

Министерство науки и высшего образования российской федерации
Пензенский государственный университет
Кафедра «Вычислительной техники»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проектированию
по курсу: «Логика и основы алгоритмизации в инженерных задачах»
на тему: «Реализация алгоритма Форда-Беллмана»

Выполнил:

студент группы 22ВВП1

Демин М.С.

Принял:

Акифьев И.В.

28.12.23г.
Отлично

Пенза 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

« » 20

ЗАДАНИЕ

на курсовое проектирование по курсу

«Логика и основы алгоритмизации в инженерных задачах»
Студенту Дашин Максим Сергеевич Группа 22 ВВВ2
Тема проекта Реализация алгоритма Форда - Беллмана

Исходные данные (технические требования) на проектирование

Разработка алгоритмов и программного обеспечения в соответствии
с данными заданием курсового проекта

Разработочная записка должна содержать:

1. Постановку задачи

2. Теоретическую часть задачи

3. Описание алгоритма поставленной задачи

4. Пример ручного расчета задачи и вычислений (на
небольшом участке работы алгоритма)

5. Описание самой программы

6. Тесты

7. Список литературы

8. Итоги программы

9. Результаты работы программы

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

Схема алгоритма в формате блок-схемы.

3. Экспериментальная часть

Тестирование программы

Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

- 1 Исследование теоретической части курсового
- 2 Разработка алгоритмов программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания " 7 " сентября 2023

Дата защиты проекта " " "

Руководитель Акифьев И.В.

Задание получил " 28 " сентября 2023 г.

Студент Демин Максим Сергеевич

Министерство науки и высшего образования российской федерации
Пензенский государственный университет
Кафедра «Вычислительной техники»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проектированию
по курсу: «Логика и основы алгоритмизации в инженерных задачах»
на тему: «Реализация алгоритма Форда-Беллмана»

Выполнил:

студент группы 22ВВП1

Демин М.С.

Принял:

Акифьев И.В.

Пенза 2023

Содержание

Реферат	6
Введение	7
1. Постановка задачи	9
2. Теоретическая часть программы	10
3. Описание алгоритма программы	12
4. Описание программы.....	15
5. Тестирование.....	23
6. Ручной расчет задачи	29
Список литературы	33
Приложение А	34
Код программы.....	34
Приложение В	43
Результаты работы программы	43

Реферат

Отчет 41 стр., 28 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, АЛГОРИТМ ФОРДА БЕЛЛМАНА, ПОИСК КРАТЧАЙШИХ РАССТОЯНИЙ

Цель исследования – разработка программы, которая будет находить кратчайшие пути от одной вершины графа до всех остальных, используя алгоритм Беллмана-Форда.

В работе рассмотрен алгоритм Беллмана-Форда, предназначенный для поиска кратчайшего пути в графе. Установлено, что с помощью данного алгоритма можно делать восстановление пути, а также проверять наличие циклов отрицательного веса в графе.

Введение

Теория графов — раздел дискретной математики, изучающий графы, одна из ветвей топологии. В самом общем смысле граф — это множество точек (вершин, узлов), которые соединяются множеством линий (рёбер, дуг).

Теория графов (то есть систем линий, соединяющих заданные точки) включена в учебные программы для начинающих математиков, поскольку:

- 1 как и геометрия, обладает наглядностью;
- 2 как и теория чисел, проста в объяснении и имеет сложные нерешённые задачи;
- 3 не имеет громоздкого математического аппарата («комбинаторные методы нахождения нужного упорядочения объектов существенно отличаются от классических методов анализа поведения систем с помощью уравнений»);
- 4 имеет выраженный прикладной характер.

На протяжении более сотни лет развитие теории графов определялось в основном проблемой четырёх красок. Решение этой задачи в 1976 году оказалось поворотным моментом истории теории графов, после которого произошло её развитие как основы современной прикладной математики.

Универсальность графов незаменима при проектировании и анализе коммуникационных сетей.

Теория графов, как математическое орудие, приложима как к наукам о поведении (теории информации, кибернетике, теории игр, теории систем, транспортным сетям), так и к чисто абстрактным дисциплинам (теории множеств, теории матриц, теории групп и так далее).

Несмотря на разнообразные, усложнённые, малопонятные и специализированные термины многие модельные (схемные, структурные) и конфигурационные проблемы переформулируются на языке теории графов, что позволяет значительно проще выявить их концептуальные трудности.

В разных областях знаний понятие «граф» может встречаться под следующими названиями:

- 1 структура (гражданское строительство);
- 2 сеть (электротехника);
- 3 социограмма (социология и экономика);
- 4 молекулярная структура (химия);

- 5 навигационная карта (картография);
 - 6 распределительная сеть (энергетика)
- и так далее.

Часто требуется производить различные операции над графами, например нахождение кратчайшего расстояния между вершинами. Для этого придуманы различные алгоритмы, один из которых Алгоритм Дейкстры.

В качестве среды разработки мною была выбрана среда PyCharm, языки программирования – Python.

Целью данной курсовой работы является разработка программы на языках программирования, которые являются широко используемыми. Именно с их помощью в данном курсовом проекте реализуется алгоритм Беллмана-Форда, осуществляющий поиск кратчайших путей в орграфе.

1. Постановка задачи

Необходимо разработать программу, которая будет искать длины кратчайших путей от вершины st (стартовой) до всех остальных вершин, используя алгоритм Беллмана-Форда.

Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности орграфа, вид орграфа и все компоненты связности орграфа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

2. Теоретическая часть программы

Граф G (рисунок 1) задается множеством вершин $1, 2, \dots, 5(N)$. и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества A ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.

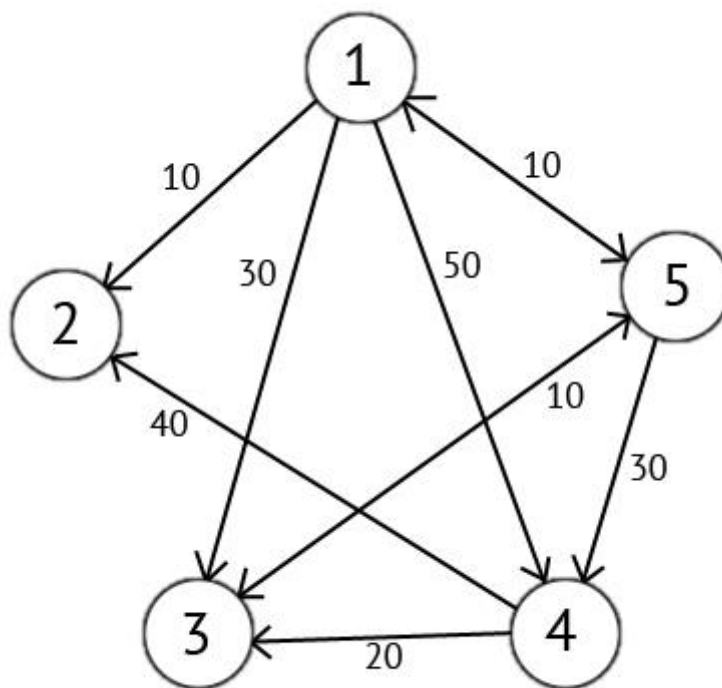


Рисунок 1 – Пример орграфа

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается весом ребра, иначе нулем.

Алгоритм Беллмана-Форда — алгоритм поиска кратчайшего пути во взвешенном графе. За время $O(|st| \cdot |E|)$ алгоритм находит кратчайшие пути от одной вершины графа до всех остальных. В отличие от алгоритма Дейкстры, алгоритм Беллмана — Форда допускает рёбра с отрицательным весом. Предложен независимо Ричардом Беллманом и Лестером Фордом.

Дан ориентированный или неориентированный граф G со взвешенными рёбрами. Длиной пути назовём сумму весов рёбер, входящих в этот путь. Требуется найти кратчайшие пути от выделенной вершины st до всех вершин графа.

Заметим, что кратчайших путей может не существовать. Так, в графе, содержащем цикл с **отрицательным суммарным весом**, существует сколько угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути). Цикл, сумма весов рёбер которого отрицательна, называется **отрицательным циклом**.

Алгоритм Беллмана-Форда — алгоритм поиска кратчайшего пути во взвешенном графе. Также как и Дейкстра алгоритм Форда-Беллмана ищет расстояние от одной вершины до всех остальных, но работает и с отрицательными ребрами. Если алгоритм Дейкстры жадно выбирает узел с минимальным весом, который еще не был обработан, и выполняет этот процесс релаксации на всех его исходящих ребрах; алгоритм Беллмана – Форда, напротив, просто ослабляет все ребра и делает это $|V| - 1$ раз, где $|V|$ количество вершин в графе. В каждом из этих повторений растет число вершин с правильно рассчитанными расстояниями, из чего следует, что в конечном итоге все вершины будут иметь свои правильные расстояния. Этот метод позволяет применять алгоритм Беллмана-Форда к более широкому классу входных данных, чем Дейкстра.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (relax, ослабление) вдоль каждого ребра (v,u) веса w . Релаксация вдоль ребра — это попытка улучшить значение $nodes[u]$ значением $nodes[v] + w$. Фактически это значит, что мы пытаемся улучшить ответ для вершины u , пользуясь ребром (v,u) и текущим ответом для вершины v .

Утверждается, что достаточно $N-1$ фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (при условии, что граф не содержит циклов отрицательного веса). Для недостижимых вершин расстояние $nodes[]$ останется равным бесконечности (заведомо большим числом).

3. Описание алгоритма программы

Для алгоритма Форда-Беллмана, в отличие от многих других графовых алгоритмов, более удобно представлять граф в виде одного списка всех рёбер. В приведённой реализации заводится массив структур данных **edge** для всех рёбер. Входными данными для алгоритма являются числа **N** (кол-во вершин), **e** (кол-во рёбер), список **edge** рёбер, и номер стартовой вершины **st**. Все номера вершин нумеруются с 0 по **N-1**. Константа **inf** обозначает число "бесконечность" — её надо подобрать таким образом, чтобы она заведомо превосходила все возможные длины путей. Массив расстояний **nodes[0...N-1]**, который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: **nodes[st] = 0**, а все остальные элементы **nodes[]** равны бесконечности (**inf**).

Этот алгоритм можно несколько ускорить: зачастую ответ находится уже за несколько фаз, а оставшиеся фазы никакой полезной работы не происходит, лишь впустую просматриваются все рёбра. Поэтому будем хранить флаг (**change**) того, изменилось что-то на текущей фазе или нет, и если на какой-то фазе ничего не произошло, то алгоритм можно останавливать.

Алгоритм Форда-Беллмана модифицирован, чтобы он не только находил длины кратчайших путей, но и позволял восстанавливать сами кратчайшие пути. Для этого заведён ещё один вектор **parent[0...N-1]**, в котором для каждой вершины хранится её "предков", т.е. предпоследняя вершина в кратчайшем пути, ведущая в неё. В самом деле, кратчайший путь до какой-то вершины **v** является кратчайшим путём до какой-то вершины **parent[v]**, к которому приписали в конец вершину **v**. Заметим, что алгоритм Форда-Беллмана работает по такой же логике: он, предполагая, что кратчайшее расстояние до одной вершины уже посчитано, пытается улучшить кратчайшее расстояние до другой вершины. Следовательно, в момент улучшения нам надо просто запоминать в **parent[]**, из какой вершины это улучшение произошло.

В цикле восстановления путей мы сначала проходимся по предкам, начиная с вершины **i**, и сохраняем весь пройденный путь в векторе **path**. В этом векторе получается кратчайший путь от **st** до **i**, но в обратном порядке, поэтому мы вызываем **reverse** от него и затем выводим.

При восстановлении пути может возникнуть момент, когда после первой итерации мы попадаем в вершину, в которой уже были до этого, это свидетельствует о наличии отрицательного цикла. В этом случае останавливаем восстановление пути и выводим сообщение о некорректности результата.

Поскольку при наличии отрицательного цикла за **N** итераций алгоритма расстояния могли уйти далеко в минус, в коде приняты дополнительные меры против такого целочисленного переполнения:

$\text{nodes}[\text{edge}[i].v] = \max(-\text{inf}, \text{nodes}[\text{edge}[i].u] + \text{edge}[i].w)$

Ниже представлен псевдокод функции **bellman_ford()**:

1. Открываем/создаём файл `bellman_ford.txt` для сохранения результатов.
2. В случае невозможности открытия/создания файла останавливаем программу.
3. Заполняем массив $\text{nodes}[0 \dots N-1] = \text{inf}$.
4. Инициализируем вектор $\text{parent}[0 \dots N-1] = -1$.
5. $\text{nodes}[\text{st}(\text{стартовая вершина})] = 0$.
6. Для $i = 0$; пока $i < N-1$; делать $i = i + 1$.
 7. Флаг `change` = 0.
 8. Для $j = 0$; пока $j < e$ (кол-во рёбер); делать $j = j + 1$.
 9. Если $\text{nodes}[\text{edge}[j].v] + \text{edge}[j].w < \text{nodes}[\text{edge}[j].u]$.
 10. $\text{nodes}[\text{edge}[j].u] = \text{наибольшее из } (-\text{inf} \text{ либо } \text{nodes}[\text{edge}[j].v] + \text{edge}[j].w)$.
 11. $\text{parent}[\text{edge}[j].u] = \text{edge}[j].v$.
 12. `change` = 1.
 13. Если `change` == 0
 14. Выходим из цикла.
15. Для $i = 0$; пока $i < N$; $i = i + 1$.
 16. Если $\text{nodes}[i] == \text{inf}$
 17. Выводим в консоль и файл, что «Путь отсутствует».
 18. Иначе
 19. Выводим в консоль и файл $\text{nodes}[i]$
 20. Если $0 \leq \text{nodes}[i] < 10$
 21. Выводим в консоль и файл Пробел.
 22. Объявляем вектор `path`.
 23. Для $\text{cur} = i$; пока $\text{cur} \neq -1$; делать $\text{cur} = \text{parent}[\text{cur}]$.
 24. Для $i2 = 0$; пока $i2 < \text{размер path}$; делать $i2 = i2 + 1$.
 25. Если $\text{cur} == \text{path}[i2]$ и $\text{размер path} > 1$.
 26. Увеличиваем на 1 размер `path` и заносим в конец `cur`.
 27. Выводим предупреждение об отрицательном цикле.
 28. Переходим на метку `negative`.

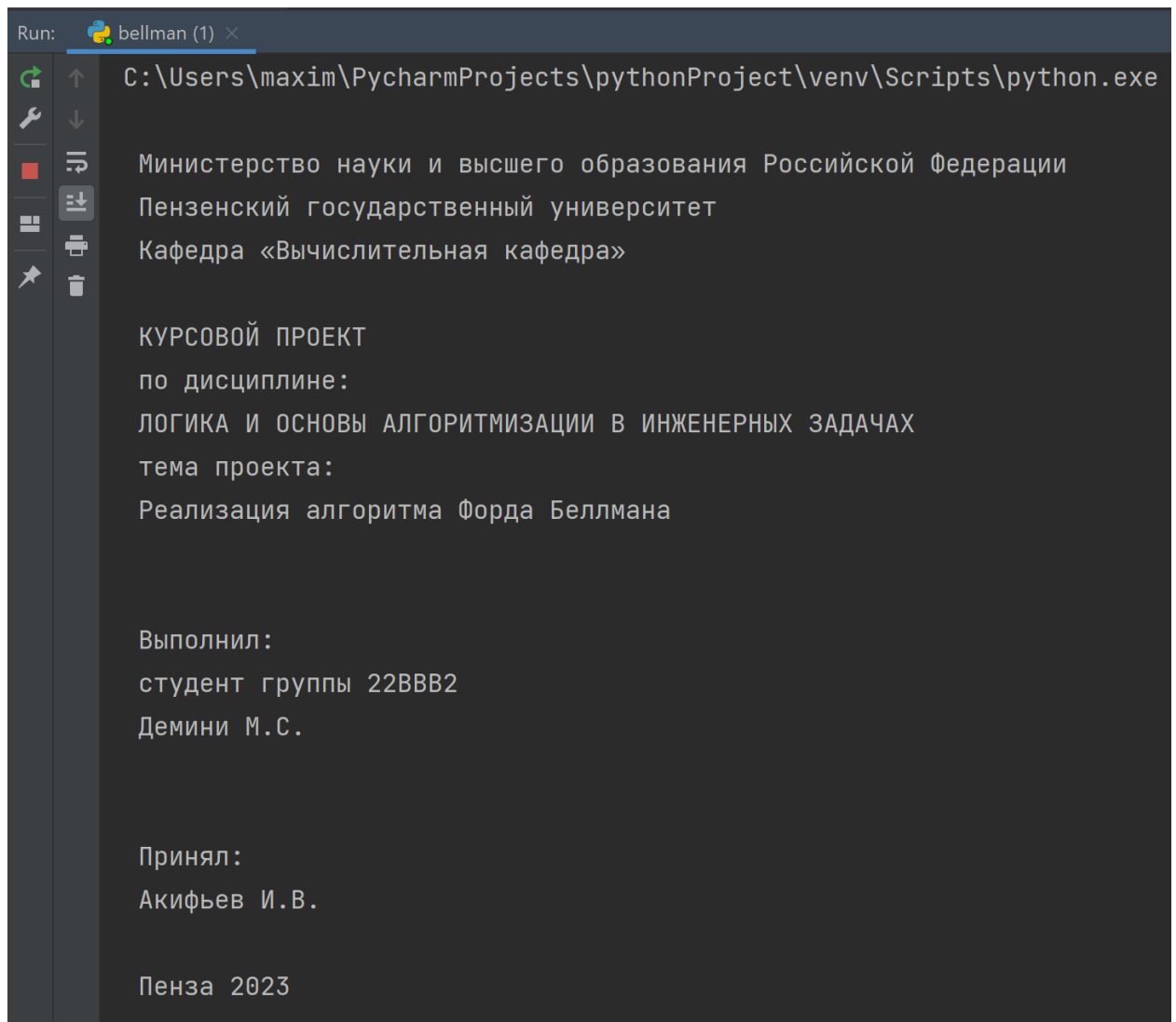
29. Увеличиваем на 1 размер path и заносим в конец cur.
30. Метка negative:
31. Меняем порядок элементов path[] на обратный.
32. Цикл вывода path[].
33. Закрываем файл bellman_ford.txt

4. Описание программы

Для написания программы использован язык программирования Python.

Данная программа является многомодульной, поскольку состоит из нескольких функций `main()`, `generation()`, `bellman_ford()`, `titulniyList()`, `wanna_more()`

Работа программы начинается с вывода титульного листа на экран:



```
Run: bellman (1) x
C:\Users\maxim\PycharmProjects\pythonProject\venv\Scripts\python.exe

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная кафедра»

КУРСОВОЙ ПРОЕКТ
по дисциплине:
ЛОГИКА И ОСНОВЫ АЛГОРИТМИЗАЦИИ В ИНЖЕНЕРНЫХ ЗАДАЧАХ
тема проекта:
Реализация алгоритма Форда Беллмана

Выполнил:
студент группы 22BVB2
Демини М.С.

Принял:
Акифьев И.В.

Пенза 2023
```

2. Титульный лист работы

Далее на экран выводится текстовое меню. Пользователю даётся на выбор 3 действия. Для выбора действия пользователю нужно нажать соответствующую клавишу на клавиатуре.

1. Ввести матрицу вручную.
2. Прочитать матрицу из файла.
3. Сгенерировать случайную матрицу.

```
Выберете способ ввода графа:  
1 - вручную  
2 - прочитать из файла input.txt  
3 - случайная генерация  
--> |
```

3. Меню программы.

Пользователь может сгенерировать случайный граф или ввести граф вручную или прочитать его из файла.

При создании случайного графа у пользователя запрашивается его размер. Размер можно ввести, используя клавиатуру:

```
3 - случайная генерация  
--> 3  
  
Укажите размер матрицы N*N:  
--> |
```

4. Ввод размера графа

После ввода размера графа спрашивается ориентированный делать или нет:


```

    Укажите размер матрицы N*N:
--> 4
    Ориентированный граф?

    Да - 1
    Нет - 2
-->

```

5. Ориентированный граф или нет

Далее выводятся матрица, представляющая граф:

```

    Нет - 2
--> 2

```

	[1]	[2]	[3]	[4]
[1]	0	3	0	2
[2]	3	0	5	0
[3]	0	5	0	5
[4]	2	0	5	0

6. Вывод матрицы для графа

Далее пользователь указывает, какая вершина графа будет стартовой:

	[1]	[2]	[3]	[4]
[1]	0	3	0	2
[2]	3	0	5	0
[3]	0	5	0	5
[4]	2	0	5	0

Стартовая вершина >

7. Стартовая вершина

После указания стартовой вершины, выводятся кратчайшие пути:

```
Стартовая вершина > 3

Список кратчайших путей:

3 -> 1 = 7    ( 3 -> 4 -> 1 )
3 -> 2 = 5    ( 3 -> 2 )
3 -> 3 = 0    ( 3 )
3 -> 4 = 5    ( 3 -> 4 )
```

8. Результат расчётов

Далее у пользователя спрашивается, продолжить или нет:

```
Продолжаем Форда?
1 - да
2 - нет
->
```

9. Продолжаем?

При построении графа в ручную у пользователя спрашивается кол-во вершин:

```
Выберете способ ввода графа:
1 - вручную
2 - прочитать из файла input.txt
3 - случайная генерация
--> 1

Количество вершин > |
```

10. Кол-во вершин при ручном построении

После ввода кол-ва вершин надо указать вес каждого ребра между вершинами:

```

Количество вершин > 3
Вес 1 -> 1 = 2
Вес 1 -> 2 = 3
Вес 1 -> 3 = 4
Вес 2 -> 1 = 19
Вес 2 -> 2 = 1
Вес 2 -> 3 = 0
Вес 3 -> 1 = 2
Вес 3 -> 2 = 3
Вес 3 -> 3 = 6

      [1]  [2]  [3]
[1]   2    3    4
[2]  19    1    0
[3]   2    3    6

Стартовая вершина > |

```

11. Ручной ввод весов ребер

После указания стартовой вершины выводятся кратчайшие пути и спрашивается о продолжении программы:

```

Стартовая вершина > 2

Список кратчайших путей:

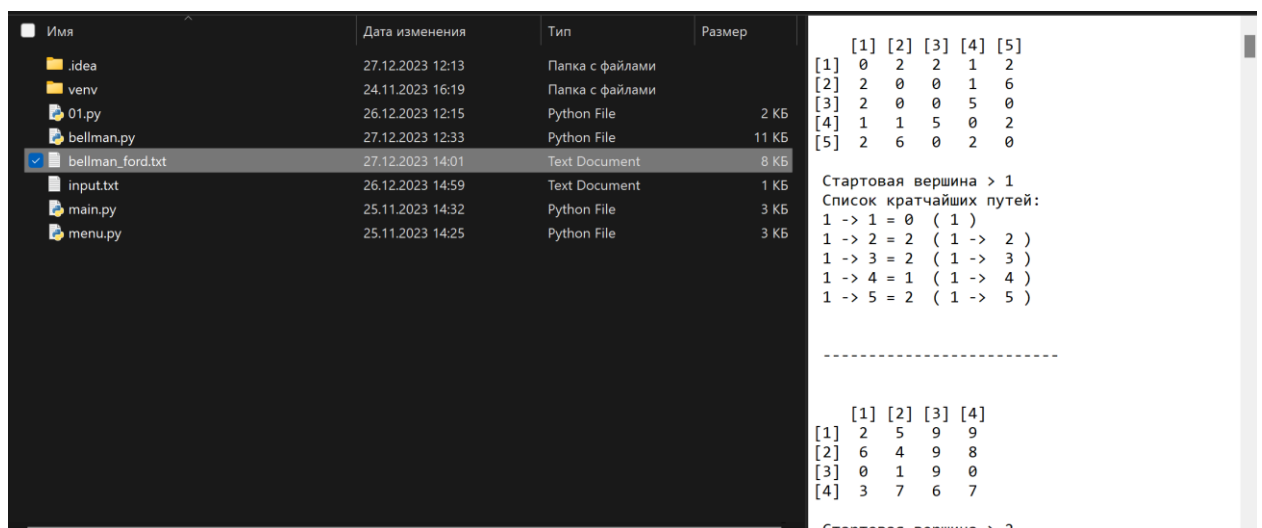
2 -> 1 = 19 ( 2 -> 1 )
2 -> 2 = 0 ( 2 )
2 -> 3 = 23 ( 2 -> 1 -> 3 )

Продолжаем Форда?
1 - да
2 - нет
->

```

12. Результат поиска путей в графе, заданном вручную

После успешного завершения программы происходит сохранение результата в текстовый документ. Данные в нем обновляются каждый раз после завершения программы.



13. Сохранение графа в файл

Также пользователь может выбрать ввод графа из файла:

```

Выберете способ ввода графа:
1 - вручную
2 - прочитать из файла input.txt
3 - случайная генерация
--> 2

```

```

Файл открыт
      [1]  [2]  [3]  [4]
[1]    5    6    5    6
[2]    5    5    1    5
[3]    4    9    6    9
[4]    0    6    0    2

```

```

Стартовая вершина >

```

14. Граф, заданный в файле

После указания стартовой вершины, выводится результат работы

```

Стартовая вершина > 3

Список кратчайших путей:

3 -> 1 = 2      ( 3 -> 1 )
3 -> 2 = 3      ( 3 -> 2 )
3 -> 3 = 0      ( 3 )
3 -> 4 = 8      ( 3 -> 1 -> 4 )

Продолжаем Форда?
1 - да
2 - нет
->

```

15. Результат работы программы с графом, введенным из файла

Граф хранится в отдельном файле, который пользователь может сам редактировать:

Имя	Дата изменения	Тип	Размер	
.idea	27.12.2023 12:13	Папка с файлами		4
venv	24.11.2023 16:19	Папка с файлами		5 6 5 6
01.py	26.12.2023 12:15	Python File	2 КБ	5 5 1 5
bellman.py	27.12.2023 12:33	Python File	11 КБ	4 9 6 9
bellman_ford.txt	27.12.2023 14:05	Text Document	9 КБ	0 6 0 2
input.txt	26.12.2023 14:59	Text Document	1 КБ	
main.py	25.11.2023 14:32	Python File	3 КБ	
menu.py	25.11.2023 14:25	Python File	3 КБ	

16. Граф заданный в файле

5. Тестирование

Тестирование проводилось и в процессе разработки, и после написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, выводом данных в файл, алгоритмом программы, взаимодействием функций.

Ниже представлены результаты работы программы.

```
bellman (1) x
Выберете способ ввода графа:
1 - вручную
2 - прочитать из файла input.txt
3 - случайная генерация
--> 1

Количество вершин > 3
Вес 1 -> 1 = 1
Вес 1 -> 2 = 0
Вес 1 -> 3 = -2
Вес 2 -> 1 = 5
Вес 2 -> 2 = 0
Вес 2 -> 3 = -2
Вес 3 -> 1 = 7
Вес 3 -> 2 = 1
Вес 3 -> 3 = -2

      [1]  [2]  [3]
[1]   1    0   -2
[2]   5    0   -2
[3]   7    1   -2

Стартовая вершина > |
```

17. Результат ручного ввода

```
Выберете способ ввода графа:
1 - вручную
2 - прочитать из файла input.txt
3 - случайная генерация
--> 3

Укажите размер матрицы N*N:
--> 4

Ориентированный граф?

Да - 1
Нет - 2
--> 2
```

	[1]	[2]	[3]	[4]
[1]	0	3	4	5
[2]	3	0	2	3
[3]	4	2	0	0
[4]	5	3	0	0

18. Результат случайно сгенерированного неориентированного графа


```
Выберете способ ввода графа:  
1 - вручную  
2 - прочитать из файла input.txt  
3 - случайная генерация  
--> 3
```

```
Укажите размер матрицы N*N:  
--> 4  
Ориентированный граф?
```

```
Да - 1  
Нет - 2  
--> 1
```

	[1]	[2]	[3]	[4]
[1]	0	0	0	4
[2]	0	0	0	0
[3]	0	0	0	1
[4]	4	0	1	0

19. Результат случайно сгенерированного ориентированного графа

```
Выберете способ ввода графа:
1 - вручную
2 - прочитать из файла input.txt
3 - случайная генерация
--> 2

Файл открыт

      [1]  [2]  [3]  [4]
[1]   5    6    5    6
[2]   5    5    1    5
[3]   4    9    6    9
[4]   0    6    0    2

Стартовая вершина > 3

Список кратчайших путей:

3 -> 1 = 4      ( 3 -> 1 )
3 -> 2 = 9      ( 3 -> 2 )
3 -> 3 = 0      ( 3 )
3 -> 4 = 9      ( 3 -> 4 )
```

20. Результат графа, прочитанного из файла

Также в программе предусмотрена проверка на некорректный ввод данных. При попытке ввода пользователем некорректного значения (такого, с каким программа не должна работать) на экране отображается сообщение об ошибке и какой ввод ожидался запрашивается повторный ввод.

Количество вершин > **апа**
 Количество вершин должно быть только целым числом
 Введите число вершин еще раз...

Количество вершин > **2**
 Вес 1 -> 1 = **наб**
 Вес должен быть только целым положительным или отрицательным числом
 Введите вес еще раз...

3
 Вес 1 -> 2 = **0.3**
 Вес должен быть только целым положительным или отрицательным числом
 Введите вес еще раз...

фывфывфыв
 Вес должен быть только целым положительным или отрицательным числом
 Введите вес еще раз...

Стартовая вершина > **ываыва**
 Введите целое число в диапазоне от 1 до 2...

3
 Введите целое число в диапазоне от 1 до 2...

2

Продолжаем Форда?

1 - да
 2 - нет
 -> **ва**

Введите целое число: '1' или '2'...

21. Проверка ввода некорректных значений.

Таблица 1 - описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
	Появление титульного листа, меню на экране и	

Запуск программы	сообщения о выборе нужной операции	Верно
Выбор генерации графа	Запрос ввода количества вершин	Верно
Ручной ввод графа	Запрос ввода веса ребер между вершинами	Верно
Сохранение результата	Результат записан в файл	Верно
Результат работы программы	Верный вывод кратчайших расстояний	Верно

В результате тестирования было выявлено, что программа работает успешно.

6. Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере взвешенного ориентированного графа с 5 вершинами (рисунок 2).

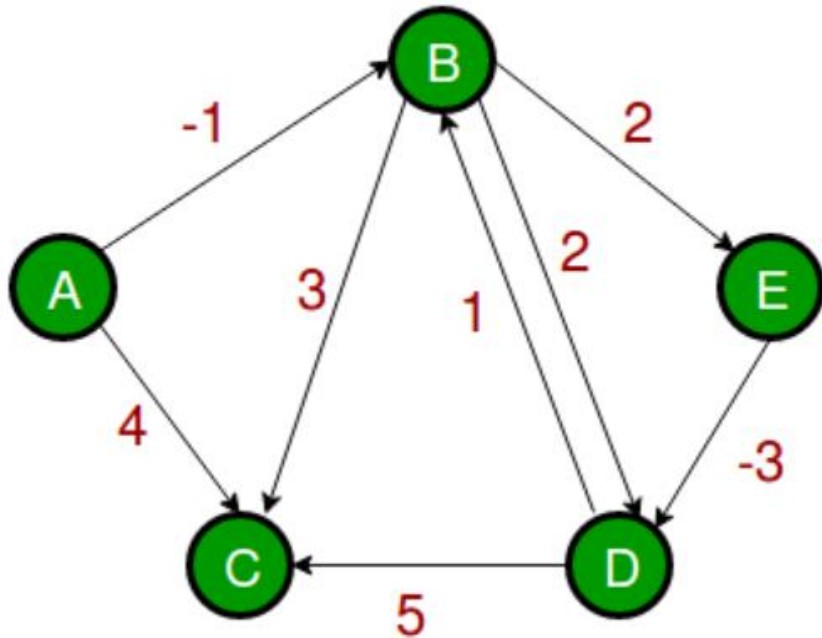


Рисунок 22 – Орграф

Пусть ребра обрабатываются в следующем порядке: (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D). Мы получаем следующие расстояния, когда проход по ребрам был совершен первый раз. Первая строка показывает начальные расстояния, вторая строка показывает расстояния, когда ребра (B, E), (D, B), (B, D) и (A, B) обрабатываются. Третья строка показывает расстояние при обработке (A, C). Четвертая строка показывает, что происходит, когда обрабатываются (D, C), (B, C) и (E, D).

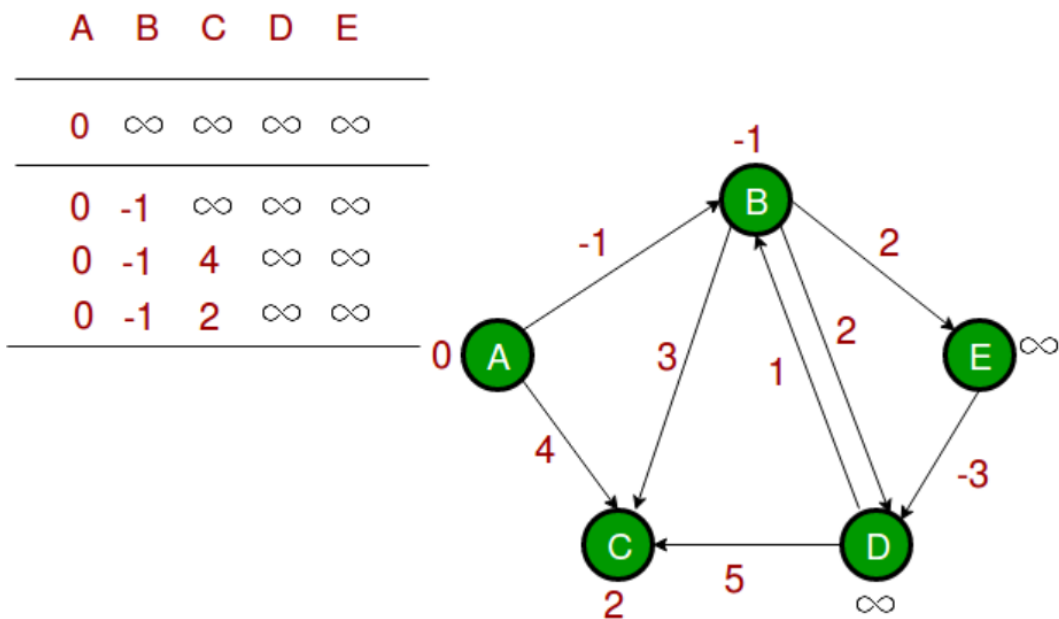


Рисунок 23 – Результат первой итерации

Первая итерация гарантирует, что все самые короткие пути будут не длиннее пути в 1 ребро. Мы получаем следующие расстояния, когда будет совершен второй проход по всем ребрам (в последней строке показаны конечные значения).

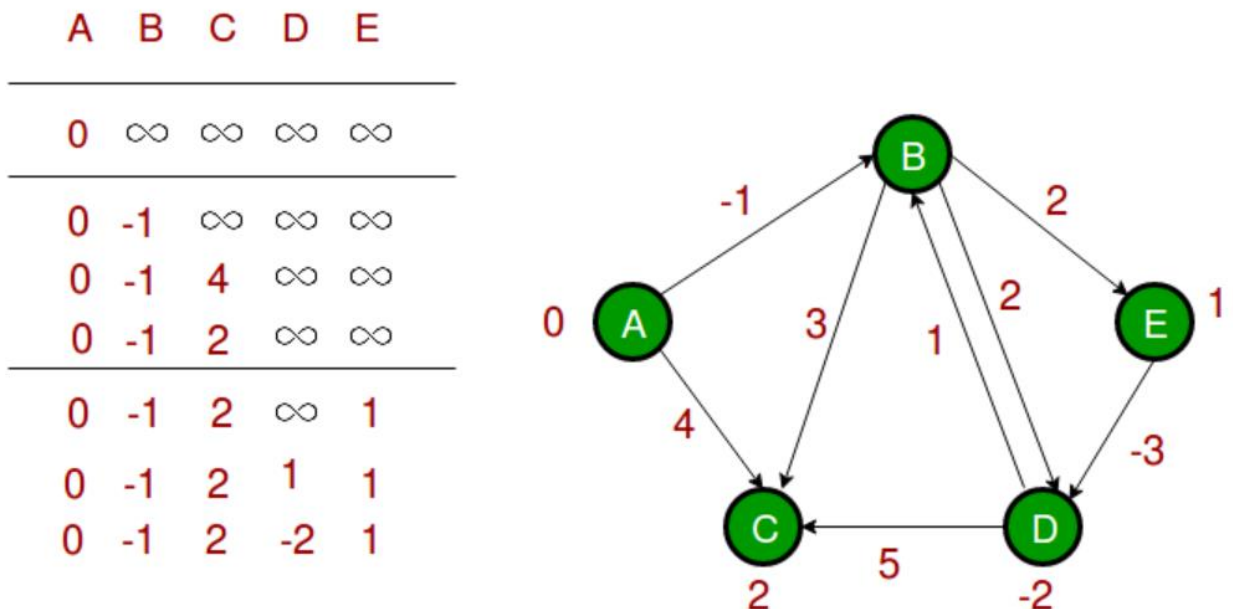


Рисунок 24 – Результат второй итерации

Используя алгоритм Беллмана-Форда, мы можем определить, есть ли отрицательный цикл в графе.

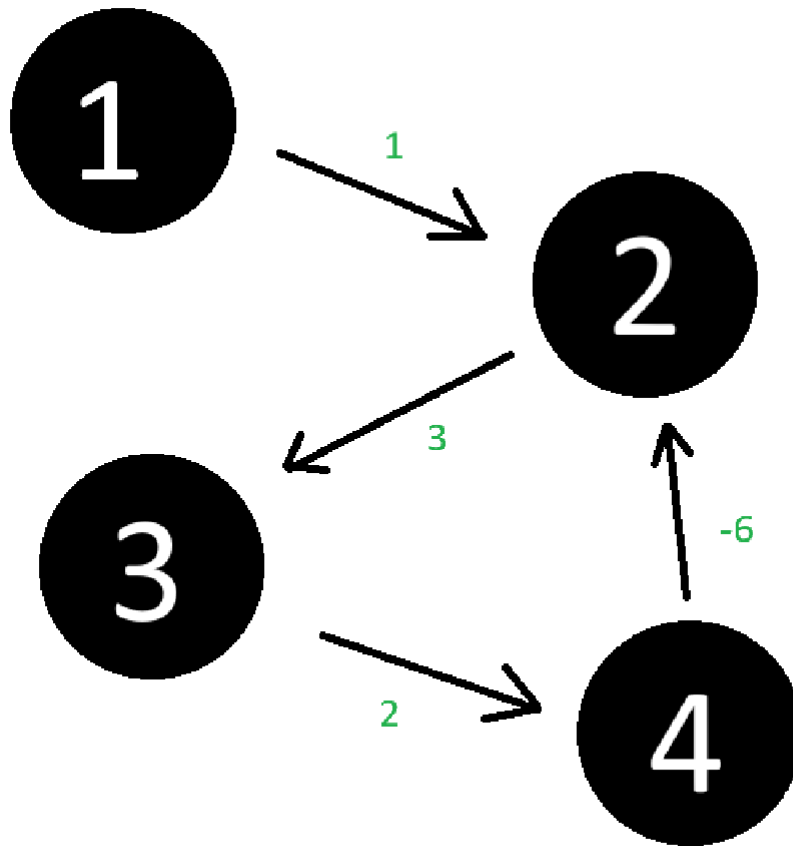


Рисунок 25 – Орграф с циклом отрицательного веса

Выберем вершину 1 как стартовую. После применения ранее разобранный алгоритм Беллмана-Форда к графу мы выясним расстояния от стартовой вершины до всех остальных вершин.

Вот как выглядит граф после $(N - 1) = 3$ итераций (рисунок 6). Это должно быть результатом, так как существует 4 ребра, нам нужно не более трех итераций, чтобы узнать кратчайший путь. После $(N - 1)$ итераций мы делаем еще одну заключительную итерацию, и, если расстояние продолжает уменьшаться, это означает, что в графе определенно есть цикл отрицательного веса.

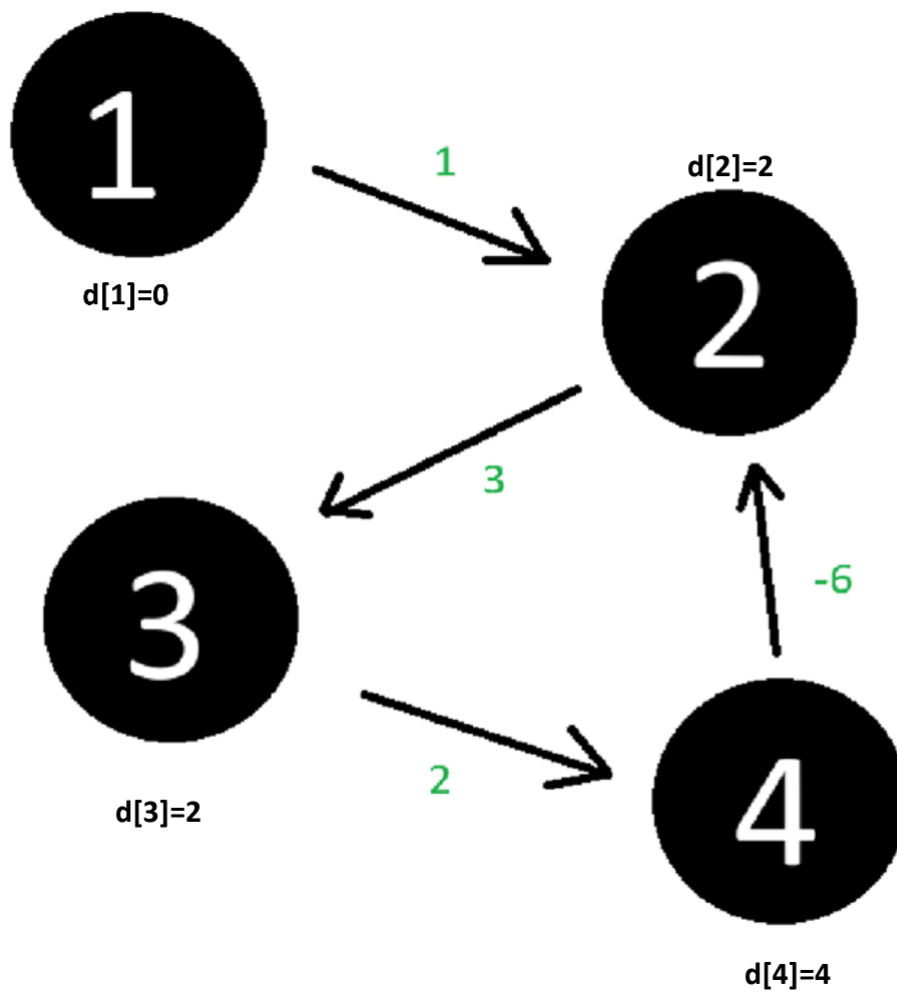


Рисунок 26 – Граф после 3 итераций

В этом примере: если мы проверим путь 2-3, $d[2] + \text{вес}[2][3]$ даст нам 1, что меньше, чем $d[3] = 2$. Поэтому мы можем заключить, что на нашем графе есть отрицательный цикл.

Список литературы

1. Официальная документация языка программирования Python версии 3.7 – <https://docs.python.org/3.7/library/random.html>
2. В. Н. Касьянов, В. А. Евстигнеев - Графы в программировании: обработка, визуализация и применение.
3. info-master.su/programming – Поляков А.В. – Основы программирования.
4. Седер Наоми – Python. Экспресс-курс. 3-е изд. (2018)
5. Роберт Седжвик - Алгоритмы на Python. Анализ структуры данных. Сортировка. Поиск. Алгоритмы на графах.
6. www.geeksforgeeks.org/bellman-ford-algorithm-dp-23 - Bellman–Ford Algorithm | DP-23

Приложение А

Код программы

```
import sys
import random
import time
from typing import List

INFTY = sys.maxsize
M1 = None
M2 = None
N = 0
nodes = []
st = 0
inf = 2000000000
Emax = 1000
edge = []

class Edges:
    def __init__(self, u, v, w):
        self.u = u
        self.v = v
        self.w = w

def generation():
    global M1, M2, N, nodes
    naprav = 0

    print("\n Укажите размер матрицы N*N: ")

    # N = int(input("--> "))

    k = True
    while k:
        N = input("--> ")
        if N.isdigit() and int(N) >= 1:
            k = False
        else:
            print(f"Введите целое число больше нуля...")
    N = int(N)
```

```

print(" Ориентированный граф?")
print("\n Да - 1 ")
print(" Нет - 2")
# c = 2
# c = input("--> ")

k = True
while k:
    c = input("--> ")
    if c.isdigit() and (int(c) == 1 or int(c) == 2):
        k = False
    else:
        print(f"Введите 1, если да, 2, если нет...")
c = int(c)

print("\n")
if c == "1":
    naprav = 1

M1 = [[0] * N for _ in range(N)]
nodes = [0] * N

random.seed()
for i in range(N):
    M1[i][i] = 0
    for j in range(i + 1, N):
        ch = random.randint(0, 9)
        ch2 = random.randint(0, 99)
        if ch < 7:
            if ch2 < 1:
                ch = -ch
            M1[i][j] = ch
        else:
            M1[i][j] = 0
        M1[j][i] = M1[i][j]

M2 = [[0] * N for _ in range(N)]

for i in range(N):
    M2[i][i] = 0
    for j in range(N):
        ch = random.randint(0, 14)
        ch2 = random.randint(0, 99)

```

```

    if ch < 12:
        if ch > 9:
            ch = ch - 6
        if ch2 < 1:
            ch = -ch
        M2[i][j] = ch
    else:
        M2[i][j] = 0

if naprav == 1:
    for i in range(N):
        for j in range(N):
            M1[i][j] = M2[i][j]

for i in range(N - 1, 0, -1):
    del M2[i]

def bellman_ford():
    global M1, N, nodes, st, inf, Emax, edge

    foute = open("bellman_ford.txt", "a")

    for i in range(N):
        nodes[i] = inf

    parent = [-1] * N
    nodes[st] = 0

    for i in range(N - 1):
        change = False

        for i in range(len(edge)):
            if nodes[edge[i].u] + edge[i].w < nodes[edge[i].v]:
                nodes[edge[i].v] = max(-inf, nodes[edge[i].u] + edge[i].w)
                parent[edge[i].v] = edge[i].u
                change = True

        if not change:
            break

    for i in range(N):
        if nodes[i] == inf:

```

```

print(f"\n {st + 1} -> {i + 1} = Путь отсутствует ")
foute.write(f"\n {st + 1} -> {i + 1} = Путь отсутствует")
else:
    print(f"\n {st + 1} -> {i + 1} = {nodes[i]}", end=" ")
    foute.write(f"\n {st + 1} -> {i + 1} = {nodes[i]} ")

    if nodes[i] >= 0 and nodes[i] < 10:
        print(" ", end=" ")

    path = []
    cur = i
    while cur != -1:
        for i2 in range(len(path)):
            if cur == path[i2] and len(path) > 1:
                path.append(cur)
                print(" (Отрицательный цикл)")
                foute.write(" (Отрицательный цикл)")
                return
        path.append(cur)
        cur = parent[cur]

    path.reverse()
    print(" (", end=" ")
    foute.write(" ( ")
    for i in range(len(path)):
        if (i + 1) != len(path):
            print(f"{path[i] + 1} -> ", end=" ")
            foute.write(f"{path[i] + 1} -> ")
        else:
            print(f"{path[i] + 1}", end=" ")
            foute.write(f"{path[i] + 1} ")
    print(")", end=" ")
    foute.write(")")
    foute.write(f"\n\n\n ----- ")

def main():
    global M1, M2, N, nodes, st, inf, Emax, edge

    fout = open("bellman_ford.txt", "a")

    if not fout:
        print("\n Ошибка открытия файла")

```

```

    return sys.exit()

fout.write("\n\n\n")

print("\n Выберите способ ввода графа:")
print(" 1 - вручную")
print(" 2 - прочитать из файла input.txt")
print(" 3 - случайная генерация")

c = input("--> ")

if c == "1":
    # print("\n Количество вершин > ")
    # N = int(input("\n Количество вершин > "))

    k = True
    while k:
        N = input("\n Количество вершин > ")
        if N.isdigit() and int(N) >= 1:
            k = False
        else:
            print("Количество вершин должно быть только целым числом\nВведите число вершин
еще раз...")
            N = int(N)

    nodes = [0] * N

    M1 = [[0] * N for _ in range(N)]

    e = 0
    for i in range(N):
        for j in range(N):
            print(f" Век {i + 1} -> {j + 1} = ", end=" ")
            # w = int(input())

            k = True
            while k:
                w = input()
                try:
                    int(w)
                    k = False
                except:
                    print(

```

```

        "Вес должен быть только целым положительным или отрицательным
числом\nВведите вес еще раз...")
        w = int(w)

        M1[i][j] = 0
        if w != 0:
            edge.append(Edges(i, j, w))
            M1[i][j] = w
            e += 1

    print("\n")
    elif c == "2":
        try:
            fin = open("input.txt", "r")
        except FileNotFoundError:
            print("\n\n Ошибка открытия файла")
            print("\n Проверьте существование файла input.txt")
            print("\n Для корректной работы программы файл должен быть заполнен в таком виде:")
            print("\n Первая строка - кол-во вершин")
            print("\n Начиная со второй - матрица смежности")
            print("\n\n Пример заполнения:")
            print("\n 3 ")
            print("\n 0 8 0 ")
            print("\n 3 0 0 ")
            print("\n 4 0 0\n ")

            fout.close()
            fout = open("input.txt", "w")
            fout.close()
            return

        print("\n\n Файл открыт")

        N = int(fin.readline())

        nodes = [0] * N
        M1 = [[0] * N for _ in range(N)]

        for i in range(N):
            line = fin.readline()
            for j, val in enumerate(line.split()):
                M1[i][j] = int(val)

```

```

fin.close()
# print("из файла: ", M1)
e = 0
for i in range(N):
    for j in range(N):
        if M1[i][j] != 0:
            edge.append(Edges(i, j, M1[i][j]))
            e += 1

elif c == "3":
    # print("\n")
    generation()
    e = 0
    for i in range(N):
        for j in range(N):
            if M1[i][j] != 0:
                edge.append(Edges(i, j, M1[i][j]))
                e += 1

else:
    main()

print(" ", end=" ")
fout.write(" ")

for i in range(N):
    print(f"[{i + 1}] ", end=" ")
    fout.write(f"[{i + 1}] ")

for i in range(N):
    print(f"\n[{i + 1}]", end=" ")
    fout.write(f"\n[{i + 1}]")

for j in range(N):
    if (i + 1) < 10:
        if M1[i][j] < 0:
            print("\b', end=" ")

    if j < 10:
        print(f"{M1[i][j]:3d} ", end=" ")
        fout.write(f"{M1[i][j]:3d} ")
    else:
        print(f" {M1[i][j]:3d} ", end=" ")

```



```

        fout.write(f" {M1[i][j]:3d} ")
    elif (i + 1) < 100:
        if j == 0:
            print(f" {M1[i][j]}", end=" ")
            fout.write(f" {M1[i][j]}")
        elif j < 10:
            print(f" {M1[i][j]}", end=" ")
            fout.write(f" {M1[i][j]}")
        else:
            print(f" {M1[i][j]}", end=" ")
            fout.write(f" {M1[i][j]}")

print("\n\n Стартовая вершина > ", end=" ")
# st = int(input())

k = True
while k:
    st = input()
    if st.isdigit() and 1 <= int(st) <= N:
        k = False
    else:
        print(f"Введите целое число в диапазоне от 1 до {N}...")
st = int(st)

fout.write(f"\n\n Стартовая вершина > {st}")
st -= 1
print("\n\n Список кратчайших путей:")
fout.write("\n Список кратчайших путей:")
fout.close()

bellman_ford()

for i in range(N - 1, 0, -1):
    del M1[i]

del M1

def titulniyList():
    print("\n Министерство науки и высшего образования Российской Федерации")
    print(" Пензенский государственный университет")
    print(" Кафедра «Вычислительная кафедра»\n")
    print(" КУРСОВОЙ ПРОЕКТ")

```

```

print(" по дисциплине:\n ЛОГИКА И ОСНОВЫ АЛГОРИТМИЗАЦИИ В ИНЖЕНЕРНЫХ ЗАДАЧАХ")
print(" тема проекта:\n Реализация алгоритма Форда Беллмана\n\n")
print(" Выполнил:\n студент группы 22ВВВ2\n Демини М.С. \n\n")
print(" Принял:\n Акифьев И.В.\n\n Пенза 2023\n")

```

```

def wanna_more(): # продолжение после окончания расчетов
    k = True
    while k:
        todo = input("\n\nПродолжаем Форда?\n 1 - да\n 2 - нет\n -> ")
        if todo.isdigit() and (int(todo) == 1 or int(todo) == 2):
            k = False
        else:
            print(f"Введите целое число: '1' или '2'...")
    todo = int(todo)
    if todo == 1:
        main()
    else:
        print("\n\nЗавершение работы...\n\n...MaxSDemin © 2023...")
        return False
    return True

```

```

if __name__ == "__main__":
    titulyList()
    main()
    while wanna_more():
        pass

```

Приложение В

Результаты работы программы

```
Ориентированный граф?

Да - 1
Нет - 2
--> 2

      [1]  [2]  [3]
[1]   0    2    3
[2]   2    0    3
[3]   3    3    0

Стартовая вершина > 2

Список кратчайших путей:

2 -> 1 = 2      ( 2 -> 1 )
2 -> 2 = 0      ( 2 )
2 -> 3 = 3      ( 2 -> 3 )

Продолжаем Форда?
1 - да
2 - нет
->
```

27. Результат случайно сгенерированного неориентированного графа

```
-----  
  
      [1] [2]  
[1]   3   2  
[2]   2   2
```

```
Стартовая вершина > 2  
Список кратчайших путей:  
2 -> 1 = 2  ( 2 -> 1 )  
2 -> 2 = 0  ( 2 )
```

```
-----  
  
      [1] [2] [3]  
[1]   0   2   3  
[2]   2   0   3  
[3]   3   3   0
```

```
Стартовая вершина > 2  
Список кратчайших путей:  
2 -> 1 = 2  ( 2 -> 1 )  
2 -> 2 = 0  ( 2 )  
2 -> 3 = 3  ( 2 -> 3 )
```

28. Результат сохранения работы программы в файл