Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЁТ №2**

**Дисциплина: Многоагентное моделирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Миков

Краснодар

2024

**Цель работы**

Написать программу на любом языке программирования, симулирующую поведение среды с агентами

**Описание задачи**

В системе есть n агентов. У каждого агента есть целевой набор патентов (может быть представлен как строка, глобальный идентификатор и т.д.)

A\_t, который ему нужно собрать для реализации некоторой глобальной задачи. Его целью является сбор всех элементов своего целевого набора.

В начале моделирования каждому агенту выдаётся непустое множество патентов A\_s. Они могут быть как из целевого набора, так и не из него.

В процессе моделирования агенты общаются между собой. Целью коммуникации является обмен патентами. При коммуникации с другим агентом, текущий спрашивает есть ли у другого один из нужных ему патентов. Если нет, то раунд коммуникации завершается. Если у другого агента есть нужный патент, то он готов им поделиться только в обмен на один из нужных для себя. Если у текущего агента есть один из подходящих для второго патентов, обмен завершается. Если агент собрал свой целевой набор, то он может безвозмездно делиться своими патентами.

Цикл моделирования заканчивается, когда у всех агентов собран целевой набор. В результате необходимо вывести список агентов с размером

их целевого набора и количества итераций и раундов коммуникаций, которые ему потребовались для его сбора.

Ремарки

Раундом коммуникации считается общение агента с агентом. Например, агент 1 пообщался с агентом 2 и агентом 3. Тогда у агента 1 будет

два раунда, а у второго и третьего по одному.

Для генераций начальных условий агента (целевой набор и множество патентов) можно сначала сгенерировать каждому агенту свой набор, затем

объединить получившиеся множества в одно, и раздать его в случайном порядке. По итогам раздачи размеры множеств A\_s каждого агента должны быть примерно одинаковыми.

**Описание решения**

Для реализации данной задачи был выбран Python, а также объектно-ориентированный подход.

Было создано 3 файла: task.txt (в котором содержалось описание задачи), classes.py (в котором содержались все необходимые классы) и main.py (в котором содержался основная логика).

Classes.py

Было решено написать 2 класса: Агент и Коробка Сообщений.

Агенты – основные объекты, который общаются между собой, а коробка сообщений – глобальный контролёр коммуникаций. Класс агент состоит из конструктора с полями: id (id агента), target\_task (целевая задача агента), important\_patents (важные патенты, которые нужны агенту для выполнения задачи; агент не меняется этими патентами), useless\_patents (ненужные для целевой задачи агента патенты; этими патентами агент может меняться), number\_of\_iteration (кол-во итераций, которые прошли, пока агент не выполнил целевую задачу) и number\_of\_communication (кол-во коммуникаций, после которого агент выполнил целевую задачу).

Методы класса Агент:

Info(): метод, который печатает основную информацию об агенте (id, целевую задачу, собранные, нужные и ненужные патенты);

Obtaining\_a\_patent(): метод, который проверяет, является ли полученный патент важным, и в зависимости от результата помещает его в один из списков;

Swap(): метод, который вызывается при получении агента патентом (увеличивает число коммуникаций и вызывает метод obtaining\_a\_patent());

Update\_state(): метод, который увеличивает кол-во итераций агента;

Checking\_task(): метод, который составляет словарь отсутствующих нужных патентов агента;

Check\_completion(): метод, который проверяет, выполнил ли агент свою целевую задачу;

Win\_info(): метод, который печатает итоговую информацию об агенте

MessageBox – ящик сообщений – ключевой объект в данной реализации программы, потому что именно в этом ящике происходят вычисления пар агентов для обмена, а также реализован механизм решения проблемы отсутствия прямого взаимного обмена между агентами. У самого ящика нет полей, но есть методы:

Consilium(): один из ключевых методов, отвечает за нахождение пар для прямого взаимного обмена между агентами (достигается перебором с условиями: берётся первый агент, и для него перебираются агенты, среди которых выбирается тот, у которого среди ненужных есть хотя бы 1 патент, который нужен первому агенту; затем для второго агента ищутся нужные патенты, и среди ненужных патентов первого агенты ищется хоты бы 1 такой; после этих действия выполняется прямой обмен); если же не удаётся найти пару для прямого обмена, то начинает составляться цепочка обменов для данного агента;

Trade(): метод, который последовательно удаляет патент у первого агента и добавляет его второму, а затем наоборот (запускается при прямом обмене);

Find\_chain(): метод, который циклично подбирает агентов по особому правилу. Ключевая особенность метода заключается в том, что из-за особенности генерации и раздачи патентов агентам всегда можно составить такую цепь, что она замкнётся на начальном агенте (если отсутствует возможность выполнить прямой обмен между первым агентом и ещё каким-то), т.е. выбирается первый агент, для которого нельзя выполнить прямой обмен; далее подбирается такой агент, среди нужных патентов которого есть тот патент, который числится ненужным у первого агента; далее выбирается третий агент так, чтобы среди нужных для него патентов был такой, который является ненужным у второго агента, и так далее;

Find\_next\_agent(): метод, который циклично вызывается в Find\_chаin; в данном методе ищется новый агент так, чтобы среди нужных ему патентов был тот, который есть у старого агента как ненужный;

Chain\_exchange(): метод, который выполняет последовательный обмен в цепочке агентов. Данный метод берёт первого агента в цепочке, удаляет у него тот патент, который является ненужным для него, но нужным для следующего в цепочке агента, и затем добавляет этот патент этому следующему агенту.

Main.py

Данный файл импортирует все классы из вышеописанного, а также содержит 2 основных функции:

Create\_agents(): данная функция создаёт нужное количество агентов и выдаёт им патенты по следующему правилу: при генерации агентов у каждого из них создаётся целевая задача; создаётся глобальный список, который содержит все патенты всех целевых задач; этот список перемешивается, и патенты начинают поочерёдно выдаваться всем агентам;

Main(): основная функция, в которой создаётся ящик сообщений и запускается основной цикл, в котором агенты поочерёдно коммуницируют, а по завершению сбора целевых задач печатают итоговую информацию.

**Код программы**

classes.py

import random as rnd

from collections import Counter

target\_str = 'abcdefghijklmnopqrstuvwxyz'

len\_target\_task = 5

class Agent:

    def \_\_init\_\_(*self*, *id\_agent*: int) -> None:

        """

        Конструктор, который создаёт объект типа Агент с нужными полями

        Args:

            id\_agent (int): id агента

        """

*self*.id = *id\_agent*                                                                               *# задаём агенту id*

*self*.target\_task = sorted([target\_str[rnd.randint(0, len(target\_str) - 1)] for i in range(len\_target\_task)])   *# создаём случайный целевой список*

*self*.important\_patents = []                                                                      *# патенты, которые агент не отдаст*

*self*.useless\_patents = []                                                                        *# патенты, которыми агент может обменяться*

*self*.number\_of\_iteration = 0                                                                     *# количество итераций агента*

*self*.number\_of\_communications = 0                                                                *# количество коммуникаций агента*

    def info(*self*) -> None:

        """

        Метод, который печатает основную информацию об агенте

        """

        missing\_important\_patents = *self*.checking\_task()       *# получение словаря недостающих патентов*

        mis\_imp\_pat\_list = []                                  *# создание списка недостающих патентов*

        for key, value in missing\_important\_patents.items():   *# перебор ключей и значений словаря недостающих патентов*

            mis\_imp\_pat\_list.extend(key \* value)               *# заполнение списка недостающих патентов*

        mis\_imp\_pat\_list = sorted(mis\_imp\_pat\_list)            *# сортировка списка недостающих патентов*

        print(f"Агент: id = {*self*.id}; целевая задача: {*self*.target\_task}; собранные патенты: {*self*.important\_patents}; ненужные патенты: {*self*.useless\_patents}, нужные патенты: {mis\_imp\_pat\_list}")

    def obtaining\_a\_patent(*self*, *patent*: str) -> None:

        """

        Метод, который проверяет, является ли полученный патент важным, и в зависимости от результата помещает его в один из списков

        Args:

            patent (str): Получаемый патент

        """

        count\_patent = *self*.target\_task.count(*patent*)                                  *# считаем количество таких патентов среди целевых*

        if count\_patent > 0 and *self*.important\_patents.count(*patent*) < count\_patent:   *# если такой патент является целевым и среди важных таких патентов не хватает*

*self*.important\_patents.append(*patent*)                                      *# то добавляем патент к важным*

*self*.important\_patents = sorted(*self*.important\_patents)                    *# сортируем список*

        else:                                                                          *# иначе*

*self*.useless\_patents.append(*patent*)                                        *# добавляем патент к бесполезным*

*self*.useless\_patents = sorted(*self*.useless\_patents)                        *# сортируем список*

    def swap(*self*, *patent*: str) -> None:

        """

        Метод, который вызывается при получении патента агентом

        Args:

            patent (str): Получаемый патент

        """

*self*.number\_of\_communications += 1   *# увеличиваем количество коммуникаций*

*self*.obtaining\_a\_patent(*patent*)      *# вызываем метод получение патента*

    def update\_state(*self*) -> None:

        """

        Метод, который увеличивает количество итераций агента

        """

*self*.number\_of\_iteration += 1   *# увеличиваем количество итераций*

    def checking\_task(*self*) -> dict:

        """

        Метод, который составляет словарь отсутствующих нужных патентов агента

        Returns:

            dict: Словарь отсутствующих патентов

        """

        target\_patent\_count = Counter(*self*.target\_task)                    *# сколько раз патенты встречаются в целевой задаче*

        collected\_patent\_count = Counter(*self*.important\_patents)           *# сколько раз патенты уже собраны*

        missing\_patents = {}                                               *# результирующий словарь*

        for patent, target\_count in target\_patent\_count.items():           *# перебор пар ключ-значение в целевых патентах*

            collected\_count = collected\_patent\_count.get(patent, 0)        *# получаем сколько уже собрано патентов, или 0, если ничего не собрано*

            if collected\_count < target\_count:                             *# если собрано меньше, чем нужно*

                missing\_patents[patent] = target\_count - collected\_count   *# добавляем недостающие патенты*

        return missing\_patents                                             *# возвращаем составленный словарь*

    def check\_completion(*self*) -> bool:

        """

        Метод, который проверяет, выполнил ли агент свою целевую задачу

        """

        if dict(Counter(*self*.target\_task)) == dict(Counter(*self*.important\_patents)):   *# проверка совпадения словарей целевой задачи и имеющихся важных патентов*

            return True

        else:

            return False

    def win\_info(*self*) -> None:

        """

        Метод, который печатает результирующую информацию при победе агентов

        """

        print(f"Агенту {*self*.id} для сбора целевого набора {*self*.target\_task} потребовалось {*self*.number\_of\_iteration} итераций и {*self*.number\_of\_communications} коммуникаций")

class Message\_Box:

    def \_\_init\_\_(*self*) -> None:

        pass

    def consilium(*self*, *list\_of\_agents*: list) -> None:

        """

        Метод, который ищет пару агентов, которые могут поменяться патентами

        Args:

            list\_of\_agents (list): Список всех агентов

        """

        list\_of\_all\_useless\_patents = []                                                                        *# список словарей всех ненужных патентов агентов*

        successful\_trade = False

        for i in range(len(*list\_of\_agents*)):                                                                    *# перебираем всех агентов*

            patents\_count = Counter(*list\_of\_agents*[i].useless\_patents)                                          *# считаем кол-во каждого ненужного патента*

            list\_of\_all\_useless\_patents.append(dict(patents\_count))                                             *# добавляем в общий список словарь для текущего агента*

        for i in range(len(*list\_of\_agents*)):                                                                    *# перебираем всех агентов*

            missing\_patents\_i = *list\_of\_agents*[i].checking\_task()                                               *# получаем словарь недостающих патентов для i-агента*

            for j in range(len(*list\_of\_agents*)):                                                                *# перебираем остальных агентов*

                if i != j:                                                                                      *# проверка на разных агентов*

                    if any(patent in list\_of\_all\_useless\_patents[j] for patent in missing\_patents\_i):           *# если среди ненужных патентов j-го агента содержится хотя бы 1 из недостающих i-го агента*

                        missing\_patents\_j = *list\_of\_agents*[j].checking\_task()                                   *# получаем словарь недостающих патентов для j-го агента*

                        if any(patent in list\_of\_all\_useless\_patents[i] for patent in missing\_patents\_j):       *# проверяем аналогично патенты для j-го агента*

*self*.trade(*list\_of\_agents*[i], *list\_of\_agents*[j])                                    *# если нашли такую пару, то выполняем для них обмен*

                            list\_of\_all\_useless\_patents[i] = dict(Counter(*list\_of\_agents*[i].useless\_patents))   *# обновляем словарь ненужных патентов i-му агенту*

                            list\_of\_all\_useless\_patents[j] = dict(Counter(*list\_of\_agents*[j].useless\_patents))   *# обновляем словарь ненужных патентов j-му агенту*

                            successful\_trade = True

                            break                                                                               *# выходим из цикла*

        if not successful\_trade:                                                                                *# если не нашлось ни одной пары для обмена*

            for i in range(len(*list\_of\_agents*)):                                                                *# перебор агентов*

                if not *list\_of\_agents*[i].check\_completion():                                                    *# проверка, что текущий агент не выполнил целевую задачу*

                    print(f"Для агента {*list\_of\_agents*[i].id} прямой обмен невозможен, ищем цепочку обменов")   *# уведомление*

                    chain = *self*.find\_chain(*list\_of\_agents*, i, list\_of\_all\_useless\_patents)                     *# строим цепочку обменов*

                    if chain:                                                                                   *# если цепь нашлась*

                        print(f"Цепочка нашлась: {chain}")                                                      *# то выводим её*

    def trade(*self*, *agent\_1*: Agent, *agent\_2*: Agent):

        """

        Метод обмена патентами между двумя агентами

        Args:

            agent\_1 (Agent): Первый агент

            agent\_2 (Agent): Второй агент

        """

        patent\_1 = 0

        for patent in *agent\_1*.useless\_patents:           *# перебираем ненужные патенты первого агента*

            if patent in *agent\_2*.checking\_task():        *# проверяем, если патент нужен второму агенту*

*agent\_1*.useless\_patents.remove(patent)   *# удаляем патент из ненужных у первого агента*

*agent\_2*.swap(patent)                     *# добавляем патент второму агенту*

                patent\_1 = patent                        *# находим первый патент*

                break                                    *# один обмен произошёл, выходим*

        patent\_2 = 0

        for patent in *agent\_2*.useless\_patents:           *# перебираем ненужные патенты второго агента*

            if patent in *agent\_1*.checking\_task():        *# проверяем, если патент нужен первому агенту*

*agent\_2*.useless\_patents.remove(patent)   *# удаляем патент из ненужных у второго агента*

*agent\_1*.swap(patent)                     *# добавляем патент первому агенту*

                patent\_2 = patent                        *# находим второй патент*

                break                                    *# второй обмен произошёл, выходим*

        print(f"Агенты {*agent\_1*.id} и {*agent\_2*.id} поменялись патентами {patent\_1} и {patent\_2}")

*agent\_1*.info()

*agent\_2*.info()

    def find\_chain(*self*, *list\_of\_agents*: list, *i*: int, *list\_of\_all\_useless\_patents*: list) -> None:

        """

        Метод, который циклично подбирает агентов так, чтобы они могли составить цепь обменов

        Args:

            list\_of\_agents (list): Список всех агентов

            i (int): Номер текущего агента в списке

            list\_of\_all\_useless\_patents (list): Список словарей всех ненужных патентов

        """

        print(f"Начинается поиск цепи для агента {*list\_of\_agents*[*i*].id}")                                       *# уведомление*

        current\_agent = *list\_of\_agents*[*i*]                                                                       *# текущий агент*

        path = [*list\_of\_agents*[*i*].id]                                                                           *# путь, по которому будем делать обмен*

        visited\_agents = []                                                                                     *# посещённые агенты (текущего не вносим, потому что им должен закончиться путь)*

        new\_current\_agent = *self*.find\_next\_agent(*list\_of\_agents*, current\_agent, path, visited\_agents)           *# получаем нового агента в цепочке*

        while path[0] != path[-1]:                                                                              *# пока первый и последний агенты в цепи не одинаковые*

            new\_current\_agent = *self*.find\_next\_agent(*list\_of\_agents*, new\_current\_agent, path, visited\_agents)   *# продолжаем находить новых агентов*

        print(f"Нашлась цепочка агентов: {path}")                                                               *# уведомление*

*self*.chain\_exchange(*list\_of\_agents*, path, *list\_of\_all\_useless\_patents*)                                  *# запуск цепочки обменов*

    def find\_next\_agent(*self*, *list\_of\_agents*: list, *current\_agent*: Agent, *path*: list, *visited\_agents*: list) -> Agent:

        """

        Метод, который ищет агента, которому нужен ненужный патент текущего агента

        Args:

            list\_of\_agents (list): Список всех агентов

            current\_agent (Agent): Агент, для которого ищется "нуждающийся"

            path (list): Список индексов агентов, по которым идёт поиск цепочки

            visited\_agents (list): Список посещённых агентов

        Returns:

            agent: "нуждающийся" агент для текущего

        """

        for useless\_patent in *current\_agent*.useless\_patents:                                         *# перебираем ненужные патенты текущего агента*

            for i in range(len(*list\_of\_agents*)):                                                     *# начинаем перебирать всех агентов*

                if *list\_of\_agents*[i] != *current\_agent* and *list\_of\_agents*[i] not in *visited\_agents*:   *# проверяем, чтобы агенты были разные и ещё не посещённые*

                    if useless\_patent in *list\_of\_agents*[i].checking\_task():                          *# нужен ли новому агенту патент*

*path*.append(*list\_of\_agents*[i].id)                                            *# записываем нового агента в путь*

*visited\_agents*.append(*list\_of\_agents*[i])                                     *# записываем нового агента в посещённые*

                        print(f"Следующий агент для цепочки найден: {*list\_of\_agents*[i].id}")         *# уведомление*

                        return *list\_of\_agents*[i]                                                     *# возвращаем нового агента*

    def chain\_exchange(*self*, *list\_of\_agents*: list, *path*: list, *list\_of\_all\_useless\_patents*: list) -> None:

        """

        Метод, который совершает обмены по цепочке

        Args:

            list\_of\_agents (list): Список всех агентов

            path (list): Путь обмена агентов

            list\_of\_all\_useless\_patents (list): Список словарей всех ненужных патентов

        """

        for i in range(len(*path*) - 1):                                                                                           *# начинаем идти по цепочке обменов*

            for j in range(len(*list\_of\_agents*)):                                                                                 *# перебираем агентов*

                if *list\_of\_agents*[j].id == *path*[i]:                                                                              *# ищем подходящего агента*

                    useless\_patent = *list\_of\_agents*[j].useless\_patents[0]                                                        *# (костыль, но с ним почему-то всё работает) берём первый ненужный патент текущего агента*

*list\_of\_agents*[j].useless\_patents.remove(useless\_patent)                                                     *# удаляем этот патент*

                    for k in range(len(*list\_of\_agents*)):                                                                         *# начинаем перебирать агентов*

                        if *list\_of\_agents*[k].id == *path*[i+1]:                                                                    *# ищем подходящего агента*

*list\_of\_agents*[k].swap(useless\_patent)                                                               *# даём ему тот патент, который удалили у предыдущего агента*

                            print(f"Агент {*list\_of\_agents*[j].id} отдал патент {useless\_patent} агенту {*list\_of\_agents*[k].id}")   *# уведомление*

*list\_of\_agents*[i].info()                                                                             *# уведомление*

*list\_of\_agents*[k].info()                                                                             *# уведомление*

*list\_of\_all\_useless\_patents*[j] = dict(Counter(*list\_of\_agents*[j].useless\_patents))                    *# корректируем общий список словарей ненужных патентов*

main.py

from classes import \*

import random as rnd

import time

def create\_agents(*n*: int, *list\_of\_agents*: list) -> None:

    """

    Функция, которая создаёт агентов и выдаёт им случайные патенты

    Args:

        n (int): Кол-во агентов

        list\_of\_agents (list): Список всех агентов

    """

    for i in range(*n*):                          *# создаём n агентов*

        agent = Agent(i + 1)                    *# создаём агента*

        agent.info()                            *# выводим информацию о нём*

*list\_of\_agents*.append(agent)            *# добавляем его в общий список*

    all\_patents = []                            *# список для всех патентов*

    for agent in *list\_of\_agents*:                *# перебираем всех агентов*

        for patent in agent.target\_task:        *# перебираем все целевые патенты каждого агента*

            all\_patents.append(patent)          *# добавляем эти патенты в общий список*

    print("Все патенты:", all\_patents)

    rnd.shuffle(all\_patents)                    *# перемешиваем все патенты*

    print("Перемешанные патенты:", all\_patents)

    patents\_per\_agent = len(all\_patents) // *n*   *# определяем по сколько патентов выдадим каждому агенту*

    for i in range(patents\_per\_agent):          *# начинаем цикл раздачи*

        for agent in *list\_of\_agents*:            *# выдаём для каждого агента*

            patent = all\_patents.pop(0)         *# удаляем первый патент из списка*

            agent.obtaining\_a\_patent(patent)    *# даём этот патент агенту*

    for agent in *list\_of\_agents*:                *# перебираем всех агентов*

        agent.info()                            *# печатаем информацию*

def main(*list\_of\_agents*: list, *t\_start*: int) -> None:

    """

    Функция, отвечающая за проведение симуляции

    Args:

        list\_of\_agents (list): Список всех агентов

        t\_start (int): Счётчик итераций

    """

    message\_box = Message\_Box()                                                                                      *# создание объекта-коробки сообщений*

    while len([agent for agent in *list\_of\_agents* if len(agent.target\_task) == len(agent.important\_patents)]) != n:   *# запуск основного цикла*

*t\_start* += 1                                                                                                 *# увеличение счётчика итераций*

        print(f"Текущая итерация: {*t\_start*}")                                                                        *# уведомление*

        message\_box.consilium(*list\_of\_agents*)                                                                        *# основная функция коммуникации агентов*

        for agent in *list\_of\_agents*:                                                                                 *# перебор агентов*

            if (len(agent.target\_task) != len(agent.important\_patents)) or agent.number\_of\_iteration == 0:           *# условие изменения состояния агентов*

                agent.update\_state()                                                                                 *# изменение состояния агента*

            agent.info()                                                                                             *# уведомление*

        time.sleep(0.5)                                                                                              *# удобство вывода*

    for agent in *list\_of\_agents*:                                                                                     *# перебор агентов*

        agent.update\_state()                                                                                         *# итоговое увеличение итерации*

        agent.win\_info()                                                                                             *# итоговая информация*

n = 5                 *# кол-во агентов*

list\_of\_agents = []   *# список всех агентов*

t\_start = 0           *# счётчик итераций симуляции*

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

    create\_agents(n, list\_of\_agents)

    main(list\_of\_agents, t\_start)

**Примеры вывода**



