

Федеральное государственное образовательное бюджетное  
учреждение высшего образования  
**«Финансовый университет при Правительстве Российской Федерации»**  
**(Финансовый университет)**  
Колледж информатики и программирования

Специальность 09.02.07 «Информационные системы и программирование»

**ОТЧЕТ  
ПО УЧЕБНОЙ ПРАКТИКЕ**

Профессиональный модуль ПМ.02 Осуществление интеграции программных  
модулей  
(наименование профессионального модуля)

Выполнил:  
обучающийся учебной группы № 4ИСИП-421

Сакович Н.Д.  
(И.О. Фамилия)

Проверил:  
руководитель практики от колледжа:

Т.Г. Аксёнова  
(И.О. Фамилия)

Е.Л. Альшакова  
(И.О. Фамилия)

Н.Н. Сафонова  
(И.О. Фамилия)

**Москва  
2024**

## Перечень заданий/работ, выполненных в ходе учебной практики

Специальность 09.02.07 «Информационные системы и программирование»  
(наименование специальности)

Профессиональный модуль ПМ.02 Осуществление интеграции программных модулей  
(наименование профессионального модуля)

№ п/п	Темы учебной практики	Выполненные задания/работы
<b>Раздел 1. Разработка программного обеспечения</b>		
1	Тема 1. Анализ выбранной предметной области	Анализ предметной области на основе плана
2	Тема 2. Разработка и оформление технического задания на программное средство	Разработка и оформление технического задания на программное средство на основе проведенного анализа предметной области
3	Тема 3. Построение архитектуры программного средства	Построение архитектуры программного средства на основе технического задания
4	Тема 4. Анализ требований и построение необходимых диаграмм	Анализ требований и построение необходимых диаграмм с использованием специализированных инструментальных средств
<b>Раздел 2. Средства разработки программного обеспечения</b>		
5	Тема 1. Разработка структуры, перечня артефактов и протоколов проекта	Разработка структуры, перечня артефактов и протоколов проекта на основе технического задания и спецификаций
6	Тема 2. Командная работа над проектом с использованием системы контроля версий	Командная работа над проектом с использованием выбранной системы контроля версий
7	Тема 3. Отладка программного проекта	Выполнение отладки программного проекта с использованием специализированных программных средств
8	Тема 4. Тестирование и анализ качества программного средства	Осуществление разработки тестовых наборов и тестовых сценариев для программного обеспечения. Проведение тестирования программного средства. Анализ качества программного средства
9	Тема 5. Документирование результатов тестирования	Составление документов по итогам тестирования
<b>Раздел 3. Математическое моделирование</b>		
10	Тема 1. Построение различных типов математических моделей	Построение математических моделей на основе анализа предметной области
11	Тема 2. Разработка модуля по построенной математической модели и его интеграция в программное обеспечение	Разработка программного модуля по построенной математической модели и его интеграция в программное обеспечение
12	Тема 3. Составление тестовых наборов для проверки работоспособности математической модели и тестирования программы	Осуществление разработки тестовых наборов для тестирования математической модели разработанной программы

## **СОДЕРЖАНИЕ**

<b>ОТЧЕТ О ВЫПОЛНЕНИИ ЗАДАНИЯ № 1 .....</b>	<b>4</b>
<b>ОТЧЕТ О ВЫПОЛНЕНИИ ЗАДАНИЯ № 2 .....</b>	<b>12</b>

Федеральное государственное образовательное бюджетное  
учреждение высшего образования  
**«Финансовый университет при Правительстве Российской Федерации»**  
**(Финансовый университет)**  
Колледж информатики и программирования

**ОТЧЕТ О ВЫПОЛНЕНИИ ЗАДАНИЯ № 1**

**Тема: Транспортная задача**

Студент: Сакович Н.Д.

Группа: 4ИСИП-421

Преподаватель: Т.Г. Аксёнова

Дата: 19.12.2024

**Цель работы:** разработать программное обеспечение для решения транспортной задачи разными методами.

**Листинг кода least\_coast:**

```
import numpy as np

def transportation_cost(supply, demand, costs):
    num_suppliers = len(supply)
    num_customers = len(demand)

    supply_remaining = np.array(supply)
    demand_remaining = np.array(demand)
    allocation = np.zeros((num_suppliers, num_customers))

    while np.sum(supply_remaining) > 0:
        supplier_index = np.argmax(supply_remaining)
        customer_index = np.argmax(demand_remaining)

        quantity = min(supply_remaining[supplier_index],
            demand_remaining[customer_index])

        allocation[supplier_index, customer_index] = quantity
        supply_remaining[supplier_index] -= quantity
        demand_remaining[customer_index] -= quantity

    total_cost = np.sum(allocation * costs)

    return allocation, total_cost

cost = np.array([
    [24, 36, 28, 16, 33],
    [42, 52, 38, 22, 46],
    [14, 58, 22, 34, 36],
    [20, 34, 40, 52, 37]
])

supply = [125, 240, 75, 330]
demand = [150, 245, 70, 340, 220]

allocation, total_cost = transportation_cost(supply, demand, cost)

print("Optimal Allocation:")
print(allocation)
print("Total Cost:", total_cost)
```

**Листинг кода least\_coast\_method:**

```
import numpy as np

def least_cost_method(supply, demand, cost):
    supply = supply.copy()
    demand = demand.copy()
    rows, cols = len(supply), len(demand)
    allocation = np.zeros((rows, cols), dtype=int)

    while np.sum(allocation) != np.sum(supply):
```

```

min_value = float('inf')
min_pos = (0, 0)

# Найти минимальный элемент в матрице стоимости
for i in range(rows):
    for j in range(cols):
        if cost[i][j] < min_value and supply[i] > 0 and demand[j] > 0:
            min_value = cost[i][j]
            min_pos = (i, j)

i, j = min_pos

# Распределить как можно больше на найденную позицию
allocation_amount = min(supply[i], demand[j])
allocation[i][j] = allocation_amount
supply[i] -= allocation_amount
demand[j] -= allocation_amount

return allocation

def print_result(allocation, cost):
    total_cost = 0
    for i in range(len(allocation)):
        for j in range(len(allocation[i])):
            total_cost += allocation[i][j] * cost[i][j]
    print("Allocation Matrix:")
    print(allocation)
    print(f"Total Transportation Cost: {total_cost} руб.")

# Данные из таблицы
cost = np.array([
    [24, 36, 28, 16, 33],
    [42, 52, 38, 22, 46],
    [14, 58, 22, 34, 36],
    [20, 34, 40, 52, 37]
])

supply = [125, 240, 75, 330]
demand = [150, 245, 70, 340, 220]

allocation = least_cost_method(supply, demand, cost)
print_result(allocation, cost)

```

#### Листинг кода northwest\_corner\_method:

```

import numpy as np

def northwest_corner_method(supply, demand, cost):
    supply = supply.copy()
    demand = demand.copy()

```

```

rows, cols = len(supply), len(demand)
allocation = np.zeros((rows, cols), dtype=int)

i = j = 0
while i < rows and j < cols:
    if supply[i] < demand[j]:
        allocation[i][j] = supply[i]
        demand[j] -= supply[i]
        i += 1
    else:
        allocation[i][j] = demand[j]
        supply[i] -= demand[j]
        j += 1

return allocation

def print_result(allocation, cost):
    total_cost = 0
    for i in range(len(allocation)):
        for j in range(len(allocation[i])):
            total_cost += allocation[i][j] * cost[i][j]
    print("Allocation Matrix:")
    print(allocation)
    print(f"Total Transportation Cost: {total_cost} руб.")

# Данные из таблицы
cost = np.array([
    [24, 36, 28, 16, 33],
    [42, 52, 38, 22, 46],
    [14, 58, 22, 34, 36],
    [20, 34, 40, 52, 37]
])

supply = [125, 240, 75, 330]
demand = [150, 245, 70, 340, 220]

allocation = northwest_corner_method(supply, demand, cost)
print_result(allocation, cost)

```

#### Листинг кода potentials\_method:

```

import numpy as np

def northwest_corner_method(supply, demand):
    supply = supply.copy()
    demand = demand.copy()
    rows, cols = len(supply), len(demand)
    allocation = np.zeros((rows, cols), dtype=int)

    i = j = 0
    while i < rows and j < cols:

```

```

    if supply[i] < demand[j]:
        allocation[i][j] = supply[i]
        demand[j] -= supply[i]
        i += 1
    else:
        allocation[i][j] = demand[j]
        supply[i] -= demand[j]
        j += 1

    return allocation

def calculate_potentials(allocation, cost):
    u = [None] * len(cost)
    v = [None] * len(cost[0])
    u[0] = 0

    while None in u or None in v:
        for i in range(len(cost)):
            for j in range(len(cost[i])):
                if allocation[i][j] > 0:
                    if u[i] is not None and v[j] is None:
                        v[j] = cost[i][j] - u[i]
                    elif u[i] is None and v[j] is not None:
                        u[i] = cost[i][j] - v[j]

    return u, v

def find_entering_variable(allocation, cost, u, v):
    min_value = 0
    entering_variable = None

    for i in range(len(cost)):
        for j in range(len(cost[i])):
            if allocation[i][j] == 0 and (u[i] + v[j] < cost[i][j]):
                value = cost[i][j] - (u[i] + v[j])
                if value < min_value:
                    min_value = value
                    entering_variable = (i, j)

    return entering_variable

def find_loop(allocation, entering_variable):
    rows, cols = len(allocation), len(allocation[0])
    loop = []
    loop.append(entering_variable)

    def find_path(start, direction, visited):
        if direction == 'horizontal':
            for j in range(cols):

```



```

        if allocation[start[0]][j] > 0 or (start[0], j) == entering_variable:
            if (start[0], j) not in visited:
                visited.append((start[0], j))
                find_path((start[0], j), 'vertical', visited)
    elif direction == 'vertical':
        for i in range(rows):
            if allocation[i][start[1]] > 0 or (i, start[1]) == entering_variable:
                if (i, start[1]) not in visited:
                    visited.append((i, start[1]))
                    find_path((i, start[1]), 'horizontal', visited)

    find_path(entering_variable, 'horizontal', loop)
    return loop

def adjust_allocation(allocation, loop):
    positions = loop[1::2]
    min_value = min(allocation[i][j] for i, j in positions)

    for k, (i, j) in enumerate(loop):
        if k % 2 == 0:
            allocation[i][j] += min_value
        else:
            allocation[i][j] -= min_value

    return allocation

def transportation_problem(supply, demand, cost):
    allocation = northwest_corner_method(supply, demand)

    while True:
        u, v = calculate_potentials(allocation, cost)
        entering_variable = find_entering_variable(allocation, cost, u, v)

        if entering_variable is None:
            break

        loop = find_loop(allocation, entering_variable)
        allocation = adjust_allocation(allocation, loop)

    return allocation

def print_result(allocation, cost):
    total_cost = 0
    for i in range(len(allocation)):
        for j in range(len(allocation[i])):
            total_cost += allocation[i][j] * cost[i][j]
    print("Allocation Matrix:")
    print(allocation)
    print(f"Total Transportation Cost: {total_cost} py6.")

```

```

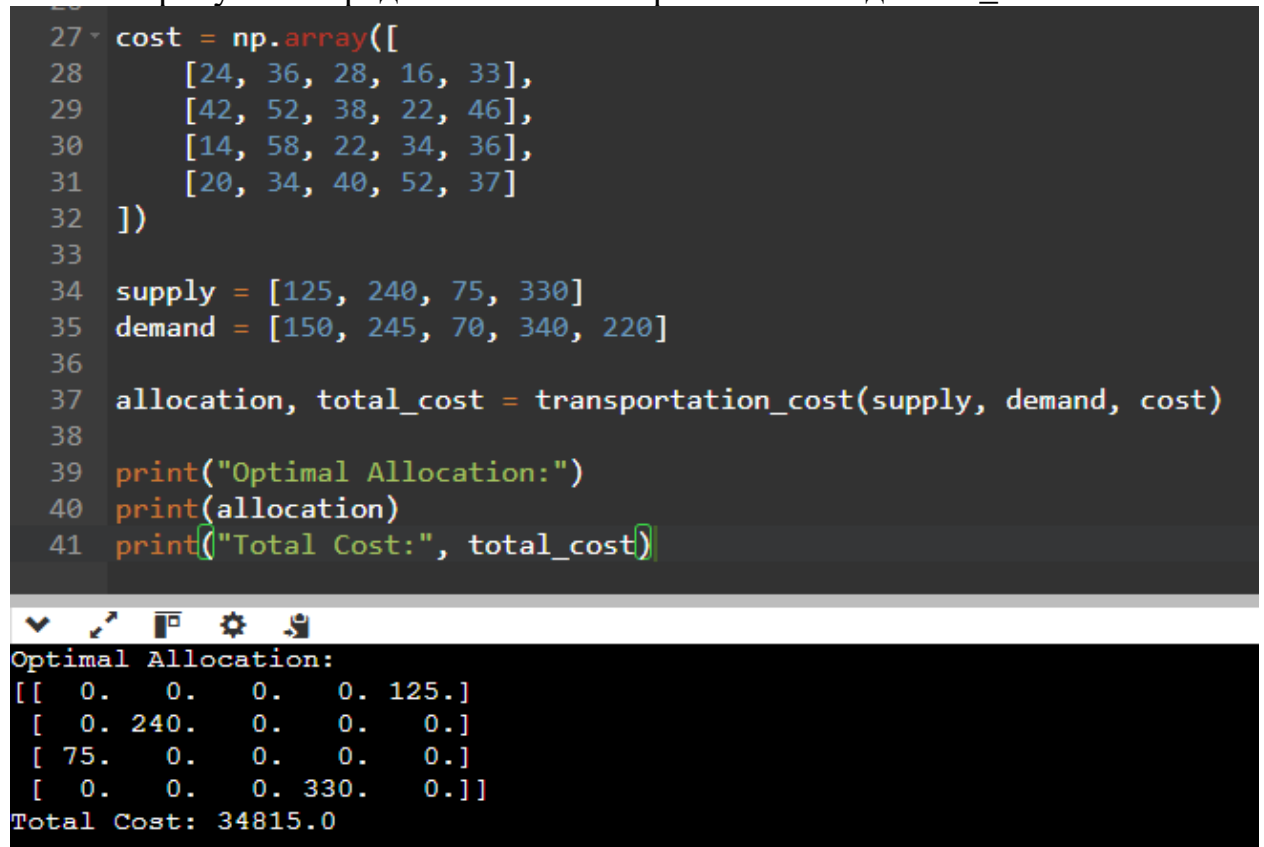
# Данные из таблицы
cost = np.array([
    [24, 36, 28, 16, 33],
    [42, 52, 38, 22, 46],
    [14, 58, 22, 34, 36],
    [20, 34, 40, 52, 37]
])

supply = [125, 240, 75, 330]
demand = [150, 245, 70, 340, 220]

allocation = transportation_problem(supply, demand, cost)
print_result(allocation, cost)

```

На рисунке 1 предоставлено тестирование метода `least_cost`:



```

27 cost = np.array([
28     [24, 36, 28, 16, 33],
29     [42, 52, 38, 22, 46],
30     [14, 58, 22, 34, 36],
31     [20, 34, 40, 52, 37]
32 ])
33
34 supply = [125, 240, 75, 330]
35 demand = [150, 245, 70, 340, 220]
36
37 allocation, total_cost = transportation_cost(supply, demand, cost)
38
39 print("Optimal Allocation:")
40 print(allocation)
41 print("Total Cost:", total_cost)

```

Optimal Allocation:

```

[[ 0.  0.  0.  0. 125.]
 [ 0. 240.  0.  0.  0.]
 [ 75.  0.  0.  0.  0.]
 [ 0.  0.  0. 330.  0.]]
Total Cost: 34815.0

```

Рисунок 1. Тестирование

На рисунке 2 предоставлено тестирование метода `northwest_corner_method`:

```
33 |
34 | cost = np.array([
35 |     [24, 36, 28, 16, 33],
36 |     [42, 52, 38, 22, 46],
37 |     [14, 58, 22, 34, 36],
38 |     [20, 34, 40, 52, 37]
39 | ])
40 |
41 | supply = [125, 240, 75, 330]
42 | demand = [150, 245, 70, 340, 220]
43 |
44 | allocation = northwest_corner_method(s
45 | print_result(allocation, cost)
```

Allocation Matrix:

[125	0	0	0	0]
[ 25	215	0	0	0]
[ 0	30	45	0	0]
[ 0	0	25	305	0]

Total Transportation Cost: 34820 руб.

Рисунок 2. Тестирование

Федеральное государственное образовательное бюджетное  
учреждение высшего образования  
**«Финансовый университет при Правительстве Российской Федерации»**  
**(Финансовый университет)**  
Колледж информатики и программирования

## **ОТЧЕТ О ВЫПОЛНЕНИИ ЗАДАНИЯ № 2**

**Тема: Проектирование базы данных и разработка объектов базы данных**

Студент: Сакович Н.Д.  
Группа: 4ИСИП-421  
Преподаватель: Т.Г. Аксёнова  
Дата: 19.12.2024

**Цель работы:** разработать программное обеспечение для решения транспортной задачи разными методами.

Листинг кода `simplex_method`:

```
import numpy as np

def simplex(c, A, b):
    """
    Решает задачу линейного программирования в канонической форме
    с помощью симплекс-метода.

    Args:
        c: Массив коэффициентов целевой функции.
        A: Матрица коэффициентов ограничений.
        b: Вектор правых частей ограничений.

    Returns:
        Кортеж (x, z), где x - оптимальное решение, z - значение целевой функции.
    """

    m, n = A.shape
    A = np.hstack((A, np.eye(m)))
    c = np.hstack((c, np.zeros(m)))

    basis = list(range(n, n + m))

    while True:
        z = np.dot(c[basis], A[:, basis].T)

        if all(z <= 0):
            break

        entering = np.argmax(z)

        ratios = b / A[:, entering]
        ratios[A[:, entering] <= 0] = np.inf

        leaving = np.argmin(ratios)

        basis[leaving] = entering

    x = np.zeros(n + m)
    x[basis] = A[:, basis].T @ np.linalg.inv(A[:, basis]) @ b

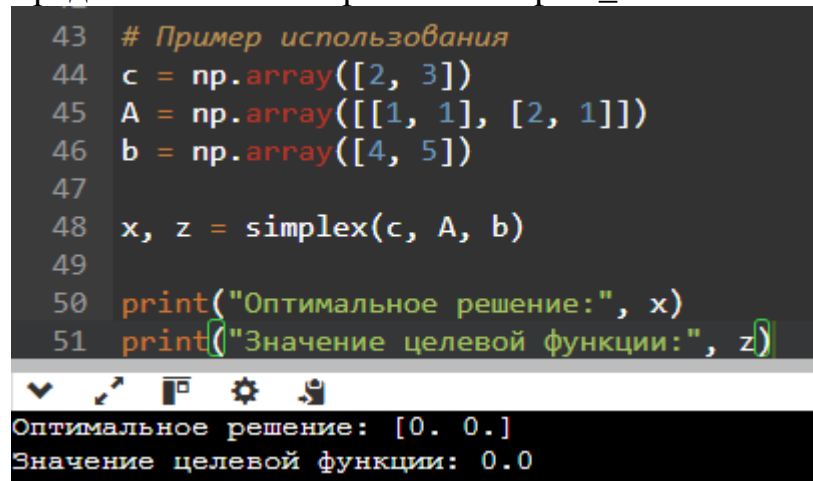
    return x[:n], np.dot(c[:n], x[:n])

# Пример использования
c = np.array([2, 3])
A = np.array([[1, 1], [2, 1]])
b = np.array([4, 5])

x, z = simplex(c, A, b)
```

```
print("Оптимальное решение:", x)
print("Значение целевой функции:", z)
```

На рисунке 3 предоставлено тестирование `simplex_method`:



```
43 # Пример использования
44 c = np.array([2, 3])
45 A = np.array([[1, 1], [2, 1]])
46 b = np.array([4, 5])
47
48 x, z = simplex(c, A, b)
49
50 print("Оптимальное решение:", x)
51 print("Значение целевой функции:", z)
```

Оптимальное решение: [0. 0.]  
Значение целевой функции: 0.0

Рисунок 3. Тестирование