Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе №6

по курсу «Анализ Алгоритмов»

на тему: «Методы решения задачи коммивояжера»

Студент ИУ7-53Б _____ Лысцев Н. Д.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Волкова Л. Л.
(Подпись, дата) (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель лабораторной работы – описание методов решения задачи коммивояжера полным перебором и на основе муравьиного алгоритма.

Для достижения поставленной цели необходимо решить следующие задачи:

1) изучить и описать задачу коммивояжера;
2) изучить и описать методы решения задачи коммивояжера — метод полного перебора и метод на основе муравьиного алгоритма;
3) разработать и реализовать программный продукт, позволяющий решить задачу коммивояжера исследуемыми методами;
4) сравнить по времени метод полного перебора и метод на основе муравьиного алгоритма.
5) обосновать полученные результаты в отчете о выполненной лабораторной работе.

Выданный индивидуальный вариант для выполнения лабораторной работы:

- неориентированый граф;
- без элитных муравьев;
- карта городов России от Калининграда до Владивостока;
- незамкнутый маршрут.

1 Аналитический раздел

В данном разделе будет рассмотрена задача коммивояжера и будут описаны алгоритмы её решения.

1.1 Задача коммивояжера

Цель задачи коммивояжера [1] заключается в нахождении самого выгодного маршрута (кратчайшего, самого быстрого, наиболее дешевого), проходящего через все заданные точки (пункты, города) по одному разу.

Условия задачи должны содержать критерий выгодности маршрута (должен ли он быть максимально коротким, быстрым, дешевым или все вместе), а также исходные данные в виде матрицы затрат (расстояния, стоимости, времени) при перемещении между рассматриваемыми пунктами.

1.2 Алгоритм полного перебора

Рассмотрим n городов и матрицу расстояний между ними. Найдем самый короткий маршрут посещения всех городов ровно по одному разу, без возвращения в первый город:

1) число вариантов для выбора первого города равно n;

2) число вариантов для выбора второго города равно $n - 1$;

3) с каждым следующим городом число вариантов уменьшается на 1;

4) число всех вариантов маршрутра равно $n!$;

5) минимальный по сумме значений матрицы расстояний вариант маршрута — искомый.

В связи со сложностью $n!$ полный перебор вариантов занимает существенное время, а при большом количестве городов становится технически невозможным.

1.3 Муравьиный алгоритм

Идея муравьиного алгоритма [2] — моделирование поведения муравьев, связанное с их способностью быстро находить кратчайший путь и адаптироваться к изменяющимся условиям, находя новый кратчайший путь.

Муравей обладает следующими чувствами:

- `зрение` — муравей k, находясь в городе i, может оценить длину (метку) D_{ij} дуги/ребра $i-j$ и привлекательность этого ребра/дуги, описываемую формулой 1.1:

$$\eta_{ij} = 1/D_{ij}, \tag{1.1}$$

- `обоняние` — муравей чует концентрацию феромона $\tau_{ij}(t)$ на ребре/дуге $i-j$ в текущий день t.

- `память` — муравей k запоминает список/кортеж посещенных за текущий день t городов — $J_k(t)$

Муравей выполняет следующую последовательность действий, пока не посетит все города:

1) находясь в городе i, муравей k выбирает следующий город j по вероятностному правилу (формула 1.2):

$$P_{ij,k}(t) = \begin{cases} 0, & j \in J_k(t) \\ \dfrac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\sum\limits_{q \notin J_k(t)} \eta_{iq}^{\alpha} \cdot \tau_{iq}^{\beta}}, & \text{если город } j \text{ необходимо посетить} \end{cases} \tag{1.2}$$

где α — параметр влияния видимости пути, β — параметр влияния феромона, τ_{ij} — число феромона на дуге/ребре $i-j$, η_{ij} — эвристическое желание посетить город j, если муравей находится в городе i. Выбор города является вероятностным, данное правило определяет ширину зоны города j. В общую зону всех не посещенных муравьем городов бросается случайное число, которое и определяет выбор муравья;

2) муравей k проходит путь по дуге/ребре $i - j$ и оставляет на нем феромон.

Информация о числе феромона на пути используется другими муравьями для выбора пути. Те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащен феромоном. Следующие муравьи будут предпочитать именно этот путь, продолжая обогащать его феромоном.

После прохождения маршрутов всеми муравьями (ночью, перед наступлением следующего дня) значение феромона на путях обновляется в соответствии со следующим правилом (1.3):

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}, \tag{1.3}$$

где $\rho \in [0,1]$ — коэффициент испарения. Чтобы найденное локальное решение не было единственным, моделируется испарение феромона.

При этом

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \Delta\tau_{ij,k}, \tag{1.4}$$

где N — число муравьев,

$$\Delta\tau_{ij,k} = \begin{cases} 0, & \text{если ребро/дуга } i-j \text{ не посещена муравьем } k \text{ в день } t \\ \frac{Q}{L_k}, & \text{иначе} \end{cases} \tag{1.5}$$

где L_k — длина маршрута (сумма меток дуг/ребер) k-го муравья в день t, Q — квота (некоторая константа) феромона 1-го муравья на день, Q выбирается соразмерной длине наименьшего маршрута (формула 1.6).

$$Q = \frac{\sum \text{недиагональных элементов матрицы смежности}}{\text{линейный размер матрицы смежности}} \tag{1.6}$$

Поскольку вероятность 1.2 перехода в заданную точку не должна быть равна нулю, необходимо обеспечить неравенство $\tau_{ij}(t)$ нулю посредством введения дополнительного минимально возможного значения феромона τ_{min} и в случае, если $\tau_{ij}(t+1)$ принимает значение, меньшее τ_{min}, откатывать значение феромона

до этой величины τ_{min}.

Вывод

В данном разделе была рассмотрена задача коммивояжера, а также способы её решения — алгоритм полного перебора и муравьиный алгоритм.

2 Конструкторский раздел

В данном разделе будут представлены схемы алгоритма полного перебора и муравьиного алгоритма.

2.1 Разработка алгоритмов

2.1.1 Алгоритм полного перебора

На рисунке 2.1 представлена схема алгоритма полного перебора для решения задачи коммивояжера.

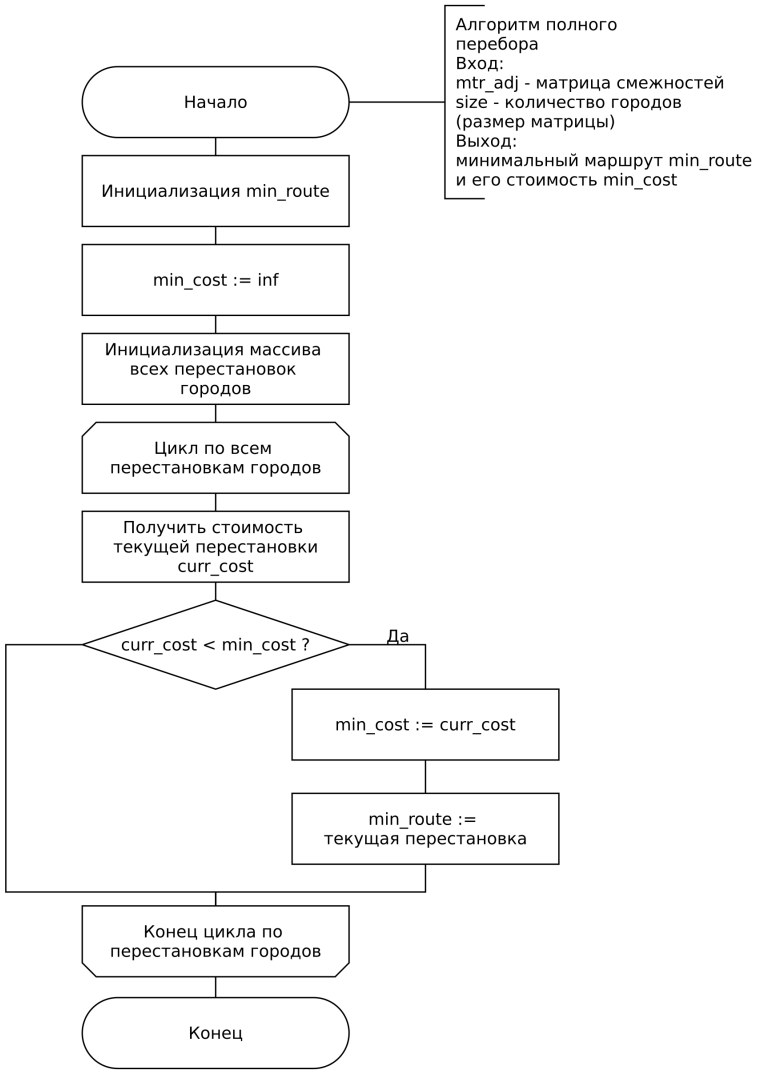

Рисунок 2.1 – Схема алгоритма полного перебора

2.1.2 Муравьиный алгоритм

На рисунке 2.2 представлена схема муравьиного алгоритма для решения задачи коммивояжера.

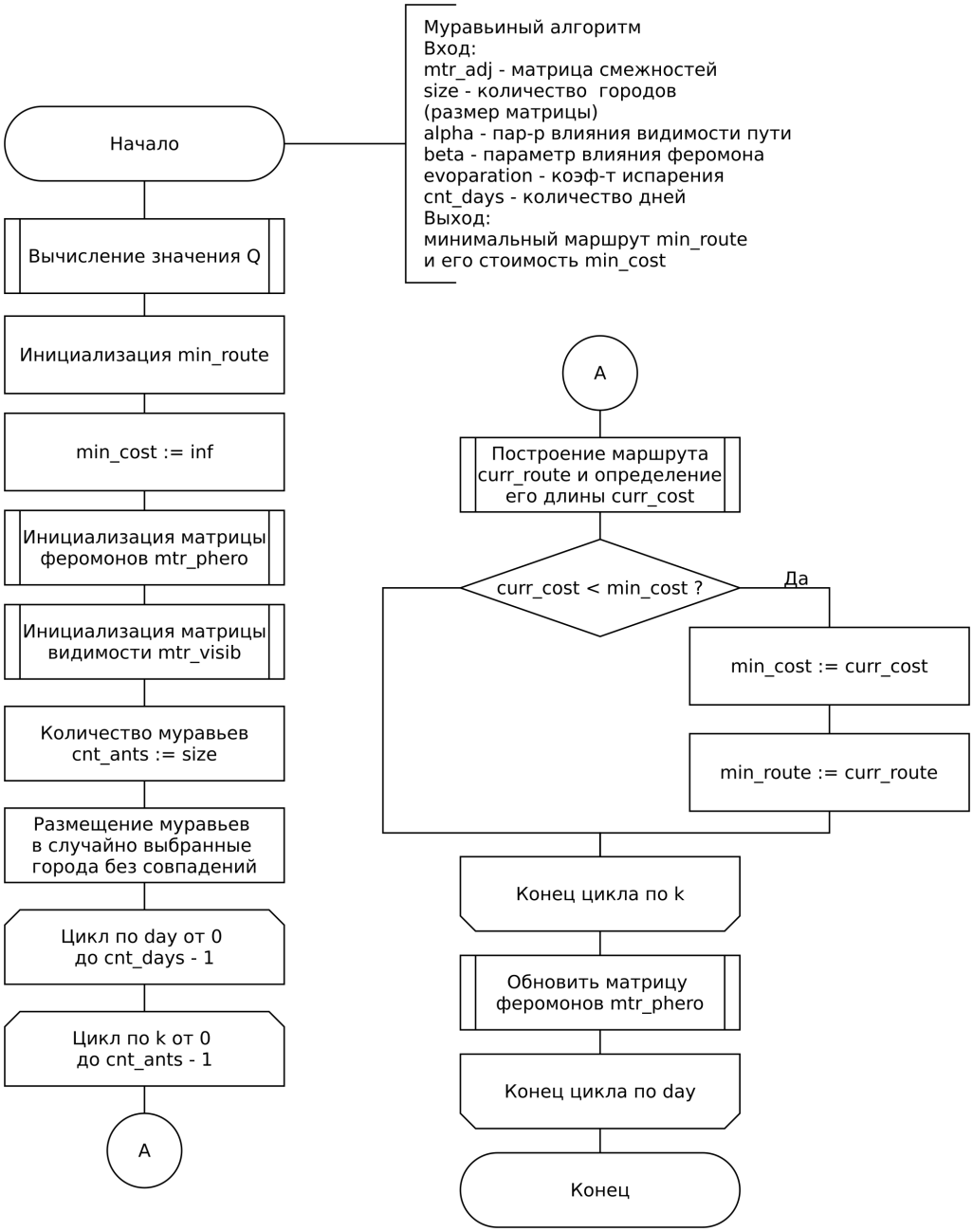

Рисунок 2.2 – Схема муравьиного алгоритма

2.2 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

Вывод

В данном разделе были представлены схемы алгоритма полного перебора и муравьиного алгоритма, была описана структура разрабатываемого ПО.

3 Технологический раздел

В данном разделе будут перечислены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к программному обеспечению

К программе предъявляется ряд требований:

- программа должна получать на вход матрицу смежности, для которой можно будет выбрать один из алгоритмов поиска оптимальных путей (полным перебором или муравьиным алгоритмом);
- программа должна позволять пользователю определять коэффициенты и количество дней для муравьиного алгоритма;
- программа должна давать возможность получить минимальную сумму пути, а также сам путь, используя один из алгоритмов.

3.2 Средства реализации

В качестве языка программирования для этой лабораторной работы был выбран *Python* [3] по следующим причинам:

- в *Python* есть библиотека *itertools* [4], содержащая функцию *permutations* для вычисления всех перестановок переданной последовательности;
- в *Python* есть библиотека *numpy* [5], предоставляющая необходимый функционал для работы с многомерными массивами;
- в *Python* есть библиотека *time* [6], содержащая функцию *process_time* для замеров процессорного времени.

В качестве функции, которая будет осуществлять замеры процессорного времени, будет использована функция *process_time* из библиотека *time* [6].

3.3 Сведения о модулях программы

Программа состоит из пяти модулей:

- `main.py` — файл, содержащий интерфейс программы и точку входа;
- `brute_force_alg.py` — файл, содержащий код алгоритма полного перебора;
- `ant_alg.py` — файл, содержащий код муравьиного алгоритма;
- `utils.py` — файл, содержащий служебные алгоритмы;
- `measurements.py` — файл, содержащий функции для выполнения замеров по времени.

3.4 Реализации алгоритмов

В листинге 3.1 представлена реализация алгоритма полного перебора.

В листингах 3.2 и 3.3 представлена реализация муравьиного алгоритма.

В листинге 3.4 представлена реализация функции вычисления константы Q.

В листинге 3.5 представлена реализация функции создания и инициализации матрицы феромонов и матрицы видимости.

В листингах 3.6 и 3.7 представлена реализация функций для подсчета необходимых вероятностей.

В листингах 3.8 и 3.9 представлена реализация функций для определения маршрута муравья.

В листингах 3.10 и 3.11 представлена реализация функции для обновления матрицы феромонов.

Листинг 3.1 – Реализация алгоритма полного перебора

```
def get_curr_cost(curr_route: list[int], mtr_adj: np.ndarray) ->
   int:
    curr_cost = 0

    for i in range(len(curr_route) - 1):
        curr_cost += mtr_adj[curr_route[i]][curr_route[i + 1]]

    return curr_cost


def brute_force_alg(mtr_adj: np.ndarray, size: int) ->
   tuple[list[int], int]:
    min_route = list()
    min_cost = float("inf")

    for perm in it.permutations(list(range(len(mtr_adj)))):
        curr_route = list(perm)

        curr_cost: int = get_curr_cost(curr_route, mtr_adj)

        if curr_cost < min_cost:
            min_route = curr_route
            min_cost = curr_cost

    return min_route, min_cost
```

Листинг 3.2 – Реализация муравьиного алгоритма (начало)

```
def ant_alg(
        mtr_adj: np.ndarray,
        size: int,
        alpha: float,
        beta: float,
        evoparation: float,
        cnt_days: int
):
    Q: float = calc_Q(mtr_adj, size)
```

Листинг 3.3 – Реализация муравьиного алгоритма (конец)

```
    min_route = list()
    min_cost = float("inf")

    mtr_phero: np.ndarray = init_mtr_phero(size)
    mtr_visib: np.ndarray = init_mtr_visib(mtr_adj, size)

    cnt_ants = size

    for day in range(cnt_days):
        ants: list[dict] = [{'tabu': [j], 'cost': 0} for j in
           range(cnt_ants)]
        for k in range(cnt_ants):
            ants[k]: dict = get_ant_route(ants[k], mtr_adj,
               mtr_phero, mtr_visib, size, alpha, beta)

            if ants[k]['cost'] < min_cost:
                min_cost = ants[k]['cost']
                min_route = ants[k]['tabu']

        mtr_phero = update_phero(mtr_phero, size, ants, Q,
           evoparation)

    return min_route, min_cost
```

Листинг 3.4 – Реализация функции для вычисления константы Q

```
def calc_Q(mtr_adj: np.ndarray, size: int) -> float:
    _sum = 0
    count = 0

    for i in range(size):
        for j in range(size):
            if i != j:
                _sum += mtr_adj[i][j]
                count += 1

    return _sum / count
```

Листинг 3.5 – Реализация функции создания и инициализации матрицы феромонов и матрицы видимости

```
def init_mtr_phero(size: int) -> np.ndarray:

    return np.ones((size, size))

def init_mtr_visib(mtr_adj: np.ndarray, size: int) -> np.ndarray:
    mtr_visib = [[(1.0 / mtr_adj[i][j] if i != j else 0) for j in
       range(size)] for i in range(size)]
    mtr_visib = np.array(mtr_visib)

    return mtr_visib
```

Листинг 3.6 – Реализация функций подсчета необходимых вероятностей (начало)

```
def get_prob_numerator(
        city_from: int,
        city_to: int,
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        alpha: float,
        beta: float
) -> float:
    P = (mtr_phero[city_from][city_to] ** alpha) *
       (mtr_visib[city_from][city_to] ** beta)

    return P

def get_probabilities(
        size: int,
        tabu_k: list,
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        alpha: float,
        beta: float
) -> np.ndarray:
    probabilities = np.zeros(size)
```

Листинг 3.7 – Реализация функций подсчета необходимых вероятностей (конец)

```
    for city in range(size):
        if city not in tabu_k:
            curr_city = tabu_k[-1]

            probabilities[city] = get_prob_numerator(curr_city,
               city, mtr_phero, mtr_visib, alpha, beta)

    denominator = sum(probabilities)

    probabilities /= denominator

    return probabilities
```

Листинг 3.8 – Реализация функции для определения следующего города

```
def get_next_city(
        tabu_k: list[int],
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        size: int,
        alpha: float,
        beta: float
) -> int:
    probabilities: np.ndarray = get_probabilities(size, tabu_k,
       mtr_phero, mtr_visib, alpha, beta)

    choice: float = random()

    probSum = probabilities[0]
    next_city = 0

    while choice > probSum and next_city < size - 1:
        next_city += 1
        probSum += probabilities[next_city]

    return next_city
```

Листинг 3.9 – Реализация функции для определения маршрута муравья

```
def get_ant_route(
        ant_k: dict,
        mtr_adj: np.ndarray,
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        size: int,
        alpha: float,
        beta: float
) -> dict:
    while len(ant_k['tabu']) != size:
        nextCity = get_next_city(ant_k['tabu'], mtr_phero,
           mtr_visib, size, alpha, beta)

        ant_k['cost'] += mtr_adj[ant_k['tabu'][-1]][nextCity]
        ant_k['tabu'].append(nextCity)

    return ant_k
```

Листинг 3.10 – Реализация функции для обновления матрицы феромонов (начало)

```
def update_phero(
        mtr_phero: np.ndarray,
        size: int,
        ants: list[dict],
        Q: float,
        evoparation: float
) -> np.ndarray:
    add_mtr_phero = np.zeros((size, size))

    for ant_k in ants:
        delta = Q / ant_k['cost']

        for i in range(size - 1):
            add_mtr_phero[ant_k['tabu'][i]][ant_k['tabu'][i + 1]]
               += delta

    mtr_phero += ((1 - evoparation) * mtr_phero + add_mtr_phero)
```

Листинг 3.11 – Реализация функции для обновления матрицы феромонов (конец)

```
    for i in range(size):
        for j in range(size):
            mtr_phero[i][j] = 0.1 if mtr_phero[i][j] <
               MIN_PHEROMONE else mtr_phero[i][j]

    return mtr_phero
```

Вывод

В данном разделе были перечислены требования к программному обеспечению, средства реализации и листинги кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задача коммивояжера — метод ветвей и границ [Электронный ресурс]. — Режим доступа: `https://galyautdinov.ru/post/zadacha-kommivoyazhera` (дата обращения: 02.02.2024).
2. М.В. Ульянов Ресурсно-эффективные компютерные алгоритмы. Разработка и анализ. — 2007.
3. Python 3.12.1 documentation [Электронный ресурс]. — Режим доступа: `https://docs.python.org/3/index.html` (дата обращения: 03.02.2024).
4. itertools - Functions creating iterators for efficient looping [Электронный ресурс]. — Режим доступа: `https://docs.python.org/3/library/itertools.html` (дата обращения: 03.02.2024).
5. NumPy documentation [Электронный ресурс]. — Режим доступа: `https://numpy.org/doc/stable/` (дата обращения: 03.02.2024).
6. time - Time access and conversions [Электронный ресурс]. — Режим доступа: `https://docs.python.org/3/library/time.html` (дата обращения: 03.02.2024).