**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине: «Оптимизация проектных решений»

на тему: **«**Простейший пример оптимизации**»**

Выполнил: студент гр. ИТП-31

Расшивалов Н.И.

Принял: преподаватель-стажёр

Карась О.В.

Гомель 2022

**Цель работы**: изучение различных методов решения простейших задач

оптимизации.

**Задание:**

1. Построить таблицу 1 (5х5). В качестве исходных данных записать

свою фамилию.

2. Найти минимальное время транспортировки, используя «жадный»

алгоритм. Написать программу, реализующую «жадный» алгоритм для

произвольной матрицы (10х10).

3. Разработать программу для нахождения минимального и максимального времени транспортировки методом полного перебора.

4. Оценить, к какому значению ближе решение, найденное в п.2.

5. Оценить вычислительную сложность алгоритмов.

**Ход работы**

На рисунке 1 представлено заполнение исходных данных.

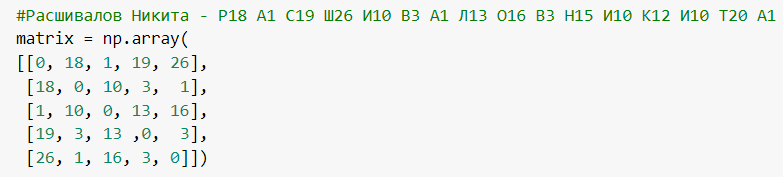


Рисунок 1 – Заполнение исходных данных

На рисунке 2 представлены результаты работы программы для матрицы 5 на 5.

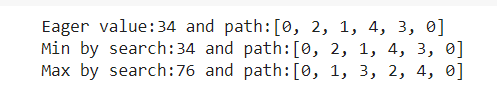


Рисунок 3 – Результаты работы программы для матрицы 5 на 5

На рисунке 3 представлены результаты работы программы для матрицы 10 на 10.

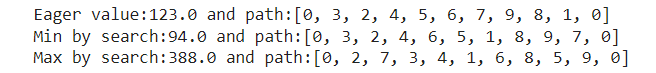


Рисунок 3 – Результаты работы программы для матрицы 10 на 10

**Тесты**

P0 p1 p2 p3 p4

P0 [0, 18, 1, 19, 26]

P1 [18, 0, 10, 3, 1]

P2 [1, 10, 0, 13, 16]

P3 [19, 3, 13, 0, 3]

P4 [26, 1, 16, 3, 0]

P0(1) – p2(10) – p1(1) – p4(3) – p3(19) - p0

Длина пути = 34

Результат полученный в программе представлен на рисунке 4.



Рисунок 4 – Результат, полученный в программе

Жадный алгоритм значительно ускоряет расчёты.

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы решена простейшая задача оптимизации, в ходе которой установлено что жадный алгоритм имеет допустимую точность вычислений при значительной экономии времени вычисления.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

**lab1\_OPR.ipynb**

import numpy as np

#Расшивалов Никита - Р18 А1 С19 Ш26 И10 В3 А1 Л13 О16 В3 Н15 И10 К12 И10 Т20 А1

matrix = np.array(

[[0, 18, 1, 19, 26],

 [18, 0, 10, 3,  1],

 [1, 10, 0, 13, 16],

 [19, 3, 13 ,0,  3],

 [26, 1, 16, 3, 0]])

def make\_matrix(size, seed=23):

 np.random.seed(seed)

 rand\_matrix = np.zeros((size, size))

 for i in range(size):

  for j in range(i, size):

    rand\_matrix[i, j] = np.random.randint(0, 50)

    rand\_matrix[j, i] = rand\_matrix[i, j]

 return rand\_matrix

#matrix = make\_matrix(10)

print(matrix)

def eager\_algorithm(matrix):

 looked\_rows = []

 path = []

 path\_length = 0

 current\_row = 0

 for i in range(matrix.shape[0]):

   first\_available = 0

   for j in range(matrix.shape[1]):

     if(j != current\_row and j not in looked\_rows):

       first\_available = j

       break

   min\_column = matrix[current\_row, first\_available]

   min\_col\_pos = first\_available

   for j in range(first\_available + 1, matrix.shape[1]):

    if(j not in looked\_rows and current\_row != j and matrix[current\_row, j] < min\_column):

      min\_column = matrix[current\_row, j]

      min\_col\_pos = j

   path\_length += min\_column

   path.append(current\_row)

   looked\_rows.append(current\_row)

   current\_row = min\_col\_pos

 path.append(path[0])

 return (path, path\_length)

def permutations(iterable):

    pool = tuple(iterable)

    n = len(pool)

    r = n

    if r > n:

        return

    indices = list(range(n))

    cycles = list(range(n, n-r, -1))

    yield tuple(pool[i] for i in indices[:r])

    while n:

        for i in reversed(range(r)):

            cycles[i] -= 1

            if cycles[i] == 0:

                indices[i:] = indices[i+1:] + indices[i:i+1]

                cycles[i] = n - i

            else:

                j = cycles[i]

                indices[i], indices[-j] = indices[-j], indices[i]

                yield tuple(pool[i] for i in indices[:r])

                break

        else:

            return

def full\_search(matrix):

  permutationItems = permutations([i for i in range(matrix.shape[0])])

  paths = []

  times = []

  for path in permutationItems:

    paths.append(path)

    times.append([])

    for i in range(matrix.shape[0]):

      time = matrix[path[i],path[0]]

      if i < matrix.shape[0] - 1:

        time = matrix[path[i],path[i + 1]]

      times[-1].append(time)

  return paths,times

def find\_min\_path(matrix):

  paths,times = full\_search(matrix)

  min\_path = sum(times[0])

  min\_pos = 0

  for i in range(1, len(paths)):

    next\_sum = sum(times[i])

    if next\_sum < min\_path:

      min\_path = next\_sum

      min\_pos = i

  full\_path = list(paths[min\_pos])

  full\_path.append(paths[min\_pos][0])

  return min\_path,full\_path

def find\_max\_path(matrix):

  paths,times = full\_search(matrix)

  max\_path = sum(times[0])

  max\_pos = 0

  for i in range(1, len(paths)):

    next\_sum = sum(times[i])

    if next\_sum > max\_path:

      max\_path = next\_sum

      max\_pos = i

  full\_path = list(paths[max\_pos])

  full\_path.append(paths[max\_pos][0])

  return max\_path,full\_path

#Расшивалов Никита - Р18 А1 С19 Ш26 И10 В3 А1 Л13 О16 В3 Н15 И10 К12 И10 Т20 А1

matrix = np.array(

[[0, 18, 1, 19, 26],

 [18, 0, 10, 3,  1],

 [1, 10, 0, 13, 16],

 [19, 3, 13 ,0,  3],

 [26, 1, 16, 3, 0]])

eager\_path,min\_eager = eager\_algorithm(matrix)

print("Eager value:" + str(min\_eager) + " and path:" + str(eager\_path))

min\_path\_len,min\_path = find\_min\_path(matrix)

print("Min by search:" + str(min\_path\_len) + " and path:" + str(min\_path))

max\_path\_len,max\_path = find\_max\_path(matrix)

print("Max by search:" + str(max\_path\_len) + " and path:" + str(max\_path))

def make\_matrix(size, seed=23):

 np.random.seed(seed)

 rand\_matrix = np.zeros((size, size))

 for i in range(size):

  for j in range(i, size):

    rand\_matrix[i, j] = np.random.randint(0, 50)

    rand\_matrix[j, i] = rand\_matrix[i, j]

 return rand\_matrix

matrix = make\_matrix(10)

eager\_path,min\_eager = eager\_algorithm(matrix)

print("Eager value:" + str(min\_eager) + " and path:" + str(eager\_path))

min\_path\_len,min\_path = find\_min\_path(matrix)

print("Min by search:" + str(min\_path\_len) + " and path:" + str(min\_path))

max\_path\_len,max\_path = find\_max\_path(matrix)

print("Max by search:" + str(max\_path\_len) + " and path:" + str(max\_path))