Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности
Кафедра высшей математики

ИНДИВИДУАЛЬНАЯ РАБОТА
по дисциплине «Методы оптимизации»

Программная реализация алгоритма решения транспортной задачи и проведение тестирования разработанной программы на заданной выборке примеров

Выполнил
студент группы КТбо3-1 _____ Беридзе И. Д.
Приняла
Доцент ИКТИБ _____ Липко Ю. Ю.

Таганрог 2023

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Транспортные задачи — это специальный класс задач из разделов линейного программирования. В задачах чаще всего описываются перевозки какого-либо товара из пункта отправления (как правило, с места производства) в пункт назначения (склад или магазин). Также при решении транспортных задач должны учитываться ограничения, накладываемые на объёмы грузов в пунктах отправления, потребность в пунктах назначения. Кроме ограничений, стоит учитывать и стоимость перевозки по заданным маршрутам.

Задачей транспортных задач является оптимизация перевозок с учётом всех ограничений. С помощью программной реализации можно существенно сократить время на вычисление целевой функции, описывающей минимальные затраты с учётом всех условий и ограничений.

ЦЕЛИ И ЗАДАЧИ

Цель работы — программная реализация алгоритма решения транспортной задачи и тестирование на заданной выборке примеров.

Задачи работы:

1. Реализовать алгоритм Северо-Западного угла для решения транспортной задачи;
2. Реализовать алгоритм минимального транспортного тарифа для решения транспортной задачи;
3. Найти целевую функцию с помощью одного из двух представленных выше алгоритмов.

АЛГОРИТМ

Для удобства восприятия алгоритм, переведённый в программу на языке Python3, представлен ниже в виде блок-схемы.

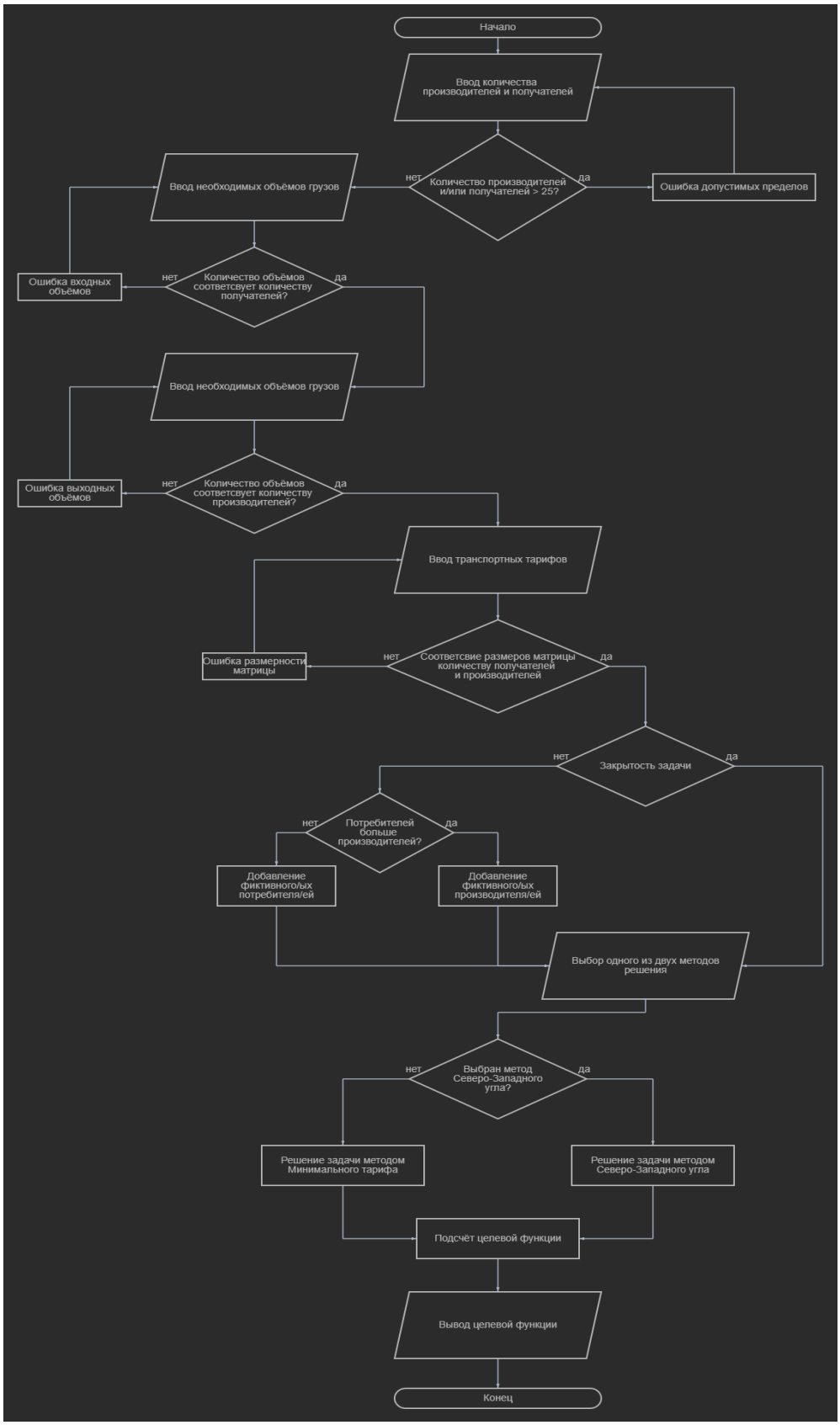

Рисунок 1 - Блок-схема

СТРУКТУРА ПРОГРАММЫ

Проект содержит в себе два файла: файл класса Transport.py и исполняемый файл main.py.

В файле Transport.py реализован класс, включающий необходимые поля (количество получателей, производителей, объёмы входных и выходных грузов, матрица транспортных тарифов, открытость/закрытость задачи, матрица с итоговым планом, целевая функция), методы класса и вспомогательные функции.

В файле main.py реализована работа с пользователем из окна командой строки.

ЛИСТИНГ ПРОГРАММЫ

Файл Transport.py:

```
# Задание условий
class Transport:

    def __init__(self):
        self.makers = 5  # Количество отправителей (производителей)
        self.customers = 3  # Количество получателей (заказчиков)
        self.needs = [60, 60, 50]  # Необходимые получателям объёмы
        self.volume = [30, 20, 55, 40, 25]  # Имеющиеся у отправителей объёмы
        self.rate = []  # Транспортный тариф
        self.close = True  # Закрытость задачи
        self.vol_out = []  # Итоговый план
        self.f_min = 0  # Целевая функция

# Установка количества производителей и заказчиков
    def m_c_set(self, make, cust):
        if make > 25 or cust > 25:
            print('Слишком много получателей и/или отправителей!')
            print('     Принимается не больше 25!\n')
            return 0
        else:
            self.makers = make
            self.customers = cust
            return 1

    # def extend(self, need_or_vol):
    #     t_need = "Введите необходимые объёмы грузов:"
    #     t_vol = "Введите имеющиеся объёмы грузов:"
    #
    #     while True:
    #         print(t_need)
    #         try:
    #

# Установка объёмов необходимых и имеющихся грузов
    def n_v_set(self):
        self.needs = []
        self.volume = []

        self.separate()
        while True:
            print("Введите необходимые объёмы грузов:")
            try:
                self.needs = list(map(int, input().split()))
            except ValueError:
                print("    Ошибка! Вы ввели не число!")
            else:
                if len(self.needs) == self.customers:
                    print("Принято!")
                    self.separate()
                    break

                else:
                    self.needs = []
                    print("Недопустимый ввод!")
        print(self.needs)

        self.separate()
        while True:
```

```
            print("Введите имеющиеся объёмы грузов:")
            try:
                self.volume = list(map(int, input().split()))
            except ValueError:
                print("    Ошибка! Вы ввели не число!")
            else:
                if len(self.volume) == self.makers:
                    print("Принято!")
                    self.separate()
                    break

                else:
                    self.volume = []
                    print("Недопустимый ввод!")
        print(self.volume)

# Установка транспортных тарифов
    def set_rate(self):
        self.rate = []

        self.separate()
        while True:
            print("Вводите построчно транспортные тарифы:\n")
            try:
                for i in range(0, self.customers):
                    self.rate.append(list(map(int, input().split())))
                    if len(self.rate[i]) != self.makers:
                        self.rate = []
                        print("    Недопустимый ввод!")
                        raise StopIteration
            except ValueError:
                print("    Ошибка! Вы ввели не число!")
            except StopIteration:
                pass
            else:
                print("Принято!")
                self.separate()
                break
        print(self.rate)

# Проверка на открытость или закрытость задачи
    def is_close(self):
        cust_vol_sum = 0
        make_vol_sum = 0
        for L in self.needs:
            cust_vol_sum += L
        for R in self.volume:
            make_vol_sum += R
        self.separate()
        if cust_vol_sum == make_vol_sum:  # Задача закрыта. Фиктивные отправители или
получатели не требуются
            print(f'Задача закрыта (сумма = {cust_vol_sum}).')
            for i in range(0, len(self.rate)):
                for j in range(0, len(self.rate[i])):
                    print(self.rate[i][j], end=' ')
                print()
        elif cust_vol_sum < make_vol_sum:  # Задача открыта. Необходимо добавить фик-
тивного заказчика
            self.close = False
            dif = make_vol_sum - cust_vol_sum
            print(f'Задача открыта (лишние {dif} у поставщиков).')
            self.customers += 1
```

```
            self.needs.append(dif)
            self.rate.insert(len(self.rate), [0 * i for i in range(self.makers)])
            for i in range(0, len(self.rate)):
                for j in range(0, len(self.rate[i])):
                    print(self.rate[i][j], end=' ')
                print()
        else:  # Задача открыта. Необходимо добавить фиктивного поставщика
            self.close = False
            dif = cust_vol_sum - make_vol_sum
            print(f'Задача открыта (недостающие {dif} у заказчиков).')
            self.makers += 1
            self.volume.append(dif)
            for i in range(0, self.customers):
                self.rate[i].append(0)
            for i in range(0, len(self.rate)):
                for j in range(0, len(self.rate[i])):
                    print(self.rate[i][j], end=' ')
                print()
        self.separate()

# Создание матрицы перевозимого объёма груза
    def volume_now(self):
        self.vol_out = [[-1] * len(self.rate[i]) for i in range(len(self.rate))]
        for i in range(0, len(self.vol_out)):
            for j in range(0, len(self.vol_out[i])):
                self.vol_out[i][j] = -1

# Метод Северо-Западного угла
    def nort_west(self):
        print("Метод Северо-Западного угла")
        for i in range(0, len(self.vol_out)):
            for j in range(0, len(self.vol_out[i])):
                self.check_set(i, j)
        self.normal_set()
        self.separate()
        self.plan_out()
        self.separate()
        self.func_out()

# Метод наименьшего транспортного тарифа
    def min_way(self):
        min_now = 1
        max_tt = self.max_tt()
        while (self.volume != [0] * len(self.volume)) and (self.needs != [0] *
len(self.needs)):
            for i in range(0, len(self.rate)):
                for j in range(0, len(self.rate[i])):
                    if self.rate[i][j] == min_now:
                        self.check_set(i, j)
                    elif self.rate[i][j] == 0 and min_now == max_tt:
                        self.check_set(i, j)
                    else:
                        continue
            min_now += 1
        self.normal_set()
        self.separate()
        self.plan_out()
        self.separate()
        self.func_out()

# Функция поиска максимального тт
    def max_tt(self):
```

```
        tt_max = 0
        for i in range(0, len(self.rate)):
            for j in range(0, len(self.rate[i])):
                if self.rate[i][j] > tt_max:
                    tt_max = self.rate[i][j]
        return tt_max

# Функция распределения объёмов грузов
    def check_set(self, i, j):
        if self.needs[i] != 0 and self.volume[j] != 0:  # Проверка необходимости
            if self.volume[j] - self.needs[i] < 0:  # Если у поставщика меньше груза,
чем требует получатель
                self.vol_out[i][j] = self.volume[j]
                self.needs[i] -= self.volume[j]
                self.volume[j] = 0
            elif self.volume[j] - self.needs[i] == 0:  # Если имеющийся объём постав-
щика равен требуемому
                self.vol_out[i][j] = self.needs[i]
                self.needs[i] = 0
                self.volume[j] = 0
            else:  # Если у поставщика больше груза, чем требует получатель
                self.vol_out[i][j] = self.needs[i]
                self.volume[j] -= self.needs[i]
                self.needs[i] = 0

# Функция нормализации итогового плана
    def normal_set(self):
        for i in range(0, len(self.vol_out)):
            for j in range(0, len(self.vol_out[i])):
                if self.vol_out[i][j] == -1:
                    self.vol_out[i][j] = 0

# Вывод итогового плана
    def plan_out(self):
        print("Итоговый план:\n")
        for i in range(0, len(self.vol_out)):
            for j in range(0, len(self.vol_out[i])):
                print(self.vol_out[i][j], end=' ')
            print()

# Вывод целевой функции
    def func_out(self):
        for i in range(0, len(self.vol_out)):
            for j in range(0, len(self.vol_out[i])):
                self.f_min += self.vol_out[i][j] * self.rate[i][j]
        print(f'f(x) -> min = {self.f_min}')

# Разделитель
    @staticmethod
    def separate():
        print('-'*32)
```

Файл main.py:

```
from Transport import Transport

A = Transport()
A.separate()
makers = 0
customers = 0
while True:
    try:
        makers = int(input("Введите количество производителей: "))
        customers = int(input("Введите количество потребителей: "))
    except ValueError:
        print("    Ошибка! Вы ввели не число!")
    else:
        if A.m_c_set(makers, customers) == 1:
            break

A.n_v_set()
A.set_rate()
A.is_close()
A.volume_now()

method = -1
while True:
    try:
        print("Выберите один из методов:")
        print(" Метод Севере-западного угла - 0")
        print(" Метод минимального тт - 1")
        method = int(input("Выбранный метод - "))
    except ValueError:
        print("    Ошибка! Неправильный ввод!")
    else:
        if method == 0:
            A.nort_west()
            break
        elif method == 1:
            A.min_way()
            break
        else:
            print("Ошибка! Что-то пошло не так...")
            break
```

ВЫВОДЫ

В ходе выполнения индивидуальной работы были подробно разобраны алгоритмы решения транспортных задач с помощью методов Северо-Западного угла и минимального транспортного тарифа, что позволило автоматизировать вычисления с помощью переноса в программу.