Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

по дисциплине «Дискретная Математика»

Выполнил
студент гр. 23508/4 Е.Г. Проценко

Проверила
ассистент Д.С. Лаврова

Санкт-Петербург
2016

1 ФОРМУЛИРОВКА ЗАДАНИЯ (ВАРИАНТ 7)

Цель работы – изучение основ теории графов, базовых понятий и определений компьютерных способов представления графов и операций над ними.

1) Написать на C++ класс, описывающий граф/орграф. Класс должен поддерживать следующую функциональность:
   - Определение числа вершин;
   - Определение числа ребер (дуг);
   - Определение степени произвольной вершины (для орграфа – полустепеней захода);
   - Определение степенной последовательности графа;
   - Определение матрицы смежности;
   - Определение матрицы инцидентности;
   - Определение списка смежности;
   - Добавления/удаление вершин в граф;
   - Добавление/удаление ребра в граф;
   - Определение дополнения графа;
   - Подразбиение ребра;
   - Стягивание графа;
   - Отождествление вершин;
   - Дублирование вершин;
   - Размножение вершин;
   - Объединение (дизъюнктивное) графов;
   - Соединение графов;
   - Произведение графов;
   - Ввод/вывод графов в текстовый файл в виде списка смежности в следующем формате: {1 <смежные вершины через пробел>…}{2 <смежные вершины через пробел>}… Например: {1 2}{2 1 4}{3}{4 2}.
2) Главная программа должна демонстрировать возможности разработанного класса. А именно:
   - Она должна позволять задавать пользователю граф в любом из трех видов (список смежности, матрица смежности или матрица инцидентности) и получать на выходе любое другое представление;
   - По запросу пользователя выводить характеристики графа (число вершин, число ребер, степенную последовательность, степень выбранной вершины);
   - Позволять пользователю выполнять реализованные в классе операции;
   - Визуализация графа – по желанию.
3) Получить у преподавателя вариант задания и представить результаты его выполнения в отчете.

Граф G_1 задан матрицей смежности, граф G_2 – матрицей инцидентности, орграф G_3 – матрицей инцидентности. В матрицах инцидентности по строкам располагаются номера вершин. Вывод результатов понимается как вывод в файл и/или на экран (в отчете также представить результат). Если выполнить операцию невозможно – в отчете необходимо обоснование.

Задание 1. Для заданных графов и орграфов вывести (определить):

а) Число вершин и число ребер (дуг);

б) Списки смежности;

в) Степенные последовательности (для орграфа – полустепени захода и исхода каждой вершины);

г) матрицу инцидентности G_1, матрицы смежности G_2 и G_3.

Задание 2.

а) добавить в граф G_1 новые вершины v_1, v_2 и удалить из него вершину v_3. Вывести результат;

б) добавить в полученный на предыдущем шаге граф ребра e_1, e_2, e_3, e_4, e_5 и удалить ребра e_6, e_7. Вывести результат в виде матрицы инцидентности;

в) построить дополнение полученного на шаге б) графа. Пусть это граф G_4 (вывести его в виде матрицы смежности);

г) добавить в орграф G_3 вершины v_4, v_5, v_6 и удалить v_7. Добавить дуги f_1, f_2, f_3, удалить дугу f_4. Вывести результат;

д) выполнить операции предыдущего пункта в обратном порядке (сначала дуги, потом вершины). Вывести результат и сравнить с предыдущим.

Задание 3.

Получить граф из имеющихся графов G_1, G_2 и G_4 по заданной формуле (списку операций). Вывести результат.

$$A(G_1)=\begin{pmatrix}0&1&1&1&1&1&2&1&1&1\\1&0&1&1&1&1&1&1&1&1\\1&1&0&1&1&1&1&1&1&1\\1&1&1&0&1&1&1&1&1&1\\1&1&1&1&2&1&1&1&1&1\\1&1&1&1&1&0&0&0&0&0\\2&1&1&1&1&0&0&0&1&0\\1&1&1&1&1&0&0&0&1&0\\1&1&1&1&1&0&1&1&0&0\\1&1&1&1&1&0&0&0&0&0\end{pmatrix},\ I(G_2)=\begin{pmatrix}0&1&1&1&1&0\\0&0&0&1&0&0\\0&1&0&0&1&0\\1&0&0&0&0&0\\0&0&1&0&0&1\\1&0&0&0&0&1\end{pmatrix},$$

$$I(G_3)=\begin{pmatrix}1&0&0&0&-1&0&0\\0&-1&0&0&1&0&0\\0&0&1&-1&0&0&0\\0&0&-1&0&0&0&1\\-1&0&0&0&0&1&0\\0&0&0&1&0&0&-1\\0&1&0&0&0&-1&0\end{pmatrix}.$$

Списки вершин, ребер и дуг для заданий:

v_i: {11, 12, 8, 8, 9, 10, 3}; e_i: {(9,11), (9,10), (4,12), (10, 12), (5,11), (5,5), (7,1)}; f_i: {(1,4), (2,7), (1,9), (3,4)}.

Получить граф: в G_4отождествить вершины 1 и 10, затем найти объединение с

2 РЕЗУЛЬТАТЫ РАБОТЫ

2.1 Задание 1

1) Задание 1 для графа G_1.

```
Матрица Смежности:
0 1 1 1 1 1 2 1 1 1
1 0 1 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1
1 1 1 1 2 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0
2 1 1 1 1 0 0 0 1 0
1 1 1 1 1 0 0 0 1 0
1 1 1 1 1 0 1 1 0 0
1 1 1 1 1 0 0 0 0 0
```

а) Число вершин и ребер

```
Command: get_e
Число ребер в графе: 39
Command: get_v
Число вершин в графе: 10
```

б) Список смежности

```
Command: print_al
Список Смежности:
{0 1 2 3 4 5 6 6 7 8 9}
{1 0 2 3 4 5 6 7 8 9}
{2 0 1 3 4 5 6 7 8 9}
{3 0 1 2 4 5 6 7 8 9}
{4 0 1 2 3 4 5 6 7 8 9}
{5 0 1 2 3 4}
{6 0 0 1 2 3 4 8}
{7 0 1 2 3 4 8}
{8 0 1 2 3 4 6 7}
{9 0 1 2 3 4}
```

в) Степенная последовательность

```
Степенная последовательность графа:
{4:11}{0:10}{2:9}{3:9}{1:9}{6:7}{8:7}{7:6}{5:5}{9:5}
```

г) Матрица инцидентности

```
Command: print_im
Матрица Инцидентности:
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 2 1 1 1 1 1 0 0
0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
```

2) Задание 1 для графа G_2.

```
Command: read_im
Command: Введите название файла из которого считать матрицу: input2.txt
Done
Command: print_im
Матрица Инцидентности:
0 1 1 1 1 0
0 0 0 1 0 0
0 1 0 0 1 0
1 0 0 0 0 0
0 0 1 0 0 1
1 0 0 0 0 1
```

а) Число вершин и ребер

```
Command: get_v
Число вершин в графе: 6
Command: get_e
Число ребер в графе: 6
```

б) Список смежности

```
Command: print_al
Список Смежности:
{0 1 2 2 4}
{1 0}
{2 0 0}
{3 5}
{4 0 5}
{5 3 4}
```

в) Степенная последовательность

```
Command: get_deg_sqnce
Степенная последовательность графа:
{0:4}{2:2}{4:2}{5:2}{1:1}{3:1}
```

г) Матрица смежности

```
Command: print_am
Матрица Смежности:
0 1 2 0 1 0
1 0 0 0 0 0
2 0 0 0 0 0
0 0 0 0 0 1
1 0 0 0 0 1
0 0 0 1 1 0
```

3) Задание 1 для графа G_3.

```
Command: print_im
Матрица Инцидентности:
1       0       0       0       -1      0       0
0       -1      0       0       1       0       0
0       0       1       -1      0       0       0
0       0       -1      0       0       0       1
-1      0       0       0       0       1       0
0       0       0       1       0       0       -1
0       1       0       0       0       -1      0
```

а) Число вершин и ребер

```
Command: get_v
Число вершин в графе: 7
Command: get_e
Число ребер в графе: 7
```

б) Список смежности

```
Command: print_al
Список Смежности:
{0 4}
{1 0}
{2 3}
{3 5}
{4 6}
{5 2}
{6 1}
```

в) Степенная последовательность

```
Command: get_deg_sqnce
Полустепени входа и исхода для каждой вершины:
{0:1,1}{1:1,1}{2:1,1}{3:1,1}{4:1,1}{5:1,1}{6:1,1}
```

г) Матрица смежности

```
Command: print_am
Матрица Смежности:
0 0 0 0 1 0 0
1 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1
0 0 1 0 0 0 0
0 1 0 0 0 0 0
```

2.2 Задание 2

a) добавить в граф G_1 новые вершины $v_1 = \{11\}$, $v_2 = \{12\}$ и удалить из него вершину $v_3 = \{8\}$. Вывести результат;

```
Command: add_v
Вершина добавлена
Command: add_v
Вершина добавлена
Command: del_v
Command: Введите номер удаляемой вершины (0 <= var <= 11): 7
Вершина удалена.
Command: print_am
Матрица Смежности:
0 1 1 1 1 1 2 1 1 0 0
1 0 1 1 1 1 1 1 1 0 0
1 1 0 1 1 1 1 1 1 0 0
1 1 1 0 1 1 1 1 1 0 0
1 1 1 1 2 1 1 1 1 0 0
1 1 1 1 1 0 0 0 0 0 0
2 1 1 1 1 0 0 1 0 0 0
1 1 1 1 1 0 1 0 0 0 0
1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
```

Не забываем, что нумерация начинается с нуля.

После того как мы удалили вершину 8, вершины с большим индексом сдвинулись вниз.

1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12 – остались

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 – их индексы

b) добавить в полученный на предыдущем шаге граф ребра $e_1 = \{9,11\}$, $e_2 = \{9,10\}$, $e_3 = \{4,12\}$, $e_4 = \{10,12\}$, $e_5 = \{5,11\}$ и удалить ребра $e_6 = \{5,5\}$, $e_7 = \{7,1\}$. Вывести результат в виде матрицы инцидентности;

До преобразований:

```
Command: print_im
Матрица Инцидентности:
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 2 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

После преобразований:

```
Command: add_e
Command: Введите номер первой вершины (0 <= var <= 10): 7
Command: Введите номер второй вершины (0 <= var <= 10): 9
Ребро добавлено.
Command: add_e
Command: Введите номер первой вершины (0 <= var <= 10): 7
Command: Введите номер второй вершины (0 <= var <= 10): 8
Ребро добавлено.
Command: add_e
Command: Введите номер первой вершины (0 <= var <= 10): 3
Command: Введите номер второй вершины (0 <= var <= 10): 10
Ребро добавлено.
Command: add_e
Command: Введите номер первой вершины (0 <= var <= 10): 8
Command: Введите номер второй вершины (0 <= var <= 10): 10
Ребро добавлено.
Command: add_e
Command: Введите номер первой вершины (0 <= var <= 10): 4
Command: Введите номер второй вершины (0 <= var <= 10): 9
Ребро добавлено.
Command: del_e
Command: Введите номер одной вершины (0 <= var <= 10): 4
Command: Введите номер другой вершины (0 <= var <= 10): 4
Ребро удалено.
Command: del_e
Command: Введите номер одной вершины (0 <= var <= 10): 6
Command: Введите номер другой вершины (0 <= var <= 10): 0
Ребро удалено.
Command: print_im
Матрица Инцидентности:
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0
0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

c) построить дополнение полученного на шаге b) графа. Пусть это граф G_4 (вывести его в виде матрицы смежности);

До преобразований:

```
Command: new
Новый граф создан. Всего графов: 2.
Command: cng
Command: Введите номер графа, с которым Вы хотите продолжить работу (0 <= var <= 1): 1
Теперь рабочим графом является граф номер: 1.
Command: cpy
Command: Введите номер графа, который вы хотите скопировать в текущий (0 <= var <= 1): 0
Граф скопирован.
Command: print_am
Матрица Смежности:
0 1 1 1 1 1 1 1 1 0 0
1 0 1 1 1 1 1 1 1 0 0
1 1 0 1 1 1 1 1 1 0 0
1 1 1 0 1 1 1 1 1 0 1
1 1 1 1 0 1 1 1 1 1 0
1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 0 0 1 0 0 0
1 1 1 1 1 0 1 0 1 1 0
1 1 1 1 1 0 0 1 0 0 1
0 0 0 0 1 0 0 1 0 0 0
0 0 0 1 0 0 0 0 1 0 0
```

После преобразований:

```
Command: 1
Дополнение графа получено.
Command: print_am
Матрица Смежности:
0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 1 1 1 1
0 0 0 0 0 1 0 0 1 1 1
0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 1 1 0 0 1 0
1 1 1 1 0 1 1 0 1 0 1
1 1 1 0 1 1 1 1 0 1 0
```

d) Добавить в орграф G_3 вершины $v_4 = \{8\}$, $v_5 = \{9\}$, $v_6 = \{10\}$ и удалить $v_7 = \{3\}$. Добавить дуги $f_1 = \{1,4\}$, $f_2 = \{2,7\}$, $f_3 = \{1,9\}$, удалить дугу $f_4 = \{3,4\}$. Вывести результат;

Считали граф:

```
Command: new
Новый граф создан. Всего графов: 3.
Command: cng
Command: Введите номер графа, с которым Вы хотите продолжить работу (0 <= var <= 2): 2
Теперь рабочим графом является граф номер: 2.
Command: read_im
Command: Введите название файла из которого считать матрицу: input3.txt
Done
Command: print_am
Матрица Смежности:
0 0 0 0 1 0 0
1 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1
0 0 1 0 0 0 0
0 1 0 0 0 0 0
```

Добавили, удалили вершины v_i:

```
Command: add_v
Вершина добавлена
Command: add_v
Вершина добавлена
Command: add_v
Вершина добавлена
Command: del_v
Command: Введите номер удаляемой вершины (0 <= var <= 9): 2
Вершина удалена.
Command: print_am
Матрица Смежности:
0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

Вершины:

1, 2, 4, 5, 6, 7, 8, 9, 10 – остались
0, 1, 2, 3, 4, 5, 6, 7, 8 – их индексы

Добавили, удалили дуги f_i:

```
Command: add_e
Command: Введите номер вершины, из которой выходит ребро (0 <= var <= 8): 0
Command: Введите номер вершины, в которую входит ребро (0 <= var <= 8): 2
Ребро добавлено.
Command: add_e
Command: Введите номер вершины, из которой выходит ребро (0 <= var <= 8): 1
Command: Введите номер вершины, в которую входит ребро (0 <= var <= 8): 5
Ребро добавлено.
Command: add_e
Command: Введите номер вершины, из которой выходит ребро (0 <= var <= 8): 0
Command: Введите номер вершины, в которую входит ребро (0 <= var <= 8): 7
Ребро добавлено.
```

Удалить ребро {3,4} невозможно, т.к. вершина 3 была удалена ранее.

Результат:

```
Command: print_am
Матрица Смежности:
0 0 1 1 0 0 0 1 0
1 0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

е) выполнить операции предыдущего пункта в обратном порядке (сначала дуги, потом вершины). Вывести результат и сравнить с предыдущим.

```
Command: del_e
Command: Введите номер одной вершины (0 <= var <= 8): 0
Command: Введите номер другой вершины (0 <= var <= 8): 2
Ребро удалено.
Command: del_e
Command: Введите номер одной вершины (0 <= var <= 8): 1
Command: Введите номер другой вершины (0 <= var <= 8): 5
Ребро удалено.
Command: del_e
Command: Введите номер одной вершины (0 <= var <= 8): 0
Command: Введите номер другой вершины (0 <= var <= 8): 7
Ребро удалено.
```

```
Command: print_am
Матрица Смежности:
0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

После добавление вершины, она встанет на последнюю позицию:

```
Command: print_am
Матрица Смежности:
0 0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Вершины:

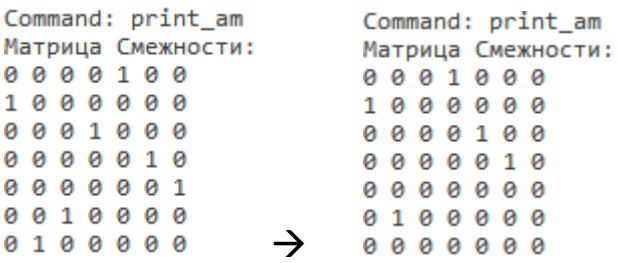
1, 2, 3, 4, 5, 6, 7, 8, 9, 10 – остались
0, 1, 9, 2, 3, 4, 5, 6, 7, 8 – их индексы

```
Command: del_v
Command: Введите номер удаляемой вершины (0 <= var <= 9): 8
Вершина удалена.
Command: del_v
Command: Введите номер удаляемой вершины (0 <= var <= 8): 7
Вершина удалена.
Command: del_v
Command: Введите номер удаляемой вершины (0 <= var <= 7): 6
Вершина удалена.
Command: print_am
Матрица Смежности:
0 0 0 1 0 0 0
1 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 0
0 1 0 0 0 0 0
0 0 0 0 0 0 0
```

Вершины:

1, 2, 3, 4, 5, 6, 7 – остались
0, 1, 9, 2, 3, 4, 5 – их индексы

До и после преобразований:



Вершина 3 переместилась вниз из-за удаления/добавлении.

При удалении вместе с вершиной удалилась дуга {3,4}.

3 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чему равна сумма степеней всех вершин графа?

По лемме о рукопожатиях $\sum_{v \in V} \deg(v) = 2|E|$

2. Докажите, что алгебраические дополнения всех элементов матрицы Кирхгофа графа равны между собой.

Определим матрицу Кирхгофа $B = B(G)$, полагая

$$B(G) = \begin{pmatrix} \deg v_1 & 0 & \dots & 0 \\ 0 & \deg v_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \deg v_n \end{pmatrix} - A(G),$$

где $A(G)$ — матрица смежности графа G.

Обозначим столбец (1, 1, ..., 1)^T длины n, состоящий из единиц, через 1.

Для матрицы Кирхгофа $B(G) = (\beta_{ij})_{n \times n}$ выполняется

$\Sigma_{j=1..n} \beta_{ij} = 0$ ($i = 1, 2, ..., n$), т.е. $B * 1 = 0$,
$\Sigma_{i=1..n} \beta_{ij} = 0$ ($j = 1, 2, ..., n$), т.е. $1^T * B = 0$.
Отсюда следует, что $\det B = 0$ и $\text{rank } B \leq n - 1$.

Если $\text{rank } B < n - 1$, то все алгебраические дополнения элементов матрицы B равны 0. Пусть $\text{rank } B = n - 1$ и C — присоединённая к B матрица, составленная из алгебраических дополнений B_{ij} элементов β_{ij}, т.е.

$$\begin{pmatrix} B_{11} & B_{21} & \dots & B_{n1} \\ B_{12} & B_{22} & \dots & B_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ B_{1n} & B_{2n} & \dots & B_{nn} \end{pmatrix}.$$

В силу свойств C получаем

$$BC = CB = (\det B)E = 0.$$

Так как $BC = 0$, любой столбец X матрицы C удовлетворяет системе $BX = 0$. Эта система линейных уравнений имеет ранг $n - 1$. Так как $B * 1 = 0$, этой системе удовлетворяет столбец 1. Следовательно, столбцы матрицы C пропорциональны столбцу 1, откуда следует

$$B_{i1} = B_{i2} = \dots = B_{in} \ (i = 1, 2, \dots, n).$$

Аналогично получаем

$$B_{1j} = B_{2j} = \dots = B_{nj} \ (j = 1, 2, \dots, n).$$

Следовательно, все элементы матрицы C одинаковы. Ч.т.д

3. Пусть G — граф, множество вершин которого совпадает с отрезком натурального ряда {1,2,...5}, а множество ребер определяется следующим условием: несовпадающие вершины v_i, и v_j смежны тогда, когда числа i и jвзаимно просты. Какой вид имеют матрица смежности, матрица инцидентности и матрица Кирхгофа?

I(G)=

	a	b	c	d	e	f	g	h	j
1	1	1	1	1	0	0	0	0	0
2	1	0	0	0	1	1	0	0	0
3	0	1	0	0	1	0	1	1	0
4	0	0	1	0	0	0	1	0	1
5	0	0	0	1	0	1	0	1	1

A(G)=

	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	0	1
3	1	1	0	1	1
4	1	0	1	0	0
5	1	1	1	1	0

K(G)=

	1	2	3	4	5
1	4	-1	-1	-1	-1
2	-1	3	-1	0	-1
3	-1	-1	4	-1	-1
4	-1	0	-1	2	0
5	-1	-1	-1	-1	4

4. Задан неориентированный граф *G*. В графе удаляются вершина и два ребра. Существенна ли последовательность выполнения операций? Нет.

5.Верно ли равенство: $Q_3 \overset{?}{=} K_2 \times Q_2$. Да

6.Графы $H = H_1 \cup H_2$ и *G* являются подграфами K_n.Выполняется ли для них соотношение:

$$H \times G = (H_1 \cup H_2) \times G = H_1 \times G \cup H_2 \times G \text{?}$$ Да

4 ВЫВОД

В данной лабораторной работе я познакомился с языком C++ и классами. Был написан класс, который работает с графами и орграфами. Методами класса являются все основные операции над графами, начиная от добавления/удаления вершин, заканчивая перемножением графов.
Во время написания программы я разобрался с теоретическим материалом и применил на практике. Все вычисления проверялись вручную. Теперь я знаю, чем отличается каждая операция от других.

Факты о работе:

- На работу было потрачено суммарно около 25 часов
- Было выпито 5 банок RedBull
- Было потрачено 3 бессонных ночи
- Это самая большая программа на C++
- Это самый большой класс, написанный мной
- По времени больше заняла только курсовая работа за 2 семестр “Покер”

5 ПРИЛОЖЕНИЕ

Листинг написанной программы:

main.cpp

```
#include "Graph.hpp"
#include <iostream>
#include <fstream>
#include <clocale>
#include <string>

int main()
{
	setlocale(LC_ALL, "Russian");
	int total = 1;
	int current = 0;
	Graph * Graph_arr = (Graph *)malloc(sizeof(Graph));
	Graph * _Graph = Graph_arr;
	_Graph->help();
	_Graph->init();



	std::string str;
	//char str[256];
	while (true)
	{
		std::cout << "Command: ";
		std::cin >> str;

		if (str.compare("help") == 0) _Graph->help();
		else if (str.compare("exit") == 0) break;
		else if (str.compare("new") == 0)
		{
			Graph_arr = (Graph *)realloc((void *)Graph_arr, (++total) * sizeof(Graph));
			_Graph = &Graph_arr[current];
			Graph_arr[total - 1].init();
			std::cout << "Новый граф создан. Всего графов: " << total << "." <<
std::endl;
		}
		else if (str.compare("change") == 0 || str.compare("cng") == 0)
		{
			int i;
			std::cout << "Command: Введите номер графа, с которым Вы хотите продолжить
работу (0 <= var <= " << total - 1 << "): ";
			std::cin >> i;
			if (i >= 0 && i <= total - 1)
			{
				_Graph = &Graph_arr[i];
				current = i;
				std::cout << "Теперь рабочим графом является граф номер: " << i <<
"." << std::endl;
			}

			else std::cout << "Неверные входные данные." << std::endl;
		}
		else if (str.compare("copy") == 0 || str.compare("cpy") == 0)
		{
			int i;
			std::cout << "Command: Введите номер графа, который вы хотите скопировать в
текущий (0 <= var <= " << total - 1 << "): ";
			std::cin >> i;
			if (i >= 0 && i <= total - 1)
			{
				_Graph->copy_from(&Graph_arr[i]);
				std::cout << "Граф скопирован." << std::endl;
			}

			else std::cout << "Неверные входные данные." << std::endl;
		}

		else if (str.compare("read_am") == 0)
		{
			std::cout << "Command: Введите название файла из которого считать матрицу:
";
			std::string str;
```

```
                        std::cin >> str;
                        str.c_str();

                        std::ifstream fin(str.c_str());
                        if (!fin.is_open()){
                                std::cout << "Такого файла не существует" << std::endl;
                                continue;
                        }

                        _Graph->read_adjacency_matrix(&str);
                        std::cout << "Done" << std::endl;
                }
                else if (str.compare("read_im") == 0)
                {
                        std::cout << "Command: Введите название файла из которого считать матрицу:
";
                        std::string str;
                        std::cin >> str;
                        str.c_str();

                        std::ifstream fin(str.c_str());
                        if (!fin.is_open()){
                                std::cout << "Такого файла не существует" << std::endl;
                                continue;
                        }

                        _Graph->read_incidence_matrix(&str);
                        std::cout << "Done" << std::endl;
                }

                /*-----------------------------------*/
                /*-----------------------------------*/
                /*-----------------------------------*/

                else if (str.compare("get_v") == 0) std::cout << "Число вершин в графе: " <<
_Graph->get_vertices() << std::endl;
                else if (str.compare("get_e") == 0) std::cout << "Число ребер в графе: " <<
_Graph->get_edges() << std::endl;
                else if (str.compare("get_deg") == 0)
                {
                        int i;
                        std::cout << "Command: Введите номер вершины: ";
                        std::cin >> i;

                        if (i >= 0 && i < _Graph->get_vertices())
                        {
                                int result[2];
                                _Graph->get_v_degree(i, result);
                                if (_Graph->is_oriented_graph())
                                {
                                        std::cout << "Полустепень исхода вершины " << i << ": " <<
result[0] << "." << std::endl;
                                        std::cout << "Полустепень захода вершины " << i << ": " <<
result[1] << "." << std::endl;
                                }
                                else
                                {
                                        std::cout << "Степень вершины " << i << ": " << result[0] <<
"."<< std::endl;
                                }
                        }
                        else std::cout << "Wrong v number" << std::endl;
                }
                else if (str.compare("get_deg_sqnce") == 0)
                {
                        int * sqnce = _Graph->get_degree_sequence();

                        if (_Graph->is_oriented_graph())
                        {
                                std::cout << "Полустепени входа и исхода для каждой вершины: " <<
std::endl;
                                int ver = _Graph->get_vertices();
                                for (int i = 0; i < ver; i++) std::cout << "{" << i << ":" <<
sqnce[i] << "," << sqnce[i + ver] << "}";
                                std::cout << std::endl;
                                continue;
                        }

                        std::cout << "Степенная последовательность графа: " << std::endl;
```

```
                        for (int i = 0; i < _Graph->get_vertices(); i++) std::cout << "{" <<
sqnce[i + _Graph->get_vertices()] << ":" << sqnce[i] << "}";
                        std::cout << std::endl;
                }

                /*-----------------------------------*/
                /*-----------------------------------*/
                /*-----------------------------------*/

                else if (str.compare("print_am") == 0)
                {
                        std::cout << "Матрица Смежности:" << std::endl;
                        _Graph->print_adjacency_matrix();
                }
                else if (str.compare("print_im") == 0)
                {
                        std::cout << "Матрица Инцидентности:" << std::endl;
                        _Graph->print_incidence_matrix();
                }
                else if (str.compare("print_al") == 0)
                {
                        std::cout << "Список Смежности:" << std::endl;
                        _Graph->print_adjacency_list();
                }
                else if (str.compare("print_out") == 0)
                {
                        _Graph->print_out();
                        std::cout << "Список смежнотей теперь в output.txt." << std::endl;
                }

                /*-----------------------------------*/
                /*-----------------------------------*/
                /*-----------------------------------*/

                else if (str.compare("add_v") == 0)
                {
                        _Graph->add_vortex();
                        std::cout << "Вершина добавлена" << std::endl;
                }
                else if (str.compare("del_v") == 0)
                {
                        int vertices = _Graph->get_vertices();

                        if (vertices > 0) std::cout << "Command: Введите номер удаляемой вершины (0
<= var <= " << vertices - 1 << "): ";
                        else
                        {
                                std::cout << "Граф пуст. В нем нечего удалять" << std::endl;
                                continue;
                        }

                        int v;
                        std::cin >> v;
                        if (v >= 0 && v <= vertices - 1)
                        {
                                _Graph->delete_vortex(v);
                                std::cout << "Вершина удалена." << std::endl;
                        }
                        else std::cout << "Неверные входные данные" << std::endl;
                }
                else if (str.compare("add_e") == 0)
                {
                        int vertices = _Graph->get_vertices();
                        if (vertices == 0)
                        {
                                std::cout << "Граф пуст. Невозможно добавить ребро." << std::endl;
                                continue;
                        }

                        int out, in;

                        if (_Graph->is_oriented_graph() == true) std::cout << "Command: Введите
номер вершины, из которой выходит ребро (0 <= var <= " << vertices - 1 << "): ";
                        else std::cout << "Command: Введите номер первой вершины (0 <= var <= " <<
vertices - 1 << "): ";
                        std::cin >> out;
                        if (out < 0 || out >= vertices)
                        {
                                std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
```

```
                        continue;
                    }
                    if (_Graph->is_oriented_graph() == true) std::cout << "Command: Введите
номер вершины, в которую входит ребро (0 <= var <= " << vertices - 1 << "): ";
                    else std::cout << "Command: Введите номер второй вершины (0 <= var <= " <<
vertices - 1 << "): ";
                    std::cin >> in;
                    if (in < 0 || in >= vertices)
                    {
                        std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                        continue;
                    }

                    _Graph->add_edge(out, in);

                    std::cout << "Ребро добавлено." << std::endl;
                }
                else if (str.compare("del_e") == 0)
                {
                    /*int e;
                    int edges = _Graph->get_edges();
                    if (edges == 0)
                    {
                        std::cout << "В графе нет ребер. Нечего удалять." << std::endl;
                        continue;
                    }

                    else std::cout << "Command: Введите номер удаляемого ребра (0 <= var <= "
<< edges - 1 << "): ";
                    std::cin >> e;
                    if (e < 0 || e >= edges)
                    {
                        std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                        continue;
                    }

                    _Graph->delete_edge(e);
                    std::cout << "Ребро удалено." << std::endl;*/

                    int v1, v2;
                    int edges = _Graph->get_edges();
                    int ver = _Graph->get_vertices();
                    if (edges == 0)
                    {
                        std::cout << "В графе нет ребер. Нечего удалять." << std::endl;
                        continue;
                    }

                    std::cout << "Command: Введите номер одной вершины (0 <= var <= " << ver -
1 << "): ";
                    std::cin >> v1;
                    if (v1 < 0 || v1 >= ver)
                    {
                        std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                        continue;
                    }

                    std::cout << "Command: Введите номер другой вершины (0 <= var <= " << ver -
1 << "): ";
                    std::cin >> v2;
                    if (v2 < 0 || v2 >= ver)
                    {
                        std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                        continue;
                    }

                    bool err = _Graph->delete_edge_2(v1, v2);
                    if (err == false) std::cout << "Ребро удалено." << std::endl;
                    else std::cout << "Такое ребро отсутствует" << std::endl;
                }

                /*---------------------------------*/
                /*---------------------------------*/
                /*---------------------------------*/

                else if (str.compare("1") == 0)
```

```
		{
			if (!_Graph->is_oriented_graph()) _Graph->complement_graph();
			else
			{
				std::cout << "Недопустимая операция для ориентированного графа" << std::endl;
				continue;
			}
			std::cout << "Дополнение графа получено." << std::endl;
		}
		else if (str.compare("2") == 0)
		{
			int e;
			int edges = _Graph->get_edges();
			if (edges == 0)
			{
				std::cout << "В графе нет ребер. Нечего подразбивать." << std::endl;
				continue;
			}

			else std::cout << "Command: Введите номер подразбиваемого ребра (0 <= var <= " << edges - 1 << "): ";
			std::cin >> e;
			if (e < 0 || e >= edges)
			{
				std::cout << "Ошибочные входные данные. Попробуйте снова." << std::endl;
				continue;
			}

			_Graph->edge_subdivision(e);
			std::cout << "Подразбиение выполнено." << std::endl;
		}

		else if (str.compare("3") == 0)
		{
			int vertices = _Graph->get_vertices();
			int count = 0;
			int * v_index = NULL;
			std::cout << "Последовательно введите неповторяющиеся вершины, которые нужно стянуть" << std::endl;
			std::cout << "Введите -1, чтобы закончить ввод вершин" << std::endl;
			std::cout << "Вершины (0 <= var <= " << vertices - 1 << "): " << std::endl;
			int i;
			bool error = false;
			while (true)
			{
				std::cin >> i;

				if (i == -1) break;

				if (!(i >= 0 && i <= vertices - 1))
				{
					std::cout << "Ошибочные данные. Такой вершины не сущестует." << std::endl;
					error = true;
					break;
				}
				if (error) break;

				for (int j = 0; j < count; j++)
				{
					if (v_index[j] == i)
					{
						std::cout << "Ошибочные данные. Вершина повторяется." << std::endl;
						error = true;
						break;
					}
				}
				if (error) break;

				v_index = (int *)realloc((void *)v_index, (++count) * sizeof(int));
				v_index[count - 1] = i;

			}
			if (error) continue;
			if (count < 2){
```

```
                        std::cout << "Ошибочные данные. Мало вершин. Нужны минимум две." <<
std::endl;
                        continue;
                }

                _Graph->graph_contraction(v_index, count);

                std::cout << "Стягивание графа выполнено." << std::endl;
        }

        else if (str.compare("4") == 0)
        {
                int x, y;
                int vertices = _Graph->get_vertices();
                if (vertices <= 1)
                {
                        std::cout << "В графе должно быть минимум 2 вершины для даннйо
операции." << std::endl;
                        continue;
                }

                std::cout << "Command: Введите номер одной вершины (0 <= var <= " <<
vertices - 1 << "): ";
                std::cin >> x;
                if (x < 0 || x >= vertices)
                {
                        std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                        continue;
                }

                std::cout << "Command: Введите номер другой вершины (0 <= var <= " <<
vertices - 1 << "): ";
                std::cin >> y;
                if (y < 0 || y >= vertices || x == y)
                {
                        if (x == y) std::cout << "Ошибочные входные данные. Вершины должны
быть разными. Попробуйте снова." << std::endl;
                        else std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                        continue;
                }

                _Graph->vertex_involving_2(x, y);
                std::cout << "Отождествление выполнено." << std::endl;
        }

        else if (str.compare("5") == 0)
        {
                int vertices = _Graph->get_vertices();

                if (vertices > 0) std::cout << "Command: Введите номер удаляемой вершины (0
<= var <= " << vertices - 1 << "): ";
                else
                {
                        std::cout << "Граф пуст. В нем нечего дублировать." << std::endl;
                        continue;
                }

                int v;
                std::cin >> v;
                if (v >= 0 && v <= vertices - 1)
                {
                        _Graph->dupliacate_vertices(v);
                        std::cout << "Дублирование выполнено." << std::endl;
                }
                else std::cout << "Неверные входные данные." << std::endl;
        }

        else if (str.compare("6") == 0)
        {
                int vertices = _Graph->get_vertices();

                if (vertices > 0) std::cout << "Command: Введите номер удаляемой вершины (0
<= var <= " << vertices - 1 << "): ";
                else
                {
                        std::cout << "Граф пуст. В нем нечего размножать." << std::endl;
                        continue;
                }
```

```
                        int v;
                        std::cin >> v;
                        if (v >= 0 && v <= vertices - 1)
                        {
                                _Graph->vertices_reproduction(v);
                                std::cout << "Размножение вершины выполнено." << std::endl;
                        }
                        else std::cout << "Неверные входные данные." << std::endl;
                }


                /*-----------------------------------*/
                /*-----------------------------------*/
                /*-----------------------------------*/


                else if (str.compare("7") == 0)
                {
                        if (total < 3)
                        {
                                std::cout << "Для данной операции нужны 3 рабочих графа. Два для
хранения исходных графов и один для сохранения туда результата операции." << std::endl;
                                std::cout << "На данный момент вы имеете всего " << total << "
графов." << std::endl;
                                continue;
                        }

                        int g1, g2;

                        std::cout << "Не повторяйтесь и не вводите номер текущего графа, т.к. туда
будет записан результат опреации" << std::endl;
                        std::cout << "Введите номер одного графа (0 <= var <= " << total - 1 << "):
";
                        std::cin >> g1;
                        if (g1 < 0 || g1 >= total || g1 == current)
                        {
                                std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                                continue;
                        }

                        std::cout << "Command: Введите номер другого графа (0 <= var <= " << total
- 1 << "): ";
                        std::cin >> g2;
                        if (g2 < 0 || g2 >= total || g2 == current || g2 == g1)
                        {
                                std::cout << "Ошибочные входные данные. Попробуйте снова." <<
std::endl;
                                continue;
                        }

                        if (Graph_arr[g1].is_oriented_graph() != Graph_arr[g2].is_oriented_graph())
                        {
                                std::cout << "Один граф ориентирован, другой нет. Так нельзя." <<
std::endl;
                                continue;
                        }

                        _Graph->graphs_union(&Graph_arr[g1], &Graph_arr[g2]);

                        std::cout << "Объединение выполнено." << std::endl;
                }

                else if (str.compare("8") == 0)
                {
                        if (total < 3)
                        {
                                std::cout << "Для данной операции нужны 3 рабочих графа. Два для
хранения исходных графов и один для сохранения туда результата операции." << std::endl;
                                std::cout << "На данный момент вы имеете всего " << total << "
графов." << std::endl;
                                continue;
                        }

                        int g1, g2;

                        std::cout << "Не повторяйтесь и не вводите номер текущего графа, т.к. туда
будет записан результат опреации" << std::endl;
```

```
                        std::cout << "Command: Введите номер одного графа (0 <= var <= " << total -
1 << "): ";
                        std::cin >> g1;
                        if (g1 < 0 || g1 >= total || g1 == current)
                        {
                                std::cout  <<  "Ошибочные  входные  данные.  Попробуйте  снова."  <<
std::endl;
                                continue;
                        }

                        std::cout << "Command: Введите номер другого графа (0 <= var <= " << total
- 1 << "): ";
                        std::cin >> g2;
                        if (g2 < 0 || g2 >= total || g2 == current || g2 == g1)
                        {
                                std::cout  <<  "Ошибочные  входные  данные.  Попробуйте  снова."  <<
std::endl;
                                continue;
                        }

                        if (Graph_arr[g1].is_oriented_graph() != Graph_arr[g2].is_oriented_graph())
                        {
                                std::cout << "Один граф ориентирован, другой нет. Так нельзя." <<
std::endl;
                                continue;
                        }

                        _Graph->graphs_connection(&Graph_arr[g1], &Graph_arr[g2]);

                        std::cout << "Соединение выполнено." << std::endl;
                }

                else if (str.compare("9") == 0)
                {
                        if (total < 3)
                        {
                                std::cout << "Для данной операции нужны 3 рабочих графа. Два для
хранения исходных графов и один для сохранения туда результата операции." << std::endl;
                                std::cout  <<  "На  данный  момент  вы  имеете  всего  "  <<  total  <<  "
графов." << std::endl;
                                continue;
                        }

                        int g1, g2;
                        std::cout << "Не повторяйтесь и не вводите номер текущего графа, т.к. туда
будет записан результат опреации" << std::endl;
                        std::cout << "Command: Введите номер одного графа (0 <= var <= " << total -
1 << "): ";
                        std::cin >> g1;
                        if (g1 < 0 || g1 >= total || g1 == current)
                        {
                                std::cout  <<  "Ошибочные  входные  данные.  Попробуйте  снова."  <<
std::endl;
                                continue;
                        }

                        std::cout << "Command: Введите номер другого графа (0 <= var <= " << total
- 1 << "): ";
                        std::cin >> g2;
                        if (g2 < 0 || g2 >= total || g2 == current || g2 == g1)
                        {
                                std::cout  <<  "Ошибочные  входные  данные.  Попробуйте  снова."  <<
std::endl;
                                continue;
                        }

                        if (Graph_arr[g1].is_oriented_graph() != Graph_arr[g2].is_oriented_graph())
                        {
                                std::cout << "Один граф ориентирован, другой нет. Так нельзя." <<
std::endl;
                                continue;
                        }

                        _Graph->product_of_graphs(&Graph_arr[g1], &Graph_arr[g2]);

                        std::cout << "Произведение графов выполнено." << std::endl;
                }
                else std::cout << "Wrong Command!" << std::endl;
        }
}
```

Graph.h

```
#ifndef GRAPH_H
#define GRAPH_H
#include <string>
//#include <vector>

class Graph
{
private:

	int ** adjacency_matrix; // матрица смежности
	int ** incidence_matrix; // матрица инцидентности // строки = номера ребра // столбец =
номера вершины
	int ** adjacency_list; // список смежности
	int vertices; // вершин
	int edges; // ребер

	//std::vector<std::string> names;

	//int * names;

	int al_v;
	int am_v;
	int im_v;
	int im_e;

	bool is_orgraph;

	/*void init_names();
	void copy_names(Graph *);*/

	void init_inc_matrix();
	void init_adj_list();
	void init_adj_matrix();
	void destroy_adj_list();
	void destroy_adj_matrix();
	void destroy_inc_matrix();
	void reinit_adj_list();
	void reinit_adj_matrix();
	void reinit_inc_matrix();

	void clear_inc_matrix(int, int);
	void clear_adj_matrix();
	void clear_all();

	int calculate_vertices(std::string *);
	void calculate_vertices_and_edges(std::string *);
	void calculate_edges();

	void fill_adjacency_list();
	void convert_inc_into_adj();
	void convert_adj_into_inc();
	void convert_adj_into_inc_orgraph();
	void convert_adj_into_inc_not_orgraph();
	void converting_from_im();
	void converting_from_am();


public:

	Graph();
	~Graph();
	void help();
	void init();
	void copy_from(Graph *);

	void print_adjacency_matrix() const;
	void print_incidence_matrix();
	void print_adjacency_list();
	void print_out();

	int get_vertices() const;
	int get_edges();
	bool is_oriented_graph();

	void read_adjacency_matrix(std::string *);
	void read_incidence_matrix(std::string *);
```

```
        void get_v_degree(int, int *);
        int * indegree_and_outcome();
        int * get_degree_sequence();

        void add_vortex();
        void delete_vortex(int);
        void add_edge(int, int);
        void delete_edge(int);
        bool delete_edge_2(int, int);

        void complement_graph(); //Дополнение графа
        void edge_subdivision(int); //Подразбиение ребра
        //void vertex_involving(int, int); //Отождествление вершин
        void vertex_involving_2(int, int); //Отождествление вершин
        void graph_contraction(int *, int); //Стягивание графа
        void dupliacate_vertices(int); //Дублирование
        void vertices_reproduction(int); //Размножение

        void graphs_union(Graph *, Graph *);
        void graphs_connection(Graph *, Graph *);
        void product_of_graphs(Graph *, Graph *);

};

#endif
```

Graph.cpp

```
#include "Graph.hpp"
#include <iostream>
#include <fstream>

Graph::Graph()
{
	init();
}

Graph::~Graph()
{
	clear_all();
}

int Graph::calculate_vertices(std::string * str)
{
	std::ifstream fin(str->c_str());

	int count;
	char c;
	for (count = 0; ; count++)
	{
		c = fin.peek();
		if (c == '\n') break;
		fin >> c;
	}
	fin.close();
	return count;
}

void Graph::calculate_vertices_and_edges(std::string * str)
{
	std::ifstream fin(str->c_str());

	char c;
	int i;
	vertices = 0;
	for (edges = 0;; edges++)
	{
		c = fin.peek();
		if (c == '\n') {
			vertices++;
			break;
		}
		if (c == EOF)
		{
			if (edges != 0) vertices++;
			fin.close();
			return;
		}
		fin >> i;
	}

	while (true)
	{
		c = fin.peek();
		if (c == '\n') vertices++;
		if (c == EOF)
		{
			fin.close();
			return;
		}
		fin >> i;
	}

	/*char c;
	for (edges = vertices = 0;; edges++)
	{
		c = fin.peek();
		if (c == '\n') vertices++;
		if (c == EOF) break;
		fin >> c;
		if (c == '-') edges--;
	}
	edges /= (++vertices);*/

	//fin.close();
```

```
}

/*-----------reading-----------*/
/*------------begin------------*/

void Graph::read_adjacency_matrix(std::string * str)
{

	clear_all();

	vertices = calculate_vertices(str);

	init_adj_matrix();


	std::ifstream fin(str->c_str());

	for (int i = 0; i < vertices; i++)
	{
		for (int j = 0; j < vertices; j++)
		{
			fin >> adjacency_matrix[i][j];

			if (i == j && (adjacency_matrix[i][j] % 2 == 1)) is_orgraph = true;
			if (i > j && (adjacency_matrix[i][j] != adjacency_matrix[j][i])) is_orgraph
= true;
		}
	}

	fin.close();

	calculate_edges();

	init_inc_matrix();
	convert_adj_into_inc();

	init_adj_list();
	fill_adjacency_list();

	//init_names();
}

void Graph::read_incidence_matrix(std::string * str)
{

	clear_all();

	calculate_vertices_and_edges(str);

	init_inc_matrix();

	std::ifstream fin(str->c_str());

	for (int i = 0; i < vertices; i++)
	{
		for (int j = 0; j < edges; j++)
		{
			fin >> incidence_matrix[i][j];
			if (incidence_matrix[i][j] == -1 && is_orgraph == false) is_orgraph = true;
		}
	}

	fin.close();

	init_adj_matrix();
	convert_inc_into_adj();

	init_adj_list();
	fill_adjacency_list();

	//init_names();
}



/*-----------reading-----------*/
/*-------------end-------------*/

/*-----------printing----------*/
/*------------begin------------*/
```

```
void Graph::print_adjacency_matrix() const
{
	/*if (is_adj_matrix == false) {
		std::cout << "Adjacency_Matrix is not ready" << std::endl;
		return;
	}*/

	//printf("\nAdjacency_Matrix\n");
	for (int i = 0; i < vertices; i++)
	{
		//printf("%d\t|", names[i]);
		for (int j = 0; j < vertices; j++)
		{
			printf("%d ", adjacency_matrix[i][j]);
		}
		printf("\n");
	}
}

void Graph::print_incidence_matrix()
{
	//if (is_inc_matrix == false) convert_adj_into_inc();

	//printf("\nIncidence_Matrix\n");
	if (get_edges() == -1) calculate_edges();

	for (int i = 0; i < vertices; i++)
	{
		//printf("%d\t|", names[i]);
		for (int j = 0; j < im_e; j++)
		{
			if (!is_orgraph) printf("%d ", incidence_matrix[i][j]);
			else printf("%d\t", incidence_matrix[i][j]);
		}
		printf("\n");
	}
}

void Graph::print_adjacency_list()
{
	//if (is_adj_list == false) fill_adjacency_list();

	//printf("\nAdjacency_List\n");

	for (int i = 0; i < vertices; i++)
	{
		//printf("{%d", names[i]);
		printf("{%d", i);
		for (int j = 0; adjacency_list[i][j] != -1; j++)
		{
			printf(" %d", adjacency_list[i][j]);
		}
		printf("}\n");
	}
}

/*-----------printing----------*/
/*-------------end-------------*/

/*-----------clearing----------*/
/*------------begin------------*/

void Graph::clear_adj_matrix()
{
	for (int i = 0; i < am_v; i++) for (int j = 0; j < am_v; j++) adjacency_matrix[i][j] = 0;
}

void Graph::clear_inc_matrix(int x, int y)
{
	for (int i = 0; i < x; i++)
	{
		for (int j = 0; j < y; j++)
		{
			incidence_matrix[i][j] = 0;
		}
	}
}

void Graph::clear_all()
```

```
{
	if (vertices < 0) return;

	for (int i = 0; i < vertices; i++) free(adjacency_matrix[i]);
	free(adjacency_matrix);

	for (int i = 0; i < vertices; i++) free(incidence_matrix[i]);
	free(incidence_matrix);

	for (int i = 0; i < vertices; i++) free(adjacency_list[i]);
	free(adjacency_list);

	adjacency_matrix = NULL;
	incidence_matrix = NULL;
	adjacency_list = NULL;
	vertices = 0;
	edges = 0;

	is_orgraph = false;

	//names.clear();
	//if (names) free(names);

	am_v = 0;
	al_v = 0;
	im_v = 0;
	im_e = 0;

}

/*-----------clearing----------*/
/*-------------end-------------*/

/*---------converting----------*/
/*-----------begin-------------*/

void Graph::fill_adjacency_list()
{

	for (int i = 0; i < vertices; i++)
	{
		int count = 0;
		for (int j = 0; j < vertices; j++)
		{
			int temp = adjacency_matrix[i][j];
			if (temp > 0)
			{
				if (i == j && !is_orgraph)
				{
					temp = temp >> 1;
				}
				temp += count;
				adjacency_list[i] = (int *)realloc((void *)adjacency_list[i], (temp
+ 1) * sizeof(int));
				while (count < temp)
				{
					adjacency_list[i][count] = j + 1;
					count++;
				}
				adjacency_list[i][count] = -1;
			}
		}
	}
}

void Graph::convert_inc_into_adj()
{
	clear_adj_matrix();

	int in, out;
	bool loop;

	int loops = 0;
	for (int i = 0; i < vertices; i++) if (adjacency_matrix) if (adjacency_matrix[i][i] > 0)
loops += (adjacency_matrix[i][i] >> 1);
	for (int j = 0; j < edges - loops; j++)
	{
		in = out = -1;
		loop = false;
```

```
                for (int i = 0; i < vertices; i++)
                {
                        int temp = incidence_matrix[i][j];
                        if (temp == 1)
                        {
                                if (is_orgraph) out = i;
                                else if (in == -1) in = i;
                                else out = i;
                        }
                        else if (temp == -1) in = i;
                        else if (temp == 2)
                        {
                                loop = true;
                                if (is_orgraph) adjacency_matrix[i][i]++;
                                else adjacency_matrix[i][i] += 2;
                        }
                }

                if (loop) continue;

                if (is_orgraph) adjacency_matrix[out][in]++;
                else
                {
                        adjacency_matrix[out][in]++;
                        adjacency_matrix[in][out]++;
                }
        }
}

void Graph::convert_adj_into_inc()
{
        (is_orgraph) ? convert_adj_into_inc_orgraph() : convert_adj_into_inc_not_orgraph();
}

void Graph::convert_adj_into_inc_orgraph()
{
        int y = 0;

        for (int i = 0; i < vertices; i++)
        {
                for (int j = 0; j < vertices; j++)
                {
                        int temp = adjacency_matrix[i][j];
                        if (temp > 0)
                        {
                                temp += y;
                                if (i == j)
                                {
                                        for (; y < temp; y++)
                                        {
                                                incidence_matrix[i][y] = 2;
                                        }
                                }
                                else
                                {
                                        for (; y < temp; y++)
                                        {
                                                incidence_matrix[i][y] = 1;
                                                incidence_matrix[j][y] = -1;
                                        }
                                }
                        }
                }
        }
}

void Graph::convert_adj_into_inc_not_orgraph()
{
        int y = 0;

        for (int i = 0; i < vertices; i++)
        {
                for (int j = i; j < vertices; j++)
                {
                        int temp = adjacency_matrix[i][j];
                        if (temp > 0)
                        {
                                if (i == j)
                                {
                                        temp = temp >> 1;
```

```
                                        temp += y;
                                        while (y < temp)
                                        {
                                                incidence_matrix[i][y] = 2;
                                                y++;
                                        }
                                }
                                else
                                {
                                        temp += y;
                                        while (y < temp)
                                        {
                                                incidence_matrix[i][y] = incidence_matrix[j][y] = 1;
                                                y++;
                                        }
                                }
                        }
                }
        }
}

void Graph::converting_from_im()
{
        im_v = vertices;
        im_e = edges;

        reinit_adj_list();
        reinit_adj_matrix();
        convert_inc_into_adj();
        fill_adjacency_list();
}

void Graph::converting_from_am()
{
        calculate_edges();

        reinit_inc_matrix();
        convert_adj_into_inc();

        //print_incidence_matrix();

        reinit_adj_list();
        fill_adjacency_list();
}

/*----------converting----------*/
/*------------end--------------*/

/*----------build-up-----------*/
/*-----------begin-------------*/

void Graph::init()
{
        adjacency_matrix = NULL;
        incidence_matrix = NULL;
        adjacency_list = NULL;
        vertices = 0;
        edges = 0;

        //names = NULL;

        is_orgraph = false;
        al_v = 0;
        am_v = 0;
        im_v = 0;
        im_e = 0;
}

void Graph::init_adj_matrix()
{
        adjacency_matrix = (int **)malloc(vertices * sizeof(int *));
        for (int i = 0; i < vertices; i++) adjacency_matrix[i] = (int *)malloc(vertices *
sizeof(int));
        am_v = vertices;

        //names = (int *)malloc(vertices * sizeof(int));
        //for (int i = 0; i < vertices; i++) names[i] = i;
        //for (int i = 0; i < vertices; i++) names.push_back();
}
```

```
/*void Graph::init_names()
{
	for (int i = 0; i < vertices; i++) names[i] = i + 1;
}*/

void Graph::init_inc_matrix()
{
	int loops = 0;
	//for (int i = 0; i < vertices; i++) if (adjacency_matrix) if (adjacency_matrix[i][i] > 0)
loops += (adjacency_matrix[i][i] >> 1);
	incidence_matrix = (int **)malloc(vertices * sizeof(int *));
	for (int i = 0; i < vertices; i++)
	{
		//incidence_matrix[i] = (int *)malloc((edges - loops) * sizeof(int));
		incidence_matrix[i] = (int *)malloc(edges * sizeof(int));
	}

	im_v = vertices;
	//im_e = edges - loops;
	im_e = edges;
	clear_inc_matrix(im_v, im_e);
}

void Graph::init_adj_list()
{


	adjacency_list = (int **)malloc(vertices * sizeof(int *));
	for (int i = 0; i < vertices; i++)
	{
		adjacency_list[i] = (int *)malloc(sizeof(int));
		adjacency_list[i][0] = -1;
	}

	al_v = vertices;
}

void Graph::destroy_adj_list()
{
	if (adjacency_list == NULL) return;
	for (int i = 0; i < al_v; i++) free(adjacency_list[i]);
	free(adjacency_list);
	adjacency_list = NULL;
}

void Graph::destroy_adj_matrix()
{
	if (adjacency_matrix == NULL) return;
	for (int i = 0; i < am_v; i++) free(adjacency_matrix[i]);
	free(adjacency_matrix);
	adjacency_matrix = NULL;
}

void Graph::destroy_inc_matrix()
{
	if (incidence_matrix == NULL) return;
	for (int i = 0; i < im_v; i++) free(incidence_matrix[i]);
	free(incidence_matrix);
	incidence_matrix = NULL;
}

void Graph::reinit_adj_list()
{
	destroy_adj_list();
	init_adj_list();
}

void Graph::reinit_adj_matrix()
{
	destroy_adj_matrix();
	init_adj_matrix();
}

void Graph::reinit_inc_matrix()
{
	destroy_inc_matrix();
	init_inc_matrix();
}

/*----------build-up----------*/
```

```
/*------------end-------------*/

int Graph::get_vertices() const
{
	return vertices;
}

int Graph::get_edges()
{
	calculate_edges();
	return edges;
}

void Graph::calculate_edges()
{
	edges = 0;
	for (int i = 0; i < vertices; i++)
	{
		int j = (is_orgraph) ? 0 : i;
		for (; j < vertices; j++)
		{
			int temp = adjacency_matrix[i][j];
			if (temp > 0)
			{
				edges += (i == j && !is_orgraph) ? (temp >> 1) : temp;
			}
		}
	}
}

void Graph::get_v_degree(int v, int *result)
{
	result[0] = result[1] = 0;
	for (int i = 0; i < vertices; i++)
	{
		result[0] += adjacency_matrix[v][i];
		if (is_orgraph) result[1] += adjacency_matrix[i][v];
	}
}

int * Graph::indegree_and_outcome()
{
	int * in_out = (int *)calloc(vertices * 2, sizeof(int));
	for (int i = 0; i < vertices; i++)
	{
		for (int j = 0; j < vertices; j++)
		{
			in_out[i] += adjacency_matrix[i][j];
			in_out[i + vertices] += adjacency_matrix[j][i];
		}
	}
	return in_out;
}

int * Graph::get_degree_sequence()
{
	if (is_orgraph) return indegree_and_outcome();

	int * sqnce = (int *)malloc(vertices*2*sizeof(int));
	for (int i = 0; i < vertices; i++)
	{
		sqnce[i] = 0;
		for (int j = 0; j < vertices; j++)
		{
			sqnce[i] += adjacency_matrix[i][j];
		}
	}

	for (int i = 0; i < vertices; i++) sqnce[i + vertices] = i;

	int max_index = -1;
	for (int i = 0; i < vertices; i++)
	{
		max_index = i;
		for (int j = i; j < vertices; j++)
		{
			if (sqnce[max_index] < sqnce[j])
			{
				int temp = sqnce[max_index];
				sqnce[max_index] = sqnce[j];
```

```
                                sqnce[j] = temp;

                                temp = sqnce[max_index + vertices];
                                sqnce[max_index + vertices] = sqnce[j + vertices];
                                sqnce[j + vertices] = temp;
                        }
                }
        }
        return sqnce;
}

void Graph::help()
{
        std::cout << std::endl;
        std::cout << "help\t-\tВыдать меню команд" << std::endl;
        std::cout << "exit\t-\tВыход из программы" << std::endl;
        std::cout << std::endl;
        std::cout << "new\t-\tСоздать новый граф" << std::endl;
        std::cout << "change (cng)\t-\tВыбрать граф для последующей работы" << std::endl;
        std::cout << "copy (cpy)\t-\tСкопировать состояние из другого графа" << std::endl;
        std::cout << std::endl;
        std::cout << "read_am\t-\tСчитать матрицу смежности из файла input.txt" << std::endl;
        std::cout << "read_im\t-\tСчитать матрицу инцидентности из файла input2.txt" << std::endl;
        std::cout << std::endl;
        std::cout << "get_v\t-\tОпределение числа вершин" << std::endl;
        std::cout << "get_e\t-\tОпределение числа ребер" << std::endl;
        std::cout << "get_deg\t-\tОпределение степени вершины" << std::endl;
        std::cout  <<  "get_deg_sqnce\t-\tОпределение  степенной  последовательности  графа"  <<
std::endl;
        std::cout << std::endl;
        std::cout << "print_am\t-\tОпределение матрицы смежности" << std::endl;
        std::cout << "print_im\t-\tОпределение матрицы инцидентности" << std::endl;
        std::cout << "print_al\t-\tОпределение списка смежности" << std::endl;
        std::cout << "print_out\t-\tВывод в файл output.txt как список смежностей" << std::endl;
        std::cout << std::endl;
        std::cout << "add_v\t-\tДобавление вершин" << std::endl;
        std::cout << "del_v\t-\tУдаление вершин" << std::endl;
        std::cout << "add_e\t-\tДобавление ребра" << std::endl;
        std::cout << "del_e\t-\tУдаление ребра" << std::endl;
        std::cout << std::endl;
        std::cout << "1\t-\tОпределить дополнения графа" << std::endl;
        std::cout << "2\t-\tПодразбиение ребра" << std::endl;
        std::cout << "3\t-\tСтягивание графа" << std::endl;
        std::cout << "4\t-\tОтождествление вершин" << std::endl;
        std::cout << "5\t-\tДублирование вершин" << std::endl;
        std::cout << "6\t-\tРазмножение вершин" << std::endl;
        std::cout << "7\t-\tОбъединение (дизъюнктивное) графов" << std::endl;
        std::cout << "8\t-\tСоединение графов" << std::endl;
        std::cout << "9\t-\tПроизведение графов" << std::endl;
        std::cout << std::endl;
}

bool Graph::is_oriented_graph()
{
        return is_orgraph;
}

void Graph::copy_from(Graph *G)
{
        vertices = G->vertices;
        reinit_adj_matrix();

        for (int i = 0; i < vertices; i++)
        {
                for (int j = 0; j < vertices; j++)
                {
                        adjacency_matrix[i][j] = G->adjacency_matrix[i][j];
                }
        }

        //copy_names(G);
        converting_from_am();
}

/*void Graph::copy_names(Graph *G)
{
        int ver = G->vertices;
        names = (int *)malloc(ver * sizeof(int));
        for (int i = 0; i < ver; i++) names[i] = G->names[i];
}*/
```

```
/*-----------------------------------------*/
/*-----------------------------------------*/
/*-----------------------------------------*/
/*-----------------------------------------*/

void Graph::add_vortex()
{
	vertices++;

	incidence_matrix = (int **)realloc((void *)incidence_matrix, vertices * sizeof(int *));
	incidence_matrix[vertices - 1] = (int *)calloc(edges, sizeof(int));

	converting_from_im();
}

void Graph::delete_vortex(int v)
{
	//Удаление всех инцидентных ребер
	for (int i = edges - 1; i >= 0; i--)
	{
		int temp = incidence_matrix[v][i];
		if (temp == 1 || temp == -1 || temp == 2)
		{
			//копирование (сдвиг) столбцов
			delete_edge(i);
		}
	}

	//Удаление самой вершины
	//
	//копирование (сдвиг) строк
	for (int k = v + 1; k < vertices; k++)
	{
		for (int l = 0; l < edges; l++)
		{
			incidence_matrix[k - 1][l] = incidence_matrix[k][l];
		}
	}

	//удаление лишнего (крайнего) столбца
	vertices--;
	for (int k = 0; k < vertices; k++)
	{
		incidence_matrix = (int **)realloc((void *)incidence_matrix, vertices * sizeof(int
*));
	}

	converting_from_im();
}

void Graph::add_edge(int out, int in)
{
	edges++;

	for (int i = 0; i < vertices; i++)
	{
		incidence_matrix[i] = (int *)realloc((void *)incidence_matrix[i], edges *
sizeof(int));
		incidence_matrix[i][edges - 1] = 0;
	}
	incidence_matrix[out][edges - 1]++;
	(is_orgraph && in != out) ? incidence_matrix[in][edges - 1]-- : incidence_matrix[in][edges
- 1]++;

	converting_from_im();
}

void Graph::delete_edge(int e)
{
	//копирование (сдвиг) столбцов
	for (int k = e + 1; k < edges; k++)
	{
		for (int l = 0; l < vertices; l++)
		{
			incidence_matrix[l][k - 1] = incidence_matrix[l][k];
		}
	}

	//удаление лишнего (крайнего) столбца
```

```
	edges--;
	for (int k = 0; k < vertices; k++)
	{
		incidence_matrix[k] = (int *)realloc((void *)incidence_matrix[k], edges *
sizeof(int));
	}
}

bool Graph::delete_edge_2(int x, int y)
{
	if (!is_orgraph)
	{
		if (adjacency_matrix[x][y] && adjacency_matrix[y][x])
		{
			adjacency_matrix[x][y]--;
			adjacency_matrix[y][x]--;
		}
		else return true;
	}
	else
	{
		if (adjacency_matrix[x][y]) adjacency_matrix[x][y]--;
		else return true;
	}

	edges--;

	converting_from_am();

	return false;
}

void Graph::print_out()
{
	std::ofstream fout("output.txt", std::ios_base::trunc);

	for (int i = 0; i < vertices; i++)
	{
		fout << "{" << i;
		//printf("{%d", i);
		for (int j = 0; adjacency_list[i][j] != -1; j++)
		{
			fout << " " << adjacency_list[i][j];
			//printf(" %d", adjacency_list[i][j]);
		}
		fout << "}";
		//printf("}\n");
	}

	fout.close();
}


/*-----------------------------------------*/
/*-----------------------------------------*/
/*-----------------------------------------*/
/*-----------------------------------------*/


void Graph::complement_graph()
{
	/*for (int i = 0; i < vertices; i++)
	{
		for (int j = 0; j < vertices; j++)
		{
			adjacency_matrix[i][j] = 1;
		}
		adjacency_matrix[i][i] = 0;
	}


	for (int i = 0; i < vertices; i++)
	{
		for (int j = 0; adjacency_list[i][j] != -1; j++)
		{
			adjacency_matrix[i][adjacency_list[i][j]] = 0;
		}
	}*/

	for (int i = 0; i < vertices; i++)
```

```
	{
		for (int j = i + 1; j < vertices; j++)
		{
			int new_val = (adjacency_matrix[i][j] > 0) ? 0 : 1;
			adjacency_matrix[i][j] = adjacency_matrix[j][i] = new_val;
		}
	}

	for (int i = 0; i < vertices; i++) adjacency_matrix[i][i] = 0;

	calculate_edges();

	converting_from_am();
}

void Graph::edge_subdivision(int e)
{
	int out, in;
	out = in = -1;
	bool loop = false;
	for (int i = 0; i < vertices; i++)
	{
		int temp = incidence_matrix[i][e];
		if (temp == 2)
		{
			loop = true;
			out = in = i;
			break;
		}
		else if (temp == 1)
		{
			if (out != -1){
				in = i;
				break;
			}
			else out = i;
		}
		else if (temp == -1) in = i;
	}

	delete_edge(e);

	vertices++;
	incidence_matrix = (int **)realloc((void *)incidence_matrix, vertices * sizeof(int));

	edges += 2;
	incidence_matrix[vertices - 1] = (int *)malloc(edges * sizeof(int));
	for (int i = 0; i < edges - 2; i++) incidence_matrix[vertices - 1][i] = 0;
	for (int i = 0; i < vertices - 1; i++)
	{
		incidence_matrix[i] = (int *)realloc((void *)incidence_matrix[i], edges *
sizeof(int));
		incidence_matrix[i][edges - 2] = incidence_matrix[i][edges - 1] = 0;
	}

	int new_var;
	new_var = (is_orgraph) ? -1 : 1;
	incidence_matrix[out][edges - 2] = incidence_matrix[vertices - 1][edges - 1] = 1;
	incidence_matrix[vertices - 1][edges - 2] = incidence_matrix[in][edges - 1] = new_var;

	converting_from_im();
}

void Graph::vertex_involving_2(int x, int y)
{
	add_vortex();
	int check;
	if (adjacency_matrix[x][x] || adjacency_matrix[y][y])
	{
		add_edge(vertices - 1, vertices - 1);
		//adjacency_matrix[vertices - 1][vertices - 1] = (is_orgraph) ? 1 : 2;
		check = adjacency_matrix[vertices - 1][vertices - 1];
	}

	for (int i = 0; i < vertices - 1; i++)
	{
		if (i != x && i != y)
		{
			if (adjacency_matrix[i][x] || adjacency_matrix[i][y]) add_edge(i, vertices-
1); //adjacency_matrix[i][vertices - 1] = 1;
```

```
                        check = adjacency_matrix[i][vertices - 1];
                        if (is_orgraph)
                        {
                                if      (adjacency_matrix[x][i]      ||      adjacency_matrix[y][i])
add_edge(vertices - 1, i); //adjacency_matrix[vertices - 1][i] = 1;
                                check = adjacency_matrix[vertices - 1][i];
                        }
                }
        }
        //print_adjacency_matrix();
        int max, min;
        if (x > y)
        {
                max = x;
                min = y;
        }
        else
        {
                max = y;
                min = x;
        }


        //converting_from_am();

        delete_vortex(max);
        //print_adjacency_matrix();
        delete_vortex(min);
        //print_adjacency_matrix();
}

void Graph::graph_contraction(int * v_index, int size)
{
        int var_1, var_2;
        var_1 = v_index[0];
        var_2 = v_index[1];

        for (int k = 2; k < size; k++)
        {
                vertex_involving_2(var_1, var_2);
                //print_incidence_matrix();
                //print_adjacency_matrix();
                var_1 = get_vertices() - 1;
                //int check;
                int v_k = v_index[k];
                for (int i = 0; i < k; i++)
                {
                        if (v_index[i] < v_k) v_index[k]--;
                        //check = v_index[k];
                }
                var_2 = v_index[k];
        }
        vertex_involving_2(var_1, var_2);

        //converting_from_im();
}

void Graph::dupliacate_vertices(int x)
{
        add_vortex();

        int loop = adjacency_matrix[x][x];
        if (is_orgraph == false) loop = loop >> 1;
        for (int i = 0; i < loop; i++) add_edge(vertices - 1, vertices - 1);

        for (int i = 0; i < vertices - 1; i++)
        {
                if (i == x) continue;

                int count = adjacency_matrix[i][x];
                if (count){
                        for (int j = 0; j < count; j++) add_edge(i, vertices - 1);
                }

                if (is_orgraph)
                {
                        int count = adjacency_matrix[x][i];
                        if (count){
                                for (int j = 0; j < count; j++) add_edge(vertices - 1, i);
                        }
```

```
                }
        }
}

void Graph::vertices_reproduction(int x)
{
        dupliacate_vertices(x);
        add_edge(x, vertices - 1);
        if (is_orgraph) add_edge(vertices - 1, x);
}


/*-----------------------------------------*/
/*-----------------------------------------*/
/*-----------------------------------------*/
/*-----------------------------------------*/


void Graph::graphs_union(Graph * G1, Graph * G2)
{
        if (G1->vertices < G2->vertices)
        {
                Graph * temp = G1;
                G1 = G2;
                G2 = temp;
        }

        vertices = G1->vertices;

        reinit_adj_matrix();
        //clear_adj_matrix();

        for (int i = 0; i < vertices; i++)
        {
                for (int j = 0; j < vertices; j++)
                {
                        adjacency_matrix[i][j] = G1->adjacency_matrix[i][j];
                }
        }
        for (int i = 0; i < G2->vertices; i++)
        {
                for (int j = 0; j < G2->vertices; j++)
                {
                        if        (G2->adjacency_matrix[i][j]        >        adjacency_matrix[i][j])
adjacency_matrix[i][j] = G2->adjacency_matrix[i][j];
                }
        }

        //print_adjacency_matrix();
        converting_from_am();
}

void Graph::graphs_connection(Graph * G1, Graph * G2)
{
        int g1_v, g2_v;
        g1_v = G1->vertices;
        g2_v = G2->vertices;
        if (g1_v < g2_v)
        {
                Graph * temp = G1;
                G1 = G2;
                G2 = temp;
        }

        vertices = g1_v + g2_v;

        reinit_adj_matrix();

        for (int i = 0; i < g1_v; i++)
        {
                for (int j = 0; j < vertices; j++)
                {
                        adjacency_matrix[i][j] = (j < G1->vertices) ? G1->adjacency_matrix[i][j] :
1;
                }
        }
        for (int i = g1_v; i < vertices; i++)
        {
                for (int j = 0; j < g1_v; j++)
                {
```

```
                        adjacency_matrix[i][j] = 1;
                }
        }
        for (int i = g1_v; i < vertices; i++)
        {
                for (int j = g1_v; j < vertices; j++)
                {
                        adjacency_matrix[i][j] = G2->adjacency_matrix[i - g1_v][j - g1_v];
                }
        }

        converting_from_am();
}

void Graph::product_of_graphs(Graph * G1, Graph * G2)
{
        if (G1->vertices < G2->vertices)
        {
                Graph * temp = G1;
                G1 = G2;
                G2 = temp;
        }

        vertices = G1->vertices * G2->vertices;

        reinit_adj_matrix();
        clear_adj_matrix();

        for (int k = 0; k < G2->vertices; k++)
        {
                int x = 0;
                for (int i = k * G1->vertices; i < (k + 1) * G1->vertices; i++, x++)
                {
                        int y = 0;
                        for (int j = k * G1->vertices; j < (k + 1) * G1->vertices; j++, y++)
                        {
                                adjacency_matrix[i][j] = G1->adjacency_matrix[x][y];
                        }
                }
        }
        for (int i = 0; i < G2->vertices; i++)
        {
                for (int j = 0; j< G2->vertices; j++)
                {
                        int value = G2->adjacency_matrix[i][j];
                        if (value > 0)
                        {
                                int start_x = i * G1->vertices;
                                int start_y = j * G1->vertices;
                                int end_x = (i + 1) * G1->vertices;
                                while (start_x < end_x)
                                {
                                        adjacency_matrix[start_x][start_y] += value;
                                        start_x++;
                                        start_y++;
                                }
                        }
                }
        }

        converting_from_am();
}
```