МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)

ИИТУС
БГТУ им. В.Г. Шухова

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4

по дисциплине: Исследования операций
тема: «Закрытая транспортная задача»

Выполнил: ст. группы ПВ-223
Дмитриев А.А.
Проверил:
Вирченко Ю.П.

Белгород 2024 г.

Цель работы: изучить математическую модель транспортной задачи, овладеть методами решения этой задачи.

Задания для подготовки к работе:

1. Изучить содержательную и математическую постановки закрытой транспортной задачи, методы нахождения первого опорного решения ее системы ограничений. Изучить понятие цикла пересчета в матрице перевозок. Овладеть распределительным методом и методом потенциалов, а также их алгоритмами.
2. Составить и отладить программы решения транспортной задачи распределительным методом и методом потенциалов.
3. Для подготовки тестовых данных решить задачу по варианту.

Вариант 2:

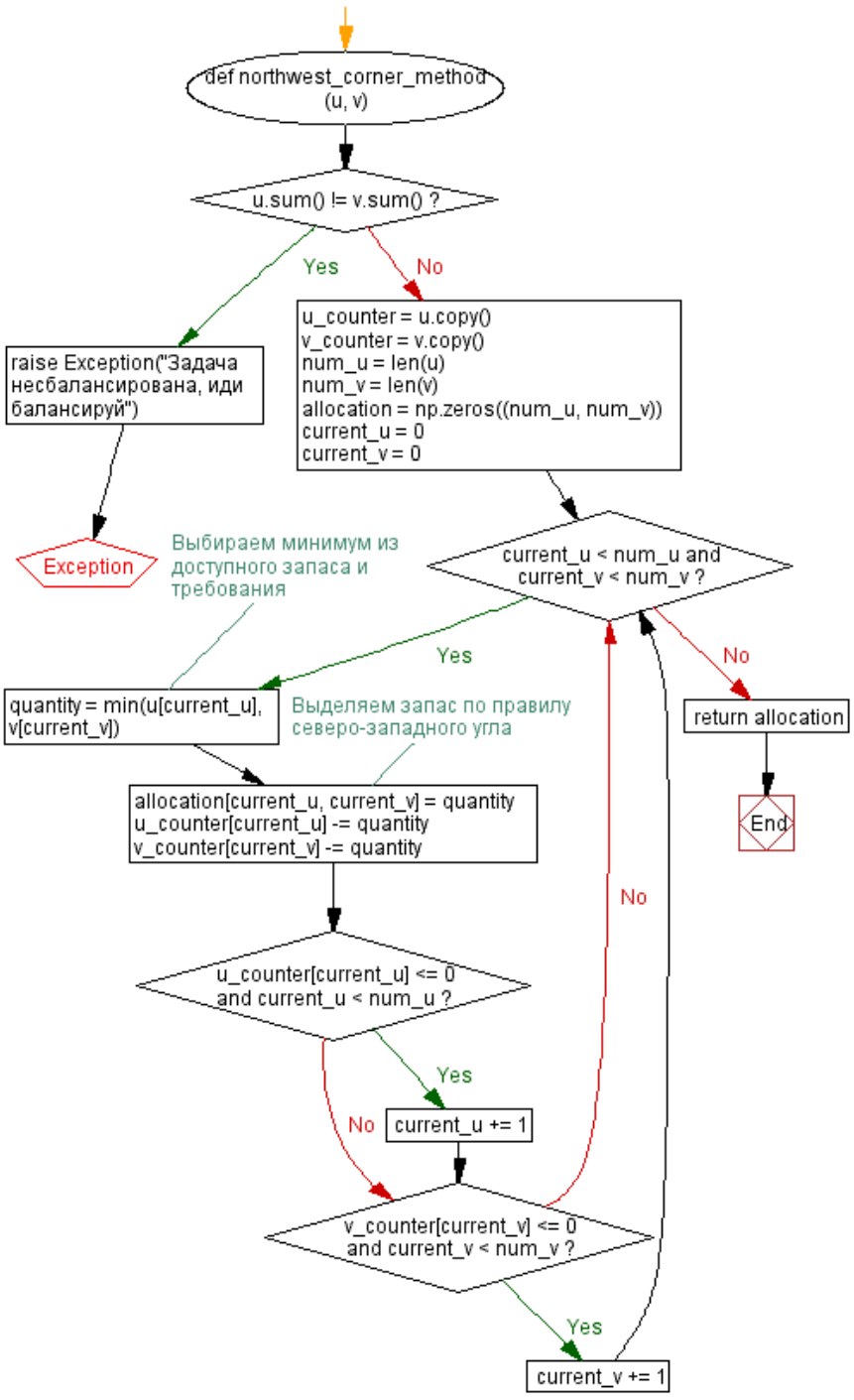
$\vec{a} = (18, 12, 22, 19)$;

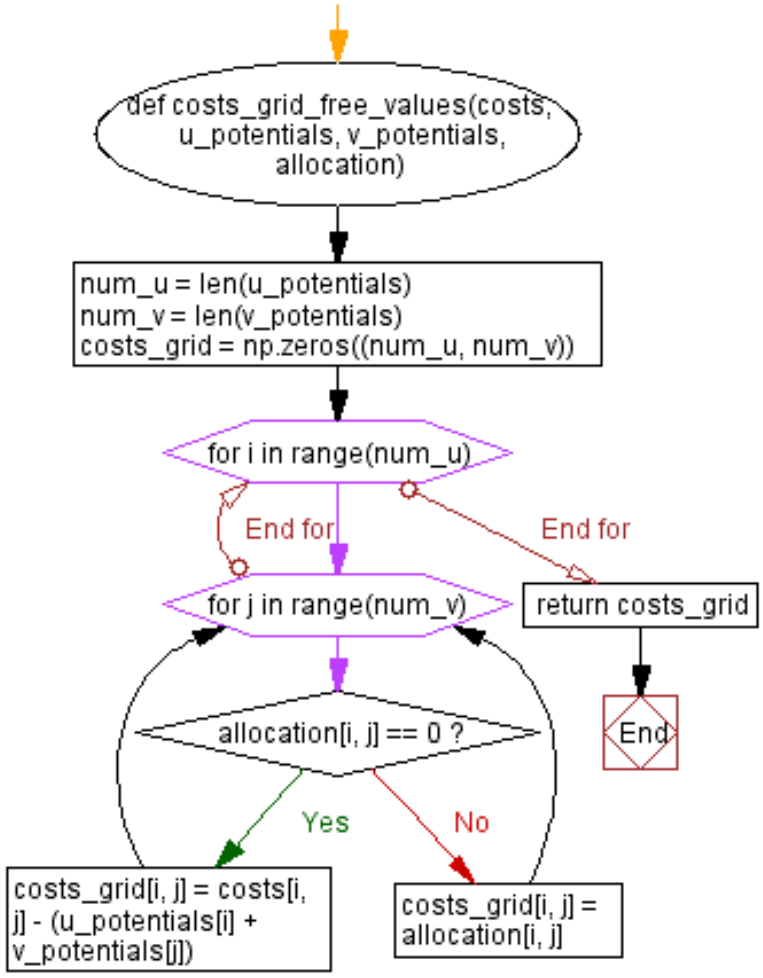
$\vec{b} = (14, 11, 17, 15, 14)$;

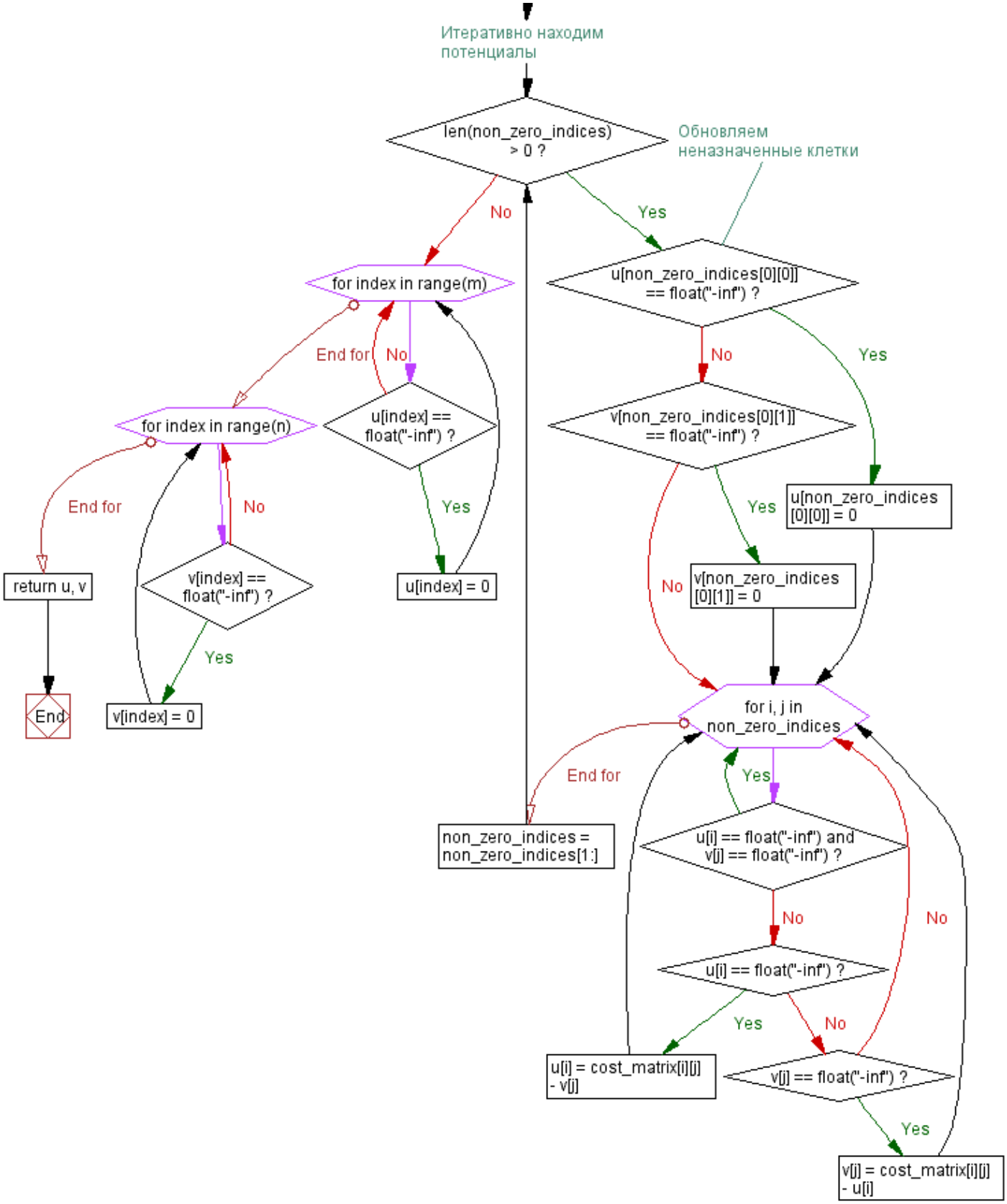
$$C = \begin{pmatrix} 9 & 21 & 22 & 14 & 10 \\ 30 & 34 & 42 & 23 & 26 \\ 8 & 17 & 30 & 27 & 9 \\ 11 & 20 & 24 & 7 & 25 \end{pmatrix}$$

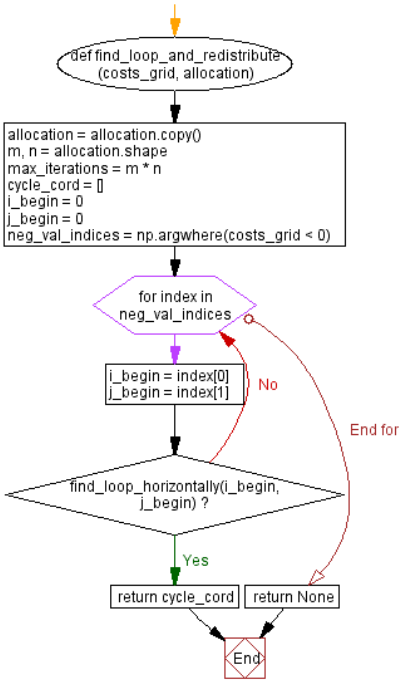
Задание:

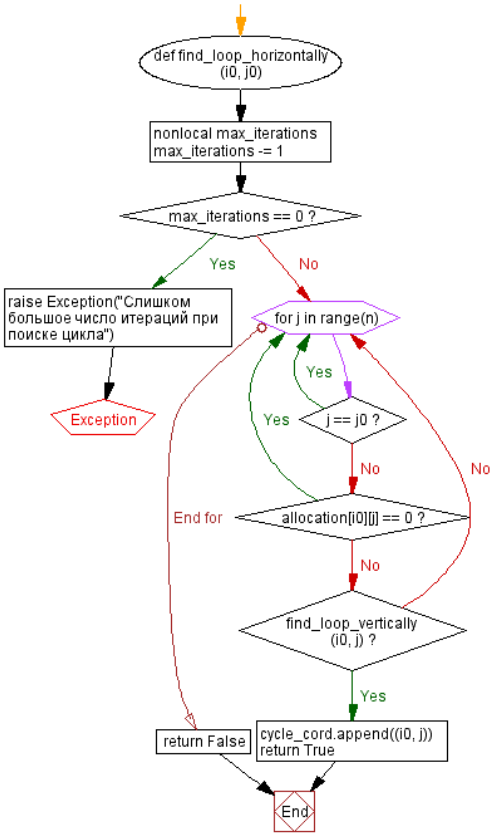
Блок схема:

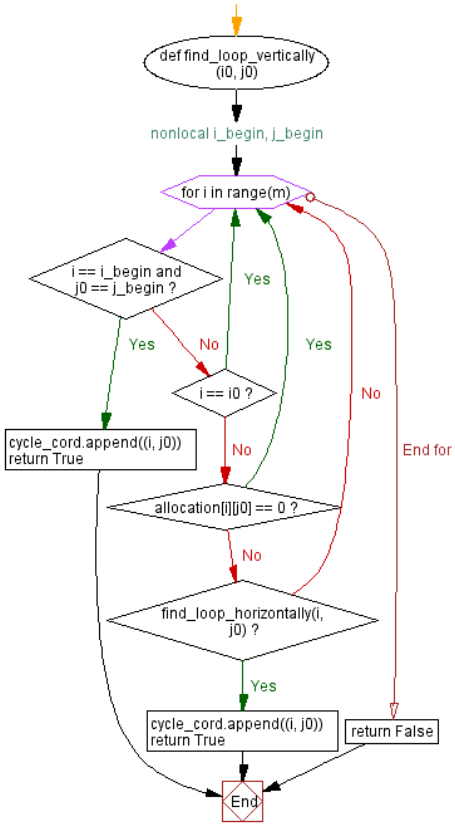


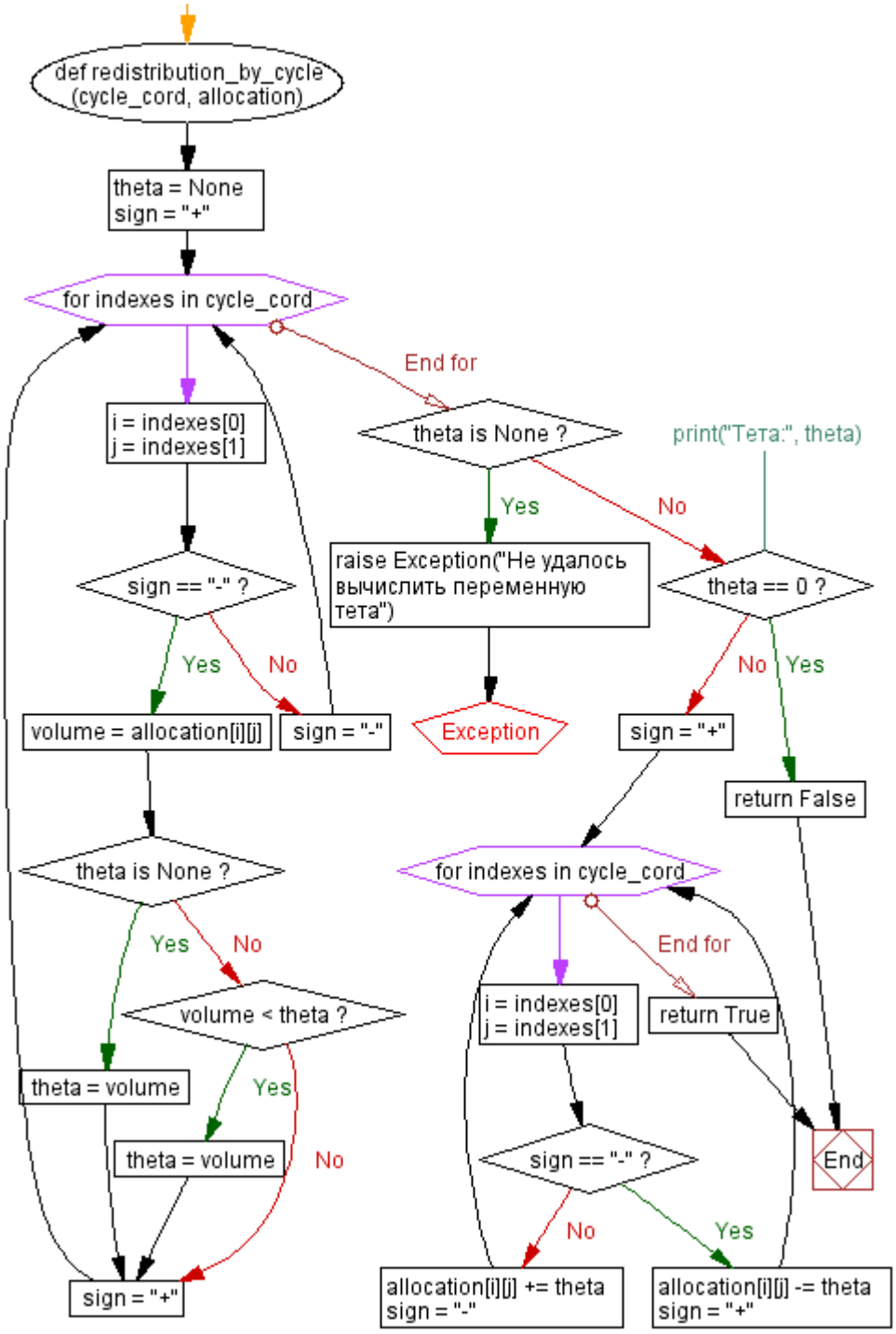

def costs_grid_free_values(costs, u_potentials, v_potentials, allocation)
num_u = len(u_potentials)
num_v = len(v_potentials)
costs_grid = np.zeros((num_u, num_v))
for i in range(num_u)
End for
End for
for j in range(num_v)
return costs_grid
End
allocation[i, j] == 0 ?
Yes
No
costs_grid[i, j] = costs[i, j] - (u_potentials[i] + v_potentials[j])
costs_grid[i, j] = allocation[i, j]

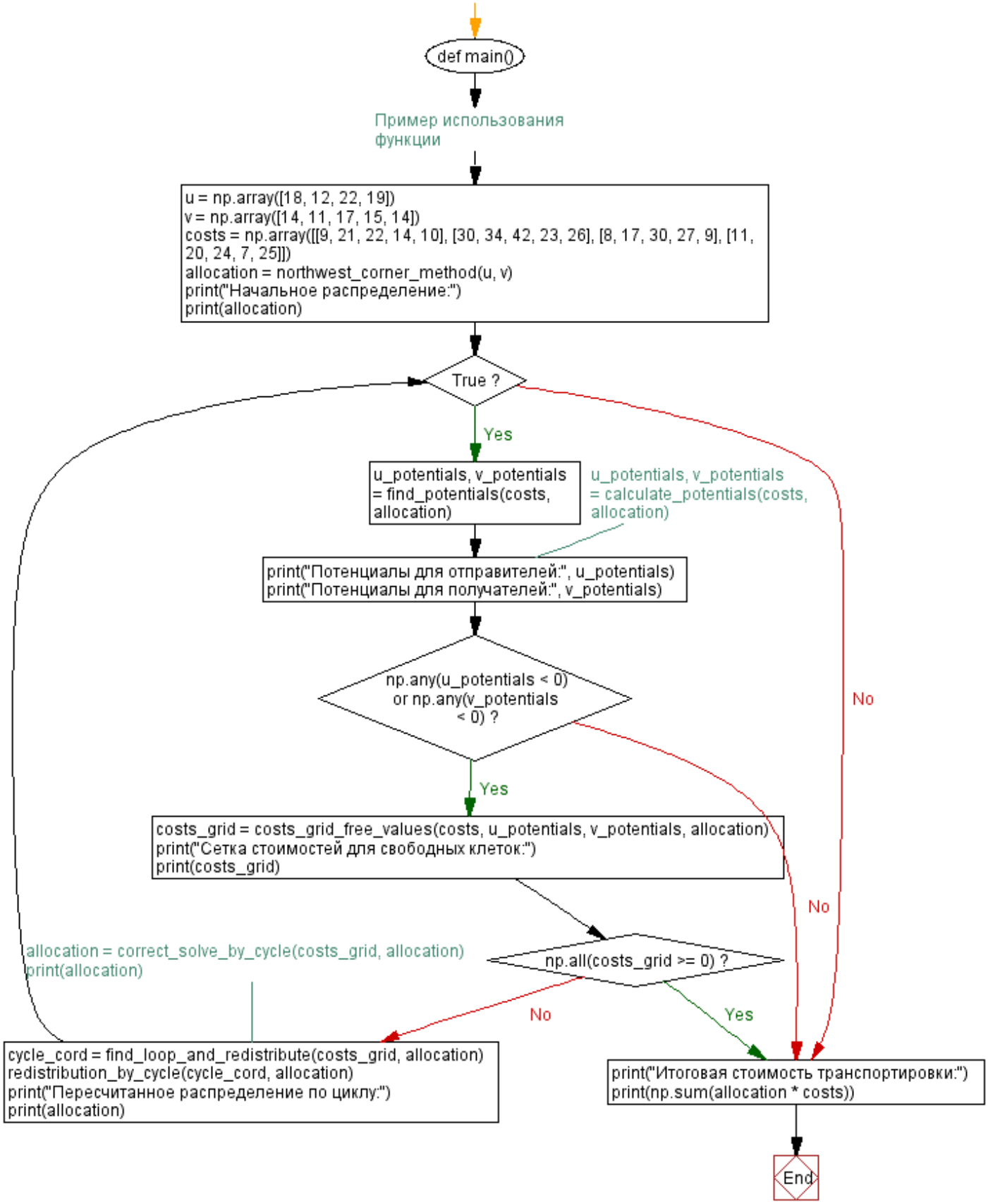

Итеративно находим потенциалы
len(non_zero_indices) > 0 ?
No
Yes
Обновляем неназначенные клетки
for index in range(m)
End for
No
u[index] == float("-inf") ?
Yes
u[index] = 0
for index in range(n)
End for
No
v[index] == float("-inf") ?
Yes
v[index] = 0
return u, v
End
u[non_zero_indices[0][0]] == float("-inf") ?
No
Yes
v[non_zero_indices[0][1]] == float("-inf") ?
Yes
u[non_zero_indices [0][0]] = 0
No
v[non_zero_indices [0][1]] = 0
for i, j in non_zero_indices
End for
Yes
u[i] == float("-inf") and v[j] == float("-inf") ?
No
No
u[i] == float("-inf") ?
Yes
No
u[i] = cost_matrix[i][j] - v[j]
v[j] == float("-inf") ?
Yes
v[j] = cost_matrix[i][j] - u[i]
non_zero_indices = non_zero_indices[1:]


def find_loop_and_redistribute (costs_grid, allocation)
allocation = allocation.copy()
m, n = allocation.shape
max_iterations = m * n
cycle_cord = []
i_begin = 0
j_begin = 0
neg_val_indices = np.argwhere(costs_grid < 0)
for index in neg_val_indices
i_begin = index[0]
j_begin = index[1]
No
End for
find_loop_horizontally(i_begin, j_begin) ?
Yes
return cycle_cord
return None
End


def find_loop_horizontally (i0, j0)
nonlocal max_iterations
max_iterations -= 1
max_iterations == 0 ?
Yes
No
raise Exception("Слишком большое число итераций при поиске цикла")
Exception
for j in range(n)
Yes
j == j0 ?
No
End for
allocation[i0][j] == 0 ?
No
find_loop_vertically (i0, j) ?
Yes
return False
cycle_cord.append((i0, j)) return True
End


def find_loop_vertically (i0, j0)
nonlocal i_begin, j_begin
for i in range(m)
i == i_begin and j0 == j_begin ?
Yes
No
cycle_cord.append((i, j0)) return True
i == i0 ?
No
End for
allocation[i][j0] == 0 ?
No
find_loop_horizontally(i, j0) ?
Yes
cycle_cord.append((i, j0)) return True
return False
End


def redistribution_by_cycle (cycle_cord, allocation)
theta = None
sign = "+"
for indexes in cycle_cord
End for
i = indexes[0]
j = indexes[1]
theta is None ?
print("Тета:", theta)
Yes
No
raise Exception("Не удалось вычислить переменную тета")
sign == "-" ?
theta == 0 ?
Yes
No
No
Yes
volume = allocation[i][j]
sign = "-"
Exception
sign = "+"
return False
theta is None ?
for indexes in cycle_cord
Yes
No
End for
volume < theta ?
i = indexes[0]
j = indexes[1]
return True
theta = volume
Yes
theta = volume
No
sign == "-" ?
End
No
Yes
sign = "+"
allocation[i][j] += theta
sign = "-"
allocation[i][j] -= theta
sign = "+"



Листинг кода:

```
import numpy as np


def northwest_corner_method(u, v):
    if u.sum() != v.sum():
        raise Exception("Задача несбалансирована, иди балансируй")

    u_counter = u.copy()
```

```
    v_counter = v.copy()

    num_u = len(u)
    num_v = len(v)

    allocation = np.zeros((num_u, num_v))

    current_u = 0
    current_v = 0
    while current_u < num_u and current_v < num_v:
        # Выбираем минимум из доступного запаса и требования
        quantity = min(u[current_u], v[current_v])

        # Выделяем запас по правилу северо-западного угла
        allocation[current_u, current_v] = quantity

        u_counter[current_u] -= quantity
        v_counter[current_v] -= quantity

        if u_counter[current_u] <= 0 and current_u < num_u:
            current_u += 1
        if v_counter[current_v] <= 0 and current_v < num_v:
            current_v += 1

    return allocation


def costs_grid_free_values(costs, u_potentials, v_potentials, allocation):
    num_u = len(u_potentials)
    num_v = len(v_potentials)

    costs_grid = np.zeros((num_u, num_v))
    for i in range(num_u):
        for j in range(num_v):
            if allocation[i, j] == 0:
                costs_grid[i, j] = costs[i, j] - (u_potentials[i] + v_potentials[j])
            else:
                costs_grid[i, j] = allocation[i, j]

    return costs_grid


def find_potentials(cost_matrix, allocation):
    m, n = allocation.shape
    u = np.full(m, float("-inf"))
    v = np.full(n, float("-inf"))

    # Выбираем неназначенные клетки
    non_zero_indices = np.argwhere(allocation > 0)

    # Итеративно находим потенциалы
    while len(non_zero_indices) > 0:
        # Обновляем неназначенные клетки
        if u[non_zero_indices[0][0]] == float("-inf"):
            u[non_zero_indices[0][0]] = 0
        elif v[non_zero_indices[0][1]] == float("-inf"):
            v[non_zero_indices[0][1]] = 0

        for i, j in non_zero_indices:
            if u[i] == float("-inf") and v[j] == float("-inf"):
                continue
            elif u[i] == float("-inf"):
                u[i] = cost_matrix[i][j] - v[j]
            elif v[j] == float("-inf"):
                v[j] = cost_matrix[i][j] - u[i]

        non_zero_indices = non_zero_indices[1:]
```

```
    for index in range(m):
        if u[index] == float("-inf"):
            u[index] = 0

    for index in range(n):
        if v[index] == float("-inf"):
            v[index] = 0

    return u, v


def find_loop_and_redistribute(costs_grid, allocation):
    allocation = allocation.copy()
    m, n = allocation.shape

    max_iterations = m * n
    cycle_cord = []
    i_begin = 0
    j_begin = 0

    neg_val_indices = np.argwhere(costs_grid < 0)

    def find_loop_horizontally(i0, j0):
        nonlocal max_iterations
        max_iterations -= 1
        if max_iterations == 0:
            raise Exception("Слишком большое число итераций при поиске цикла")

        for j in range(n):
            if j == j0:
                continue
            if allocation[i0][j] == 0:
                continue
            if find_loop_vertically(i0, j):
                cycle_cord.append((i0, j))
                return True
        return False

    def find_loop_vertically(i0, j0):
        # nonlocal i_begin, j_begin
        for i in range(m):
            if i == i_begin and j0 == j_begin:
                cycle_cord.append((i, j0))
                return True
            if i == i0:
                continue
            if allocation[i][j0] == 0:
                continue
            if find_loop_horizontally(i, j0):
                cycle_cord.append((i, j0))
                return True
        return False

    for index in neg_val_indices:
        i_begin = index[0]
        j_begin = index[1]
        if find_loop_horizontally(i_begin, j_begin):
            return cycle_cord

    return None


def redistribution_by_cycle(cycle_cord, allocation):
    theta = None
    sign = "+"
    for indexes in cycle_cord:
```

```
        i = indexes[0]
        j = indexes[1]
        if sign == "-":
            volume = allocation[i][j]
            if theta is None:
                theta = volume
            else:
                if volume < theta:
                    theta = volume
            sign = "+"
        else:
            sign = "-"

    if theta is None:
        raise Exception("Не удалось вычислить переменную тета")

    # print("Тета:", theta)
    if theta == 0:
        return False

    sign = "+"
    for indexes in cycle_cord:
        i = indexes[0]
        j = indexes[1]
        if sign == "-":
            allocation[i][j] -= theta
            sign = "+"
        else:
            allocation[i][j] += theta
            sign = "-"

    return True


def main():
    # Пример использования функции
    u = np.array([18, 12, 22, 19])
    v = np.array([14, 11, 17, 15, 14])

    costs = np.array([[9, 21, 22, 14, 10],
                      [30, 34, 42, 23, 26],
                      [8, 17, 30, 27, 9],
                      [11, 20, 24, 7, 25]])

    allocation = northwest_corner_method(u, v)
    print("Начальное распределение:")
    print(allocation)

    while True:
        u_potentials, v_potentials = find_potentials(costs, allocation)
        # u_potentials, v_potentials = calculate_potentials(costs, allocation)
        print("Потенциалы для отправителей:", u_potentials)
        print("Потенциалы для получателей:", v_potentials)

        if np.any(u_potentials < 0) or np.any(v_potentials < 0):
            costs_grid = costs_grid_free_values(costs, u_potentials, v_potentials,
allocation)
            print("Сетка стоимостей для свободных клеток:")
            print(costs_grid)

            if np.all(costs_grid >= 0):
                break

            # allocation = correct_solve_by_cycle(costs_grid, allocation)
            # print(allocation)

            cycle_cord = find_loop_and_redistribute(costs_grid, allocation)
```

```
            redistribution_by_cycle(cycle_cord, allocation)
            print("Пересчитанное распределение по циклу:")
            print(allocation)
        else:
            break

    print("Итоговая стоимость транспортировки:")
    print(np.sum(allocation * costs))


if __name__ == '__main__':
    main()
```

Тестовые данные:

Результат работы программы:

Начальное распределение:

[[14. 11. 0. 0. 0.]

 [0. 0. 12. 0. 0.]

 [0. 0. 17. 15. 0.]

 [0. 0. 0. 0. 14.]]

Потенциалы для отправителей: [0.   0. -12.   0.]

Потенциалы для получателей: [9. 21. 42. 39. 25.]

Сетка стоимостей для свободных клеток:

[[14. 11. -20. -25. -15.]

 [21. 13. 12. -16.   1.]

 [11.   8. 17. 15. -4.]

 [2. -1. -18. -32. 14.]]

Пересчитанное распределение по циклу:

[[14. 11. 0. 0. 0.]

 [0. 0. 0. 12. 0.]

 [0. 0. 29. 3. 0.]

 [0. 0. 0. 0. 14.]]

Потенциалы для отправителей: [0. 0. 4. 0.]

Потенциалы для получателей: [9. 21. 0. 23. 25.]

Решение вручную:

Поставщики	Потребители				
	14	11	17	15	14
18	9	21	22	14	10
12	30	34	42	23	26
22	8	17	30	27	9
19	11	20	24	7	25

Для первого опорного плана воспользуемся методом наименьшей стоимости, чтобы за меньшее количество шагов прийти к оптимальному решению.

U	V				
	14	11	17	15	14
18	9	21/7	22/5	14	10/6
12	30	34	42/12	23	26
22	8/14	17	30	27	9/8
19	11	20	24	7/15	25

Рассчитаем потенциалы для оценки оптимальности решения. Пусть $u_1 = 0$.
($u_j + v_i = c_{ji}$)

U	V				
	14(9)	11(21)	17(22)	15(8)	14(10)
18(0)	9	21/7	22/5	14	10/6
12(20)	30	34	42/12	23	26
22(-1)	8/14	17	30	27	9/8
19(-1)	11	20	24	7/15	25

Видим отрицательные потенциалы. Определим оценки незадействованных маршрутов. ($d_{ij} = c_{ij} – (u_i + v_j)$)

U	V				
	14	11	17	15	14
18	9/0	21	22	14/6	10
12	30/1	34/-7	42	23/-5	26/-4
22	8	17/-3	30/9	27/20	9
19	11/3	20	24/3	7	25/16

По циклу скорректируем решение.

U	V				
	14	11	17	15	14
18	9/0	21/7	22/5	14/6	10/6
12	30/1	34/-7	42/12	23/-5	26/-4
22	8/14	17/-3	30/9	27/20	9/8
19	11/3	20/4	24/3	7/15	25/16

U	V				
	14	11	17	15	14
18	9	21/0	22/12	14	10/6
12	30	34/7	42/5	23	26
22	8/14	17	30	27	9/8
19	11	20	24	7/15	25

Проверим оптимальность.

U	V				
	14(9)	11(14)	17(22)	15(1)	14(10)
18(0)	9	21	22/12	14	10/6
12(20)	30	34/7	42/5	23	26
22(-1)	8/14	17	30	27	9/8
19(6)	11	20	24	7/15	25

План не оптимален. Повторяем шаги.

U	V				
	14	11	17	15	14
18	9/0	21	22	14/13	10
12	30	34/7	42	23/2	26/-4
22	8	17/4	30/9	27/27	9
19	11/-4	20	24/-4	7	25/9

U	V

	14	11	17	15	14
18	9/0	21/7	22/12	14/13	10/6
12	30/1	34/7	42/5	23/2	26/-4
22	8/14	17/4	30/9	27/27	9/8
19	11/-4	20/4	24/-4	7/15	25/9

U	V				
	14	11	17	15	14
18	9	21	22/17	14	10/1
12	30	34/7	42	23	26/5
22	8/14	17	30	27	9/8
19	11	20/4	24	7/15	25

U	V				
	14(9)	11(18)	17(22)	15(5)	14(10)
18(0)	9	21	22/17	14	10/1
12(16)	30	34/7	42	23	26/5
22(-1)	8/14	17	30	27	9/8
19(2)	11	20	24	7/15	25

U	V				
	14	11	17	15	14
18	9/0	21/3	22	14/9	10
12	30/5	34	42/4	23/2	26
22	8	17/0	30/9	27/23	9
19	11/0	20	24/0	7	25/13

Отрицательных оценок нет. Был получен ответ.

0	0	17	0	1
0	7	0	0	5
14	0	0	0	8
0	0	0	15	0

Вывод: В ходе лабораторной работы была изучена математическая модель транспортной задачи, также овладели методами решения этой задачи.