«Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчёт защищён с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Астахова А.В.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Отчёт

Лабораторной работе №2

«Управление проектами. Расчет параметров событий и работ сетевого  
графика. Определение критического пути»

Студент группы ПИ 92 В.М. Шульпов

Преподаватель доцент, к. эк. н. Астахова А.В.

Барнаул 2022

**Задание**.  
1. Изучить параграф 2.5 (лекция 2) и Приложение к работе.  
(см. в Илиас «Лекция 2» и файл с материалами учебника: часть 2)  
2. Разработать тестовые примеры для выполнения работы.  
3. Разработать алгоритм расчета параметров событий и работ СГ с учетом  
требований Приложения к работе и данного пункта задания.  
При разработке алгоритма и программы не должны использоваться  
стандартные библиотеки работы с графами и программы расчета сетевых  
графиков.  
Составить и отладить программу на выбранном языке программирования,  
которая обращается к программе, разработанной в лабораторной работе 2, и  
выводит на экран монитора:  
– исходный список работ СГ;  
– таблицу «Параметры событий СГ»:

|  |  |  |
| --- | --- | --- |
| Событие | Сроки свершения события | Резерв времени события |
| ранний | поздний |  |

–- таблицу «Параметры работ СГ»:

|  |  |  |
| --- | --- | --- |
| Работа | Продолжительность работы | Резервы времени работы |
| полный | независимый |  |

– список работ всех критических путей СГ, которые определяются с учетом  
необходимого и достаточного условия отнесения пути к критическому;  
– длину критического пути.  
4. Оформить отчет, внеся все необходимые комментарии в программу. Защитить  
работу Теория:

1. Ранний (или ожидаемый) срок tp(i) свершения i-го события равен продолжительности максимального пути, предшествующему этому событию.

tp(i) = max(Lni), где Lni – любой путь, предшествующий i-му событию, т.е. пусть от исходного до i-го события сети.

Если событие j имеет несколько предшествующих путей, следовательно, несколько предшествующих событий i, то ранний срок свершения события j равен

Структура графа:

self.fictive\_start\_top = -100 # шифр фиктивной стартовой вершины  
self.fictive\_end\_top = -101 # шифр фиктивной конечной вершины  
self.graph\_table = {'arc\_start': [],  
 'arc\_end': [],  
 'weight': [],  
 'is\_visited': [], } # таблица, содержащая дуги и веса графа  
self.struct\_graph\_table = {'arc\_start': [],  
 'arc\_end': [],  
 'weight': [],  
 'is\_visited': [], } # упорядоченная таблица, содержащая дуги и веса графа  
self.first\_top = None # первая вершина графа  
self.last\_top = None # последняя вершина графа  
self.ways\_num = 0 # количество путей (работ) графа  
self.struct\_graph\_ways\_num = 0 # количество путей (работ) упорядоченного графа  
self.current\_way = [] # текущий путь для вывода всех полных путей

0) инициализация и получение исходного графа из файла происходит в конструкторе \_\_init\_\_(self, file\_in)

1) метод print\_row(self, row\_index, sorted\_graph=False)печатает строку таблицу по индексу

2) метод print\_graph(self, sorted\_graph=False)печатает всю таблицу графа, используя функцию print\_row. Есть параметр функции, который указывает какой граф выводить (упорядоченный или нет).

3) метод add\_row\_in\_graph\_table(self, start, end, weight, is\_visited=False, adding\_is\_in\_sort=False) добавляет строку в таблицу графа. Есть параметр функции, который указывает в какой граф добавлять (упорядоченный или нет). При добавлении метод увеличивает счётчик количества строк соответствующего графа на 1.

4) метод def copy\_row\_to\_struct\_graph\_table(self, index) копирует строку по индексу из неупорядоченного графа и добавляет её в конец упорядоченного. Метод вызывает метод add\_row\_in\_graph\_table.

5) метод delete\_row\_from\_graph\_table(self, index) удаляет строку из неупорядоченного графа. Уменьшает счётчик количества строк в неупорядоченном графе на 1.

6) метод search\_first\_top(self) ищет начальную вершину СГ.

Проходит во внешнем цикле по всем строкам таблицы путей графа  
Во внутреннем цикле проверяется, не является ли стартовой (в неё входит дуга, кроме петель) текущая вершина. Если найдена стартовая вершина (и не посещенная), проверяется, первая ли она такая. Если первая, то происходит запоминание её шифра и переход к следующей вершине. Если стартовая вершина уже существует, то происходит добавление фиктивной вершины (добавляется дополнительная запись в таблицу СГ.

7) метод delete\_top\_loops(self, index) проверяет строку на наличие петли и удаляет строку, если петля есть. Возвращает счётчик цикла: если было удаление, то возвращает 0, чтобы запустить цикл сначала, если не было удаления, то возвращает тот же индекс цикла и ничего не изменяется. (так как нет goto)

8) метод check\_duplication(self, i, j) проверяет наличие дублирования строки (с одинаковым и разным весом). При разных весах предлагает выбрать, какую работу удалить. При одинаковом удаляет автоматически вторую (разницы какую удалять нет). Метод аналогично возвращает счётчик цикла для запуска цикла сначала. (так как нет goto).

9) метод search\_last\_top(self) работает аналогично методу с поиском первой вершины. В нём происходит поиск последней вершины. Если вершина не одна, создается фиктивная конечная вершина.

10) метод optimize\_graph(self) вызывает методы delete\_top\_loops, check\_duplication, search\_last\_top и управляет порядком их вызова. Когда нужн повторно запустить метод, когда нет он понимает по возвращаемым значениям из этих функций (так как нет goto).

11) метод struct\_graph(self) упорядочивает граф, используя очередь (проходит граф в ширину). Новый упорядоченный граф хранится другом объекте.

12) метод recursive\_search(self) рекурсивно проходит по графу.

Если последний элемент в текущем пути равен последнему элементу в графе, запускается цикл печати пути, удаляется последняя вершина из текущего пути и происходит выход из функции.

Иначе происходит вход в цикл( i от 0 до количества строк в таблице графа). Если стартовая вершина i-ой работы равна последней работе в текущем пути, то добавляем в текущий путь конечную вершину i-ой работы и рекурсивно запускаем recursive\_search.

После этого цикла удаляется последняя вершина из текущего пути.

13) метод search\_full\_ways(self) находит все полные пути графа, вызывая рекурсивный метод recursive\_search(self). Перед этим вызовом находится первая строка, в которой хранится первая вершина. Добавляет эту вершину в текущий путь.

14) код основой программы

if \_\_name\_\_ == "\_\_main\_\_":  
 in\_file = "input.csv"  
 graph = Graph(in\_file)  
 graph.print\_graph()  
 graph.search\_first\_top()  
 graph.optimize\_graph()  
 graph.print\_graph()  
 graph.struct\_graph()  
 graph.print\_graph(sorted\_graph=True)  
 graph.search\_full\_ways()

**Тесты:**

**Тест 1**

Чтение из файла проведено успешно

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 20 |

| 2 | 3 | 10 |

| 2 | 3 | 30 |

| 4 | 8 | 10 |

| 4 | 3 | 15 |

| 3 | 5 | 1 |

| 2 | 3 | 15 |

| 1 | 1 | 1 |

Первая вершина: None

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение первой вершины

создание фиктивной первой вершины

оптимизация:

ошибка! работа (2, 3) дублируется

работа имеет 2 веса: 10 и 30

Введите нужный вариант (1 или 2):1

оптимизация:

ошибка! работа (2, 3) дублируется

работа имеет 2 веса: 30 и 15

Введите нужный вариант (1 или 2):2

найдена петля (1, 1)! автоматическое удаление работы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 20 |

| 2 | 3 | 30 |

| 4 | 8 | 10 |

| 4 | 3 | 15 |

| 3 | 5 | 1 |

| -100 | 1 | 0 |

| -100 | 4 | 0 |

Первая вершина: -100

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение последней вершины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 20 |

| 2 | 3 | 30 |

| 4 | 8 | 10 |

| 4 | 3 | 15 |

| 3 | 5 | 1 |

| -100 | 1 | 0 |

| -100 | 4 | 0 |

Первая вершина: -100

Последняя вершина: 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение последней вершины

вершины графа 5 и 8 конечные

Ввести фиктивную конечную вершину (иначе удалить) (1-да, 2-нет): 2

удаление вершины 8

нахождение последней вершины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 20 |

| 2 | 3 | 30 |

| 4 | 3 | 15 |

| 3 | 5 | 1 |

| -100 | 1 | 0 |

| -100 | 4 | 0 |

Первая вершина: -100

Последняя вершина: 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

упорядочивание графа

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| -100 | 1 | 0 |

| -100 | 4 | 0 |

| 1 | 2 | 20 |

| 4 | 3 | 15 |

| 2 | 3 | 30 |

| 3 | 5 | 1 |

Первая вершина: -100

Последняя вершина: 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полные пути графа:

-100 1 2 3 5

-100 4 3 5

**Тест 2**

Чтение из файла проведено успешно

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 20 |

| 2 | 3 | 10 |

| 2 | 3 | 30 |

| 1 | 5 | 5 |

| 4 | 3 | 15 |

| 3 | 5 | 1 |

Первая вершина: None

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение первой вершины

создание фиктивной первой вершины

оптимизация:

ошибка! работа (2, 3) дублируется

работа имеет 2 веса: 10 и 30

Введите нужный вариант (1 или 2):2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 20 |

| 2 | 3 | 10 |

| 1 | 5 | 5 |

| 4 | 3 | 15 |

| 3 | 5 | 1 |

| -100 | 1 | 0 |

| -100 | 4 | 0 |

Первая вершина: -100

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение последней вершины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 20 |

| 2 | 3 | 10 |

| 1 | 5 | 5 |

| 4 | 3 | 15 |

| 3 | 5 | 1 |

| -100 | 1 | 0 |

| -100 | 4 | 0 |

Первая вершина: -100

Последняя вершина: 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

упорядочивание графа

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| -100 | 1 | 0 |

| -100 | 4 | 0 |

| 1 | 2 | 20 |

| 1 | 5 | 5 |

| 4 | 3 | 15 |

| 2 | 3 | 10 |

| 3 | 5 | 1 |

Первая вершина: -100

Последняя вершина: 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полные пути графа:

-100 1 2 3 5

-100 1 5

-100 4 3 5

**Тест 3**

Чтение из файла проведено успешно

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 10 | 9 | 1 |

| 10 | 8 | 1 |

| 10 | 10 | 1 |

| 5 | 4 | 1 |

| 5 | 3 | 1 |

| 4 | 8 | 1 |

| 10 | 9 | 1 |

| 9 | 11 | 1 |

| 9 | 8 | 1 |

| 3 | 9 | 1 |

Первая вершина: None

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение первой вершины

создание фиктивной первой вершины

оптимизация:

ошибка! работа (10, 9) дублируется

работа имеет одинаковый вес 1; автоматическое удаление

найдена петля (10, 10)! автоматическое удаление работы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 10 | 9 | 1 |

| 10 | 8 | 1 |

| 5 | 4 | 1 |

| 5 | 3 | 1 |

| 4 | 8 | 1 |

| 9 | 11 | 1 |

| 9 | 8 | 1 |

| 3 | 9 | 1 |

| -100 | 10 | 0 |

| -100 | 5 | 0 |

Первая вершина: -100

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение последней вершины

вершины графа 8 и 11 конечные

Ввести фиктивную конечную вершину (иначе удалить) (1-да, 2-нет): 2

удаление вершины 11

нахождение последней вершины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 10 | 9 | 1 |

| 10 | 8 | 1 |

| 5 | 4 | 1 |

| 5 | 3 | 1 |

| 4 | 8 | 1 |

| 9 | 8 | 1 |

| 3 | 9 | 1 |

| -100 | 10 | 0 |

| -100 | 5 | 0 |

Первая вершина: -100

Последняя вершина: 8

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

упорядочивание графа

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| -100 | 10 | 0 |

| -100 | 5 | 0 |

| 10 | 9 | 1 |

| 10 | 8 | 1 |

| 5 | 4 | 1 |

| 5 | 3 | 1 |

| 9 | 8 | 1 |

| 4 | 8 | 1 |

| 3 | 9 | 1 |

Первая вершина: -100

Последняя вершина: 8

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полные пути графа:

-100 10 9 8

-100 10 8

-100 5 4 8

-100 5 3 9 8

Process finished with exit code 0

**Тест 4**

C:\Python38-32\python.exe C:/Users/Victor/MyFolder/AltSTU\_repo/6\_sem/Dev-and-implementation-of-projects\_\_Astahova/lab1/project/main.py

Чтение из файла проведено успешно

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 10 |

| 1 | 2 | 20 |

| 1 | 2 | 30 |

| 1 | 2 | 40 |

| 2 | 3 | 100 |

| 2 | 4 | 100 |

| 2 | 3 | 50 |

Первая вершина: None

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение первой вершины

оптимизация:

ошибка! работа (1, 2) дублируется

работа имеет 2 веса: 10 и 20

Введите нужный вариант (1 или 2):2

ошибка! работа (1, 2) дублируется

работа имеет 2 веса: 10 и 30

Введите нужный вариант (1 или 2):2

ошибка! работа (1, 2) дублируется

работа имеет 2 веса: 10 и 40

Введите нужный вариант (1 или 2):1

оптимизация:

ошибка! работа (2, 3) дублируется

работа имеет 2 веса: 100 и 50

Введите нужный вариант (1 или 2):1

оптимизация:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 40 |

| 2 | 4 | 100 |

| 2 | 3 | 50 |

Первая вершина: 1

Последняя вершина: None

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение последней вершины

вершины графа 4 и 3 конечные

Ввести фиктивную конечную вершину (иначе удалить) (1-да, 2-нет): 1

создание фиктвной последней вершины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 40 |

| 2 | 4 | 100 |

| 2 | 3 | 50 |

| 4 | -101 | 0 |

| 3 | -101 | 0 |

Первая вершина: 1

Последняя вершина: -101

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение последней вершины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 40 |

| 2 | 4 | 100 |

| 2 | 3 | 50 |

| 4 | -101 | 0 |

| 3 | -101 | 0 |

Первая вершина: 1

Последняя вершина: -101

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

нахождение последней вершины

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 40 |

| 2 | 4 | 100 |

| 2 | 3 | 50 |

| 4 | -101 | 0 |

| 3 | -101 | 0 |

Первая вершина: 1

Последняя вершина: -101

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

упорядочивание графа

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вывод таблицы путей графа

| --A-- | --B-- | --Вес-- |

| 1 | 2 | 40 |

| 2 | 4 | 100 |

| 2 | 3 | 50 |

| 4 | -101 | 0 |

| 3 | -101 | 0 |

Первая вершина: 1

Последняя вершина: -101

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полные пути графа:

1 2 4 -101

1 2 3 -101

**Код:**

**import** csv  
  
  
**class** Graph:  
  
 **def** \_\_init\_\_(self, file\_in):  
 self.fictive\_start\_top = -100 *# шифр фиктивной стартовой вершины* self.fictive\_end\_top = -101 *# шифр фиктивной конечной вершины* self.graph\_table = {**'arc\_start'**: [],  
 **'arc\_end'**: [],  
 **'weight'**: [],  
 **'is\_visited'**: [],  
 **'layer'**: [], } *# таблица, содержащая дуги и веса графа* self.struct\_graph\_table = {**'arc\_start'**: [],  
 **'arc\_end'**: [],  
 **'weight'**: [],  
 **'is\_visited'**: [], } *# упоряденная таблица, содержащая дуги и веса графа* self.first\_top = **None** *# первая вершина графа* self.last\_top = **None** *# последняя вершина графа* self.ways\_num = 0 *# количество путей (работ) графа* self.struct\_graph\_ways\_num = 0 *# количество путей (работ) упорядоченного графа* self.current\_way = [] *# текущий путь для вывода всех полных путей* **with** open(file\_in) **as** File:  
 reader = csv.reader(File, delimiter=**';'**)  
 **for** row **in** reader:  
 self.add\_row\_in\_graph\_table(int(row[0]), int(row[1]), int(row[2]))  
 print(**'Чтение из файла проведено успешно'**)  
  
 **def** add\_row\_in\_graph\_table(self, start, end, weight, is\_visited=**False**, adding\_is\_in\_sort=**False**):  
 **if** adding\_is\_in\_sort:  
 self.struct\_graph\_table[**'arc\_start'**].append(start)  
 self.struct\_graph\_table[**'arc\_end'**].append(end)  
 self.struct\_graph\_table[**'weight'**].append(weight)  
 self.struct\_graph\_table[**'is\_visited'**].append(is\_visited)  
 self.struct\_graph\_ways\_num += 1  
 **else**:  
 self.graph\_table[**'arc\_start'**].append(start)  
 self.graph\_table[**'arc\_end'**].append(end)  
 self.graph\_table[**'weight'**].append(weight)  
 self.graph\_table[**'is\_visited'**].append(is\_visited)  
 self.ways\_num += 1  
  
 **def** copy\_row\_to\_struct\_graph\_table(self, index):  
 start = self.graph\_table[**'arc\_start'**][index]  
 end = self.graph\_table[**'arc\_end'**][index]  
 weight = self.graph\_table[**'weight'**][index]  
 is\_visited = self.graph\_table[**'is\_visited'**][index]  
 self.add\_row\_in\_graph\_table(start, end, weight, is\_visited, adding\_is\_in\_sort=**True**)  
  
 **def** delete\_row\_from\_graph\_table(self, index):  
 self.graph\_table[**'arc\_start'**].pop(index)  
 self.graph\_table[**'arc\_end'**].pop(index)  
 self.graph\_table[**'weight'**].pop(index)  
 self.graph\_table[**'is\_visited'**].pop(index)  
 self.ways\_num -= 1  
  
 **def** print\_graph(self, sorted\_graph=**False**):  
 *""" печать таблицы |A|B|time|"""* print(**'\_'** \* 100)  
 print(**'Вывод таблицы путей графа'**)  
 print(**'|{start:^10}|{end:^10}|{weight:^10}|'**.format(start=**'--A--'**, end=**'--B--'**, weight=**'--Вес--'**))  
 **for** row\_index **in** range(0, self.ways\_num):  
 self.print\_row(row\_index, sorted\_graph)  
 print(**f'Первая вершина: {**self.first\_top**}'**)  
 print(**f'Последняя вершина: {**self.last\_top**}'**)  
 print(**'\_'** \* 100)  
  
 **def** print\_row(self, row\_index, sorted\_graph=**False**):  
 **if** sorted\_graph:  
 print(**'|{start:^10}|{end:^10}|{weight:^10}|'**.format(  
 start=self.struct\_graph\_table[**'arc\_start'**][row\_index],  
 end=self.struct\_graph\_table[**'arc\_end'**][row\_index],  
 weight=self.struct\_graph\_table[**'weight'**][row\_index]))  
 **else**:  
 print(**'|{start:^10}|{end:^10}|{weight:^10}|'**.format(  
 start=self.graph\_table[**'arc\_start'**][row\_index],  
 end=self.graph\_table[**'arc\_end'**][row\_index],  
 weight=self.graph\_table[**'weight'**][row\_index]))  
  
 **def** search\_first\_top(self):  
 *""" поиск первой вершины графа """* print(**'нахождение первой вершины'**)  
 **for** i **in** range(0, self.ways\_num): *# итерация 1 по таблице* is\_start\_top = **True  
 for** j **in** range(0, self.ways\_num): *# итерация 2 по таблице для поиска первой вершины  
 # условие, что в вершину входит дуга (кроме петель), если входит, то это не стартовая вершина* **if** (self.graph\_table[**'arc\_start'**][i] == self.graph\_table[**'arc\_end'**][j] **and** self.graph\_table[**'arc\_start'**][i] != self.graph\_table[**'arc\_start'**][j]):  
 is\_start\_top = **False  
 break** *# если это непосещенная первая вершина* **if** is\_start\_top **and not** self.graph\_table[**'is\_visited'**][i]:  
 *# если это первая первая вершина (не изменялась после инициализации)* **if** self.first\_top **is None**:  
 self.first\_top = self.graph\_table[**'arc\_start'**][i]  
 **else**:  
 print(**'создание фиктивной первой вершины'**)  
 self.add\_row\_in\_graph\_table(self.fictive\_start\_top, self.first\_top, 0)  
 self.add\_row\_in\_graph\_table(self.fictive\_start\_top, self.graph\_table[**'arc\_start'**][i], 0)  
 **if** self.first\_top != self.fictive\_start\_top:  
 self.first\_top = self.fictive\_start\_top  
  
 **for** index **in** range(0, self.ways\_num):  
 **if** self.graph\_table[**'arc\_start'**][index] == self.graph\_table[**'arc\_start'**][i]:  
 self.graph\_table[**'is\_visited'**][index] = **True  
  
 def** delete\_top\_loops(self, index):  
 *""" удаление петель """* start\_top = self.graph\_table[**'arc\_start'**][index]  
 end\_top = self.graph\_table[**'arc\_end'**][index]  
 **if** start\_top == end\_top:  
 print(**f'найдена петля ({**start\_top**}, {**end\_top**})! автоматическое удаление работы'**)  
 self.delete\_row\_from\_graph\_table(index)  
 **return** 0  
 **return** index  
  
 **def** check\_duplication(self, i, j):  
 *""" проверка дубликатов работ """* start\_work = self.graph\_table[**'arc\_start'**][i]  
 starts\_are\_equal = start\_work == self.graph\_table[**'arc\_start'**][j]  
 end\_work = self.graph\_table[**'arc\_end'**][i]  
 ends\_are\_equal = end\_work == self.graph\_table[**'arc\_end'**][j]  
 *# если есть дубликаты* **if** starts\_are\_equal **and** ends\_are\_equal **and** i != j:  
 print(**f'ошибка! работа ({**start\_work**}, {**end\_work**}) дублируется'**)  
 *# если веса разные* **if** self.graph\_table[**'weight'**][i] != self.graph\_table[**'weight'**][j]:  
 weight\_i = self.graph\_table[**'weight'**][i]  
 weight\_j = self.graph\_table[**'weight'**][j]  
 print(**f'работа имеет 2 веса: {**weight\_i**} и {**weight\_j**}'**)  
 choice = input(**'Введите нужный вариант (1 или 2):'**)  
 **if** choice == **'1'**:  
 self.delete\_row\_from\_graph\_table(i)  
 self.optimize\_graph()  
 **return None  
 else**:  
 self.delete\_row\_from\_graph\_table(j)  
 **return** 0  
 *# если веса одинаковые* **else**:  
 print(**f'работа имеет одинаковый вес {**self.graph\_table[**"weight"**][i]**}; автоматическое удаление'**)  
 self.delete\_row\_from\_graph\_table(j)  
 **return** 0  
  
 **def** search\_last\_top(self):  
 *""" поиск последней вершины графа """* print(**'нахождение последней вершины'**)  
 **for** i **in** range(0, self.ways\_num): *# итерация 1 по таблице* is\_last\_top = **True  
 for** j **in** range(0, self.ways\_num): *# итерация 2 по таблице для поиска последней вершины  
 # условие, что из вершины выходит дуга (кроме петель), если выходит, то это не последняя вершина* **if** (self.graph\_table[**'arc\_end'**][i] == self.graph\_table[**'arc\_start'**][j] **and** self.graph\_table[**'arc\_end'**][i] != self.graph\_table[**'arc\_end'**][j]):  
 is\_last\_top = **False  
 break** *# если это непосещенная последняя вершина* **if** is\_last\_top **and not** self.graph\_table[**'is\_visited'**][i]:  
 *# если это первая последняя вершина (не изменялась после инициализации)* **if** self.last\_top **is None**:  
 self.last\_top = self.graph\_table[**'arc\_end'**][i]  
 *# отметка всех работ с таким шифром как посещенные* **for** k **in** range(0, self.ways\_num):  
 **if** self.graph\_table[**'arc\_end'**][k] == self.graph\_table[**'arc\_end'**][i]:  
 self.graph\_table[**'is\_visited'**][k] = **True  
 else**:  
 *# если это не одна и та же вершина* **if** self.last\_top != self.graph\_table[**'arc\_end'**][i]:  
 print(**f'вершины графа {**self.last\_top**} и {**self.graph\_table[**"arc\_end"**][i]**} конечные'**)  
 choice = input(**'Ввести фиктивную конечную вершину (иначе удалить) (1-да, 2-нет): '**)  
 **if** choice == **'1'**:  
 print(**'создание фиктвной последней вершины'**)  
 self.add\_row\_in\_graph\_table(self.last\_top, self.fictive\_end\_top, 0)  
 self.add\_row\_in\_graph\_table(self.graph\_table[**'arc\_end'**][i], self.fictive\_end\_top, 0)  
 **if** self.last\_top != self.fictive\_end\_top:  
 self.last\_top = self.fictive\_end\_top  
  
 **for** index **in** range(0, self.ways\_num):  
 **if** self.graph\_table[**'arc\_end'**][index] == self.graph\_table[**'arc\_end'**][i]:  
 self.graph\_table[**'is\_visited'**][index] = **True** *# удаление одной из вершин* **else**:  
 print(**f'удаление вершины {**self.graph\_table[**"arc\_end"**][i]**}'**)  
 self.delete\_row\_from\_graph\_table(i)  
 self.last\_top = **None  
 for** index **in** range(0, self.ways\_num):  
 self.graph\_table[**'is\_visited'**][index] = **False** self.search\_last\_top()  
 **break****def** optimize\_graph(self):  
 *""" оптимизация графа (удаление петель, одинаковых работ, поиск последней вершины"""* print(**'оптимизация:'**)  
 i = 0  
 **while** i < self.ways\_num:  
 *# one* i = self.delete\_top\_loops(i)  
 *# two* j = 0  
 **while** j < self.ways\_num: *# сравнение всех вершин СГ* temp = self.check\_duplication(i, j)  
 *# None при запуске optimize\_graph рекурсивно из check\_duplication(i, j). Иначе обнуляется счётчик цикла.* **if** temp **is not None**:  
 j = temp  
 j += 1  
 i += 1  
 self.print\_graph()  
 self.search\_last\_top()  
 **for** index **in** range(0, self.ways\_num):  
 self.graph\_table[**'is\_visited'**][index] = **False****def** find\_layer(self, layer\_level, top\_queue):  
 *# отметка слоев этого уровня и нахождение слоев следующего уровня* self.find\_layer(layer\_level + 1, top\_queue)  
  
 **def** struct\_graph(self):  
 print(**'упорядочивание графа'**)  
 vertex\_queue = [self.first\_top]  
 **while** self.ways\_num != self.struct\_graph\_ways\_num:  
 **for** i **in** range(0, self.ways\_num):  
 **if** vertex\_queue[0] == self.graph\_table[**'arc\_start'**][i] **and not** self.graph\_table[**'is\_visited'**][i]:  
 self.copy\_row\_to\_struct\_graph\_table(i)  
 vertex\_queue.append(self.graph\_table[**'arc\_end'**][i])  
 self.graph\_table[**'is\_visited'**][i] = **True** vertex\_queue.pop(0)  
  
 **def** search\_full\_ways(self):  
 *""" нахождение всех полных путей графа """* self.current\_way = [self.first\_top]  
 print(**'полные пути графа:'**)  
 **for** i **in** range(0, self.ways\_num):  
 **if** self.graph\_table[**'arc\_start'**][i] == self.first\_top:  
 self.current\_way.append(self.graph\_table[**'arc\_end'**][i]) *# запомнить следуюущую вершину в списке путей* self.recursive\_search()  
  
 **def** recursive\_search(self):  
 *""" рекурсивный обход графа """* **if** self.current\_way[len(self.current\_way) - 1] == self.last\_top:  
 **for** i **in** range(0, len(self.current\_way)):  
 print(**f'{**self.current\_way[i]**} '**, end=**''**)  
 print()  
 self.current\_way.pop() *# удаление последней вершины, возврат по стеку* **return  
  
 for** i **in** range(0, self.ways\_num):  
 **if** self.graph\_table[**'arc\_start'**][i] == self.current\_way[len(self.current\_way) - 1]:  
 self.current\_way.append(self.graph\_table[**'arc\_end'**][i]) *# запомнить вершину* self.recursive\_search()  
 self.current\_way.pop() *# удаление последней вершины* **return  
  
  
if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 in\_file = **"input.csv"** graph = Graph(in\_file)  
 graph.print\_graph()  
 graph.search\_first\_top()  
 graph.optimize\_graph()  
 graph.print\_graph()  
 graph.struct\_graph()  
 graph.print\_graph(sorted\_graph=**True**)  
 graph.search\_full\_ways()