«Найти самый длинный путь»

**Содержание**

Введение

1. Подготовка окружения

2. Алгоритм поиска длинного пути

2.1. Способы поиска

2.2. Блок-схема

3 Поиск пути

3.1. Подготовка данных

3.2. Операции над данными и поиск

3.3. Поиск пути

3.4. Результат

4. Проверка верности результата

4.1. Проверка других элементов самой «жирной» вершины.

4.2. Выборочная проверка пути у вершин с меньшим количеством связей.

5. Результат проверки

6. Заключение

**Введение**

Поиск самого длинного пути является задачей без решения, но если ставить определенные условия, то можно прийти к некоторому результату. В работе будет описан мой подход для реализации данной задачи, методы, которыми я пользовался для получения ответа, а также, будет проделан ряд проверок на верность решения.

**1. Подготовка окружения**

Вся работа выполнялась на *ОС Ubuntu 20.04*

**Для выполнения задания необходимо установить:**

1. Apache Spark. *Использовалась версия 3.3.0*

2. PySpark. *Для работы на языке Python.*

3. Jupyter Notebook. *Для удобства работы.*

**Библиотеки для работы:**

1. Pysspark.sql. *Для создания сессии Sparka*

2. GraphFrame. *Для создания графа*

3. Networkx. *Для работы с графом + визуализация.*

4. Matplotlib. *Для визуализации.*

**Данные, которые обрабатываем**:

1. Были приложены к заданию.

**2. Алгоритм поиска длинного пути**

**2.1. Способы поиска**

Я думал о двух способах решения:

1. **Перебор всех комбинаций src и dst**. Перебор каждой вершины. Не рационально.

2. **Перебор только «явных» dst и src.** Явные (я также называю их *жирными*), то есть значения, которые точно участвуют в связях и при этом имеют их большое, по сравнению с другими, количество.

**Раскрою второй алгоритм:**

1. Создание графа

2. Создание списка вершин.

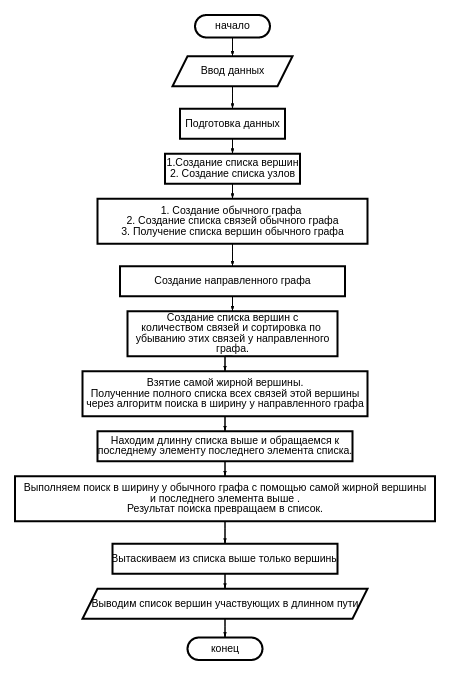
3. Перебор вершин на поиск самой «жирной», путем считывания количества связей через функцию bfs\_tree библиотеки networkx.

4. В найденной вершине взять первую (начало) и последнюю (конец) вершину. Совершить поиск пути через функцию bfs библиотеки GraphFrame.

Вывод:

1. Выбор оптимального способа поиска путем перебора «жирных» значений.

**2.2. Блок-схема**



Вывод:

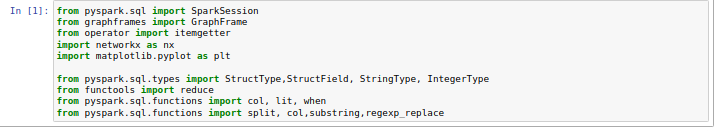
1. Сделана визуализация алгоритма.

**3. Поиск длинного пути**

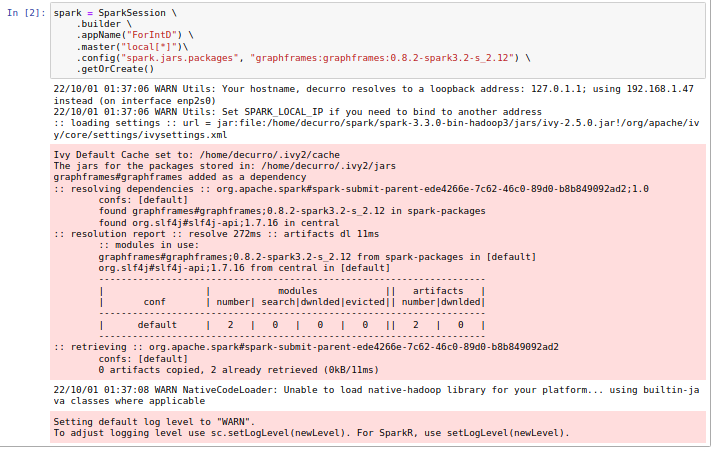
**3.1. Подготовка данных**

**Цель:** подготовить пространство для работы. Создать обычный и направленный граф из предоставленных данных.

Подключение необходимых для работы библиотек.



Создание сессии в Spark с подключением дополнительного пакета для работы с GraphFrame.

Загрузка данных, с которыми будет работа. Файл в формате csv.

Создание схемы для этого файла.



Удаление столбца cnt, так как он не будет участвовать в работе.



Удаление Null и не уникальных значений.

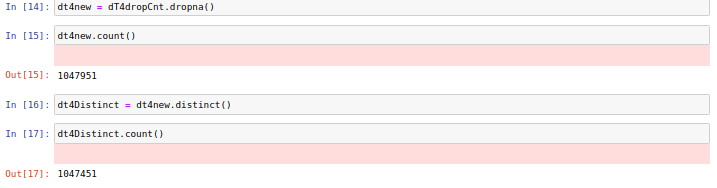
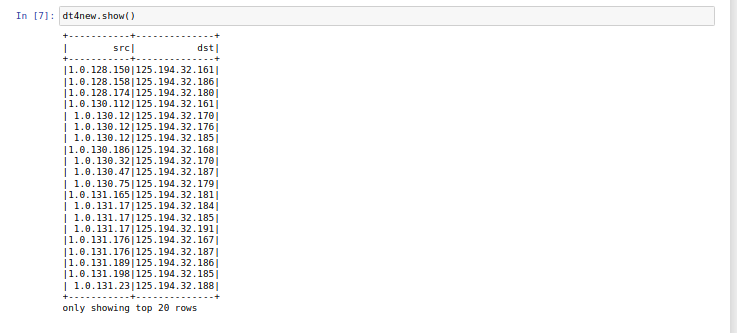
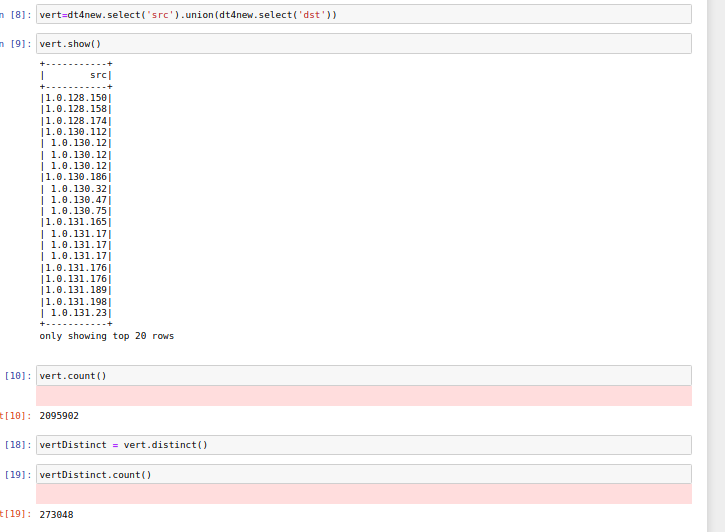


Таблица после проделанных выше операций.

 Объединение столбцов src и dst, удаление не уникальных значений для создание графа.



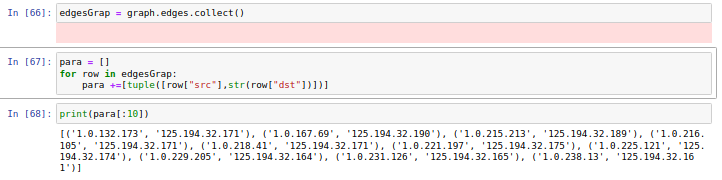
Количество вершин сократилось в ~77 раз.

Создание графа (graph) из уникальных вершин.



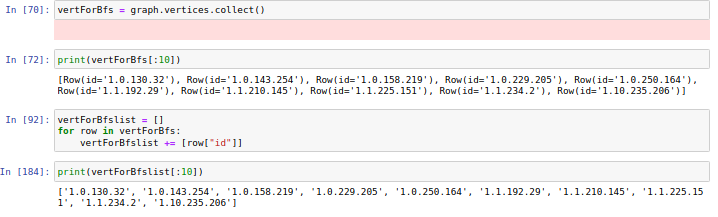


Создание списка связей (para), графа (graph).

 Создание направленного графа(grapRisDi) с помощью библиотеки networkx.



Создание списка вершин (vertForBfslist), графа (graph).



На данном этапе подготовка данных завершена

Вывод:

1. Созданы граф — graph, направленный граф - grapRisDi

2. Получен список вершин графа (graph) - vertForBfslist

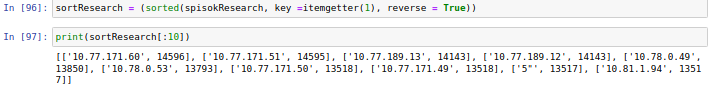
**3.2. Операции над данными**

**Цель:** найти вершину с самым большим деревом и определить последний элемент этой вершины.

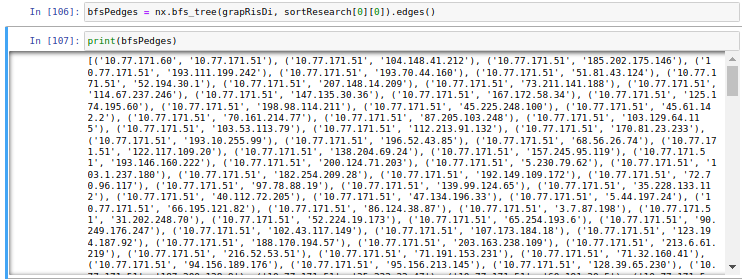
Создание сортированного списка списков вершин и количества связей по убыванию (sortResearch), c помощью функции bfs\_tree (поиск в ширину) библиотеки networkx.

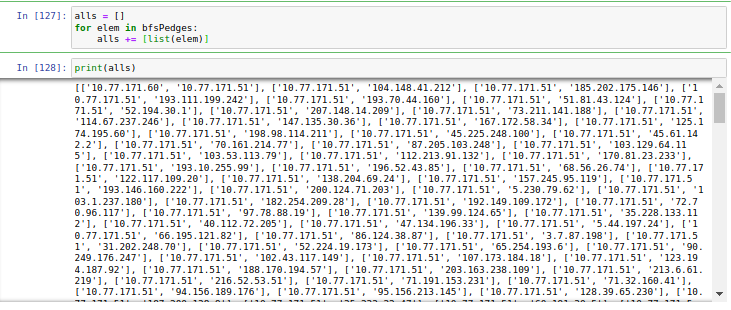




 Самая «жирная» вершина будет в sortResearch[0][0].

Создание полного списка связей (bfsPedges) найденной вершины.

 Перевод списка кортежей в список (alls).



Нахождение последнего элемента списка дерева.

 Операция над данными закончена.

Вывод:

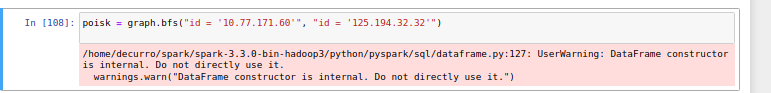
1. Найдена самая «жирная» вершина — sortResearch[0][0]

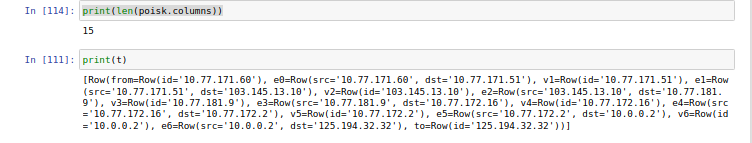
2. Найден последний элемент списка дерева самой «жирной» вершины.

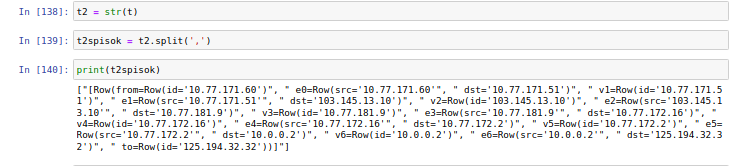
**3.3 Поиск пути**

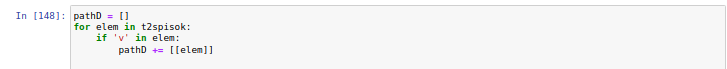
**Цель:** найти самый длинный путь с помощью вершины, где самое большое дерево, и последнего элемента этого дерева. Вывести вершины этого пути.

Через функцию bfs (поиск в ширину) из библиотеки GraphFrame, с помощью «жирной» вершины и последнего элемента списка дерева ищем путь (poisk).

Превращение полученного результата в список(t), и подсчет столбцов.

 Разбитие списка на элементы.

 Создание списка только вершин (pathD), участвующих в пути.

 Список вершин.

 Операции над данными закончены. Длинный путь найден.

Вывод:

1. Получен список вершин самого длинного пути

**3.4. Результат**

**Цель:** визуализировать полученный путь.

Визуализация полученного результата с помощью библиотеки networkx и matplotlib.

Для этого необходимо:

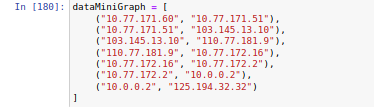
1. Создать направленный граф(grapRisDi2).

2. Самая «жирная» вершина → sortResearch[0][0]

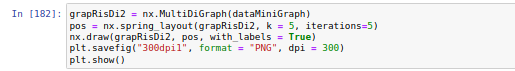
3. Вершины, полученные через функцию bfs (список pathD)

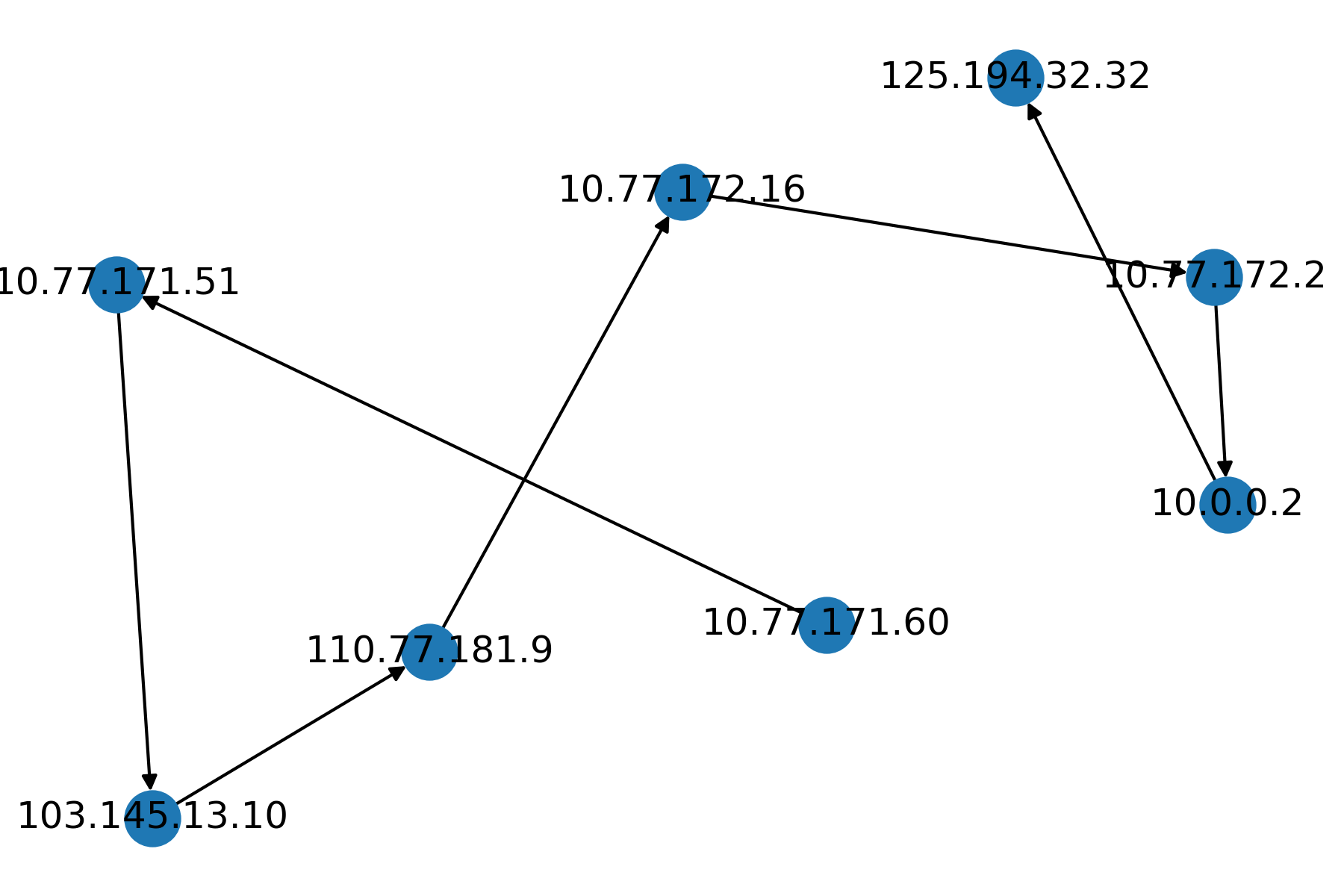
4. Конечная вершина из списка, полученная через функцию bfs\_tree → alls[14595][1]

Создание направленного графа.



Визуализация.





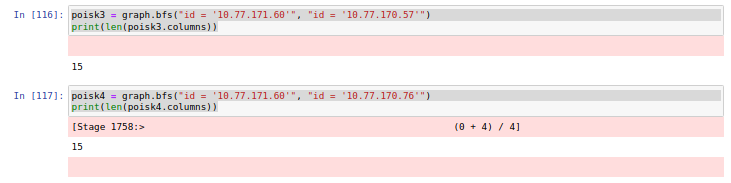
Вывод: результате получился путь с 6 вершинами (на считая начало и конец).

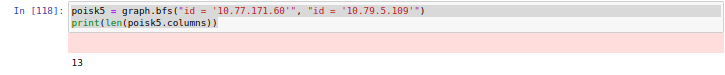
**4. Проверка верности результата**

**4.1. Проверка других элементов самой «жирной» вершины.**

**Цель:** Проверить крайние элементы дерева на наличие длинного пути, чтобы понять, является ли найденный самый длинный путь единственным, а так же возможность наличия пути длиннее.





В результате у этой «жирной» вершины есть ещё 2 длинных пути.

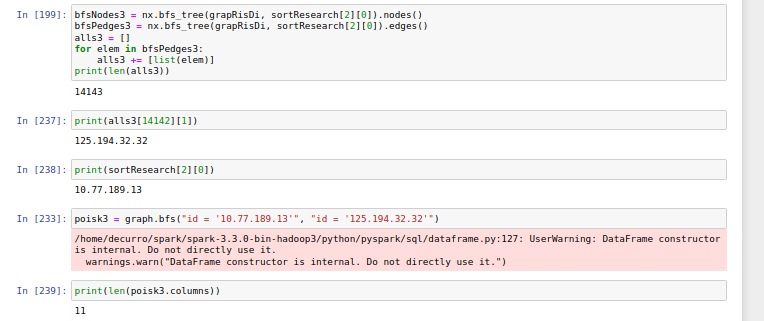
Вывод:

1. Самый длинный путь не один.

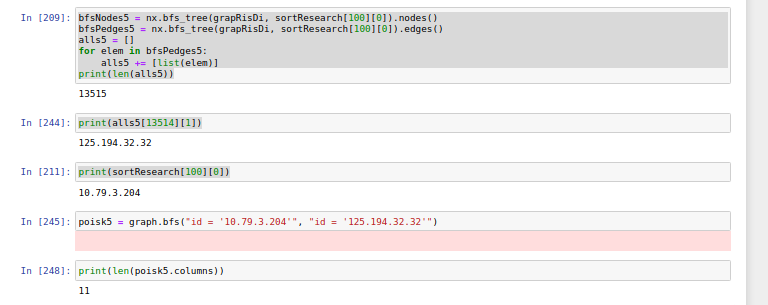
**4.2. Выборочная проверка пути у вершин с меньшим количеством связей.**

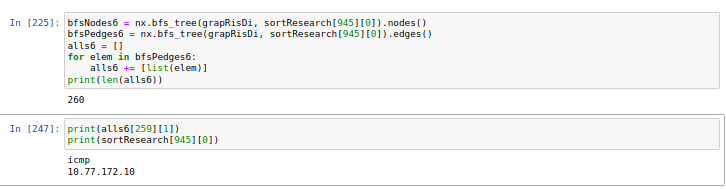
**Цель:** проверить наличие самого длинного пути у вершин, где дерево имеет меньшее количество связей, чем у самого большого.

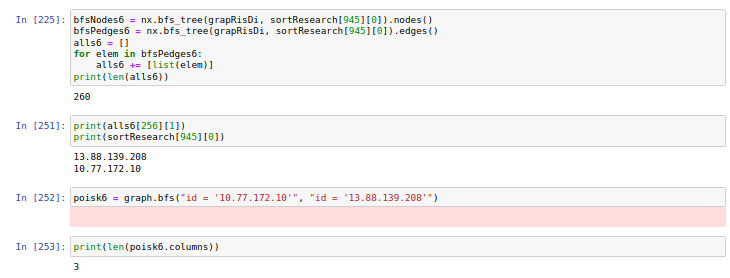












Результат проверки: путь, длиннее, чем у самой «жирной» вершины, не найден.

Вывод:

1. Длиннее путь не найден

**5. Результат проверки**

1. Самый длинный путь не изменился.

2. "Жирность" не гарантирует наличие самого длинного пути.

3. Несовершенство алгоритма отсева некорректных значений.

**6. Заключение**

В результате выполнения задания по поиску самого длинного пути, был получен путь длинною в 6 вершин (не считая начало и конец).

Путь визуализирован.

В результате проверки самый длинный путь не был оспорен.

Необходимо проработать алгоритм отсева некорректных значений