**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: Кратчайшие пути в графе: коммивояжёр.**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2023

* 1. **Цель работы.**
  2. Написать программу, решающую задачу коммивояжёра при помощи алгоритма ближайшего соседа и алгоритма Литтла.
  3. **Задание:**

Решить ЗК двумя методами в соответствии с вариантом: 1) Методом ВиГ. 2) Приближённым методом.

Дано: матрица весов графа, все веса неотрицательны; стартовая вершина.  
Найти: путь коммивояжёра (последовательность вершин) и его стоимость.

3. МВиГ: Алгоритм Литтла с модификацией: после приведения матрицы, к  
нижней оценке добавляется максимум двух нижних оценок суммарного веса  
оставшихся фрагментов пути: 1) полусуммы весов двух легчайших рёбер по  
всем построенным кускам; 2) веса МОД.

Приближённый алгоритм: АБС.

Пояснение к варианту 3. Кусок - это построенная цепочка, содержащая от 1 до |V| вершин, которая будет включена в решение. При вычислении первой  
оценки для каждого куска брать минимальное ребро, инцидентное одному  
концу, и минимальное ребро, инцидентное другому концу. При вычислении  
второй оценки в качестве "вершин" при построении МОД используются куски,  
при этом в это МОД могут входить только рёбра, инцидентные концам кусков.

* 1. **Выполнение работы.**

Для решения задачи были использованы два разных алгоритма для решения задачи коммивояжёра:

* Алгоритм ближайшего соседа:

Начиная со стартового города, выбираем следующий город, стоимость перехода в который из текущего города минимальна. В случае, если выбранный город уже посещён, поиск города для переезда продолжается. На каждой итерации посещенный город добавляется в результирующий список. Алгоритм завершает работу, когда не осталось не посещенных городов.

В ходе работы алгоритма, для каждого текущего города осуществляется поиск минимального пути через рассмотрение всех доступных для перехода рёбер, поэтому сложность по времени равна O(), где n – число городов. В свою очередь, сложность по памяти также равна O(), где n – число городов. Это связано с тем, что мы храним матрицу размера n\*n.

Для реализации алгоритма была написана функция *nn\_algorythm(start, matrix)*. Функция принимает на вход стартовый город *start* и матрицу весов *matrix*. Создаётся список не посещённых городов *lst* и запускается цикл, который будет работать до тех пор, пока список *lst* не пуст. В теле цикла производится поиск наименьшего пути из текущего города и добавление выбранного для посещения города в результирующий список.

Для псевдослучайной генерации матрицы весов была написана функция *generate(size)*, принимающая на вход размер генерируемой матрицы. В теле функции запускается цикл, в котором генерируются числа и добавляются в строку *row*, которая, в свою очередь, после заполнения, добавляется в матрицу. По завершении цикла все сгенерированные строки записываются в файл matrix.txt.

Код программы см. в приложении А.

**Тестирование.**

Ниже представлено тестирование программы.

Результаты тестирования алгоритма ближайшего соседа представлены в табл. 1.

Таблица 1 – Результаты прогона программы с разными входными данными.

| Входные данные | Выходные данные | Комментарии |
| --- | --- | --- |
| 6  2  inf 29 16 23 2 40  8 inf 33 20 26 42  4 43 inf 49 15 39  27 15 17 inf 31 4  41 13 16 10 inf 17  20 3 16 13 18 inf | 83  2-1-5-4-6-3-2 | Верно |
| 5  1  inf 35 42 35 13  24 inf 6 50 34  28 7 inf 20 44  16 1 26 inf 12  28 49 15 39 inf | 101  1-5-3-2-4-1 | Верно |
| 7  5  inf 12 11 43 48 10 17  29 inf 35 27 3 44 5  4 26 inf 43 5 44 44  39 41 29 inf 34 41 38  7 44 27 32 inf 23 43  39 19 30 1 41 inf 38  30 32 40 36 42 3 inf | 120  5-1-6-4-3-2-7-5 | Верно |



Рисунок 1 – Демонстрация промежуточных выводов для алгоритма ближайшего соседа.

* 1. **Выводы.**

Написана программа, которая решает задачу коммивояжёра при помощи алгоритма ближайшего соседа. Также была оценена сложность данного алгоритма.

**ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: neighbour.py

import math

from generator import \*

def nn\_algorythm(start, matrix):

lst = list((i for i in range(0, len(matrix[0]) + 1)))[1::] # список непосещенных городов

cost = 0 # общая цена пути

res = [start] # цепочка посещенных городов

counter = 1

while len(lst) != 0:

print(f'Итерация №{counter}')

minlen = math.inf # минимальный длина пути из рассматриваемого города

minel = start # минимальный путь из рассматриваемого города

for j in lst:

if not math.isinf(matrix[res[-1] - 1][j - 1]) and matrix[res[-1] - 1][j - 1] < minlen:

tmp = j

if (len(lst) != 1 and tmp == start) or (tmp in res and tmp != start):

continue

minel = j # изменение текущего минимального ребра

minlen = matrix[res[-1] - 1][j - 1] # изменение длины текущего минимального ребра

print(f'Для перехода выбрано ребро ({res[-1]}, {minel}), длина которого равна {minlen}')

lst.remove(minel) # удаление города из списка не посещённых

res.append(minel) # добавление города в результирующий список

cost += minlen # увеличение длины пройденного пути

print(f'Путь на текущей итерации имеет следующий вид:')

print(res)

print(f'Длина составленного куска равна {cost}')

print()

counter += 1

print(f'Решение найдено!')

return cost, res

def show\_matrix(matrix):

for i in range(len(matrix)):

for j in range(len(matrix)):

print(matrix[i][j], end=' ')

print()

size = int(input())

start = int(input())

generate(size) # генерация матрицы весов

matrix = []

i = 0

with open('matrix.txt', 'r') as f:

for line in f.readlines(): # устанавливаем бесконечности на главной диагонали

row = list(map(int, line.split(' ')))

row[i] = math.inf

matrix.append(row)

i += 1

show\_matrix(matrix)

print()

cost, way = nn\_algorythm(start, matrix)

print(cost) # длина пройденного пути

print(way[0], end='')

for i in range(1, len(way)): # демонстрация полученной цепочки

print('-', way[i], end='', sep='')

Название файла: generator.py

import random

def generate(size):

matrix = []

for i in range(size):

row = []

for j in range(size):

num = random.randint(1, 50)

row.append(num)

matrix.append(row)

with open('matrix.txt', 'w') as f:

for row in matrix:

f.write(' '.join(map(str, row)) + '\n')