**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

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

Тема: Задача Коммивояжёра

МВиГ: Алгоритм Литтла с модификацией

Приближённый алгоритм: АБС

**Вариант – 2**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Березовский М.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025

## Цель работы

Изучить задачу Коммивояжёра, МВиГ, Алгорит Литтла, приближённые алгоритмы решения. Применить полученные знания на практике, реализовав программный код.

## 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.

2 МВиГ: Алгоритм Литтла с модификацией: после приведения матрицы, к нижней оценке веса решения добавляется нижняя оценка суммарного веса остатка пути на основе МОД. Приближённый алгоритм: АБС.

Замечание к варианту 2: Начинать АБС со стартовой вершины.

## Выполнение работы

**Описание алгоритма для решения задачи**

**Метод ветвей и границ: Алгоритм Литтла с модификацией.**

Задача коммивояжёра (Traveling Salesman Problem, TSP) — это классическая NP-трудная задача комбинаторной оптимизации, в которой необходимо найти кратчайший маршрут, проходящий через все вершины графа ровно один раз и возвращающийся в начальную точку.

Реализованный алгоритм использует метод ветвей и границ (Branch and Bound), а конкретнее алгоритм Литтла с модификацией, который позволяет отсекать неоптимальные маршруты и минимизировать перебор возможных вариантов.

**Основные этапы алгоритма**

Редукция матрицы расстояний (reduce\_matrix) - Из каждой строки и столбца вычитается минимальный элемент, чтобы в каждой строке и столбце присутствовал хотя бы один ноль.

Это приводит к снижению общей стоимости решения и формирует нижнюю границу стоимости пути.

Выбор ребра с наибольшей оценкой (get\_greatest\_zero) - Среди нулей в редуцированной матрице выбирается наилучшее ребро, основываясь на сумме минимальных элементов в соответствующей строке и столбце. Этот выбор минимизирует будущие увеличения стоимости маршрута.

Расчёт нижней границы с использованием минимального остовного дерева (prim\_mst, calculate\_bound) - Оценка стоимости оставшегося маршрута производится с помощью минимального остовного дерева (МОД), что даёт более точную нижнюю границу.

Ветвление и поиск (branching) - Рекурсивно строится дерево решений, в котором:

Одно направление ветвления принимает выбранное ребро,

Другое — запрещает его, проверяя, есть ли более дешёвый путь.

Если в процессе поиска текущая стоимость пути превышает текущий best\_cost, ветка отсекается.

Перебор всех возможных стартовых вершин (solve\_little) - Запуск алгоритма для каждого возможного начального города и поиск глобального оптимального решения.

**АБС:**

Алгоритм ближайшего соседа (АБС) - это жадный алгоритм, который строит приближённое решение задачи коммивояжёра. В отличие от алгоритма Литтла, он не гарантирует нахождение оптимального маршрута, но выполняется за полиномиальное время и подходит для больших входных данных.

**Основные этапы алгоритма**

Инициализация - Задаётся начальный город start. Создаётся множество непосещённых городов unvisited\_vertexes, содержащее все вершины, кроме стартовой. Начальный маршрут route содержит только стартовую вершину.

Общая стоимость пути path\_cost инициализируется нулём.

Жадный выбор ближайшего города - Пока остаются непосещённые вершины, выбирается город с минимальным расстоянием от текущего (cur).

Этот город добавляется в маршрут, а его расстояние к path\_cost. Удаляется из множества unvisited\_vertexes.

Возврат в стартовую вершину - Когда все города пройдены, маршрут замыкается, добавляя start в конец. Финальная стоимость маршрута увеличивается на расстояние от последнего посещённого города до стартового.

Вывод результата **-** Возвращается найденный маршрут и его стоимость.

**Оптимизация алгоритма.**

Метод ветвей и границ позволяет значительно сократить количество перебираемых вариантов, но требует эффективного управления структурой данных и оценкой нижней границы. В коде используется несколько оптимизаций:

Редукция матрицы уменьшает вычислительные затраты, сразу снижая стоимость пути.

Использование минимального остовного дерева (МОД) в оценке нижней границы помогает быстрее отсекать заведомо невыгодные маршруты.

Глубина рекурсии ограничивается отсечением по стоимости, что предотвращает избыточное ветвление.

Работа с матрицами через NumPy, что значительно ускоряет обработку.

**Оценка сложности алгоритма во времени и памяти в нотации O.**

МВиГ:

Редукция матрицы выполняется за O(n²), так как требуется пройтись по n строкам и n столбцам.

Поиск наилучшего нуля выполняется за O(n²), так как нужно проверить все возможные ребра.

Оценка нижней границы с помощью MST в худшем случае выполняется за O(n²), используем алгоритм Прима.

Рекурсивное ветвление имеет наихудшую сложность O(n!), так как в теории рассматриваются все перестановки городов.

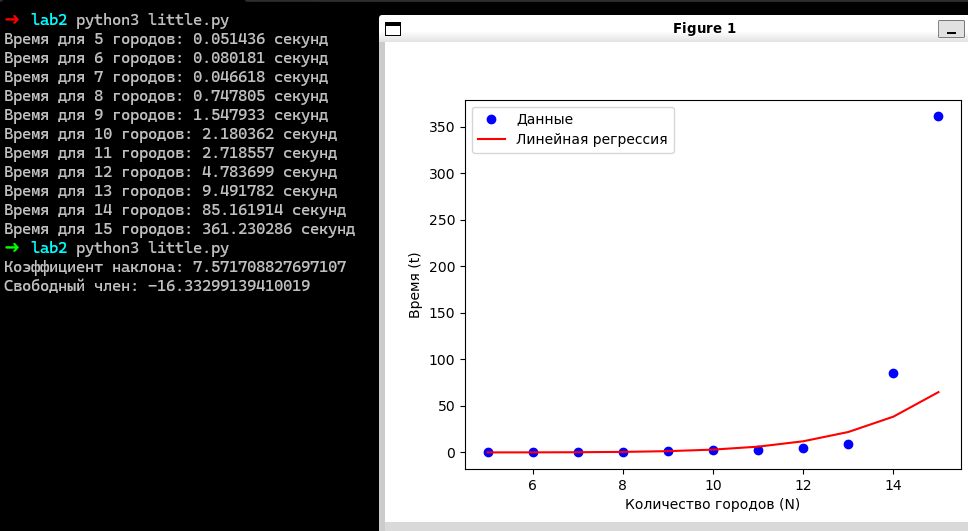
Итого, асимптотическая сложность алгоритма в худшем случае: O(n!)

**Оценка по памяти:**

Матрица n x n требует O(n²).

Рекурсивный стек может достигать глубины O(n).

В худшем случае требуется хранить O(n!) состояний, но за счёт отсечения ветвлений это число сильно уменьшается и становится O(k^n)



**АБС:**

Жадный выбор ближайшего соседа: O(n²), так как на каждом шаге происходит O(n) перебор оставшихся городов, а шагов O(n).

Общая временная сложность: O(n²)

Оценка по памяти:

Матрица n × n: O(n²).

Список маршрута O(n).

Множество непосещённых вершин O(n).

Итого, общая сложность по памяти: O(n²)

**Тестирование. Демонстрация граничных случаев алгоритма.**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 3  -1 1 3  3 -1 1  1 2 -1 | 0 1 2  3.0 | Верный вывод |
|  | 4  -1 3 4 1  1 -1 3 4  9 2 -1 4  8 9 2 -1 | 0 3 2 1  6.0 | Верный вывод |
|  | 6  -1 25 40 31 27  5 -1 17 30 25  19 15 -1 6 1  9 50 24 -1 6  22 8 7 10 -1 | 0 1 2 4 3  62.0 | Верный вывод |

**Выводы**

В зависимости от требований к точности и времени работы, следует выбирать соответствующий алгоритм. Если важно получить оптимальное решение, но допустимо долгое вычисление, то метод Литтла подходит лучше. Если же требуется быстрое приближённое решение на больших данных, то АБС является более практичным вариантом.

Разработанный программный код см. в приложении А

# Приложение А Исходный код программы

Название файла: little.py

import numpy as np

INF = np.inf

best\_cost = INF

best\_path = []

def reduce\_matrix(matrix):

delta = 0

reduced\_matrix = np.copy(matrix)

size = reduced\_matrix.shape[0]

for i in range(size):

row\_min = np.min(reduced\_matrix[i])

if row\_min == INF:

continue

delta += row\_min

reduced\_matrix[i] -= row\_min

for j in range(size):

col\_min = np.min(reduced\_matrix[:, j])

if col\_min == INF:

continue

delta += col\_min

reduced\_matrix[:, j] -= col\_min

return reduced\_matrix, delta

def get\_greatest\_zero(matrix, current\_node):

size = matrix.shape[0]

degrees = []

for j in range(size):

if matrix[current\_node][j] == 0:

row\_min = np.min(matrix[current\_node][np.arange(size) != j])

row\_min = row\_min if row\_min != INF else 0

col\_min = np.min(matrix[np.arange(size) != current\_node, j])

col\_min = col\_min if col\_min != INF else 0

degrees.append((current\_node, j, row\_min + col\_min))

return sorted(degrees, key=lambda x: -x[2])

def prim\_mst(submatrix):

size = submatrix.shape[0]

if size <= 1:

return 0

selected = {0}

mst\_cost = 0

while len(selected) < size:

min\_edge = INF

best\_vertex = None

for u in selected:

for v in range(size):

if v not in selected and matrix[u, v] < min\_edge:

min\_edge = matrix[u, v]

best\_vertex = v

selected.add(best\_vertex)

mst\_cost += min\_edge

return mst\_cost

def calculate\_bound(pieces, matrix):

if not pieces:

return 0

size = len(pieces)

submatrix = np.full((size, size), INF)

for i in range(size):

for j in range(i + 1, size):

start\_i, end\_i = pieces[i]

start\_j, end\_j = pieces[j]

w = min(matrix[end\_i][start\_j], matrix[end\_j][start\_i])

submatrix[i][j] = submatrix[j][i] = w

return prim\_mst(submatrix)

def branching(matrix, path, cost, pieces, start):

global best\_cost, best\_path

n = matrix.shape[0]

if len(path) == n:

final\_cost = cost + matrix[path[-1]][start]

if final\_cost < best\_cost:

best\_cost = final\_cost

best\_path = path + [start]

return

bound = calculate\_bound(pieces, matrix)

if cost + bound >= best\_cost:

return

current\_node = path[-1]

zero\_degrees = get\_greatest\_zero(matrix, current\_node)

if not zero\_degrees:

return

for current, next\_node, \_ in zero\_degrees:

if next\_node in path or matrix[current][next\_node] == INF:

continue

new\_pieces = pieces.copy()

merged = False

for i, (s, e) in enumerate(new\_pieces):

if e == current:

new\_pieces[i] = (s, next\_node)

merged = True

elif s == next\_node:

new\_pieces[i] = (current, e)

merged = True

if not merged:

new\_pieces.append((current, next\_node))

new\_mat = np.copy(matrix)

new\_mat[current, :] = INF

new\_mat[:, next\_node] = INF

new\_mat[next\_node, current] = INF

reduced\_mat, reduction\_cost = reduce\_matrix(new\_mat)

new\_cost = cost + matrix[current][next\_node] + reduction\_cost

branching(reduced\_mat, path + [next\_node], new\_cost, new\_pieces, start)

new\_mat\_right = np.copy(matrix)

new\_mat\_right[current][next\_node] = INF

reduced\_right, reduction\_right = reduce\_matrix(new\_mat\_right)

branching(reduced\_right, path, cost + reduction\_right, pieces, start)

break

def solve\_little(matrix):

global best\_cost, best\_path

all\_best\_cost = INF

all\_best\_path = []

n = matrix.shape[0]

for start in range(n):

best\_cost = INF

best\_path = []

reduced\_mat, init\_cost = reduce\_matrix(matrix)

branching(reduced\_mat, [start], init\_cost, [], start)

if best\_cost < all\_best\_cost:

all\_best\_cost = best\_cost

all\_best\_path = best\_path

return all\_best\_path, all\_best\_cost

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input())

matrix = []

for \_ in range(n):

row = list(map(int, input().split()))

matrix.append([float('inf') if x == -1 else float(x) for x in row])

matrix = np.array(matrix, dtype=np.float64)

path, cost = solve\_little(matrix)

print(" ".join(map(str, path[:-1])))

print(cost)

Название файла: abs.py

import numpy as np

from matrix import \*

def tsp\_abs(matrix, start):

    N = len(matrix)

    route = [start]

    path\_cost = 0

    cur = start

    unvisited\_vertexes = set(range(N)) - {start}

    while unvisited\_vertexes:

        d = []

        for v in unvisited\_vertexes:

            d.append(matrix[cur][v])

        next\_vertex = min(unvisited\_vertexes, key=lambda v: matrix[cur][v])

        path\_cost += matrix[cur][next\_vertex]

        route.append(next\_vertex)

        unvisited\_vertexes.remove(next\_vertex)

        cur = next\_vertex

    path\_cost += matrix[cur][start]

    route.append(start)

    return route, path\_cost

n = int(input("Введите количество городов (N): "))

sym = input("Симметричная матрица? (y/n): ").strip().lower() == 'y'

matrix = generate\_matrix(n, symmetric=sym)

save\_matrix(matrix, "data.txt")

loaded\_matrix = load\_matrix("data.txt")

print(loaded\_matrix)

start = int(input("Введите индекс стартовой вершины (от 0 до N-1): "))

route, cost = tsp\_abs(loaded\_matrix, start)

print("Найденный маршрут: ", route)

print("Стоимость маршрута: ", cost)

Название файла: matrix.py

import numpy as np

def generate\_matrix(n, symmetric=False):

    if symmetric:

        A = np.random.randint(1, 100, size=(n, n)).astype(float)

        A = (A + A.T) // 2

    else:

        A = np.random.randint(1, 100, size=(n, n)).astype(float)

    np.fill\_diagonal(A, np.inf)

    return A

def save\_matrix(matrix, filename="data.txt"):

    np.savetxt(filename, matrix, delimiter=',', fmt='%g')

def load\_matrix(filename="data.txt"):

    return np.loadtxt(filename, delimiter=',')