**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
2. Математическая модель 5
2.1. Решение алгоритмом Дейкстры 5
3. Алгоритмы 7
3.1. Алгоритм решения задачи нахождения кратчайшего пути методом Дейкстры. 7
3.1.1. Описание входных данных. 7
3.1.2. Описание алгоритма решения 7
3.1.3. Описание выходных данных 10
4. Варианты использования системы 10
4.1. ВИ 1 10
4.2. ВИ 2 11
4.3. ВИ 3 13
4.4. ВИ 4 14
5. Архитектура решения 16
5.2. Функции считывания информации 16
5.3. Функции обработки информации 20
5.4. Функции вывода информации 27
6. Тестирование 29
6.1. Проверка №1 матрицы 3x3: 29
6.1.1. Тест №1 кодом Python: 29
6.1.2. Тест №1 онлайн-калькулятором: 30
6.2. Проверка №2 матрицы 3x3: 30

6.2.1. Тест №2 кодом Python:.....31
6.2.2. Тест №2 онлайн-калькулятором:.....32
6.3. Проверка №3 матрицы 3x3:.....32
6.3.1. Тест №3 кодом Python:.....33
6.3.2. Тест №3 онлайн-калькулятором:.....34
6.4. Проверка №4 матрицы 3x3:.....35
6.4.1. Тест №4 кодом Python:.....35
6.4.2. Тест №4 онлайн-калькулятором:.....36
6.5. Проверка №5 матрицы 3x3:.....37
6.5.1. Тест №5 кодом Python:.....37
6.5.2. Тест №5 онлайн-калькулятором:.....38
7. Заключение.....39

1. 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.

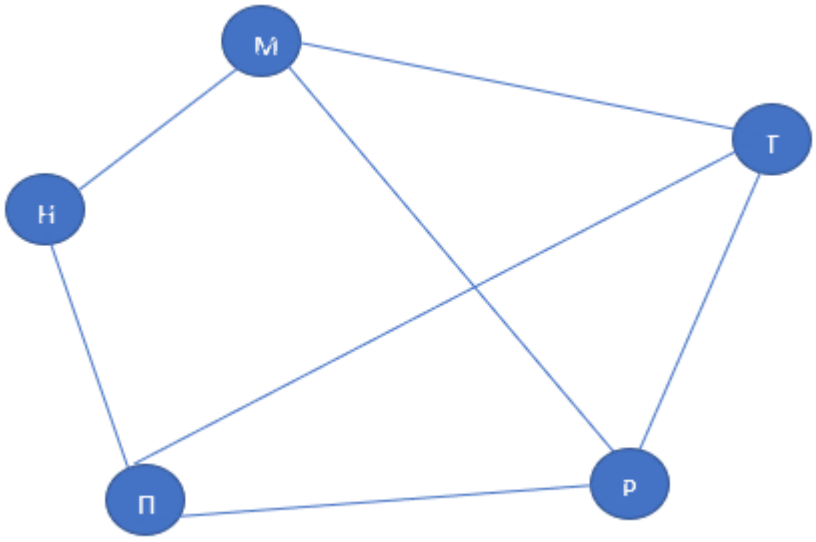
Заказчик так же предоставил нам стоимостную матрицу, выраженную в тыс. руб:

	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:



2. 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.

3. 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.

```
Каким способом вы хотите ввести данные? 1
Введите узлы графа через запятую (без пробела): М,П,Т,Р,Н
Количество ребер графа (не более 10): 4
Введите весовые коэф. в формате: A-B-n, где A - начальная точка, B - конечная, n - цена: М-П-10
Введите весовые коэф. в формате: A-B-n, где A - начальная точка, B - конечная, n - цена: П-Т-5
Введите весовые коэф. в формате: A-B-n, где A - начальная точка, B - конечная, n - цена: Р-Н-14
Введите весовые коэф. в формате: A-B-n, где A - начальная точка, B - конечная, n - цена: Т-Р-3
Начальная точка: М
Конечная точка: Н
```

Рисунок 1.Образец заполнения

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

```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 4
Введите 1 сроку весовой матрицы (значения через запятую): 0,1,2,3
Введите 2 сроку весовой матрицы (значения через запятую): 1,0,3,2
Введите 3 сроку весовой матрицы (значения через запятую): 2,3,0,1
Введите 4 сроку весовой матрицы (значения через запятую): 3,2,1,0
Весовая матрица:
    1  2  3  4
1  0  1  2  3
2  1  0  3  2
3  2  3  0  1
4  3  2  1  0
Начальная точка: 1
Конечная точка: 4
```

Рисунок 2.Образец заполнения

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

```
Каким способом вы хотите ввести данные? 3
Введите количество узлов графа: 4
Введите количество ребер графа (не более 6): 6
Весовая матрица:
       1     2     3     4
1   0.0  99.0  88.0  89.0
2  99.0   0.0  28.0  12.0
3  88.0  28.0   0.0  77.0
4  89.0  12.0  77.0   0.0
Начальная точка: 3
Конечная точка: 1
```

Рисунок 3. Образец заполнения

Импорт входных данных из csv файла:

- Весовая матрица с названиями столбцов и строк.

	A	B	C	D	E	F	G	H	I	J
1	0	0	0	0	20	11	39	0	0	0
2	0	0	0	79	0	0	76	98	0	0
3	0	0	0	0	0	0	0	5	0	0
4	0	79	0	0	0	65	0	61	0	0
5	20	0	0	0	0	0	0	0	0	0
6	11	0	0	65	0	0	71	0	0	0
7	39	76	0	0	0	71	0	0	0	0
8	0	98	5	61	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

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: М-П-Т-Р-Н. Стоимость: 32
```

Полученный путь 1

4. 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».

```
Каким способом вы хотите ввести данные? 1
Введите узлы графа через запятую (без пробела): М,П,Т,Р,Н
Количество ребер графа (не более 10): 4
Введите весовые коэф. в формате: А-В-n, где А - начальная точка, В - конечная, n - цена: М-П-10
Введите весовые коэф. в формате: А-В-n, где А - начальная точка, В - конечная, n - цена: П-Т-5
Введите весовые коэф. в формате: А-В-n, где А - начальная точка, В - конечная, n - цена: Р-Н-14
Введите весовые коэф. в формате: А-В-n, где А - начальная точка, В - конечная, n - цена: Т-Р-3
Начальная точка: М
Конечная точка: Н
Итерация №1
Наименьший путь к точке П равен 10
Итерация №2
Наименьший путь к точке М равен 20
Итерация №3
Наименьший путь к точке Т равен 15
Итерация №4
Наименьший путь к точке П равен 20
Итерация №5
Наименьший путь к точке Р равен 18
Итерация №6
Наименьший путь к точке Т равен 21
Итерация №7
Наименьший путь к точке Н равен 32
Итерация №8
Наименьший путь к точке Р равен 46

Полученный путь: М-П-Т-Р-Н. Стоимость: 32
```

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.

4.2. 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».

```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 4
Введите 1 сроку весовой матрицы (значения через запятую): 0,1,2,3
Введите 2 сроку весовой матрицы (значения через запятую): 1,0,3,2
Введите 3 сроку весовой матрицы (значения через запятую): 2,3,0,1
Введите 4 сроку весовой матрицы (значения через запятую): 3,2,1,0
Весовая матрица:
    1  2  3  4
1  0  1  2  3
2  1  0  3  2
3  2  3  0  1
4  3  2  1  0
Начальная точка: 1
Конечная точка: 4
Итерация №1
Наименьший путь к точке 2 равен 1.0
Итерация №2
Наименьший путь к точке 3 равен 2.0
Итерация №3
Наименьший путь к точке 4 равен 3.0
Итерация №4
Наименьший путь к точке 1 равен 2.0
Итерация №5
Наименьший путь к точке 3 равен 4.0
Итерация №6
Наименьший путь к точке 4 равен 3.0
Итерация №7
Наименьший путь к точке 1 равен 4.0
Итерация №8
Наименьший путь к точке 2 равен 5.0
Итерация №9
Наименьший путь к точке 4 равен 3.0
Итерация №10
Наименьший путь к точке 1 равен 6.0
Итерация №11
Наименьший путь к точке 2 равен 5.0
Итерация №12
Наименьший путь к точке 3 равен 4.0

Полученный путь: 1-4. Стоимость: 3.0
```

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.

4.3. 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.

```
Каким способом вы хотите ввести данные? 3
Введите количество узлов графа: 4
Введите количество ребер графа (не более 6): 6
Весовая матрица:
       1     2     3     4
1   0.0  99.0  88.0  89.0
2  99.0   0.0  28.0  12.0
3  88.0  28.0   0.0  77.0
4  89.0  12.0  77.0   0.0
Начальная точка: 3
Конечная точка: 1
Итерация №1
Наименьший путь к точке 1 равен 88
Итерация №2
Наименьший путь к точке 2 равен 28
Итерация №3
Наименьший путь к точке 4 равен 77
Итерация №4
Наименьший путь к точке 1 равен 127
Итерация №5
Наименьший путь к точке 3 равен 56
Итерация №6
Наименьший путь к точке 4 равен 40
Итерация №7
Наименьший путь к точке 1 равен 129
Итерация №8
Наименьший путь к точке 2 равен 52
Итерация №9
Наименьший путь к точке 3 равен 117
Итерация №10
Наименьший путь к точке 2 равен 187
Итерация №11
Наименьший путь к точке 3 равен 176
Итерация №12
Наименьший путь к точке 4 равен 177

Полученный путь: 3-1. Стоимость: 88
```

ВИЗ 1

4.4. 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:

```
Введите путь к файлу: /Users/evgeniakiseleva/Desktop/1.csv
```

CSV 1

Вводим начальную и конечную точку.

Получаем результат. Пример:

```
Каким способом вы хотите ввести данные? 4
Введите путь к файлу: /Users/evgeniakiseleva/Desktop/1.csv
Весовая матрица:
      1   2  3   4   5   6   7   8  9 10
1     0   0  0   0  20  11  39   0  0  0
2     0   0  0  79   0   0  76  98  0  0
3     0   0  0   0   0   0   0   5  0  0
4     0  79  0   0   0  65   0  61  0  0
5    20   0  0   0   0   0   0   0  0  0
6    11   0  0  65   0   0  71   0  0  0
7    39  76  0   0   0  71   0   0  0  0
8     0  98  5  61   0   0   0   0  0  0
9     0   0  0   0   0   0   0   0  0  0
10    0   0  0   0   0   0   0   0  0  0
Начальная точка: 2
Конечная точка: 6
Полученный путь: 2-7-1-6. Стоимость: 126.0
```

ВИЗ 2

5. Архитектура решения

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

5.2. 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)

1.2 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)

1.3 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: B)

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: A)

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

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: A)

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

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: A)

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

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:

```
# Ручной ввод
if way == 1:
    nodes = input('Введите узлы графа через запятую (без пробела): ')
    nodes = nodes.split(',')

    # Формирования словаря с ключами-узлами графа
    init_graph = {}
    for node in nodes:
        init_graph[node] = {}

    # Число сочетаний
    max_sum_edge = factorial(len(nodes)) / (2 * factorial(len(nodes) - 2))
    sum_edge = int(input('Количество ребер графа (не более {}): '.format(int(max_sum_edge))))

    # Ввод весовых коэффициентов и помещение их в словарь nodes
    for i in range(sum_edge):
        edge = input('Введите весовые коэф. в формате: A-B-n, где A - начальная точка, B - конечная, n - цена: ')
        edge = edge.split('-')
        init_graph[edge[0]][edge[1]] = int(edge[2])


elif way == 2:
    sum_rows = int(input('Введите количество узлов графа: '))

    matrix = []
    for i in range(sum_rows):
        input_rows = input('Введите {} сроку весовой матрицы (значения через запятую): '.format(i+1))
        input_rows = input_rows.split(',')
        matrix.append(input_rows)

    nodes, init_graph = output_nodes_init_graph(matrix)
```

```
# Случайная генерация
elif way == 3:
    sum_nodes = int(input('Введите количество узлов графа: '))

    # заполняем список узлов значениями (имена узлов)
    nodes = []
    for i in range(sum_nodes):
        nodes.append(str(i+1))

    max_sum_edge = (factorial(sum_nodes)) / (2 * factorial(sum_nodes - 2)) # Число сочетаний узлов графа
    sum_edge = int(input('Введите количество ребер графа (не более {}): '.format(int(max_sum_edge))))
    comb = list(combinations(nodes, 2)) # Сочетания узлов графа

    # Формирования словаря с ключами-узлами графа
    init_graph = {}
    for node in nodes:
        init_graph[node] = {}

    # Заполняем словарь случайными весовыми коэффициентами
    matrix = np.zeros((sum_nodes,sum_nodes)) # весовая матрица
    for i in range(sum_edge):
        random_edge = random.choice(comb)
        edge_rate = random.randint(0,100)
        matrix[int(random_edge[0]) - 1][int(random_edge[1]) - 1] = edge_rate
        matrix[int(random_edge[1]) - 1][int(random_edge[0]) - 1] = edge_rate

        init_graph[random_edge[0]][random_edge[1]] = edge_rate
        comb.remove(random_edge)

    matrix = pd.DataFrame(matrix)
    matrix.columns = nodes
    matrix.index = nodes
    print('Весовая матрица: \n', matrix)
```

```
elif way == 4:
    matrix = []
    file_way = input('Введите путь к файлу: ')
    with open(file_way, 'r', newline='') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=';')
        for row in spamreader:
            matrix.append(row)
    nodes, init_graph = output_nodes_init_graph(matrix)

start_node_input = input('Начальная точка: ')
target_node_input = input('Конечная точка: ')


graph = Graph(nodes, init_graph)

previous_nodes, shortest_path = dijkstra_algorithm(graph=graph, start_node=start_node_input)
print_result(previous_nodes, shortest_path, start_node=start_node_input, target_node=target_node_input)
```

Код в питоне 1

5.3. 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:

```
class Graph(object):
    def __init__(self, nodes, init_graph):
        self.nodes = nodes
        self.graph = self.construct_graph(nodes, init_graph)

    def construct_graph(self, nodes, init_graph):
        '''
        Этот метод обеспечивает симметричность графика. 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.
        '''
        graph = {}
        for node in nodes:
            graph[node] = {}

        graph.update(init_graph)

        for node, edges in graph.items():
            for adjacent_node, value in edges.items():
                if graph[adjacent_node].get(node, False) == False:
                    graph[adjacent_node][node] = value
        return graph

    def get_nodes(self):
        "Возвращает узлы графа"
        return self.nodes

    def get_outgoing_edges(self, node):
        "Возвращает соседей узла"
        connections = []
        for out_node in self.nodes:
            if self.graph[node].get(out_node, False) != False:
                connections.append(out_node)
        return connections
```

Код в питоне 2

```
    def value(self, node1, node2):
        "Возвращает значение ребра между двумя узлами."
        return self.graph[node1][node2]
def dijkstra_algorithm(graph, start_node):
    unvisited_nodes = list(graph.get_nodes())
    # Мы будем использовать этот словарь, чтобы сэкономить на посещении каждого узла и обновлять его по мере продвижения по графику
    shortest_path = {}

    # Мы будем использовать этот словарь, чтобы сохранить кратчайший известный путь к найденному узлу
    previous_nodes = {}

    k = 1 # номер итерации
    #print('\n')

    # Мы будем использовать max_value для инициализации значения "бесконечности" непосещенных узлов
    max_value = sys.maxsize
    for node in unvisited_nodes:
        shortest_path[node] = max_value
    # Однако мы инициализируем значение начального узла 0
    shortest_path[start_node] = 0

    # Алгоритм выполняется до тех пор, пока мы не посетим все узлы
    while unvisited_nodes:
        # Приведенный ниже блок кода находит узел с наименьшей оценкой
        current_min_node = None
        for node in unvisited_nodes: # Итерация по узлам
            if current_min_node == None:
                current_min_node = node
            elif shortest_path[node] < shortest_path[current_min_node]:
                current_min_node = node
            #print('shortest_path ',shortest_path)

        # Приведенный ниже блок кода извлекает соседей текущего узла и обновляет их расстояния
        neighbors = graph.get_outgoing_edges(current_min_node)
```

```
        for neighbor in neighbors:
            tentative_value = shortest_path[current_min_node] + graph.value(current_min_node, neighbor)
            if tentative_value < shortest_path[neighbor]:
                shortest_path[neighbor] = tentative_value
                # Мы также обновляем лучший путь к текущему узлу
                previous_nodes[neighbor] = current_min_node
            #print('tentative_value ', tentative_value, '\n')

            print('Итерация №{}\nНаименьший путь к точке {} равен {}'.format(k, neighbor, tentative_value))
            k += 1
            time.sleep(0.5)

        # После посещения его соседей мы отмечаем узел как "посещенный"
        unvisited_nodes.remove(current_min_node)
    return previous_nodes, shortest_path

def print_result(previous_nodes, shortest_path, start_node, target_node):
    path = []
    node = target_node

    while node != start_node:
        path.append(node)
        node = previous_nodes[node]

    # Добавить начальный узел вручную
    path.append(start_node)
    print("\nПолученный путь: {}. Стоимость: {}".format("-".join(reversed(path)), shortest_path[target_node]))


def output_nodes_init_graph(matrix):
    # заполняем список узлов значениями (имена узлов)
    nodes = []
    for i in range(len(matrix[0])):
        nodes.append(str(i+1))
```

Код в питоне 3

```
# Формирования словаря с ключами-узлами графа
init_graph = {}
for node in nodes:
    init_graph[node] = {}

# Весовая матрица matrix
matrix = pd.DataFrame(matrix)
matrix.columns = nodes
matrix.index = nodes
print("Весовая матрица: \n", matrix)

# Заполняем словарь init_graph значениями
for start_value, row in enumerate(matrix, start=1):
    for value in range(start_value, len(matrix.columns)+1):
        if int(matrix[row][value-1]) == 0:
            continue
        else:
            init_graph[row][str(value)] = float(matrix[row][value-1])

return nodes, init_graph
```

Код в питоне 4

5.4. 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:

```
print('Итерация №{}\nНаименьший путь к точке {} равен {}'.format(k, neighbor, tentative_value))
k += 1
time.sleep(0.5)
```

Фрагмент, отвечающий за визуализацию 1

6. 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:

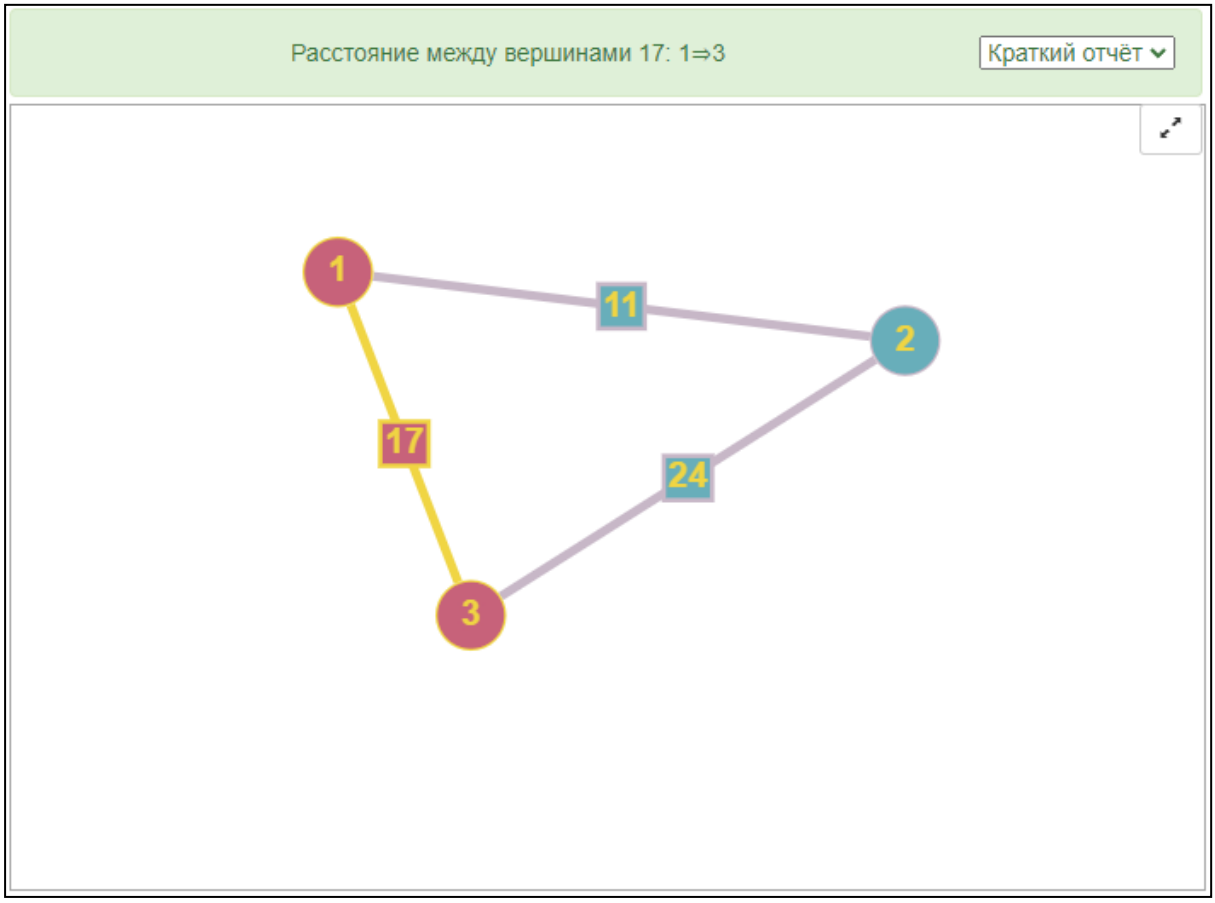
```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 3
Введите 1 сроку весовой матрицы (значения через запятую): 0,11,17
Введите 2 сроку весовой матрицы (значения через запятую): 11,0,24
Введите 3 сроку весовой матрицы (значения через запятую): 17,24,0
Весовая матрица:
     1   2   3
1   0  11  17
2  11   0  24
3  17  24   0
Начальная точка: 1
Конечная точка: 3
Итерация №1
Наименьший путь к точке 2 равен 11.0
Итерация №2
Наименьший путь к точке 3 равен 17.0
Итерация №3
Наименьший путь к точке 1 равен 22.0
Итерация №4
Наименьший путь к точке 3 равен 35.0
Итерация №5
Наименьший путь к точке 1 равен 34.0
Итерация №6
Наименьший путь к точке 2 равен 41.0

Полученный путь: 1-3. Стоимость: 17.0
```

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 3x3:

6.2.1. Тест №2 кодом Python:

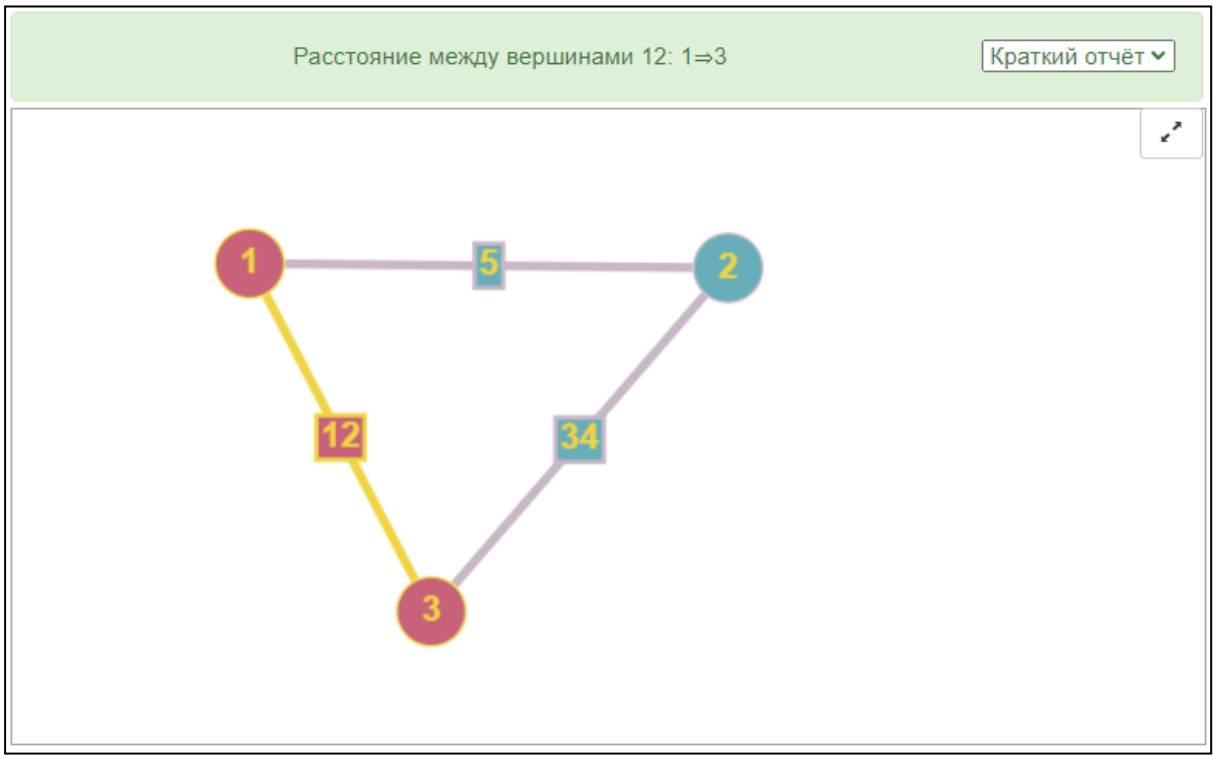
```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 3
Введите 1 сроку весовой матрицы (значения через запятую): 0,5,12
Введите 2 сроку весовой матрицы (значения через запятую): 5,0,34
Введите 3 сроку весовой матрицы (значения через запятую): 12,34,0
Весовая матрица:
     1   2   3
1   0   5  12
2   5   0  34
3  12  34   0
Начальная точка: 1
Конечная точка: 3
Итерация №1
Наименьший путь к точке 2 равен 5.0
Итерация №2
Наименьший путь к точке 3 равен 12.0
Итерация №3
Наименьший путь к точке 1 равен 10.0
Итерация №4
Наименьший путь к точке 3 равен 39.0
Итерация №5
Наименьший путь к точке 1 равен 24.0
Итерация №6
Наименьший путь к точке 2 равен 46.0

Полученный путь: 1-3. Стоимость: 12.0
```

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:

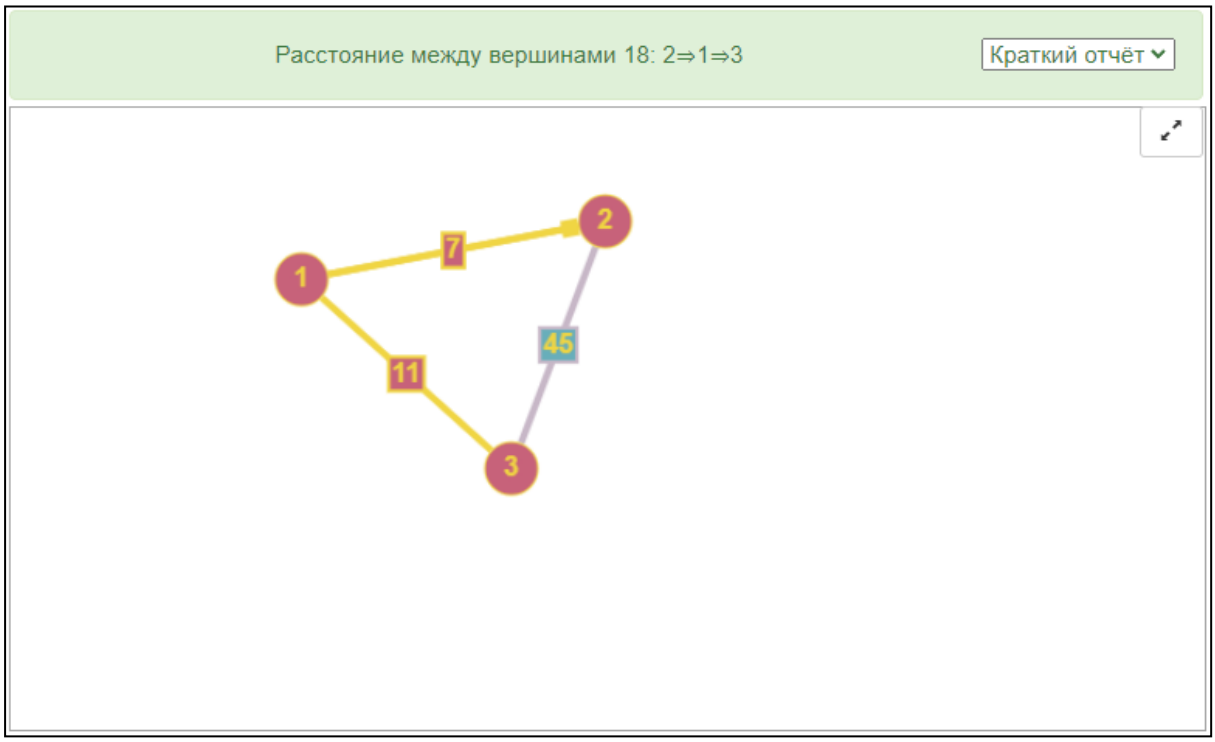
```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 3
Введите 1 сроку весовой матрицы (значения через запятую): 0,7,11
Введите 2 сроку весовой матрицы (значения через запятую): 7,0,45
Введите 3 сроку весовой матрицы (значения через запятую): 11,45,0
Весовая матрица:
     1   2   3
1   0   7  11
2   7   0  45
3  11  45   0
Начальная точка: 2
Конечная точка: 3
Итерация №1
Наименьший путь к точке 1 равен 7.0
Итерация №2
Наименьший путь к точке 3 равен 45.0
Итерация №3
Наименьший путь к точке 2 равен 14.0
Итерация №4
Наименьший путь к точке 3 равен 18.0
Итерация №5
Наименьший путь к точке 1 равен 29.0
Итерация №6
Наименьший путь к точке 2 равен 63.0

Полученный путь: 2-1-3. Стоимость: 18.0
```

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:

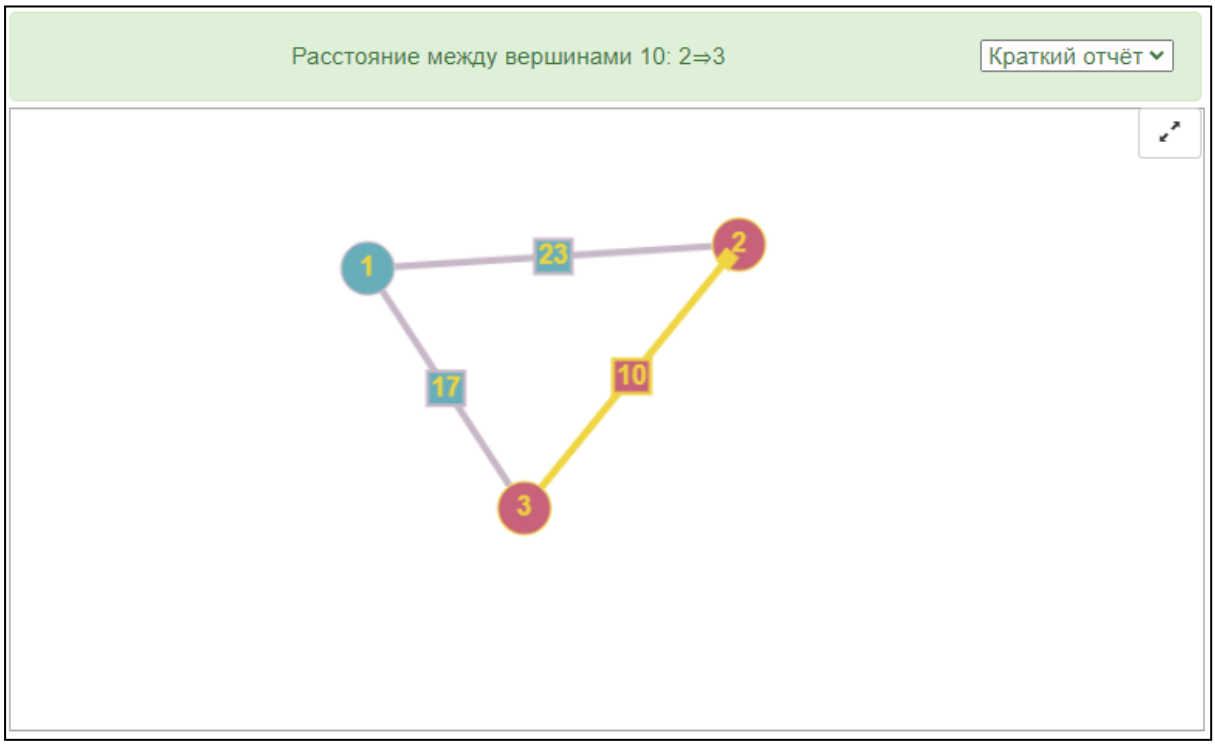
```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 3
Введите 1 сроку весовой матрицы (значения через запятую): 0,23,17
Введите 2 сроку весовой матрицы (значения через запятую): 23,0,10
Введите 3 сроку весовой матрицы (значения через запятую): 17,10,0
Весовая матрица:
     1   2   3
1   0  23  17
2  23   0  10
3  17  10   0
Начальная точка: 2
Конечная точка: 3
Итерация №1
Наименьший путь к точке 1 равен 23.0
Итерация №2
Наименьший путь к точке 3 равен 10.0
Итерация №3
Наименьший путь к точке 1 равен 27.0
Итерация №4
Наименьший путь к точке 2 равен 20.0
Итерация №5
Наименьший путь к точке 2 равен 46.0
Итерация №6
Наименьший путь к точке 3 равен 40.0

Полученный путь: 2-3. Стоимость: 10.0
```

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:

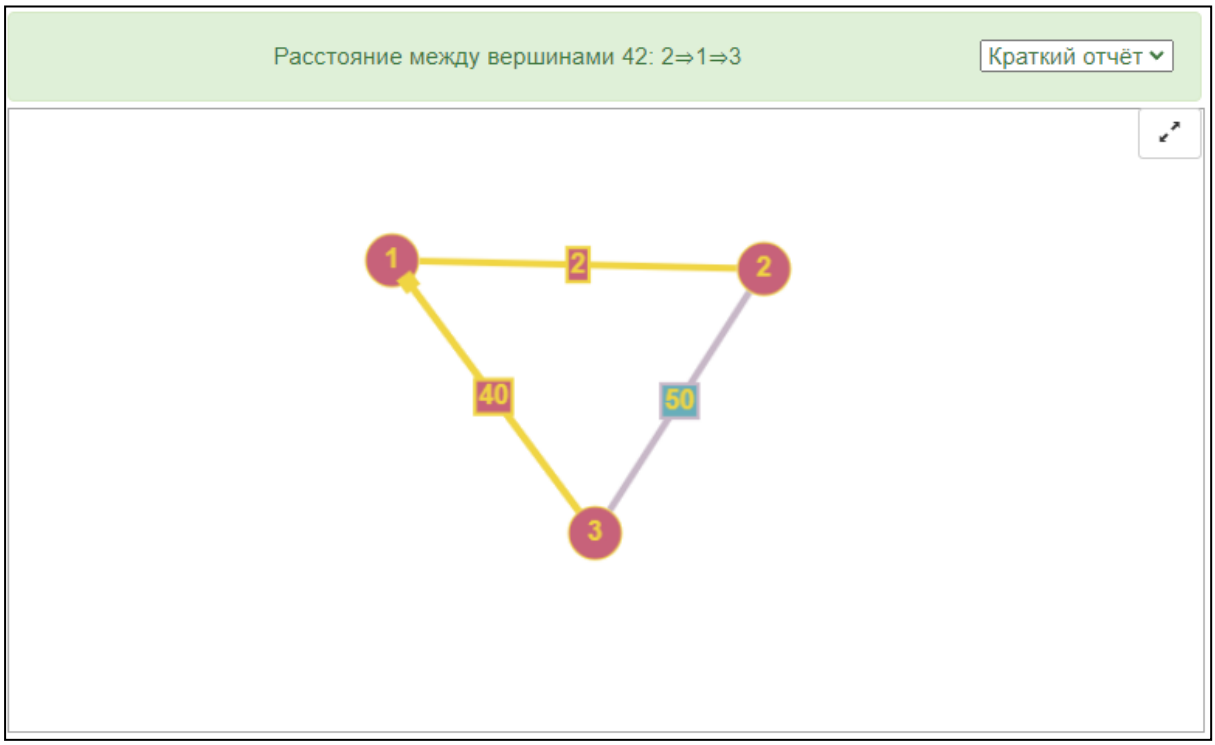
```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 3
Введите 1 сроку весовой матрицы (значения через запятую): 0,2,40
Введите 2 сроку весовой матрицы (значения через запятую): 2,0,50
Введите 3 сроку весовой матрицы (значения через запятую): 40,50,0
Весовая матрица:
     1   2   3
1   0   2  40
2   2   0  50
3  40  50   0
Начальная точка: 2
Конечная точка: 3
Итерация №1
Наименьший путь к точке 1 равен 2.0
Итерация №2
Наименьший путь к точке 3 равен 50.0
Итерация №3
Наименьший путь к точке 2 равен 4.0
Итерация №4
Наименьший путь к точке 3 равен 42.0
Итерация №5
Наименьший путь к точке 1 равен 82.0
Итерация №6
Наименьший путь к точке 2 равен 92.0

Полученный путь: 2-1-3. Стоимость: 42.0
```

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/

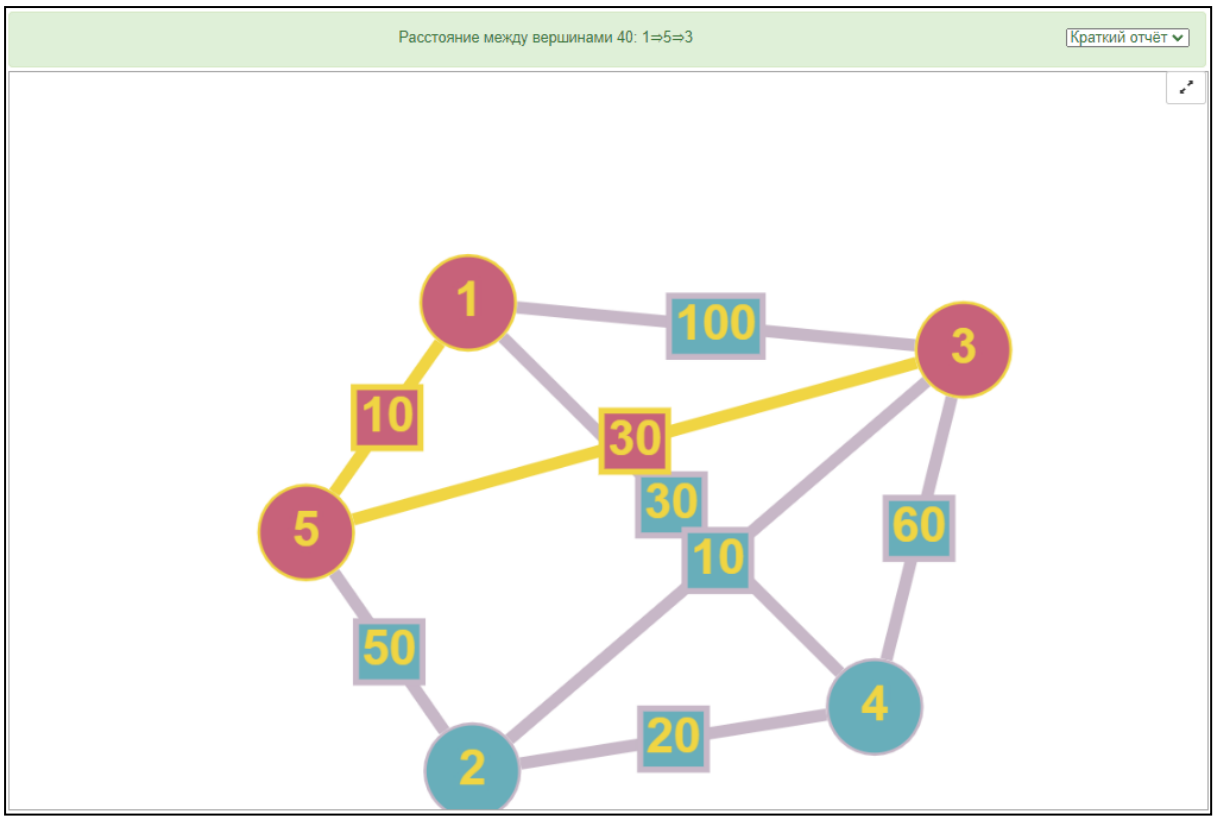
7. Заключение

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:

```
Каким способом вы хотите ввести данные? 2
Введите количество узлов графа: 5
Введите 1 сроку весовой матрицы (значения через запятую): 0,0,100,30,10
Введите 2 сроку весовой матрицы (значения через запятую): 0,0,10,20,50
Введите 3 сроку весовой матрицы (значения через запятую): 100,10,0,60,30
Введите 4 сроку весовой матрицы (значения через запятую): 30,20,60,0,0
Введите 5 сроку весовой матрицы (значения через запятую): 10,50,30,0,0
Весовая матрица:
      1   2    3   4   5
1    0   0  100  30  10
2    0   0   10  20  50
3  100  10    0  60  30
4   30  20   60   0   0
5   10  50   30   0   0
Начальная точка: 1
Конечная точка: 3
Итерация №1
Наименьший путь к точке 3 равен 100.0
Итерация №2
Наименьший путь к точке 4 равен 30.0
Итерация №3
Наименьший путь к точке 5 равен 10.0
Итерация №4
Наименьший путь к точке 1 равен 20.0
Итерация №5
Наименьший путь к точке 2 равен 60.0
Итерация №6
Наименьший путь к точке 3 равен 40.0
Итерация №7
Наименьший путь к точке 1 равен 60.0
Итерация №8
Наименьший путь к точке 2 равен 50.0
Итерация №9
Наименьший путь к точке 3 равен 90.0
Итерация №10
Наименьший путь к точке 1 равен 140.0
Итерация №11
Наименьший путь к точке 2 равен 50.0
Итерация №12
Наименьший путь к точке 4 равен 100.0
Итерация №13
Наименьший путь к точке 5 равен 70.0
Итерация №14
Наименьший путь к точке 3 равен 60.0
Итерация №15
Наименьший путь к точке 4 равен 70.0
Итерация №16
Наименьший путь к точке 5 равен 100.0

Полученный путь: 1-5-3. Стоимость: 40.0
```

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.