\_with\_number(self.\_\_enter\_number())  
  
  
def parse\_row(string): # Parse row of input data  
 return tuple(map(int, string.split()))  
  
  
def get\_input(file\_name): # Get data from input file  
 edges\_list = []  
 with open(file\_name, "r") as input\_file:  
 vertex, edges = parse\_row(input\_file.readline())  
 data = input\_file.readlines()  
 for line in data:  
 if line != '\n':  
 edges\_list.append(parse\_row(line))  
 return vertex, edges, edges\_list  
  
  
def print\_matrix(matrix): # Print matrix  
 for row in matrix:  
 for i in row:  
 print("{:>3}".format(i), end=" ")  
 print()  
  
  
def main(): # Start of program  
 vertices\_amount, edges\_amount, edges\_list = get\_input("input.txt")  
 while 1:  
 type\_of\_graph = input("Is graph directed?(y/n) : ")  
 if type\_of\_graph == "y":  
 graph = DirectedGraph(vertices\_amount, edges\_amount, edges\_list)  
 else:  
 graph = NotDirectedGraph(vertices\_amount, edges\_amount, edges\_list)  
 graph.start\_menu()  
 if input("If you want to exit type 'exit' else press any key...") == 'exit':  
 break  
  
  
main()

1. Результати виконання:









**Вхідний файл:**

5 4  
1 5  
1 1  
2 3  
3 4  
2 4