

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
 «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

|  |  |
| --- | --- |
| Заведующий кафедрой | «ПОВТ и АС» |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | В.В. Долгов |

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

**ОТЧЕТ**

по лабораторно-практической работе по дисциплине «Исследование операций» по кафедре «Программное обеспечение вычислительной техники и автоматизированных системы»

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_А. А. Кизогян\_\_\_

подпись, дата

Обозначение отчета УП.81.0000.000 Группа \_\_\_\_ВМО31\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление 02.03.03 Математическое обеспечение и администрирование информационных систем

Профиль Математическое обеспечение и администрирование информационных систем

Преподаватель: проф. Никитина Алла Валерьевна

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата подпись преподавателя

Ростов-на-Дону

2023

# Содержание

1. Лабораторная работа № 6 3
   1. Краткие теоретические сведения……………………………………...3
   2. Аналитическое решение……………………………………………….5
   3. Решение задачи линейного программирования стандартными средствами Mathcad…………………………………………………...10
   4. Решение полученное с помощью программного средства……….....11
2. Вопросы к защите лабораторной работы………………………………...12
3. Вывод…………………………………………………………………...….13
4. Приложение А Листинг программы реализующий симплекс-метод…..14

**Лабораторная работа №6**

**Тема работы:** **Метод ветвей и границ. Задача Коммивояжера**

**Цель работы**: познакомиться с методом ветвей и границ на примере задачи коммивояжера.

**Задание:** решить задачу коммивояжера в системе Mathсad.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 6 | 7 | 7 | 4 |
| 6 | # | 6 | 1 | 3 |
| 7 | 6 | # | 7 | 7 |
| 7 | 1 | 7 | # | 3 |
| 4 | 3 | 7 | 3 | # |

**Краткие теоретические сведения**

**Задача Коммивояжера**

Метод ветвей и границ

Задача коммивояжера —одна из самых известных оптимизационных задач. Ее цель заключается в нахождении самого выгодного маршрута (кратчайшего, самого быстрого, наиболее дешевого), проходящего через все заданные точки (пункты, города) по одному разу, с последующим возвратом в исходную точку.

Условия задачи должны содержать критерий выгодности маршрута (т. е. должен ли он быть максимально коротким, быстрым, дешевым или все вместе), а также исходные данные в виде матрицы затрат (расстояния, стоимости, времени и т. д.) при перемещении между рассматриваемыми пунктами. Особенности задачи в том, что она довольно просто формулируется и найти хорошие решения для нее также относительно просто, но вместе с тем поиск действительно оптимального маршрута для большого набора данных — непростой и ресурсоемкий процесс.

Метод ветвей и границ — один из методов дискретной оптимизации, являющийся развитием метода полного перебора, но отличающийся от него отсевом в процессе вычисления подмножеств неэффективных решений. Впервые был предложен в 1960 году английским профессором Алисой Лэнд и австралийским математиком Элисон Дойг.

Для решения задачи коммивояжера методом ветвей и границ необходимо выполнить следующий алгоритм (последовательность действий):

1. Построение матрицы с исходными данными — в таблицу заносятся расстояния ( между городами (в ячейки типа A-A, B-B и т. д. ставится символ M — условно бесконечно большое число); при этом строкам соответствуют города отбытия, а столбцам города прибытия;
2. Нахождение минимумов по строкам — в каждой строке определяется минимальное число () и выписывается в отдельный столбец;
3. Редукция строк — из значений ячеек каждой строки вычитаем соответствующий минимум (), не затрагивая при этом клетки с M;
4. Нахождение минимумов по столбцам — в каждом столбце определяется минимальное число () и выписывается в отдельную строку;
5. Редукция столбцов — из значений ячеек каждого столбца вычитаем соответствующий минимум (), не затрагивая при этом клетки с M;
6. Нахождение корневой нижней границы (делаем это только один раз, в следующие разы пункт 6 пропускаем) — вычисляем нижнюю границу (минимально возможную на текущем этапе длину маршрута) в стартовой (корневой) точке решения, как сумму найденных ранее минимумов () и начинаем построение графа (схемы) решения с внесения в него корневой вершины;
7. Вычисление оценок нулевых клеток — считаем оценки () для каждой ячейки с нулями, как сумму минимумов по строке и столбцу, в которых располагается нулевая клетка, не учитывая при этом саму нулевую клетку;
8. Выбор нулевой клетки с максимальной оценкой — ищем среди нулевых клеток обладающую наибольшей оценкой (если таких ячеек несколько, выбираем любую), и получаем пару ветвей (вариантов) решения задачи: с включением в маршрут отрезка пути относящегося к выбранной ячейке и без включения;
9. Редукция матрицы — вычеркиваем относящиеся к выбранной клетке строку и столбец, а также заменяем значение ячейки соответствующей обратному пути на M;
10. Вычисление нижней границы первой ветви (включающей отрезок пути) — вновь находим минимумы по строкам, проводим редукцию строк, находим минимумы по столбцам, проводим редукцию столбцов, после чего вычисляем локальную нижнюю границу, как сумму предыдущей локальной нижней границы и минимумов (), и добавляем вершину в граф;
11. Вычисление нижней границы второй ветви (не включающей отрезок пути) — считаем локальную нижнюю границу, как сумму предыдущей локальной нижней границы и оценки выбранной ранее нулевой клетки (), и добавляем вершину в граф;
12. Выбор ветви с минимальным значением нижней границы — среди еще не ветвившихся вершин выбираем обладающую минимальным значением локальной нижней границы (вне зависимости от того, какую ветвь рассматриваем в данный момент);
13. Если полный маршрут еще не найден, продолжаем решение, если найден — переходим к пункту 10 — если маршрут еще не найден, то ход дальнейшего решения зависит от выбранной ветви: (а) первая ветвь — переходим к пункту **7**, (б) вторая ветвь — в клетку с максимальной оценкой ставим M и переходим к пункту 2, (в) другая ветвь — возвращаемся к соответствующим ей этапу решения и таблице данных;
14. Построение полного маршрута и определение его длины — соединяем все найденные ранее отрезки пути в полный маршрут и считаем его общую длину (данные берем из исходной таблицы) .

**Аналитическое решение**

Начальное условие задачи Коммивояжера дано в виде таблице представленной в таблице 1.

Таблица 1 – начальное условие

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 6 | 7 | 7 | 4 |
| 6 | # | 6 | 1 | 3 |
| 7 | 6 | # | 7 | 7 |
| 7 | 1 | 7 | # | 3 |
| 4 | 3 | 7 | 3 | # |

Возьмем в качестве произвольного маршрута:

Тогда

Для определения нижней границы множества воспользуемся **операцией редукции** или приведения матрицы по строкам, для чего необходимо в каждой строке матрицы найти минимальный элемент.

Это приведет к виду, представленной в таблице 2.

Таблица 2 – добавление в задачу Коммивояжера столбец

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |  |
| **1** | # | 6 | 7 | 7 | 4 | 4 |
| **2** | 6 | # | 6 | 1 | 3 | 1 |
| **3** | 7 | 6 | # | 7 | 7 | 6 |
| **4** | 7 | 1 | 7 | # | 3 | 1 |
| **5** | 4 | 3 | 7 | 3 | # | 3 |

Затем вычитаем  из элементов рассматриваемой строки. В связи с этим во вновь полученной матрице в каждой строке будет как минимум один ноль. Можно это увидеть в таблице 3.

Таблица 3 – 1-й этап решения задачи Коммивояжера

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |
| **1** | # | 2 | 3 | 3 | 0 |
| **2** | 5 | # | 5 | 0 | 2 |
| **3** | 1 | 0 | # | 1 | 1 |
| **4** | 6 | 0 | 6 | # | 2 |
| **5** | 1 | 0 | 4 | 0 | # |

Такую же операцию редукции проводим по столбцам, для чего в каждом столбце находим минимальный элемент:

После вычитания минимальных элементов получаем полностью редуцированную матрицу, где величины и называются **константами приведения**. Решение данных действий можно увидеть в таблице 4.

Таблица 4 – редуцированная матрица данной задачи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |
| **1** | # | 2 | 0 | 3 | 0 |
| **2** | 4 | # | 2 | 0 | 2 |
| **3** | 0 | 0 | # | 1 | 1 |
| **4** | 5 | 0 | 3 | # | 2 |
| **5** | 0 | 0 | 1 | 0 | # |

Сумма констант приведения определяет нижнюю границу:

; Элементы матрицы соответствуют расстоянию от пункта до пункта . Поскольку в матрице городов, то является матрицей с неотрицательными элементами .

**Определяем ребро ветвления** и разобьем все множество маршрутов относительно этого ребра на два подмножества () и ().

С этой целью для всех клеток матрицы с нулевыми элементами заменяем поочередно нули на **#(**бесконечность) и определяем для них сумму образовавшихся констант приведения, они приведены в скобках, см. в таблицу 5.

Таблица 5 – определение ребер ветвления

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |  |
| **1** | # | 2 | 0(1) | 3 | 0(1) | 0 |
| **2** | 4 | # | 2 | **0(2)** | 2 | 2 |
| **3** | 0(0) | 0(0) | # | 1 | 1 | 0 |
| **4** | 5 | 0(2) | 3 | # | 2 | 2 |
| **5** | 0(0) | 0(0) | 1 | 0(0) | # | 0 |
|  | 0 | 0 | 1 | 0 | 1 | 0 |

Наибольшая сумма констант приведения равна для ребра , следовательно, множество разбивается на два подмножества и .

Исключение ребра проводим путем замены элемента на M, после чего осуществляем очередное приведение матрицы расстояний для образовавшегося подмножества , в результате получим редуцированную матрицу. Получаем таблицу 6.

Таблица 6 – подготовленная матрица для последующего удаления строки и столбца данной задачи

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |  |
| **1** | # | 2 | 0 | 3 | 0 | 0 |
| **2** | 4 | # | 2 | # | 2 | 2 |
| **3** | 0 | 0 | # | 1 | 1 | 0 |
| **4** | 5 | 0 | 3 | # | 2 | 0 |
| **5** | 0 | 0 | 1 | 0 | # | 0 |
|  | 0 | 0 | 0 | 0 | 0 | 2 |

Нижняя граница гамильтоновых циклов этого подмножества:

Включение ребра проводится путем исключения всех элементов 2-ой строки и 4-го столбца, в которой элемент  заменяем на **#**, для исключения образования негамильтонова цикла.

В результате получим другую сокращенную матрицу , которая подлежит операции приведения.

После операции приведения сокращенная матрица будет иметь вид, представленной в виде таблицы 7.

Таблица 7 – новая матрица с удаленными столбцом и строкой

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **5** |  |
| **1** | # | 2 | 0 | 0 | 0 |
| **3** | 0 | 0 | # | 1 | 0 |
| **4** | 5 | # | 3 | 2 | 2 |
| **5** | 0 | 0 | 1 | # | 0 |
|  | 0 | 0 | 0 | 0 | 2 |

Нижняя граница подмножества равна:

Поскольку нижние границы подмножества и подмножества равны, то ребро включаем в маршрут с новой границей .

Определяем ребро ветвления. См в таблицу 8.

Таблица 8 – определение ребер ветвления новой матрицы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **5** |  |
| **1** | # | 2 | **0(1)** | 0(0) | 0 |
| **3** | 0(0) | 0(0) | # | 1 | 0 |
| **4** | 3 | # | 1 | 0(1) | 1 |
| **5** | 0(0) | 0(0) | 1 | # | 0 |
|  | 0 | 0 | 1 | 0 | 0 |

Исключение ребра  **#**.

См. в таблицу 9.

Таблица 9 – подготовленная матрица для последующего удаления строки и столбца данной задачи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **5** |  |
| **1** | # | 2 | # | 0 | 0 |
| **3** | 0 | 0 | # | 1 | 0 |
| **4** | 3 | # | 1 | 0 | 0 |
| **5** | 0 | 0 | 1 | # | 0 |
|  | 0 | 0 | 1 | 0 | 1 |

Включение ребра (1,3): d31=М.

См. в таблицу 10.

Таблица 10 – новая матрица с удаленной строкой и столбцом

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **5** |  |
| **3** | # | 0 | 1 | 0 |
| **4** | 3 | # | 0 | 0 |
| **5** | 0 | 0 | # | 0 |
|  | 0 | 0 | 0 | 0 |

Ребро (1,3) включаем в маршрут с новой границей H=21.

Определяем ребро ветвления.

Исключение ребра **#**. Это представлено в виде таблицы 11.

Таблица 11 – матрица с добавленными

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **5** |  |
| **3** | M | 0 | 1 | 0 |
| **4** | 3 | M | M | 3 |
| **5** | 0 | 0 | M | 0 |
|  | 0 | 0 | 1 | 4 |

Включение ребра **#**. Получим в итоге таблицу 12.

Таблица 12 – финальный этап решения задачи

|  |  |  |  |
| --- | --- | --- | --- |
|  | **1** | **2** |  |
| **3** | M | 0 | 0 |
| **5** | 0 | 0 | 0 |
|  | 0 | 0 | 0 |

Ребро включаем в маршрут с новой границей.

В соответствии с этой матрицей включаем в гамильтонов маршрут ребра и .

В результате по дереву ветвлений гамильтонов цикл образуют ребра:

Длина маршрута равна .

**Решение задачи линейного программирования стандартными средствами Mathcad**

На рисунке 1 и 2 приведено решение данной задачи в среде Mathcad, это обусловенно тем, что это необходимо для проверки решений найденных аналитическим методом и программным средством.

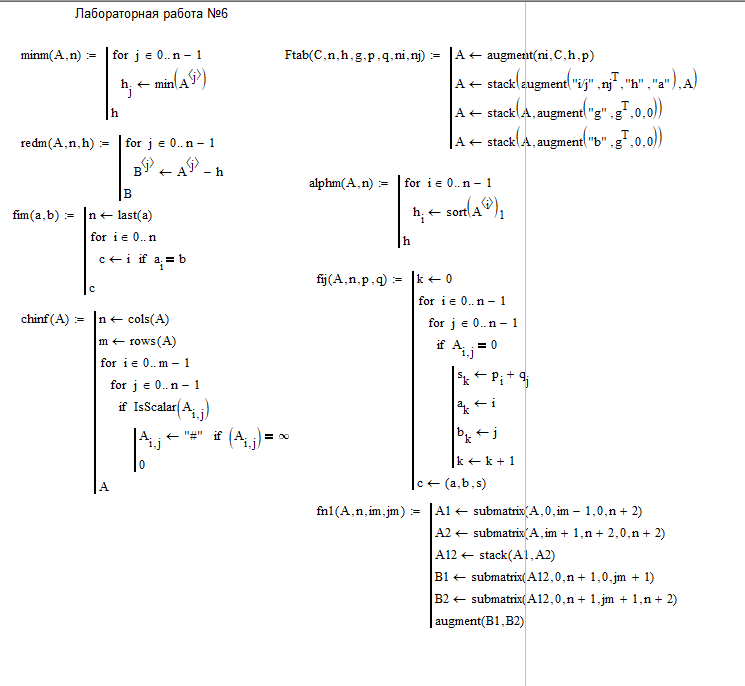


Рисунок 1- Программное средство в среде Mathcad

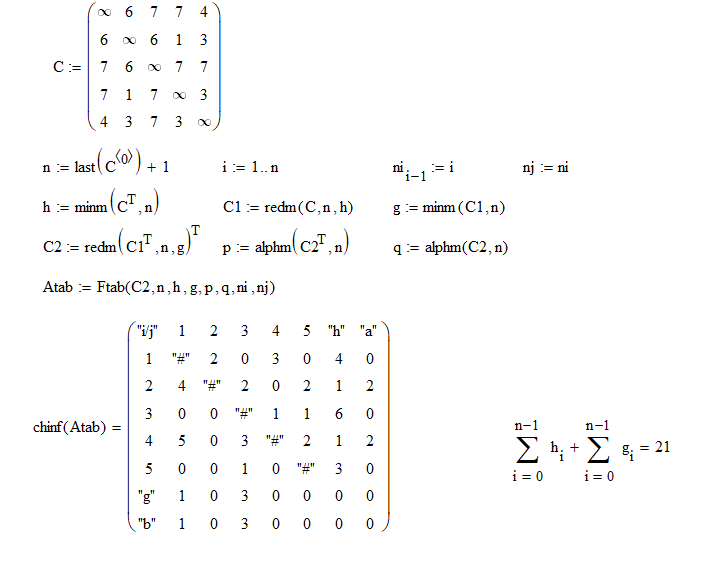


Рисунок 2 – Решение в среде Mathcad

**Решение полученное с помощью программного средства**

Благодаря программному средству, можно найти решение задачи Коммивояжера с помощью метода ветвей и границ. Описание программы приведено в Приложении А. Решение и вывод представлены на рисунках 3 и 4.

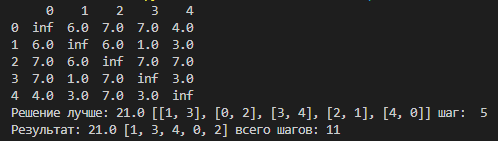


Рисунок 3- Программная реализация и результат работы программы

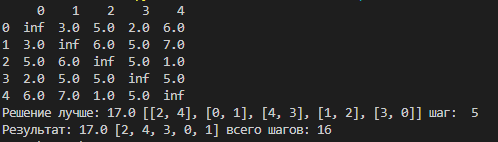


Рисунок 4- Программная реализация и результат работы программы другого варианта

**Вопросы к защите лабораторной работы**

1. **В чем заключается метод исчерпывающего перебора?**

Метод исчерпывающего перебора — это метод решения задачи, при котором перебираются все возможные варианты решения, чтобы найти оптимальное. Для этого пробуются все возможные комбинации и оценивается каждая из них с помощью заданной функции цели.

1. **При решении каких задач прибегают к методу исчерпывающего перебора?**

Метод исчерпывающего перебора применяется при решении задач, где требуется проверить все возможные комбинации или варианты решения. Это может включать поиск оптимального решения, определение всех возможных комбинаций или перестановок элементов, проверка всех условий для нахождения решения и т.д.

1. **В чем заключается поиск с возвратом?**

Поиск с возвратом в задаче коммивояжера заключается в нахождении оптимального пути, проходящего через все заданные города только один раз и возвращающегося в исходный город. При этом используется перебор всех возможных вариантов пути с последующим отслеживанием и сравнением их стоимости, чтобы найти минимальную стоимость обхода. Поиск с возвратом позволяет ограничить количество вариантов пути, исключая те, которые уже имеют большую стоимость или нарушают условие посещения каждого города только один раз.

1. **Суть метода ветвей и границ.**

Суть метода ветвей и границ заключается в решении оптимизационных задач с ограничениями путем разбиения пространства поиска на более мелкие подпространства, называемые ветвями, и оценке границ оптимального значения функции на каждом подпространстве.

Метод ветвей и границ обычно применяется для задач целочисленного программирования, где требуется найти оптимальное целочисленное решение задачи. Пространство поиска разбивается путем введения дополнительных ограничений, которые задаются для переменных, принимающих целочисленные значения. Ветви между частями пространства поиска создаются путем добавления ограничений в форме неравенств, которые разделяют допустимые решения на две части.

На каждом подпространстве выполняется решение задачи линейного программирования, которая является релаксацией исходной задачи, то есть задачи с отсутствием ограничений на цело численность переменных. Полученное оптимальное значение функции является нижней границей для этого подпространства.

Затем происходит оценка верхних границ, которая может быть достигнута на каждом подпространстве. Для этого проводится оценка значений функции на оставшихся ветвях, что позволяет отсеять те ветви, которые уже не могут дать более оптимальное решение, чем текущее лучшее решение.

Процесс разбиения пространства поиска и оценки границ продолжается до тех пор, пока не будет достигнуто оптимальное решение либо превышено допустимое время или вычислительные ресурсы.

**Вывод**

Метод ветвей и границ — это алгоритм решения задачи коммивояжера, который позволяет найти оптимальный путь с минимальной суммой длин всех ребер, проходящего через все вершины графа.

В ходе работы алгоритма происходит последовательное разбиение задачи на более простые подзадачи, которые решаются и объединяются для получения общего решения.

Алгоритм продолжает свое выполнение, пока не пройдет через все вершины графа. Как только все вершины посещены, происходит вычисление длины полученного пути. Если найденный путь является лучшим, чем предыдущий наименьший путь, он сохраняется как текущее оптимальное решение.

Однако метод ветвей и границ не гарантирует нахождение оптимального решения во всех случаях, так как требует перебора всех возможных комбинаций путей. Это делает алгоритм неэффективным для больших графов, где количество перебираемых комбинаций растет экспоненциально.

Тем не менее, метод ветвей и границ является важным алгоритмическим инструментом для решения задачи коммивояжера и может использоваться в комбинации с другими методами для нахождения приближенного или точного решения.

**Приложение А Листинг программы**

import pandas as pd

import numpy as np

import random

import matplotlib.pyplot as plt

import networkx as nx

import sys

if not sys.warnoptions:

    import warnings

    warnings.simplefilter("ignore")

pd.set\_option('display.max\_rows', 500)

pd.set\_option('display.max\_columns', 500)

pd.set\_option('display.width', 1000)

#------------------------------------------------------------------------------------

def distance(x1, y1, x2, y2):

    return ((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2) \*\* 0.5

#------------------------------------------------------------------------------------

def in\_deep\_matrix(p, y, x):

    # Возвращает новую матрицу меньшего размера, за вычитом строки и столбца

    return p.drop([y], axis = 0).drop([x], axis = 1)

#------------------------------------------------------------------------------------

def reduction\_matrix(p):

    # Производим редуцирование матрицы, возвращаем нижнюю границу

    bottom\_line = 0

    # Находим минимум по каждой строке и вычитаем его

    for index, row in p.iterrows():

        min = row.min()

        if np.isinf(min):

            return np.inf

        bottom\_line += min

        for key, value in row.iteritems():

             p[key][index] -= min

    # Находим минимум по каждому столбцу и вычитаем его

    for key, value in p.iteritems():

        min = value.min()

        if np.isinf(min):

            return np.inf

        bottom\_line += min

        for index, row in value.iteritems():

            p[key][index] -= min

    return bottom\_line

#------------------------------------------------------------------------------------

def partition\_matrix(p):

    # Ищем элемент для разбиения матрицы на m1 и m2

    # Для этого производим оценку нулевых элементов матрицы

    max\_sum = 0

    index\_i = None

    index\_j = None

    for index, row in p.iterrows():

        for key, value in p.iteritems():

            if p[key][index] == 0:

                min\_i = np.inf

                min\_j = np.inf

                for k, v in p[key].items():  # по столбецу

                    if k != index and v < min\_i:

                        min\_i = v

                for k, v in p.loc[index].items():

                    if k != key and v < min\_j:

                        min\_j = v

                l = min\_i + min\_j

                if l > max\_sum:

                    max\_sum = l

                    index\_i = index

                    index\_j = key

    return [index\_i, index\_j, max\_sum]

#------------------------------------------------------------------------------------

def reverse\_index(l, i, j):

    # Находим обрытный индекс для матрицы

    def in\_dict(d, v):

        while v in d:

            v = d[v]

        return v

    ln = len(l)

    d1 = {l[k][0]: l[k][1] for k in range(0, ln, 1)}

    d2 = {l[k][1]: l[k][0] for k in range(0, ln, 1)}

    return [in\_dict(d1, i), in\_dict(d2, j)]

#------------------------------------------------------------------------------------

def evaluation\_matrix(p, res, bottom\_line):

    # Оценка матрицы, поиск решения

    if len(p) == 1:

        res['steps'] += 1

        bottom\_line += p.iat[0, 0]

        # Если текущее решение лучше, запоминаем его

        if bottom\_line < res['global\_min']:

            res['global\_min'] = bottom\_line

            res['local\_result'].append([p.index[0], p.columns[0]])

            res['best\_result'] = res['local\_result'].copy()

            print('Решение лучше:', bottom\_line, res['best\_result'], 'шаг: ', res['steps'])

        return

    # Производим редуцирование матрицы, возвращаем минимальную нижнюю границу

    bottom\_line += reduction\_matrix(p)

    if np.isinf(bottom\_line):

        return

    max\_sum = 0

    while True:

        res['steps'] += 1

        # Находим элемент для разбиения на подмножества m1 и m2

        i, j, max\_sum = partition\_matrix(p)

        # Больше нет элементов для разбиения

        if i is None:

            return

        v\_len = len(res['local\_result'])

        # Рассматриваем m1 (соглашаемся на разбиение по элементу)

        if bottom\_line < res['global\_min']:

            res['local\_result'].append([i, j])

            p1 = in\_deep\_matrix(p, i, j)

            # Вычёркиваем обратный элемент только для матрицы большей чем 2х2, чтоб не получить inf

            if len(p1) > 2:

                i\_reverse, j\_reverse = reverse\_index(res['local\_result'], i, j)

                p1[j\_reverse][i\_reverse] = np.inf

            evaluation\_matrix(p1, res, bottom\_line)

        # Рассматриваем m2

        if res['global\_min'] < bottom\_line + max\_sum:

            return

        p[j][i] = np.inf  # Исключаем не выбранный элемент

        res['local\_result'] = res['local\_result'][:v\_len]  # Обрезаем список пути

#------------------------------------------------------------------------------------

def return\_res(res):

    l = res['best\_result']

    if l:

        d = {l[k][0]: l[k][1] for k in range(len(l))}

        li =[]

        a = l[0][0]

        for v in range(len(l)):

            li.append(a)

            a = d[a]

        return li

    else:

        return []

#------------------------------------------------------------------------------------

#random.seed(1)

n = 5

v1 = []

points = {}

for i in range(n):

    points[i] = (random.randint(1,10), random.randint(1,10))

input\_matrix = []

for i, vi in points.items():

    m1 = []

    for j, vj in points.items():

        if i==j:

             m1.append(np.inf)

        else:

            m1.append(int(distance(vi[0], vi[1], vj[0], vj[1])))

            v1.append([i,j,int(distance(vi[0], vi[1], vj[0], vj[1]))])

    input\_matrix.append(m1.copy())

plt.figure(figsize=(6, 6))

graph = nx.Graph()

graph.add\_nodes\_from(points)

# Добавляем дуги в граф

for i in v1:

    graph.add\_edge(i[0], i[1], weight=i[2])

f1 = pd.DataFrame(input\_matrix)

print(f1)

# Инициализация массива решений

res = {'global\_min':np.inf, 'best\_result': [], 'local\_result':[], 'steps':0}

# Запуск нахождения решения

evaluation\_matrix(f1, res, 0)

print('Результат:', res['global\_min'], return\_res(res), 'всего шагов:', res['steps'])

# Рисуем всё древо

nx.draw(graph, points, width=1, edge\_color="#C0C0C0")

# Рисуем оптимальный путь

nx.draw\_networkx(graph, points, with\_labels=True, edgelist=res['best\_result'], arrows=True, edge\_color="blue", width=5)