

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

|  |
| --- |
| **Институт информационных технологий (ИТ)** |
| **Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)** |

**ОТЧЁТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №5**

**по дисциплине «Структуры и алгоритмы обработки данных»**

Тема: **«**Основные алгоритмы работы с графами.»

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет представлен к рассмотрению:  Студентка группы ИНБО-01-20 | «1» ноября 2021 г. |  | Тульцова А.Д. |
| (подпись) | | | |
| Преподаватель | «1» ноября 2021 г. |  | Сорокин А.В. |
|  | (подпись) | | |

Москва, 2021 г.

CОДЕРЖАНИЕ

[Цель работы 3](#_Toc86702501)

[Постановка задачи 3](#_Toc86702502)

[Подход к решению. 3](#_Toc86702503)

[Алгоритмы операций на псевдокоде. 5](#_Toc86702504)

[Код программы. 8](#_Toc86702505)

[Тестирование программы. 12](#_Toc86702506)

[Вывод 20](#_Toc86702507)

[Список информационных источников 21](#_Toc86702508)

# Цель работы

Получение практических навыков по выполнению операций над структурой данных «граф».

# Постановка задачи

Выполнить разработку программы управления графом, в соответствии с вариантом, на основе класса «Граф». Предусмотреть в качестве данных: количество вершин в графе, структура для хранения графа.

**Вариант 11.**

|  |  |
| --- | --- |
| Представление графа в памяти | Задачи варианта |
| Матрица смежности | Ввод с клавиатуры графа (применение операции вставки ребра в граф).  Определить, является ли граф связанным.  Составить программу нахождения кратчайшего пути в графе от заданной вершины к другой заданной вершине методом «Дейкстры». |

**Дано:**

Произвольный граф (ориентированный или неориентированный, связный или несвязный) с известным количеством вершин.

**Результат.**

Отображение графа в виде матрицы смежности.

Реализованные операции варианта.

## Подход к решению.

1. Разработан класс графа, реализующий, согласно варианту, следующие методы: создание графа посредством применения операций вставки ребра в граф, вывод матрицы смежности графа, проверка графа на связность, нахождение величины кратчайшего пути и вывод кратчайшего пути от заданной вершины к другой заданной вершине методом «Дейкстры».
2. Разработан класс узла графа, содержащий информационную часть и поле индекса.
3. Разработан консольный пользовательский интерфейс для тестирования работоспособности программы.
4. Разработаны методы обработки графа:
   1. Метод вставки ребра в граф – добавление ребра с заданным весом между двумя заданными вершинами;
   2. Метод получения индекса узла – возврат индекса для заданного узла графа;
   3. Метод проверки на сильную связность – нахождение количества достижимых вершин для заданного узла графа через обход «в глубину» для проверки на сильную связность;
   4. Метод проверки на слабую связность – нахождение количества достижимых вершин для заданного узла графа через обход «в глубину» для проверки на слабую связность;
   5. Метод проверки графа на связность – запуск метода проверки на сильную связность для неориентированного графа или общее исследование связности для ориентированного графа;
   6. Метод «Дейкстры» – поиск величин кратчайших путей до каждой вершины для заданной вершины;
   7. Метод восстановления кратчайшего пути – возврат кратчайшего маршрута из одной заданной вершины в другую;
   8. Метод нахождения величины кратчайшего пути – возврат величины кратчайшего пути между двумя заданными вершинами графа, найденного с помощью метода алгоритма Дейкстры.
5. Разработаны методы приложения для тестирования:
   1. Метод для тестирования вывода графа – организация ввода графа и вывода его матрицы смежности;
   2. Метод для тестирования нахождения кратчайшего пути – организация нахождения кратчайшего пути от одной заданной вершины к другой с помощью алгоритма Дейкстры и вывода результатов работы алгоритмов в консоль;
   3. Метод для тестирования проверки графа на связность – организация проверки графа на связность и вывода результата работы алгоритмов в консоль.

## Алгоритмы операций на псевдокоде.

**Метод вставки ребра в граф:**

процедура connect(первая\_вершина, вторая\_вершина, вес := 1):  
 первая\_вершина := self.get\_index\_from\_node(первая\_вершина)

вторая\_вершина := self.get\_index\_from\_node(вторая\_вершина)  
 матрица\_смежности[первая\_вершина][вторая\_вершина] := вес

**Метод получения индекса узла:**

функция get\_index\_from\_node(узел):  
 если принадлежность(узел, int):  
 возврат узел  
 иначе:  
 возврат узел.индекс

**Метод проверки на сильную связность:**

функция connectivity\_DFS(целочисленный индекс\_узла, visited := массив[]):  
 visited.добавить\_элемент(индекс\_узла)  
 для каждого i от 0 до длина(матрица\_смежности) - 1:  
 если матрица\_смежности[индекс\_узла][i] есть не None и i не в visited:  
 connectivity\_DFS(i, visited)  
 возврат длина(visited)

**Метод проверки на слабую связность:**

функция weak\_connectivity\_DFS(целочисленный индекс\_узла, visited := массив[]):  
 visited.добавить\_элемент(индекс\_узла)  
 для каждого i от 0 до длина(матрица\_смежности) - 1:  
 если (матрица\_смежности[индекс\_узла][i] есть не None или матрица\_смежности[i][индекс\_узла] есть не None) и i не в visited:  
 weak\_connectivity\_DFS(i, visited)  
 возврат длина(visited)

**Метод проверки графа на связность:**

функция connectivity(булевский directed):  
 если directed есть Ложь:  
 для каждого индекс\_узла от 0 до длина(матрица\_смежности) – 1:  
 visited := множество()  
 если connectivity\_DFS(индекс\_узла, visited) < длина(матрица\_смежности):  
 возврат массив[Ложь]  
 иначе:  
 для каждого индекс\_узла от 0 до длина(матрица\_смежности) – 1:  
 visited := множество()  
 если connectivity\_DFS(индекс\_узла, visited) < длина(матрица\_смежности):  
 если weak\_connectivity\_DFS(индекс\_узла, visited) < длина(матрица\_смежности):  
 возврат массив[Ложь]  
 иначе:  
 возврат массив[Истина, Ложь]  
 возврат массив[Истина, Истина]

**Метод «Дейкстры»:**

функция dijkstra(узел):  
 узел := get\_index\_from\_node(узел)  
 из collections импортировать default\_словарь  
 граф := default\_словарь(список)  
 для каждого row от 0 до длина(матрица\_смежности) - 1:  
 для каждого col от 0 до длина(матрица\_смежности) - 1:  
 если матрица\_смежности[row][col] есть не None и матрица\_смежности[row][col] != 0:  
 граф[row] := граф[row] + [(матрица\_смежности[row][col], col)]  
 nodes\_to\_visit := массив[]  
 nodes\_to\_visit.добавить\_элемент\_в\_конец((0, узел))  
 visited := множество()  
 min\_dist := {i: ∞ для каждого i от 0 до длина(матрица\_смежности) - 1}  
 min\_dist[узел] := 0  
 пока длина(nodes\_to\_visit) > 0:

вес, текущий\_узел := минимальный\_элемент(nodes\_to\_visit)  
 nodes\_to\_visit.удалить\_элемент((вес, текущий\_узел))  
 если текущий узел в visited:  
 принудительный\_запуск\_следующего\_прохода\_цикла  
 visited.добавить\_элемент(текущий\_узел)  
 для след\_вес, след\_узел в graph[текущий\_узел]:  
 если вес + след\_вес < min\_dist[след\_узел] и след\_узел не в visited:  
 min\_dist[след\_узел] := вес + след\_вес  
 nodes\_to\_visit. добавить\_элемент\_в\_конец((вес + след\_вес, след\_узел))  
 возврат min\_dist

**Метод восстановления кратчайшего пути:**

функция path\_restoring(узел1, узел2):  
 visited := [None] \* длина(матрица\_смежности)  
 узел1 := self.get\_index\_from\_node(узел1)  
 узел2 := self.get\_index\_from\_node(узел2)  
 visited[0] := узел2 + 1  
 пред\_индекс := 1  
 вес := shortest\_path(узел1, узел2)  
 пока узел2 != узел1:  
 для каждого i от 0 до длина(матрица\_смежности) - 1:  
 если матрица\_смежности[i][узел2] есть не None и матрица\_смежности[i][узел2] != 0:  
 temp := вес - матрица\_смежности[i][узел2]  
 если temp == shortest\_path(узел1, i):  
 вес := temp  
 узел2 := i  
 visited[пред\_индекс] := i + 1  
 пред\_индекс := пред\_индекс + 1

пока None в visited:

visited.удалить\_элемент(None)  
 возврат развернуть\_массив(visited)

**Метод нахождения величины кратчайшего пути:**

функция shortest\_path(узел1, узел2):  
 узел2 := get\_index\_from\_node(узел2)  
 возврат dijkstra(узел1)[узел2]

## Код программы.

Класс узла графа:

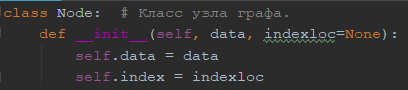


Рисунок 1 – Класс узла графа.

Класс графа:



Рисунок 2 – Класс графа.

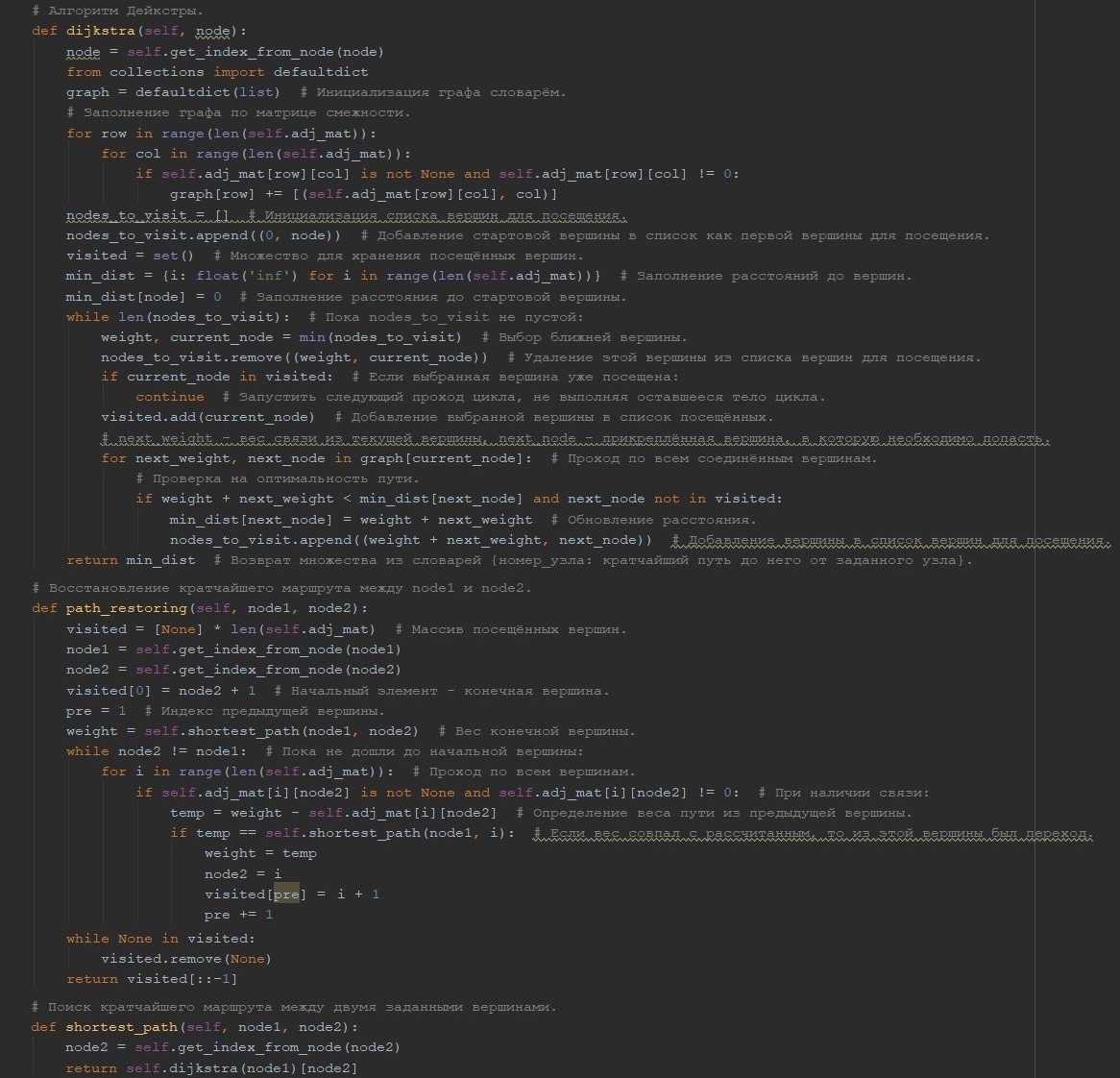


Рисунок 3 – Класс графа (продолжение).

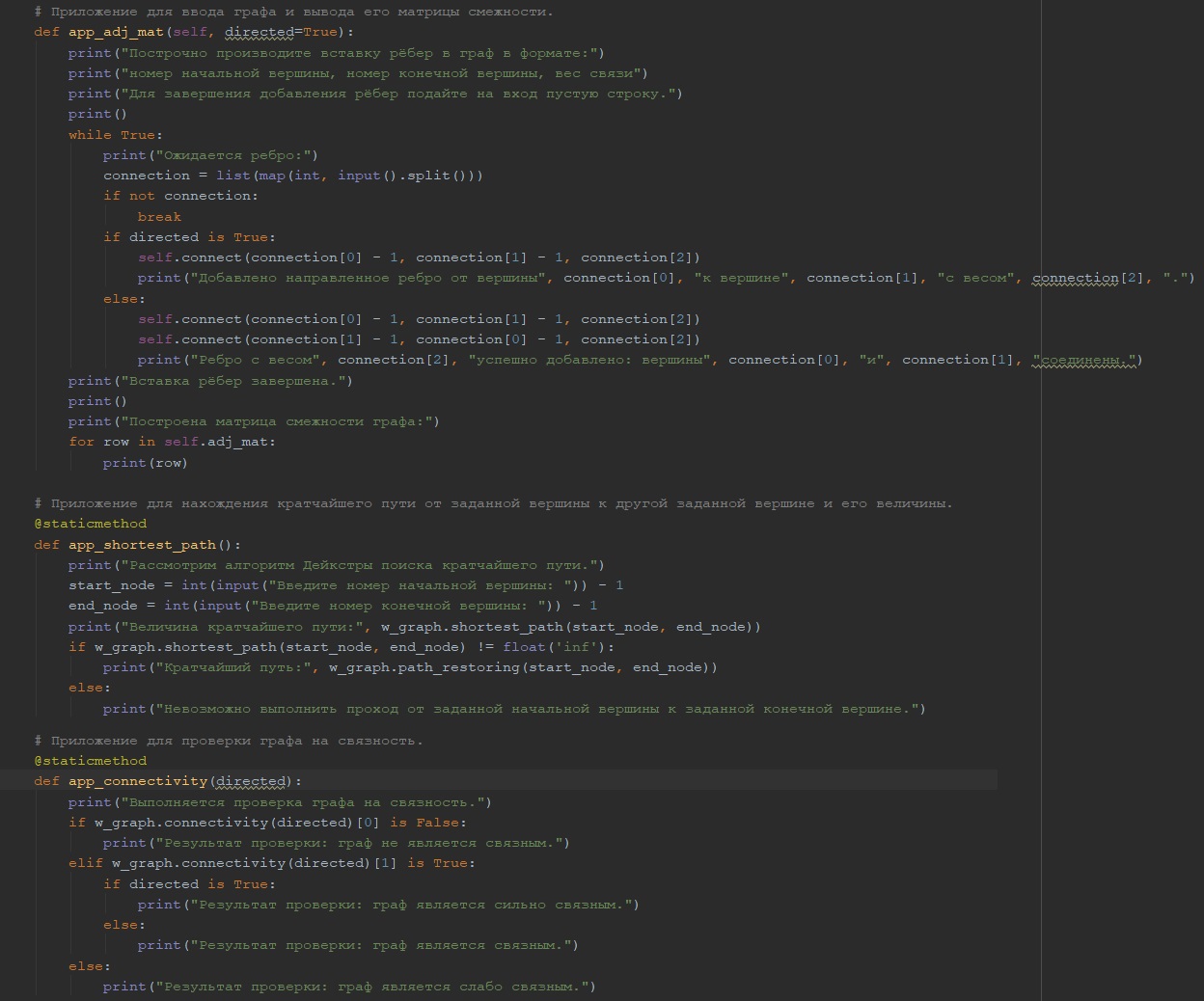


Рисунок 4 – Класс графа (продолжение).

Основная функция для тестирования:

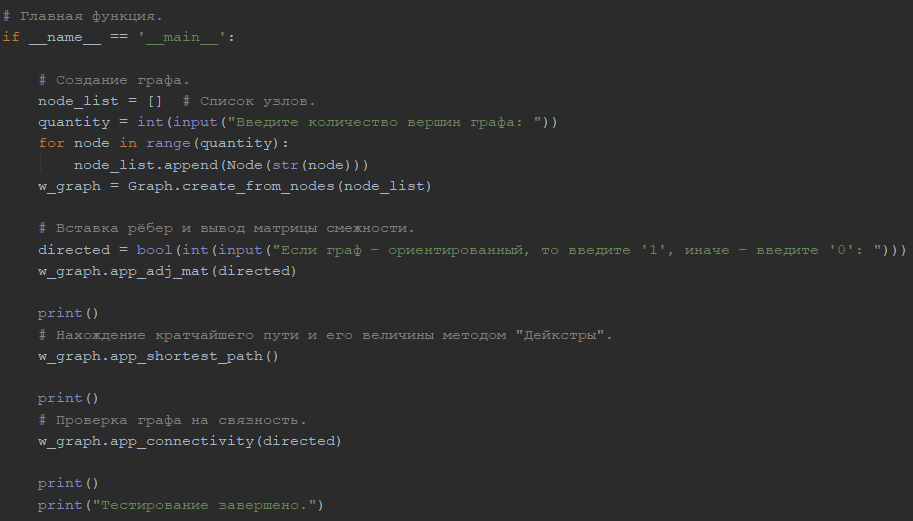


Рисунок 5 – Основная функция.

## Тестирование программы.

**Тестирование программы на слабо связном ориентированном графе.**

Для тестирования был выбран следующий граф (который является ориентированным):

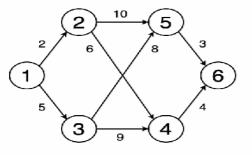


Рисунок 6 – Ориентированный граф для тестирования программы.

Было запущено тестирование: через консоль введено количество вершин графа и указан тип графа, после чего построчно производились операции вставки ребра в граф.

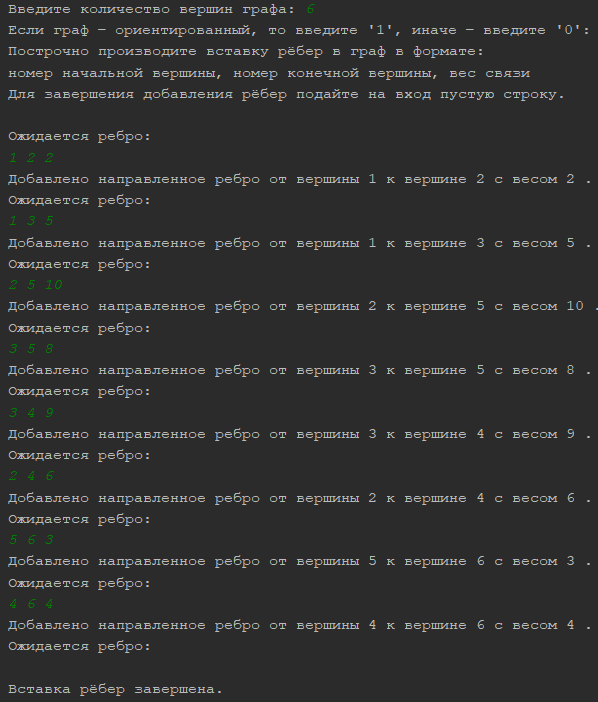


Рисунок 7 – Ввод графа посредством операций вставки ребра в граф.

В результате работы программы был выведен граф в виде матрицы смежности, в которой каждая строка является набором длин исходящих путей в каждую вершину графа для соответствующей вершины графа. Если с какой-либо вершиной отсутствует непосредственная связь, то в ячейке хранится None. Также следует отметить, что путь от любой вершины до самой себя равен 0.

Результат работы программы:

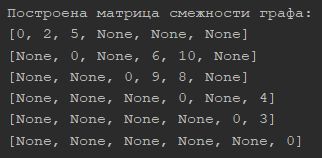


Рисунок 8 – Вывод матрицы смежности графа.

Далее программа считывает две вершины графа для нахождения величины кратчайшего пути и вывода кратчайшего пути от одной заданной вершины к другой заданной вершине методом «Дейкстры».

Для тестирования были взяты вершины «1» и «6». По графическому изображению графа видно, что кратчайшим путём от вершины «1» к вершине «6» является путь: 1 – 2 – 4 – 6. Величина кратчайшего пути: 12.

Результат работы программы это подтверждает:

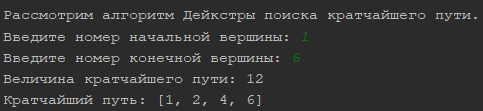


Рисунок 9 – Вывод кратчайшего пути и его величины методом «Дейкстры».

После этого программа выполнила проверку графа на связность.

В случае связности ориентированный граф может быть сильно связным или слабо связным.

Необходимым и достаточным условием сильной связности ориентированного графа является возможность прохода до любой вершины графа для каждой вершины графа.

По графическому изображению графа видно, что он не является сильно связным. Так, для вершины «6» не выполняется условие сильной связности, так как из неё невозможно осуществить проход в другие вершины графа.

С другой стороны, если представить рёбра данного графа неориентированными, то граф будет связным, поскольку для каждой вершины появится возможность прохода к любой другой вершине графа.

Таким образом, данный ориентированный граф является слабо связным.

Результат выполнения программы это подтверждает:



Рисунок 10 – Проверка графа на связность.

**Тестирование программы на неориентированном графе.**

Проверим теперь работу программы с неориентированным графом. Вообще говоря, работа алгоритмов по обработке неориентированного графа не отличается от работы алгоритмов по обработке ориентированного графа – для указания двусторонней связи можно дважды выполнить операцию вставки ребра в граф: сначала от первой вершины ко второй, затем от второй к первой.

Однако, благодаря методу приложения для ввода графа, стало возможным не только упростить ввод неориентированного графа, но и инкапсулировать его в удобный интерфейс.

Рассмотрим в качестве примера следующий неориентированный граф:

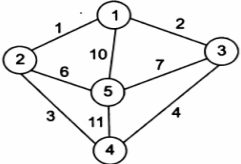


Рисунок 11 – Неориентированный граф для тестирования программы.

Было запущено тестирование: через консоль введено количество вершин графа и указан тип графа, после чего построчно производились операции вставки ребра в граф, при этом для создания ребра было достаточно указывать две вершины в любом порядке.

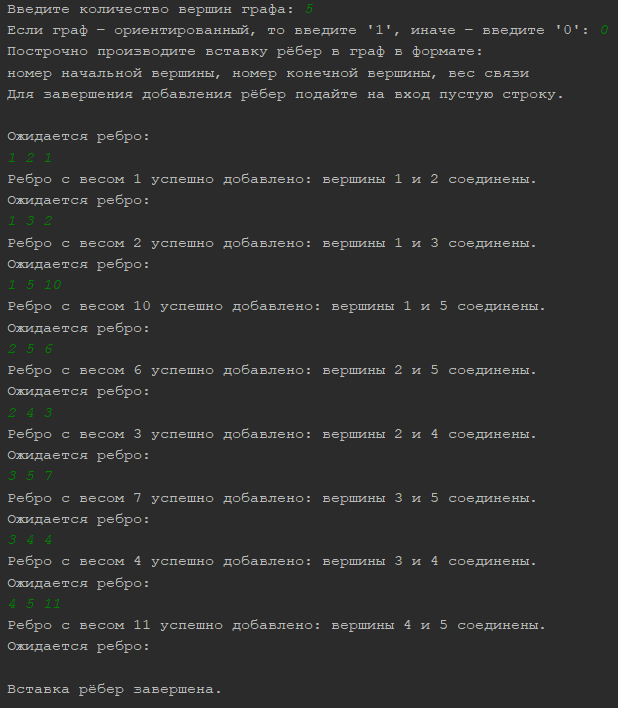


Рисунок 12 – Ввод графа посредством операций вставки ребра в граф.

В результате работы программы была построена матрица смежности графа:

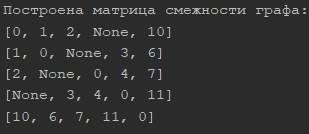


Рисунок 13 – Вывод матрицы смежности графа.

Далее программа считывает две вершины графа для нахождения величины кратчайшего пути и вывода кратчайшего пути от одной заданной вершины к другой заданной вершине методом «Дейкстры».

Для тестирования были взяты вершины «1» и «4». По графическому изображению графа видно, что кратчайшим путём от вершины «1» к вершине «4» является путь: 1 – 2 – 4. Величина кратчайшего пути: 4.

Результат работы программы это подтверждает:

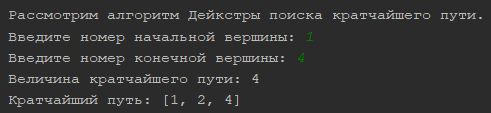


Рисунок 14 – Вывод кратчайшего пути и его величины методом «Дейкстры».

После этого программа выполнила проверку графа на связность.

По графическому изображению графа видно, что он является связным, поскольку для любой вершины данного графа существует хотя бы один проход в каждую вершину графа.

Результат выполнения программы подтверждает связность графа:

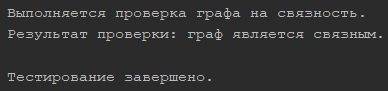


Рисунок 15 – Проверка графа на сильную связность.

**Тестирование программы на сильно связном ориентированном графе.**

Теперь проведём тестирование для сильно связного ориентированного графа:

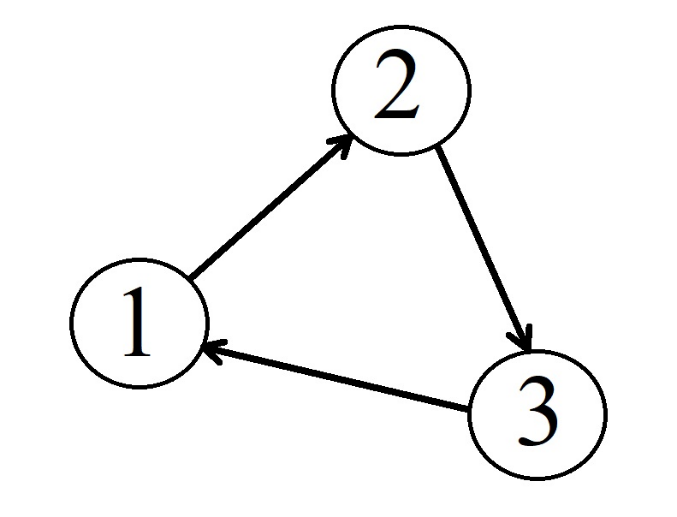


Рисунок 16 – Пример сильно связного ориентированного графа.

Осуществим ввод данных:

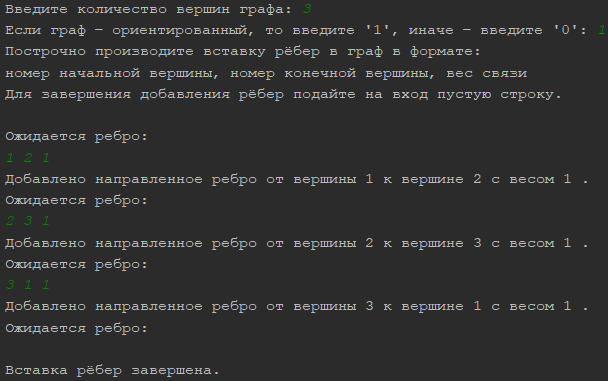


Рисунок 17 – Ввод графа посредством операций вставки ребра в граф.

Получаем матрицу смежности графа:

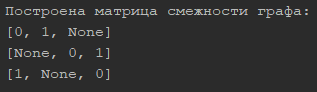


Рисунок 18 – Вывод матрицы смежности графа.

Найдём кратчайший путь от вершины «2» к вершине «1» (веса рёбер примем равными единице): 2 – 3 – 1. Величина кратчайшего пути: 2.

Результат работы программы это подтверждает:

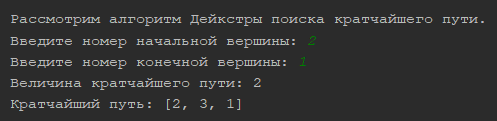


Рисунок 19 – Вывод кратчайшего пути и его величины методом «Дейкстры».

Наконец, результат выполнения программы подтверждает сильную связность графа:



Рисунок 20 – Проверка графа на связность.

**Тестирование программы на несвязном графе.**

Пусть дан несвязный граф:

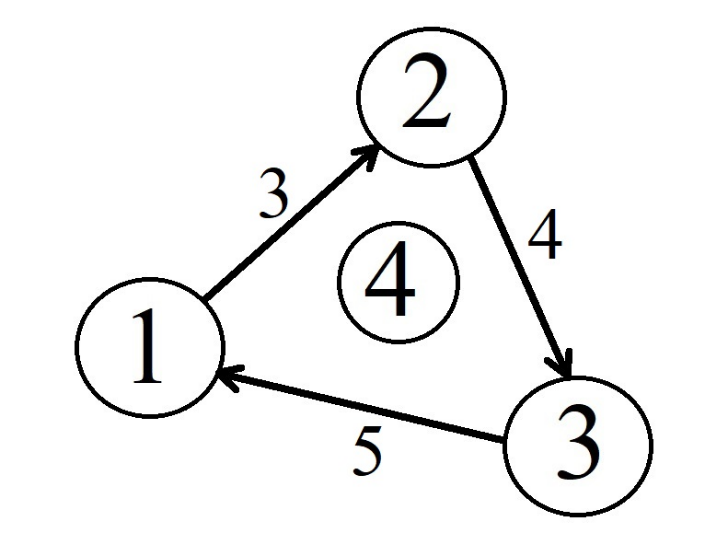


Рисунок 21 – Пример несвязного графа.

Произведём ввод данных:

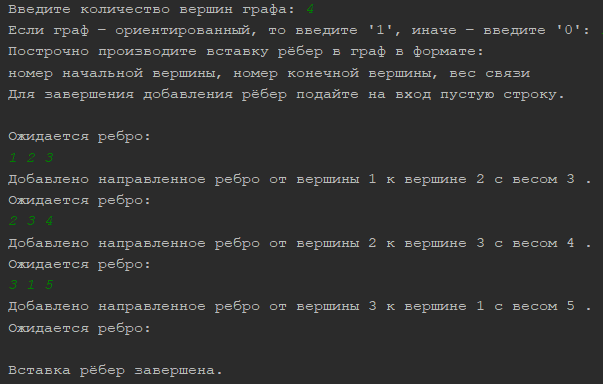


Рисунок 22 – Ввод графа посредством операций вставки ребра в граф.

Произведён вывод матрицы смежности графа:

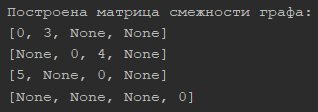


Рисунок 23 – Вывод матрицы смежности графа.

Для вершин «1», «2», «3» не существует путей к вершине «4», поэтому при попытки найти кратчайший путь, например, от вершины «1» к вершине «4» программа выводит исключение:

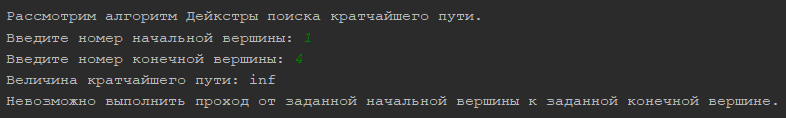


Рисунок 24 – Вывод кратчайшего пути и его величины методом «Дейкстры».

Результат выполнения программы подтверждает несвязность графа:

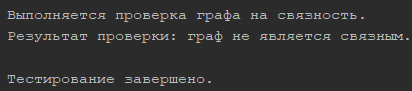


Рисунок 25 – Проверка графа на связность.

Таким образом, тестирование прошло успешно: все поставленные задачи были выполнены.

# Вывод

В ходе работы были приобретены умения и навыки разработки и реализации операций над структурой данных «граф».

# Список информационных источников

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных» /

Л. А. Скворцова, МИРЭА – Российский технологический университет, 2021.