**Федеральное государственное образовательное**

**бюджетное учреждение**

**высшего образования**

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ**

**ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ»**

**(Финансовый университет)**

**Факультет**

**информационных технологий и анализа больших данных**

**Департамент «Бизнес-информатика»**

**Домашнее задание № 2**

«Решения задачи Коммивояжера при помощи алгоритма Дейкстры»

Студенты группы БИ20-8:

Луканина Полина

Аверкин Никита

Филимонова Арина

Совин Владимир

Горшков Георгий

Киселева Евгения

Руководитель:

Аксенов Дмитрий Андреевич

**Москва 2022**

### Оглавление

[1. Постановка задачи (физическая модель) 4](#_Toc98409993)

[2. Математическая модель 5](#_Toc98409994)

[2.1. Решение алгоритмом Дейкстры 11](#_Toc98410000)

[3. Алгоритмы 13](#_Toc98410001)

[3.1. Алгоритм решения задачи нахождения кратчайшего пути методом Дейкстры. 13](#_Toc98410002)

[3.1.1. Описание входных данных. 13](#_Toc98410003)

[3.1.2. Описание алгоритма решения 15](#_Toc98410004)

[3.1.3. Описание выходных данных 17](#_Toc98410005)

[4. Варианты использования системы 17](#_Toc98410006)

[4.1. ВИ 1 17](#_Toc98410007)

[4.2. ВИ 2 18](#_Toc98410008)

[4.3. ВИ 3 20](#_Toc98410009)

[4.4. ВИ 4 21](#_Toc98410010)

[5. Архитектура решения 23](#_Toc98410011)

[5.2. Функции считывания информации 23](#_Toc98410012)

[5.3. Функции обработки информации 25](#_Toc98410013)

[5.4. Функции вывода информации 31](#_Toc98410014)

[6. Тестирование 32](#_Toc98410015)

[6.1. Проверка №1 матрицы 3x3: 32](#_Toc98410016)

[6.1.1. Тест №1 кодом Python: 32](#_Toc98410017)

[6.1.2. Тест №1 онлайн-калькулятором: 33](#_Toc98410018)

[6.2. Проверка №2 матрицы 3х3: 34](#_Toc98410019)

[6.2.1. Тест №2 кодом Python: 34](#_Toc98410020)

[6.2.2. Тест №2 онлайн-калькулятором: 35](#_Toc98410021)

[6.3. Проверка №3 матрицы 3x3: 36](#_Toc98410022)

[6.3.1. Тест №3 кодом Python: 36](#_Toc98410023)

[6.3.2. Тест №3 онлайн-калькулятором: 37](#_Toc98410024)

[6.4. Проверка №4 матрицы 3x3: 38](#_Toc98410025)

[6.4.1. Тест №4 кодом Python: 38](#_Toc98410026)

[6.4.2. Тест №4 онлайн-калькулятором: 39](#_Toc98410027)

[6.5. Проверка №5 матрицы 3x3: 40](#_Toc98410028)

[6.5.1. Тест №5 кодом Python: 40](#_Toc98410029)

[6.5.2. Тест №5 онлайн-калькулятором: 41](#_Toc98410030)

[7. Заключение 42](#_Toc98410031)

# Постановка задачи (физическая модель)

Компания ООО «Аспера» занимается продажей обуви по всей России. По всей России располагается 6 магазинов данной сети: Москва, Санкт-Петербург, Тверь, Рязань, Нижний Новгород и Казань. Распределительный центр находится в Москве. Необходимо составить наиболее оптимальный маршрутный лист для водителя грузовика с условием того, чтобы он проехал наиболее выгодный по стоимости путь из города М в город Т.

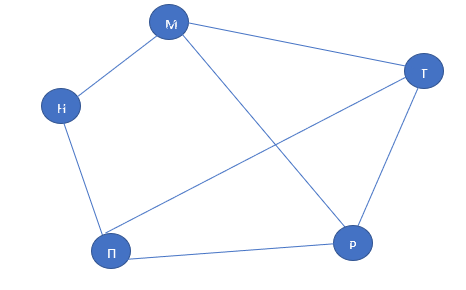
Заказчик так же предоставил нам стоимостную матрицу, выраженную в тыс. руб:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **М** | **П** | **Т** | **Р** | **Н** |
| **М** | - | - | 100 | 30 | 10 |
| **П** | - | - | 10 | 20 | 50 |
| **Т** | 100 | 10 | - | 60 | 30 |
| **Р** | 30 | 20 | 60 | - | - |
| **Н** | 10 | 50 | 30 | - | - |

Стоимостная матрица 1

Где М - Москва, П – Санкт-Петербург, Т – Тверь, Р- Рязань, Н – Нижний Новгород.

Так, построим следующий граф G (V, D), где v – вершины графа (пути/города), d – грани графа (стоимости маршрутов), описывающий количество различных путей между городами:



# Математическая модель

### 2.1. Решение алгоритмом Дейкстры

Алгоритм Дейкстры строит множество вершин S, для которых выгоднейшие пути от источника заведомо известны. При этом на каждом шаге к множеству S добавляется та из оставшихся вершин, стоимость пути до которой наименьшее. Кроме того, на каждом шаге алгоритм Дейкстры использует алгоритм D, в который записывает длинны выгоднейших особых путей для каждой вершины. Когда множество с будет содержать все вершины графа, то есть для всех вершин будут найдены особые пути, тогда массив d будет содержать длинны наивыгоднейших путей от источника каждой вершины. При этом D[v] = min(D[v], D[w] + C [w, v]), где v – предыдущая вершина, w – текущая вершина, C[w,v] – стоимость маршрута от v до w. Так, получим следующую таблицу:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | w | D[Н] | D[П] | D[Р] | D[Т] |
| {М} | М | **10** | - | 30 | 100 |
| {М, Н} | Н |  | 60 | **30** | 100 |
| {М, Н, Р} | Р |  | **50** |  | 90 |
| {М, Н, Р, П} | П |  |  |  | **60** |
| {М, Н, Р, П, Т} | Т | 10 | 50 | 30 | 60 |

Стоимостная матрица 2

Получим в последней строке таблицы суммы выгоднейших путей от точки М до любой другой точки, в частности получим оптимальный маршрут от М до Т, который равен 60 тыс. руб.

Помимо алгоритма Дейкстры, данную задачу решить генетическим алгоритмом. Для этого сгенерируем начальную популяцию (множество вариантов пути и их вес (стоимость маршрута). Далее выбираем случайным образом пару комбинаций и применяем к ним алгоритм скрещивания:

|  |  |
| --- | --- |
| Было (родители) | Стало (потомки) |
| М > П > Н > Р > Т | М > П > Р > П > Т |
| М > Н > Р > П > Т | М > Н > Р > Т > П |

Комбинации скрещивания 1

Затем выбираем еще одно случайную последовательность городов, и применяем к ним метод мутацию:

|  |  |
| --- | --- |
| Было (до мутации) | Стало (после мутации) |
| М > Н > П > Р > Т | М > Р > П > Н > Т |

Случайная последовательность городов 1

После всех проведенных алгоритмов добавляем получившиеся маршруту к основному списку, при этом заменяя их на те, где показатель стоимости наихудший. Повторяем этот цикл до тех пор, пока все новоиспеченные пути не вытиснут старые. В результате получим самый выгодный маршрут из М в Т, который будет равняться 60 тыс. руб.

# Алгоритмы

В данном блоке представлено программное решение технического задания. Для этого были разработаны различные алгоритмы, два алгоритма основываются на методе Дейкстры и один алгоритм на генетическом методе. Чтобы получить конкретный ответ, необходимо следовать следующим указаниям в описании к каждому алгоритму. Для начала следует запустить код в выбранной программе.

Обратите внимание, что очень важно вводить значения строго согласно требованиям в инструкции.

## 3.1. Алгоритм решения задачи нахождения кратчайшего пути методом Дейкстры.

### 3.1.1. Описание входных данных.

Вид входных данных зависит от способа, которым будут вводится данными. Данный алгоритм позволяет самим ввести данные, или ввести с помощью CSV файла.

Для ручного ввода входными данными являются:

* Количество узлов, название узлов графа, количество ребер графа, весовые коэффициенты, начальная и конечная точка отправления.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.Образец заполнения

* Количество узлов, весовая матрица, начальная и конечная точка отправления.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.Образец заполнения

* Количество узлов, количество ребер графа, начальная и конечная точка отправления.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3. Образец заполнения

Импорт входных данных из csv файла:

* Весовая матрица с названиями столбцов и строк.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 4. Образец заполнения csv файла

### 3.1.2. Описание алгоритма решения

После того как данные введены, программе необходимо преобразовать данные для дальнейшего использования.

Шаг 1: необходимо обеспечивается симметричность графа. Другими словами, если существует путь от узла A к B со значением V, должен быть путь от узла B к узлу A со значением V.

Шаг 2: необходимо вернуть узлы графа, соседей узла, а также значение ребра между двумя узлами.

Шаг 3: принимается весовая матрица и стартовый узел и непосредственно реализует алгоритм Дейкстры. То есть, записываются длинны выгоднейших особых путей для каждой вершины. Когда множество с будет содержать все вершины графа, то есть для всех вершин будут найдены особые пути, тогда массив будет содержать длинны наивыгоднейших путей от источника каждой вершины.

Шаг 4: необходимо посетить все узлы и найти кратчайший путь к каждому из них. Следует отметить, что он будет выполняться до тех пор, пока мы не посетим все узлы.

Шаг 5: необходимо инициализировать состояние объекта, то есть место, где мы определяем начальное и первичное состояние нашего объекта.

### 3.1.3. Описание выходных данных

В конце программа рассчитывает полученный путь и стоимость, а также выводит на основании наших данных весовую матрицу. Ответ будет представлен в формате:

Полученный путь: значение. Стоимость: значение.

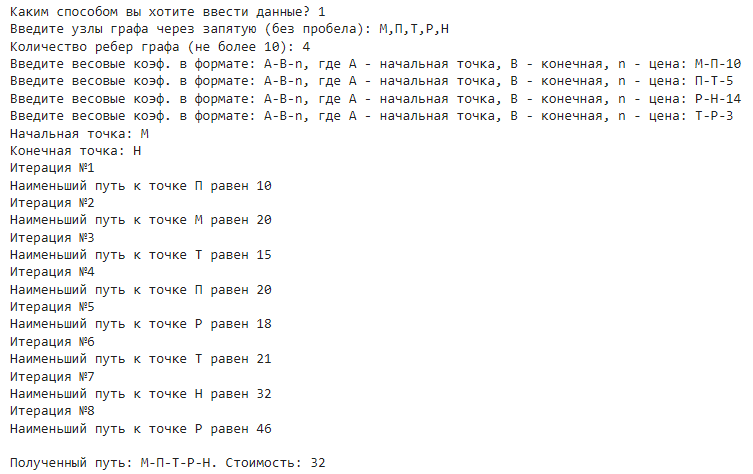
Полученный путь 1

# Варианты использования системы

В нашей системе есть 4 варианта варианты использования.

## 4.1. ВИ 1

Данный вариант использования включает в себя ручной ввод данных с клавиатуры. Для того, чтобы его активировать в графу «Каким способом вы хотите ввести значения?» надо ввести цифру «1».



Образец заполнения 1

В строчке «введите узлы графа» указываются наименования точек следования пути грузовика.

В строчке «кол-во ребер графа» указывается количество вариантов следования грузовика, но с ограничением 10. Ограничения связаны с количеством сочетаний узлов на графе (одному и тому же ребру не могут быть присвоены разные коэффициенты).

Далее необходимо поэтапно расписать весовые коэффициенты формате A-B-n 0, где A – первый конец ребра, B – второй конец ребра, n – весовой коэффициент.

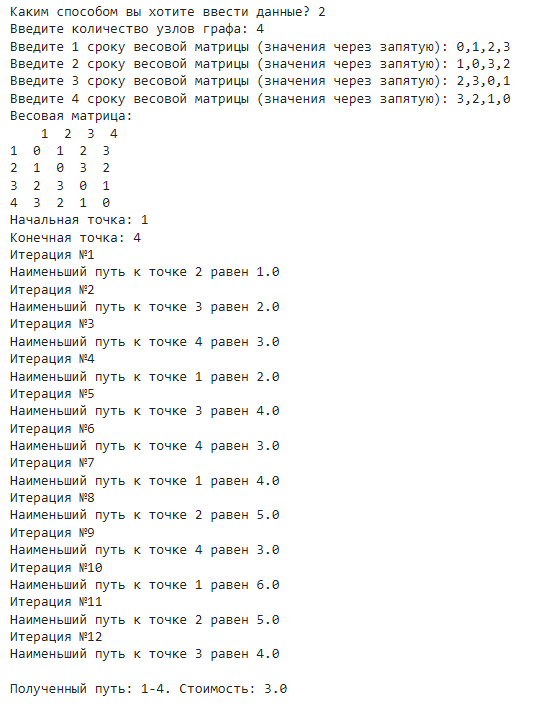
После этого вводим начальную точку (пункт отправления) и конечную (пункт прибытия).

После этого от пользователя требуется лишь нажатие клавиши «Enter» и на экране выведется оптимальное решение.

## ВИ 2

Данный вариант подразумевает под собой так же ручной ввод данных, но в формате матрицы.

Для того чтобы выбрать данный способ ввода информации необходимо в строке «Каким способом вы хотите ввести значения?» ввести «2».



ВИ2 1

Далее нужно ввести необходимое количество графов.

Далее последуют области для заполнения нужных данных, стоит учитывать, что в графу «введите n строку весовой матрицы» вводятся данные из матрицы обязательно построчно слева направо.

После этого от пользователя требуется лишь нажатие клавиши «Enter» и на экране выведется оптимальное решение.

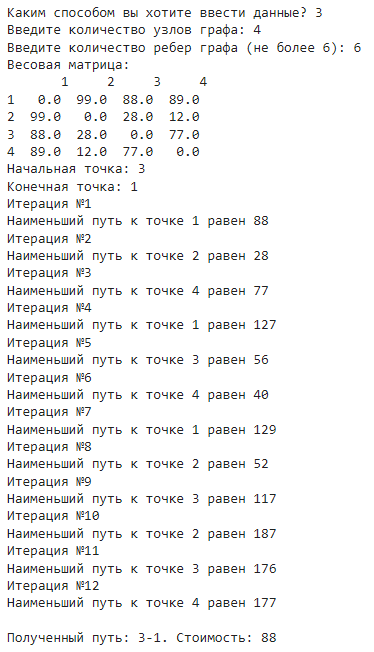
## ВИ 3

Этот способ позволяет сгенерировать случайную матрицу, для его выбора в первой строчке необходимо ввести «3».

После этого вводим нужно количество графов (точек, которые нужно проехать). Их имена определяются автоматически и соответствуют порядковому номеру.

Вводим количество ребер графа, максимальное число указано в скобках. Ограничения связаны с количеством сочетаний узлов на графе (одному и тому же ребру не могут быть присвоены разные коэффициенты). Сгенерированные значения выводятся на экран в виде весовой матрицы.

Вводим начальную и конечную точку маршрута и получаем результат.



ВИ3 1

## ВИ 4

Данный вариант использования включает в себя ввод данных с помощью файла csv. Для того, чтобы его активировать в графу «Каким способом вы хотите ввести значения?» надо ввести цифру «3».

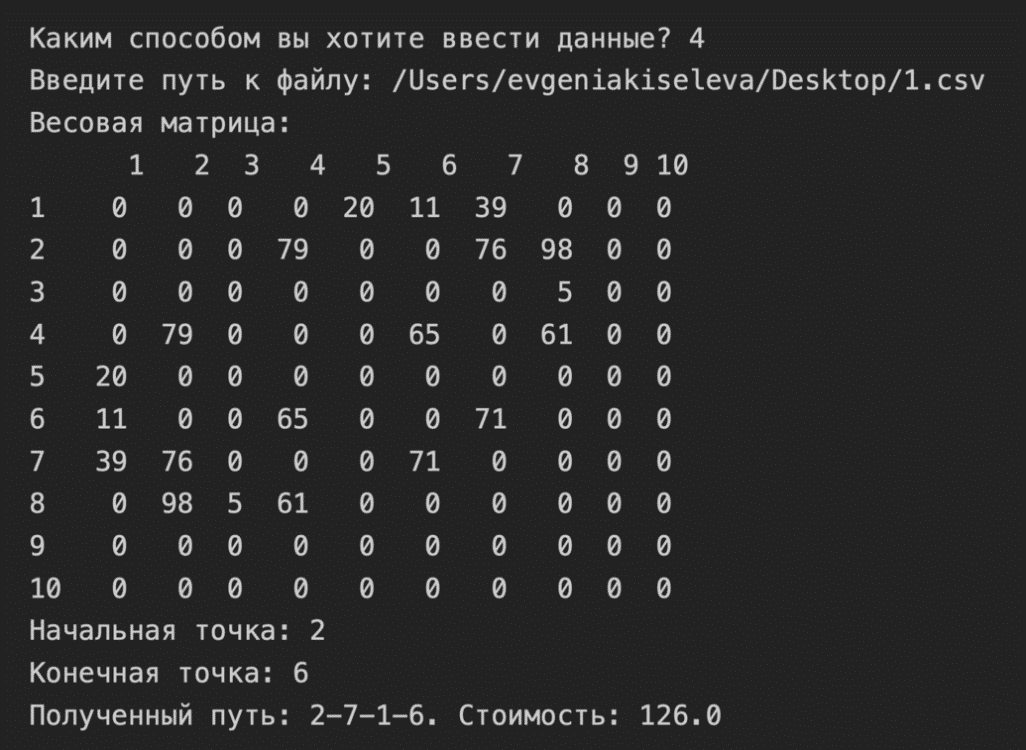
Появляется окно, в котором вводим путь к csv файлу. Например:



CSV 1

Вводим начальную и конечную точку.

Получаем результат. Пример:



ВИ3 2

# Архитектура решения

Для решения задачи использовались методы(функции), которые можно разделить на 3 принципиальных кода.

## Функции считывания информации

После запуска программы необходимо определиться каким способом будет происходить ввод данных: 1 – ручной ввод (формат указания пути – «A-B-n»); 2 – ручной ввод (в формате матрицы); 3 – случайные числа; 4 – файл csv.

Далее выберете описание ввода, который вы выбрали:

1. **Если введено «1»:**

1.1 После того, как вы выбрали этот тип ввода, программа попросит вас ввести количество узлов. Необходимо ввести название узлов графа через запятую. (Например: A, B, C, D, E)

* 1. Затем вводим количество ребер графа, максимальное число указано в скобках. Ограничения связаны с количеством сочетаний узлов на графе (одному и тому же ребру не могут быть присвоены разные коэффициенты). (Например: 4)
  2. Поэтапно вводим весовые коэффициенты. Например, A-B-10, где A – первый конец ребра, B – второй конец ребра, 10 – весовой коэффициент.

1.4 Вводим начальную точку (пункт отправления) и конечную (пункт прибытия). (Например: В)

**2. Если введено «2»:**

2.1 После успешного выбора типа ввода данных, вам необходимо ввести количество узлов графа. (Например: 3)

2.2 После этого, исходя из того какое у вас количество узлов, в программу необходимо ввести построчно матрицу, значения которой разделены запятой и без пробела , количество строк будет определено согласно количеству узлов, также как и количество значений в матрице.(Например, если вы ввели в предыдущей строке – 3 (т.е. количество узлов), то необходимо в каждую строку матрицы ввести три числа, к примеру, 1,0,3)

2.3 Затем нужно ввести начальную точку. (Например: А)

2.4 После этого введите конечную точку. (Например: С)

**3. Если введено «3»:**

3.1 После успешного выбора типа ввода данных, вам необходимо ввести количество узлов графа. (Например: 3). Обратите внимание, что имена при этом типе ввода будут выбраны автоматически и будут соответствовать порядковому номеру.

3.2 После этого вводим количество ребер графа, максимальное число указано в скобках. Ограничения связаны с количеством сочетаний узлов на графе (одному и тому же ребру не могут быть присвоены разные коэффициенты). Сгенерированные значения выводятся на экран в виде весовой матрицы.

3.3 Затем нужно ввести начальную точку. (Например: А)

3.4 После этого введите конечную точку. (Например: С)

**4. Если введено «4»:**

4.1 После успешного выбора типа ввода данных, вам необходимо указать путь к файлу csv. (Например: /Users/204299/Desktop/1.csv)

4.2 Затем нужно ввести начальную точку. (Например: А)

4.3 После этого введите конечную точку. (Например: С)

Входные параметры:

* нет входных параметров

Выходные параметры:

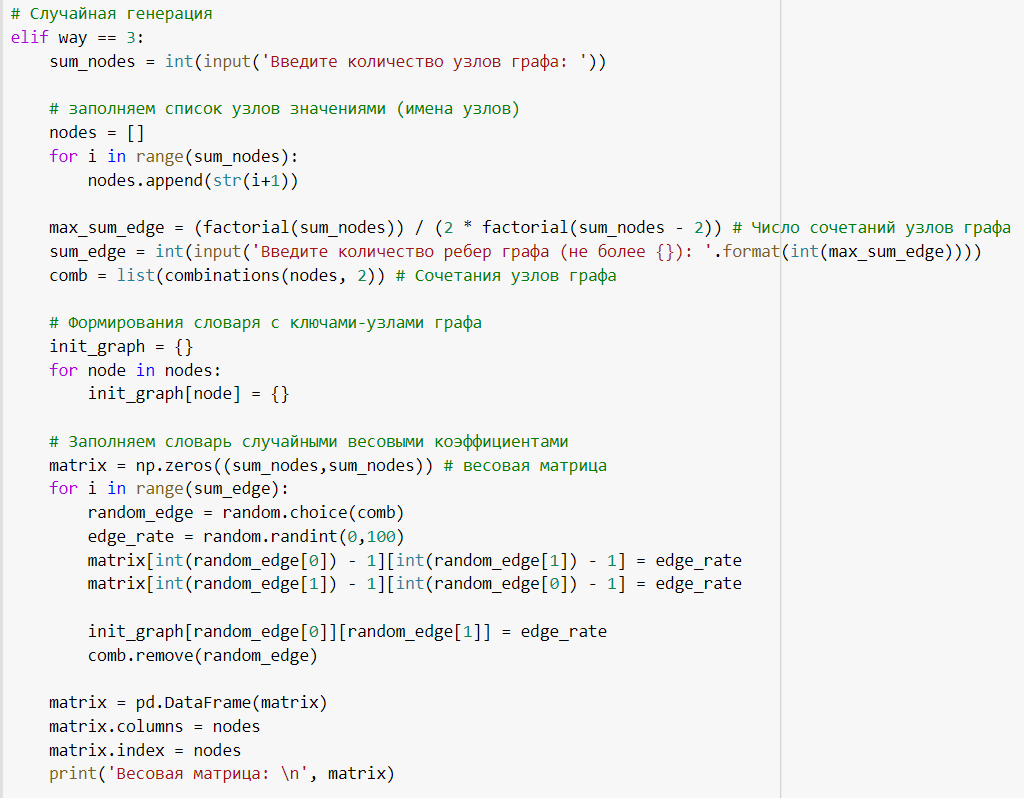
* nodes – список с узлами графа (тип данных: list);
* init\_graph – словарь с весовыми коэффициентами (тип данных: dict);

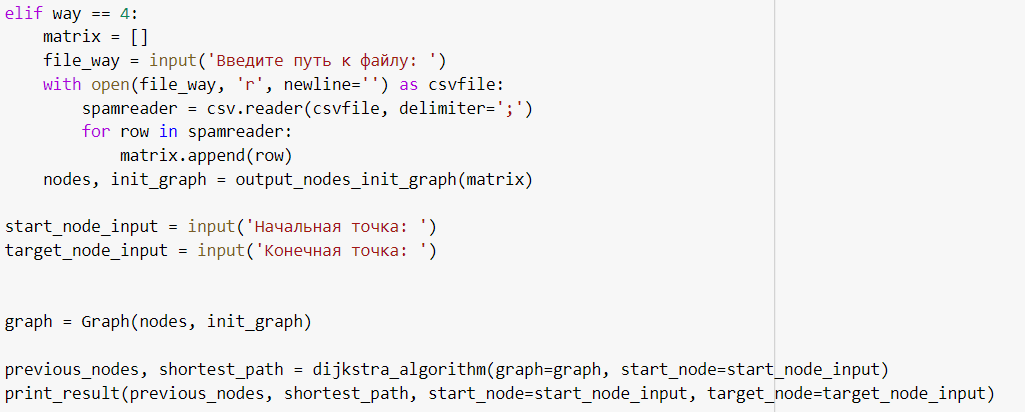
Переменные, затрагиваемые в ходе работы:

* way – значение выбранного типа ввода (тип данных: int);
* node – узел графа (тип данных: int);
* max\_sum\_edge - максимальное количество ребер графа (тип данных: int);
* sum\_edge - количество ребер графа (тип данных: int);
* edge – весовые коэффициенты (тип данных: text);
* nodes – список с узлами графа (тип данных: list);
* init\_graph – словарь с весовыми коэффициентами (тип данных: dict);
* sum\_rows – количество узлов графа (тип данных: int);
* matrix – значения матрицы (тип данных: list);
* input\_rows – значения весовой матрицы (тип данных: int);
* comb- сочетание узлов графа (тип данных: list);
* random\_edge – случайное значение узла (тип данных: int);
* edge\_rate - весовой коэффициент (тип данных: int);

Часть кода, отвечающая за считывание информации:







Код в питоне 1

## Функции обработки информации

После того, как вы введете все необходимые данные, программа их получит и начнет первичную обработку.

В случае ручного ввода в формате указания пути с помощью метода split, введенные данные через запятую будут разделены. Это необходимо для формирования словаря. Если говорить о ручном вводе в формате матрицы, то данные о значениях весовой матрицы будут внесены в пустой список matrix.

В случае случайной генерации, сначала будет заполнен список узлов значениями, а затем будет сформирован словарь с ключами-узлами графа, затем будет заполнен словарь случайными весовыми коэффициентами.

В случае ввода данных с помощью файла, программа его открывает и читает с помощью функции open и метода csv. reader и составляет матрицу, в которую помещаются построчные значения файла csv.

После всех необходимых преобразований с данными начинает работать класс «Graph» с различными функциями.

С помощью def construct\_graph, обеспечивается симметричность графа. Другими словами, если существует путь от узла A к B со значением V, должен быть путь от узла B к узлу A со значением V.

С помощью def get\_nodes, возвращаются узлы графа, функция def get\_outgoing\_edges возвращаются соседи узла, а функция def value возвращается значение ребра между двумя узлами.

Затем реализуется функция def dijkstra\_algorithm, находящаяся отдельно от класса. Она принимает в себя весовую матрицу и стартовый узел и непосредственно реализует алгоритм Дейкстры.

В этой функции есть два словаря: shortest\_path – для экономии посещении каждого узла и обновления его по мере продвижения по графику; previous\_nodes - для сохранения кратчайшего известного пути к найденному узлу.

Затем в этой функции выполняется алгоритм while unvisited\_nodes, который посещает все узлы и находит кратчайший путь к каждому из них. Следует отметить, что он будет выполняться до тех пор, пока мы не посетим все узлы. Более того, внутри этого алгоритма с помощью print ('Итерация № {} \nНаименьший путь к точке {} равен {} ’. format (k, neighbor, tentative\_value)), реализуется визуализация.

Одной из важных частей кода является класс Graph. В него входит:

Функция def \_\_init\_\_:

Что делает: инициализирует состояние объекта, то есть место, где мы определяем начальное и первичное состояние нашего объекта.

Входные параметры:

* Nodes - список с узлами графа (тип данных: list);
* init\_graph - словарь с весовыми коэффициентами (тип данных: dict);

Выходные параметры:

* Nodes - список с узлами графа (тип данных: list);
* init\_graph - словарь с весовыми коэффициентами (тип данных: dict);

Переменные, затрагиваемые в ходе работы:

* Nodes - список с узлами графа (тип данных: list);
* init\_graph - словарь с весовыми коэффициентами (тип данных: dict);

Функция def construct\_graph:

Что делает: Этот метод обеспечивает симметричность графика

Входные параметры:

* Nodes - список с узлами графа (тип данных: list);
* init\_graph - словарь с весовыми коэффициентами (тип данных: dict);

Выходные параметры:

* graph – словарь, содержащий значения графа;

Переменные, затрагиваемые в ходе работы:

* node - узел графа (тип данных: int);
* edges - – весовые коэффициенты (тип данных: text);

Функция def get\_nodes:

Что делает: возвращает узлы графа

Входные параметры:

* нет входных параметров

Выходные параметры:

* nodes - список с узлами графа (тип данных: list);

Переменные, затрагиваемые в ходе работы:

Функция def get\_outgoing\_edges:

Что делает: возвращает соседей узла

Входные параметры:

* node - узел графа (тип данных: int);

Выходные параметры:

* connections – список соседей узла (тип данных: list);

Переменные, затрагиваемые в ходе работы:

* connections – список соседей узла (тип данных: list);
* node - узел графа (тип данных: int);

Функция def value:

Что делает: возвращает ребра между двумя узлами

Входные параметры:

* node1, node2 - узел графа (тип данных: int);

Выходные параметры:

* node1, node2 - узел графа (тип данных: int);

Переменные, затрагиваемые в ходе работы:

* node1, node2 - узел графа (тип данных: int);

Основной функцией, вычисляющей по методу Дейкстры, является функция def dijkstra\_algorithm:

Входные данные:

* graph – словарь, содержащий значения графа;
* start\_node - узел графа (тип данных: int);

Выходные данные:

* shortest\_path – словарь, для сохранения посещения каждого узла (тип данных: list);
* previous\_nodes – словарь, чтобы сохранить известный кратчайший путь к найденному узлу (тип данных: list);

Переменные, затрагиваемые в ходе работы:

* shortest\_path – словарь, для сохранения посещения каждого узла (тип данных: list);
* previous\_nodes – словарь, чтобы сохранить известный кратчайший путь к найденному узлу (тип данных: list);
* max\_value – для инициализации значения бесконечности не посещённых узлов (тип данных: int);
* node - узел графа (тип данных: int);

Функция def print\_result:

Что делает: выводит результат

Входные данные:

* shortest\_path – словарь, для сохранения посещения каждого узла (тип данных: list);
* previous\_nodes – словарь, чтобы сохранить известный кратчайший путь к найденному узлу (тип данных: list);

Выходные данные:

* shortest\_path – словарь, для сохранения посещения каждого узла (тип данных: list);

Переменные, затрагиваемые в ходе работы:

* node - узел графа (тип данных: int);
* shortest\_path – словарь, для сохранения посещения каждого узла (тип данных: list);
* previous\_nodes – словарь, чтобы сохранить известный кратчайший путь к найденному узлу (тип данных: list);

Функция def output\_nodes\_init\_graph:

Что делает: заполняет словарь с весовыми коэффициентами

Входные данные:

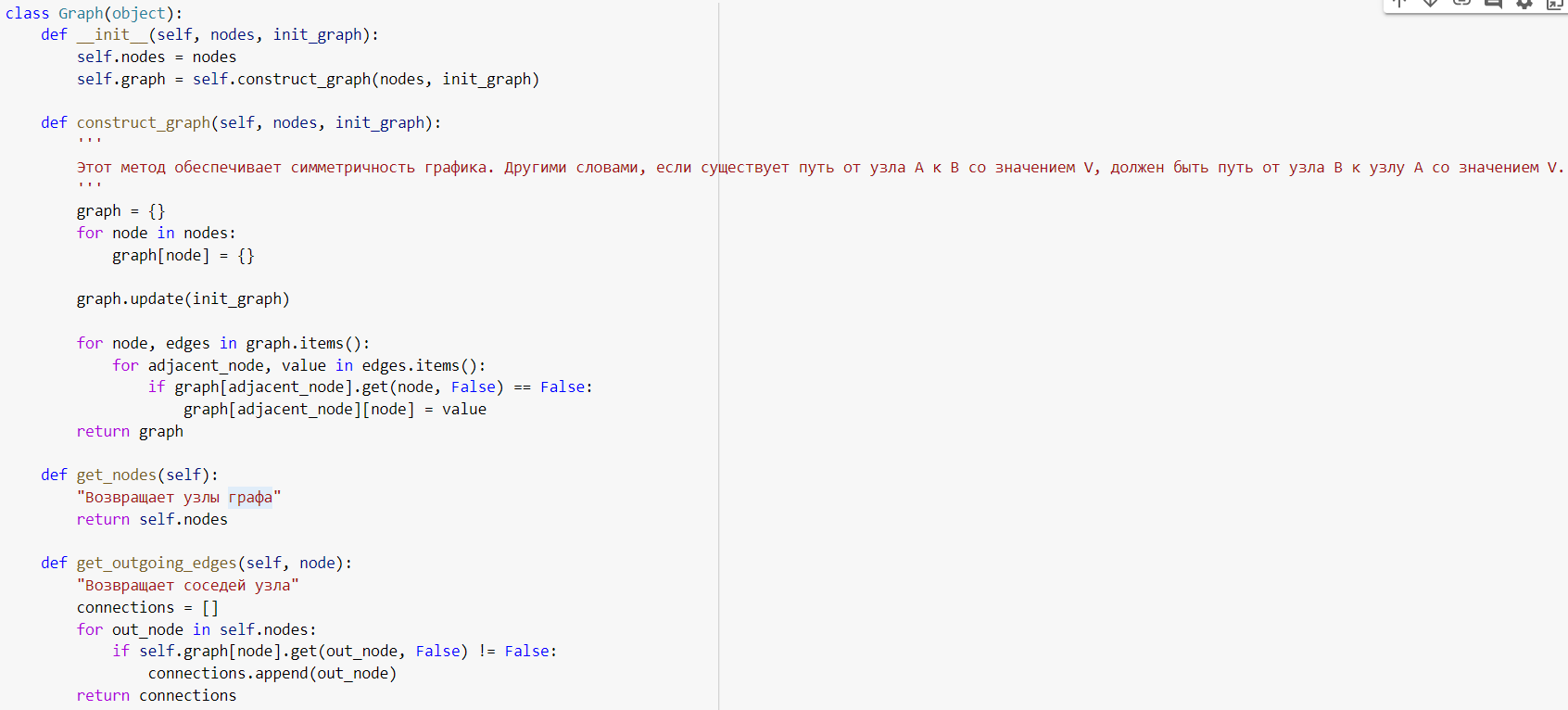
* matrix – значения матрицы (тип данных: list);

Выходные данные:

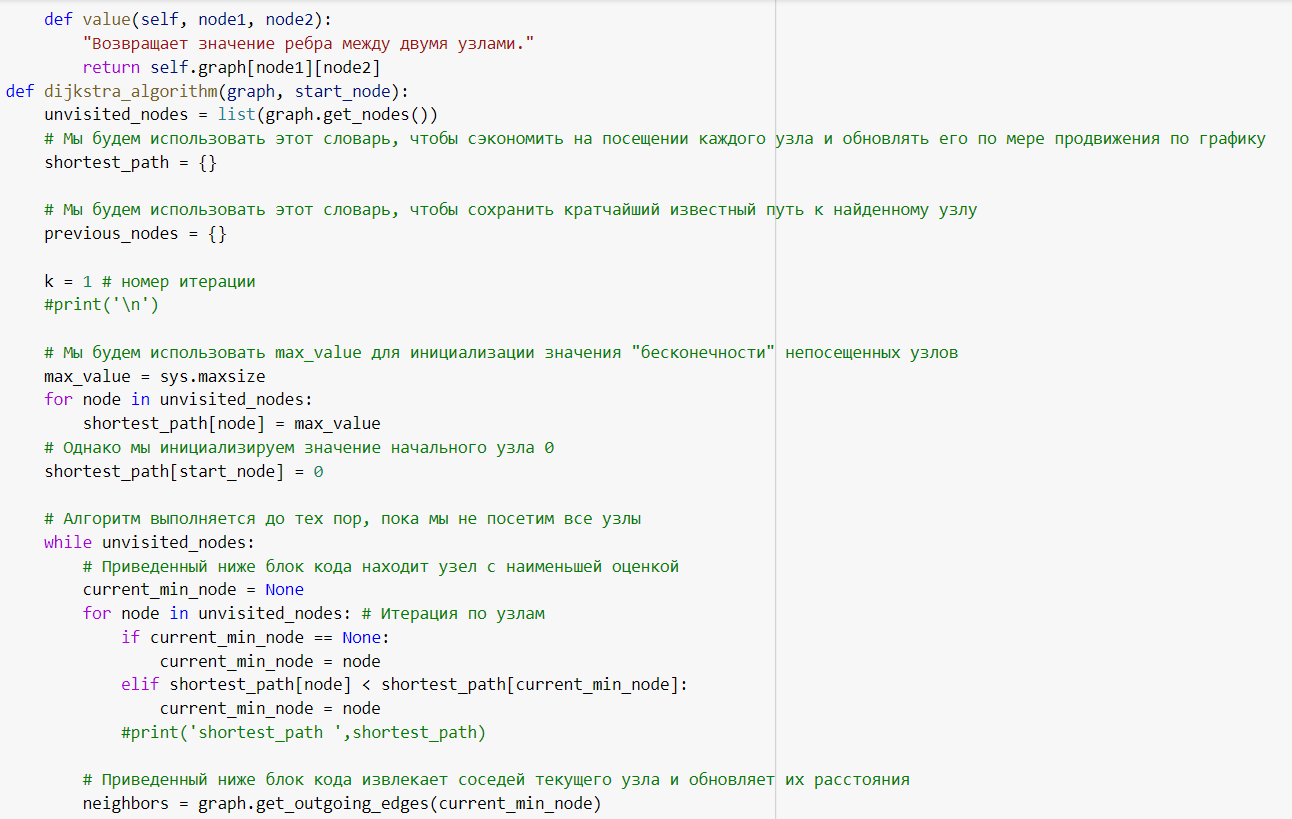
* node - узел графа (тип данных: int);
* init\_graph – словарь с весовыми коэффициентами (тип данных: dict);

Переменные, затрагиваемые в ходе работы:

* node - узел графа (тип данных: int);
* init\_graph – словарь с весовыми коэффициентами (тип данных: dict);
* Nodes - список с узлами графа (тип данных: list);
* matrix – значения матрицы (тип данных: list);

Фрагмент кода, отвечающая за обработку информации:

Код в питоне 2





Код в питоне 3



Код в питоне 4

## Функции вывода информации

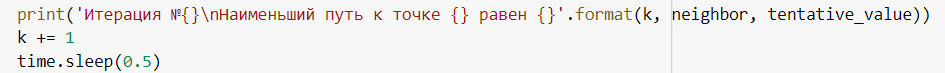
Метод вывода информации (он заключен внутри каждой функции)

Что делает: осуществляет вывод необходимой информации

Вывод информации осуществляется с помощью функции print ()  
Затрагиваемые переменные:

* Nodes - список с узлами графа (тип данных: list);
* init\_graph - словарь с весовыми коэффициентами (тип данных: dict);
* graph – словарь, содержащий значения графа;
* connections – список соседей узла (тип данных: list);
* shortest\_path – словарь, для сохранения посещения каждого узла (тип данных: list);
* previous\_nodes – словарь, чтобы сохранить известный кратчайший путь к найденному узлу (тип данных: list);
* matrix – значения матрицы (тип данных: list);

Также имеется визуализация, которая выводится через print:



Фрагмент, отвечающий за визуализацию 1

# Тестирование

Проведём тестирование нашей программы и сравним полученные показатели, чтобы сделать вывод о предпочтительном варианте использования программы под условия заказчика.

## 6.1. Проверка №1 матрицы 3x3:

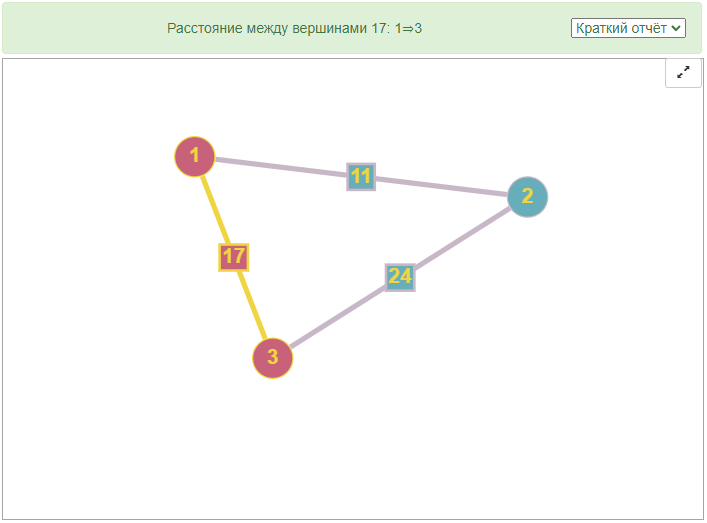
### 6.1.1. Тест №1 кодом Python:

Изображение выглядит как текст

Автоматически созданное описание

Выбираем метод ввода информации. В этом случае используем матричный тип и прописываем значения в необходимые поля. Программа ищет наикротчайший путь между заданными точками, перебирая все возможные вариации. Интеграции служат визуализацией данного перебора значений, показывая перемещение из начальной точки в следующую, из следующей в дальнейшую и так пока не насидеться самый кратчайший путь.

### 6.1.2. Тест №1 онлайн-калькулятором:



Онлайн-калькулятор 1

Строим оптимальный путь в графе на основе выбранного дата сета, а затем запускаем калькулятор.

6.2. Проверка №2 матрицы 3х3:

### 6.2.1. Тест №2 кодом Python:

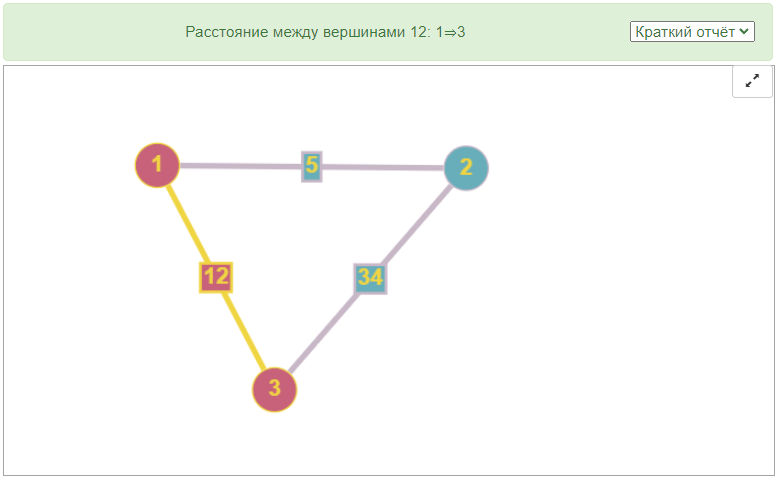
Изображение выглядит как текст

Автоматически созданное описание

Ответ в питоне 1

Выбираем метод ввода информации. В этом случае используем матричный тип и прописываем значения в необходимые поля. Программа ищет наикротчайший путь между заданными точками, перебирая все возможные вариации. Интеграции служат визуализацией данного перебора значений, показывая перемещение из начальной точки в следующую, из следующей в дальнейшую и так пока не насидеться самый кратчайший путь.

### 6.2.2. Тест №2 онлайн-калькулятором:



Онлайн-калькулятор 2

Строим оптимальный путь в графе на основе выбранного дата сета, а затем запускаем калькулятор.

## 6.3. Проверка №3 матрицы 3x3:

### 6.3.1. Тест №3 кодом Python:

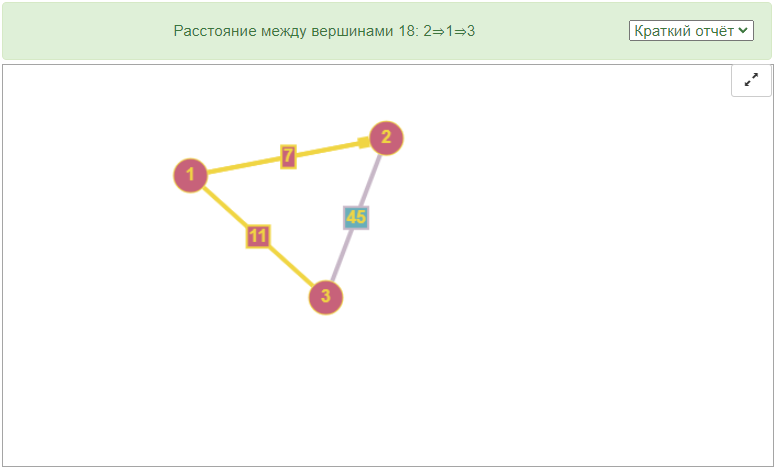
Изображение выглядит как стол

Автоматически созданное описание

Ответ в питоне 2

Выбираем метод ввода информации. В этом случае используем матричный тип и прописываем значения в необходимые поля. Программа ищет наикротчайший путь между заданными точками, перебирая все возможные вариации. Интеграции служат визуализацией данного перебора значений, показывая перемещение из начальной точки в следующую, из следующей в дальнейшую и так пока не насидеться самый кратчайший путь.

### 6.3.2. Тест №3 онлайн-калькулятором:



Онлайн-калькулятор 3

Строим оптимальный путь в графе на основе выбранного дата сета, а затем запускаем калькулятор.

## 6.4. Проверка №4 матрицы 3x3:

### 6.4.1. Тест №4 кодом Python:

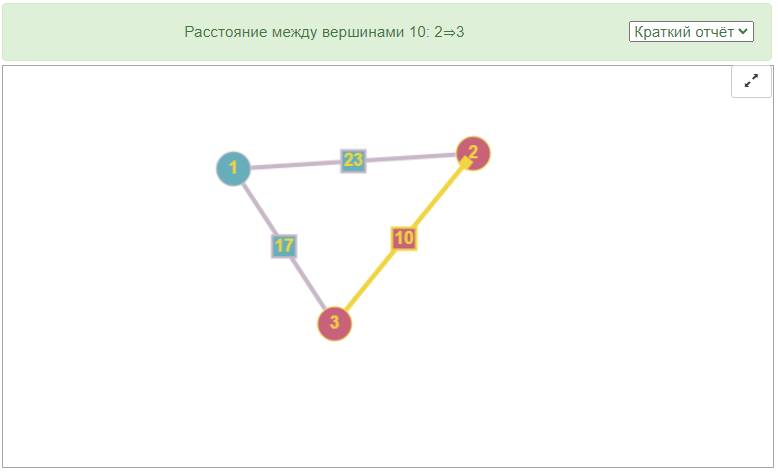
Изображение выглядит как текст

Автоматически созданное описание

Ответ в питоне 3

Выбираем метод ввода информации. В этом случае используем матричный тип и прописываем значения в необходимые поля. Программа ищет наикротчайший путь между заданными точками, перебирая все возможные вариации. Интеграции служат визуализацией данного перебора значений, показывая перемещение из начальной точки в следующую, из следующей в дальнейшую и так пока не насидеться самый кратчайший путь.

### 6.4.2. Тест №4 онлайн-калькулятором:



Онлайн-калькулятор 4

Строим оптимальный путь в графе на основе выбранного дата сета, а затем запускаем калькулятор.

## 6.5. Проверка №5 матрицы 3x3:

### 6.5.1. Тест №5 кодом Python:

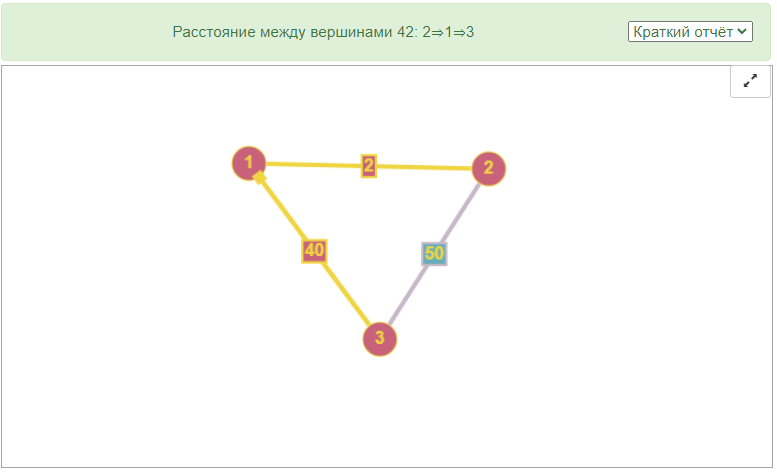
Изображение выглядит как текст

Автоматически созданное описание

Ответ в питоне 4

Выбираем метод ввода информации. В этом случае используем матричный тип и прописываем значения в необходимые поля. Программа ищет наикротчайший путь между заданными точками, перебирая все возможные вариации. Интеграции служат визуализацией данного перебора значений, показывая перемещение из начальной точки в следующую, из следующей в дальнейшую и так пока не насидеться самый кратчайший путь.

### 6.5.2. Тест №5 онлайн-калькулятором:



Онлайн-калькулятор 5

Строим оптимальный путь в графе на основе выбранного дата сета, а затем запускаем калькулятор.

Для подсчёта времени на выполнение алгоритмов будет использовано время, затраченное на ввод необходимых данных в предназначенные поля и расчёт данных самой программой с дальнейшим выводом их пользователю на экран.

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Python** | **Онлайн-калькулятор** |
| Тестирование №1 | ~20 секунд | ~30 секунд |
| Тестирование №2 | ~18 секунд | ~27 секунд |
| Тестирование №3 | ~17 секунд | ~25 секунд |
| Тестирование №4 | ~17 секунд | ~24 секунды |
| Тестирование №5 | ~15 секунд | ~21 секунда |

Затраченное время 1

Решения на рисунках под названиями «Онлайн-калькулятор 1, 2, 3, 4, 5» получены с помощью онлайн-калькулятора: <https://graphonline.ru/>

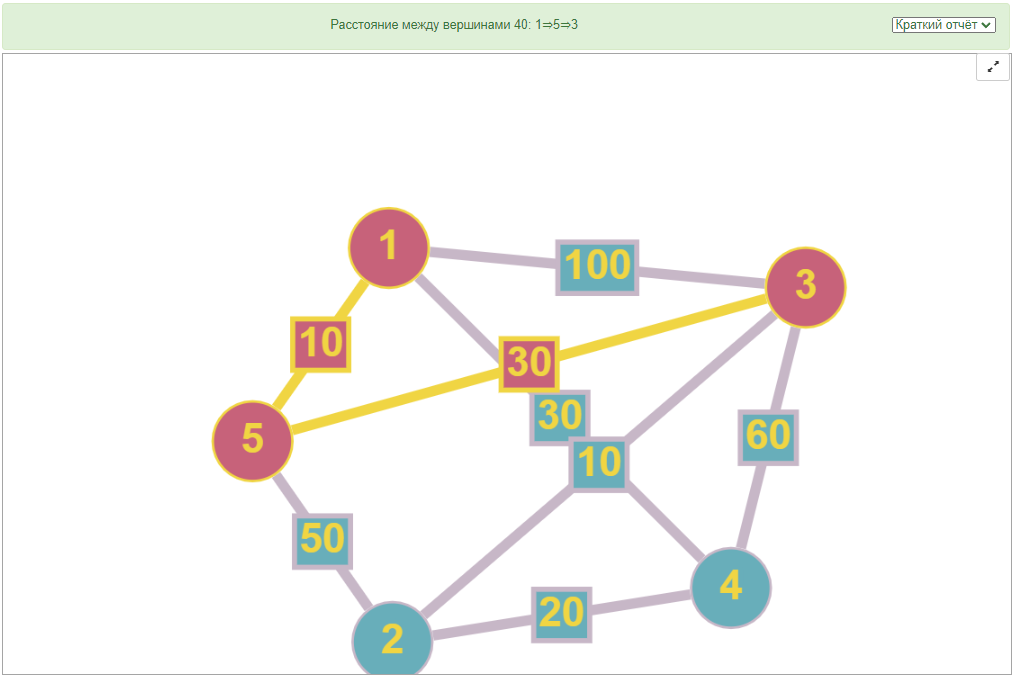
# Заключение

Наш представленный код решает поставленную задачу. На основании тестирования данного алгоритма можно сделать вывод о том, что Python выводит самое оптимальное решение достаточно быстро. Ниже представлено решение поставленной задачи через Python и проверка в онлайн-калькуляторе:

Изображение выглядит как стол

Автоматически созданное описание

Ответ в питоне 5



Онлайн-калькулятор 6

Теперь сравним два алгоритма по критериям: эффективности, скорости использования алгоритма, простоты использования, надёжности в разрезе человеческого фактора и точности предоставляемого решения.

|  |  |  |
| --- | --- | --- |
| **Критерий** | **Python** | **Онлайн-калькулятор** |
| Эффективность | Высокая | Высокая |
| Скорость использования алгоритма | Высокая | Средняя |
| Простота использования | Высокая | Средняя |
| Надёжность (человеческий фактор) | Высокая | Средняя |
| Точность | Высокая | Высокая |

Критерии сравнения 1

Мы считаем, что представленный метод – “Python” лучше, потому что он удобнее, быстрее и проще, а ещё имеет импорт .csv файлов и случайный ввод данных. Написанный код может помочь сэкономить затраты заказчику на транспортировку. Улучшением кода, к примеру, может послужить добавление визуализации и время выполнения запроса. Представленный метод «Python» можно интерпретировать для просчёта позиционных игр и лабиринта.