**Лабораторная работа №2**

**Графы**

Составить программу, решающую задачу ниже. Входные данные находятся в файле **input.txt**. Для задачи изобразить на экране соответствующий граф.

**Вариант 17 (5):** Город расположен на нескольких островах, связанных между собой мостами. Между двумя островами может быть более одного моста. Выяснить, можно ли, выйдя из дома, вернуться обратно, пройдя по каждому мосту только один раз. Если это возможно, то предложить один из путей.

Технические требования:

Количество островов **N ≤ 30**. Острова пронумерованы числами от **1** до **N**.

Формат файла **input.txt**:

1-я строка – количество островов, последующие строки содержат по два натуральных числа – номера островов, связанных мостом.

**Текст программы**

import networkx as nx  
import matplotlib.pyplot as plt  
import copy  
  
  
def find\_euler\_cycle(a):  
 stack = [0] *# Начало с стартовой вершины* cycle = [] *# Массив для хранения Эйлерова цикла* while stack:  
 v = stack[-1]  
 found = False  
 for u in range(len(a[v])):  
 if a[v][u] > 0: *# Если есть непосещённое ребро* stack.append(u) *# Добавление вершины в стек* a[v][u] -= 1 *# Ребро помечается как посещённое* a[u][v] -= 1  
 found = True  
 break  
 if not found: *# Если из текущей вершины нет исходящих рёбер* cycle.insert(0, stack.pop() + 1) *# Вершина добавляется в массив удаляется из стека* return cycle *# Цикл возвращается*def draw\_graph(a):  
 g = nx.Graph()  
 n = len(a)  
  
 *# Добавляем все вершины, начиная с 1* g.add\_nodes\_from(range(1, n+1))  
  
 *# Создаем пустой граф для рёбер* edge\_graph = nx.Graph()  
  
 *# Заполняем пустой граф рёбрами* for i in range(n):  
 for j in range(i+1, n):  
 if a[i][j] != 0:  
 edge\_graph.add\_edge(i+1, j+1, weight=a[i][j])  
  
 *# Располагаем все вершины близко друг к другу* pos = nx.circular\_layout(g)  
  
 *# Рисуем вершины* nx.draw(g, pos, with\_labels=True, node\_color='lightblue', edge\_color='gray', node\_size=500, font\_size=10)  
  
 *# Рисуем рёбра* for edge in edge\_graph.edges(data=True):  
 nx.draw\_networkx\_edges(g, pos, edgelist=[edge], width=edge[2]['weight'])  
  
 *# Рисуем подписи на рёбрах* labels = nx.get\_edge\_attributes(edge\_graph, 'weight')  
 nx.draw\_networkx\_edge\_labels(g, pos, edge\_labels=labels, label\_pos=0.7)  
  
 plt.show()  
  
  
def main():  
 try:  
 with open("input8.txt", "r") as f:  
 n = int(f.readline().strip()) *# количество островов* try:  
 if n > 30:  
 raise ValueError  
  
 except ValueError:  
 print("Число N больше 30")  
 return 1  
  
 a = [[0] \* n for \_ in range(n)] *# создание матрицы смежности* copy\_a = [[0] \* n for \_ in range(n)]  
  
 try:  
 for line in f: *# запись в матрицу смежности* u, v = map(int, line.split())  
 if u > n or v > n:  
 raise ValueError  
 a[u - 1][v - 1] += 1  
 a[v - 1][u - 1] += 1  
  
 except ValueError:  
 print("Номер одной из вершин больше числа N")  
 return 1  
  
 copy\_a = copy.deepcopy(a)  
  
 except FileNotFoundError:  
 print("Ошибка открытия файла!")  
 return 1  
  
 for row in a: *# вывод матрицы смежности* print(' '.join(map(str, row)))  
  
 connected = True *# проверка на связность* euler = True *# проверка на Эйлеров цикл* if connected:  
 for i in range(n):  
 degree = sum(a[i])  
 if degree == 0:  
 connected = False  
 euler = False  
 break  
 if degree % 2 != 0:  
 euler = False  
 break  
 else:  
 euler = False  
  
 if connected:  
 print("Граф связный")  
 else:  
 print("Граф несвязный")  
  
 if euler:  
 print("Граф содержит Эйлеров цикл")  
 cycle = find\_euler\_cycle(a)  
 print("Эйлеров цикл:", cycle)  
 else:  
 print("Граф не содержит Эйлеров цикл")  
  
 draw\_graph(copy\_a) *# отображение графа*if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Теория**

**Граф** – любое конечное множество точек (вершин), некоторые из которых соединены ребрами.

Задача в данной лабораторной работе представляет собой вариацию задачи о Кёнигсбергских мостах. Данная задача (и аналогичные ей) решаются с помощью поиска **Эйлерова цикла** в графе. Эйлеров цикл – это такой цикл, по которому можно обойти все ребра ровно 1 раз и затем вернутся в начальную вершину. То есть это именно то, что требуется в задаче.

Для существования **Эйлерова цикла** в графе необходимо и достаточно, чтобы соблюдались 2 условия:

1. Граф должен быть **связным**, то есть не иметь изолированных вершин – вершин, которые не связаны хотя бы с еще одной.
2. Все вершины должны иметь **четную степень**, т. е. из всех вершин должно исходить четное количество ребер.

Для проверки на связность нужно проверить все строки или столбцы матрицы смежности. Если хотя бы одна строка (столбец) состоит только из нулей, значит эта вершина является **изолированной**

Степень вершины также легко можно подсчитать по матрице смежности, просто сложив все значения в каждой строке, либо в каждом столбце. Во всех строках (столбцах) эта сумма должна быть **четной**

После того, как мы убедились в том, что граф действительно имеет Эйлеров цикл, нам необходимо найти один из таких путей. Это возможно сделать с помощью **обхода графа в глубину**.

**Описание программы**

Функция **find\_euler\_cycle** (поиск Эйлерова цикла в графе) – ищет один из вариантов пути через все ребра в Эйлеровом цикле с помощью модифицированной версии обхода в глубину. В качестве аргумента принимает матрицу смежности **a**

1. Инициализируется стек **stack**. В него добавляется стартовая вершина **0**
2. Инициализируется массив **cycle** для хранения вершин Эйлерова цикла
3. Пока стек **stack** не станет пустым:
   1. Получаем вершину **v** из вершины стека, при этом не удаляя ее из него
   2. Присваиваем флагу **found**, который отслеживает нахождение непосещенного ребра, значение **False**
   3. Перебираются все вершины, смежные с вершиной **v**. Если есть непосещенное ребро от вершины **v** к вершине **u**, оно записывается в стек **stack**. Затем это ребро отмечается как посещенное путем уменьшения его веса в матрице смежности. Флаг **found** принимает значение **True**. Цикл завершается
   4. Если значение флага равно **False** (т. е. из вершины **v** нет исходящих ребер), значит была достигнута вершина, из которой нет дальнейших переходов. Вершина **v** добавляется в массив **cycle** и удаляется из стека **stack**
4. Когда стек **stack** становится пустым, функция возвращает массив **cycle**

Функция **draw\_graph** (работает на базе библиотеки для работы с графами **NetworkX**) – выводит граф на экран. В качестве аргумента принимает матрицу смежности **a**

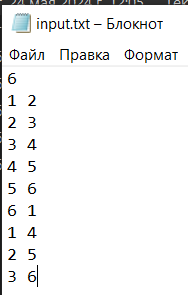
1. Создается пустой неориентированный граф **g**
2. Переменной **n** присваивается значение – количество вершин в графе, которое берется из матрицы смежности **a**
3. В граф **g** добавляются все вершины от **1** до **n**
4. Создается пустой неориентированный граф **edge\_graph** для вершин
5. Граф **edge\_graph** заполняется ребрами, проходя по матрице смежности **a**
6. Вершины из графа **g** располагаются и рисуются на экране
7. Ребра из графа **edge\_graph** рисуются на экране
8. В словарь **labels** записываются значения весов ребер (количество мостов между 2 островами)
9. Значения весов из словаря **labels** выводятся на соответствующих ребрах

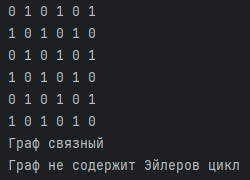
Функция **main** (основная часть программы):

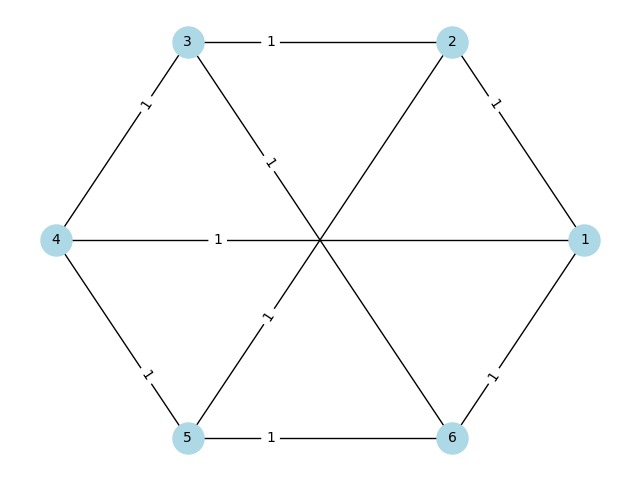
1. Файл **input.txt** открывается для чтения. Если файла нет – вызывается исключение, выводится надпись *«Ошибка открытия файла!»* и выполнение программы завершается.
2. Считывается 1 строка из файла и присваивается переменной **n**. Если **n** оказалось больше **30** – вызывается исключение, выводится надпись *«Число N больше 30»* и выполнение программы завершается.
3. Инициализируется матрица смежности **a** размером **n\*n** и ее копия **copy\_a**. Всем элементам обеих матриц присваиваются начальные значения **0**.
4. Программа читает остальные строки файла. Первое число записывается в переменную **u**, второе – в переменную **v**. В матрице смежности **a** значение элементов **a[u-1][v-1]** и **a[v-1][u-1]** увеличивается на **1**.
5. Матрица смежности **a** копируется в матрицу **copy\_a**.
6. Матрица смежности **a** выводится на экран
7. Переменным **connected** и **euler** присваиваются начальные значения **True**
8. Находятся и записываются в переменную **degree** степени всех вершин – сумма каждой строки матрицы смежности **a**.
9. Если какая-то из степеней равна 0, это значит, что данная вершина изолирована. Переменным **connected** и **euler** присваивается значение **False**, выполнение цикла прекращается. Если **degree** не равно нулю – переменные **connected** и **euler** так и остаются со значениями **True**
10. Значение **degree** проверяется на четность путем нахождения остатка от деления на **2**. Если значение остатка не равно нулю – переменной **euler** присваивается значение **False** и выполнение цикла прекращается. Мы делаем вывод, что в графе нет Эйлерова цикла. Если степень четная – переменная **euler** так и остается со значением **True**
11. Если **connected = True**, выводится надпись *«Граф связный»*, иначе – надпись *«Граф несвязный»*
12. Если **euler = True**, выводится надпись *«Граф содержит Эйлеров цикл»*. Далее программа находит один из вариантов пути через все ребра графа с помощью функции **find\_euler\_cycle** и выводит его на экран
13. Если **euler = False**, выводится надпись *«Граф не содержит Эйлеров цикл»*.
14. С помощью функции **draw\_graph** граф выводится на экран. В качестве аргумента данная функция принимает матрицу смежности. Но туда мы передаем не саму матрицу **a**, а ее копию **copy\_a**, т. к. оригинальная матрица была изменена при выполнении функции **find\_euler\_cycle**

**Результаты тестов**

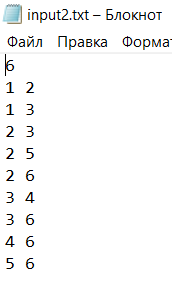
Связный граф без Эйлерова цикла на 6 элементов:

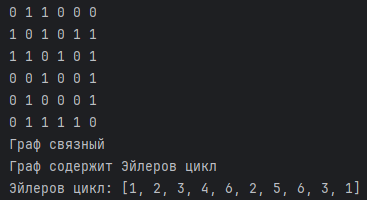


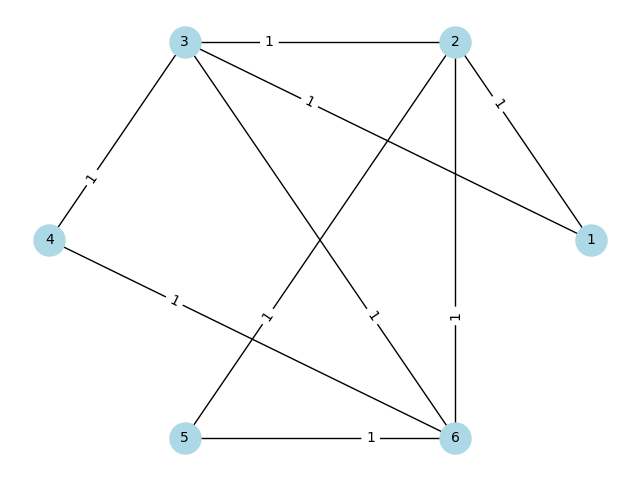


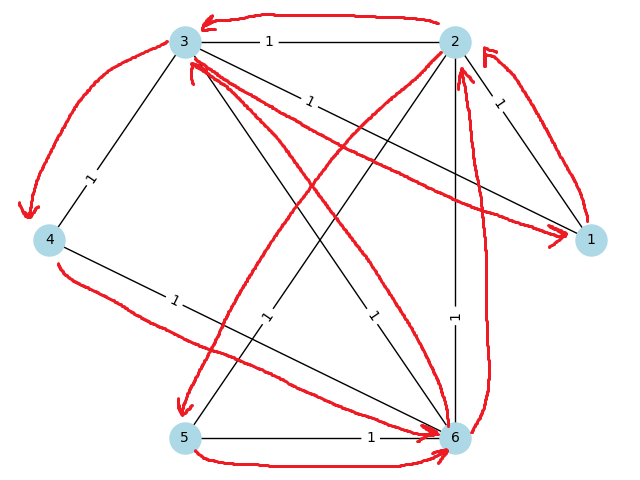


Связный граф с Эйлеровым циклом на 6 элементов:

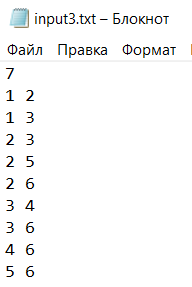


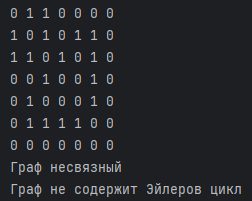


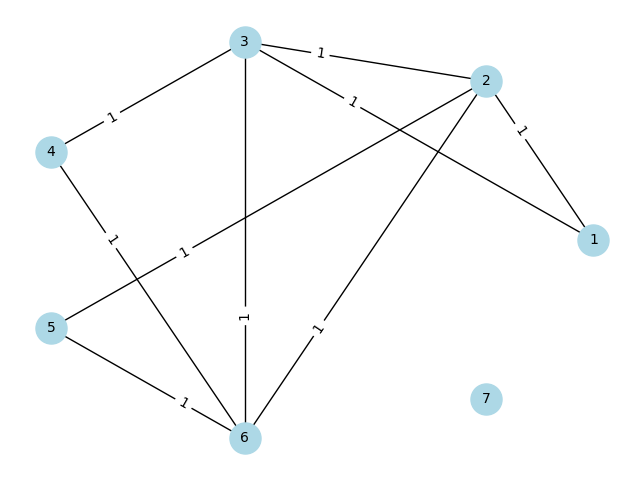




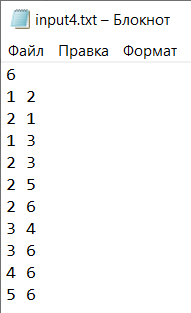
Несвязный граф на 7 элементов:

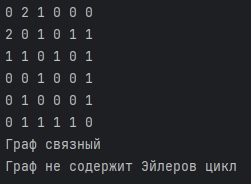


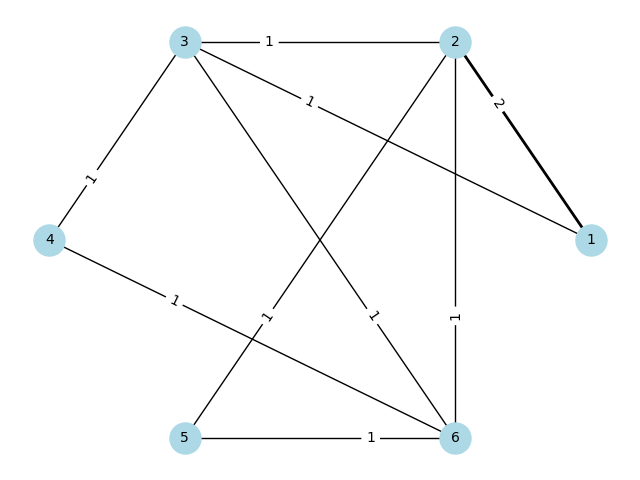




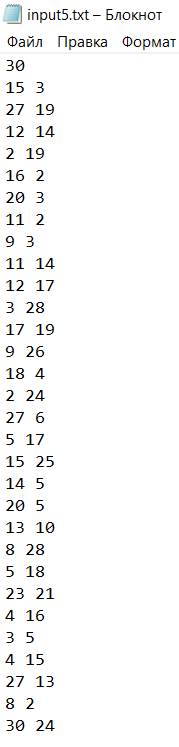
Граф с несколькими мостами:

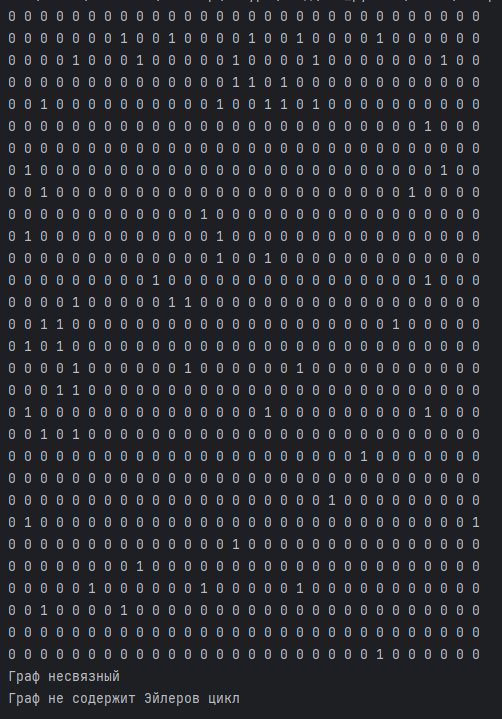


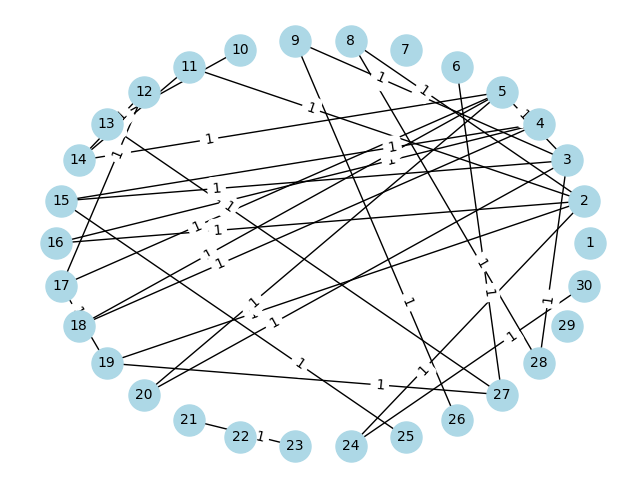




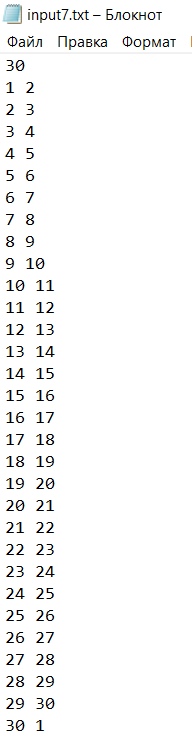
Несвязный граф на 30 элементов:



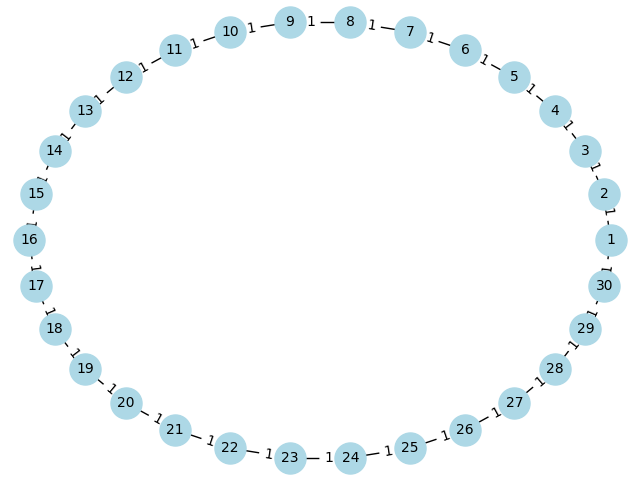


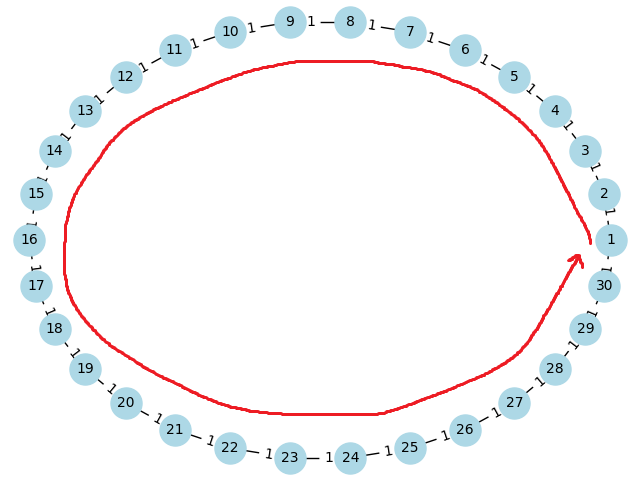


Связный граф с Эйлеровым циклом на 30 элементов:

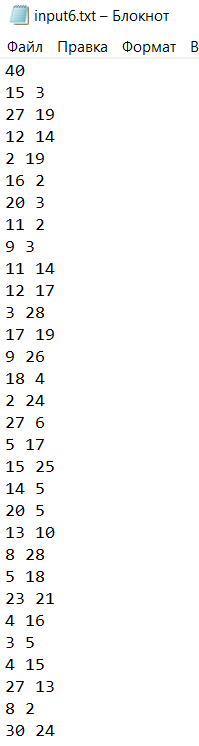






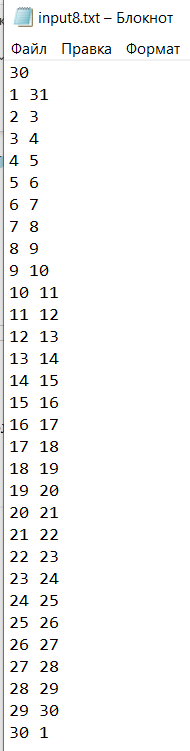


Граф с более, чем 30 элементами:





Файл, где вершина больше числа N





**Вывод**

В ходе выполнения лабораторной работы была написана программа, которая проверяет граф на связность и ищет в нем Эйлеров цикл.

Также программа находит один из путей по всем ребрам в Эйлеровом графе и выводит граф на экран.