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)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №4

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

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
проф. Вирченко Юрий Петрович

Белгород 2024 г.

Лабораторная работа №4

Закрытая транспортная задача

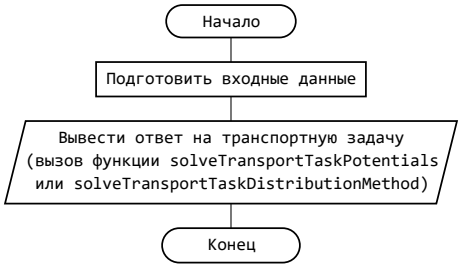
Цель работы: изучить математическую модель транспортной задачи, овладеть методами решения этой задачи.

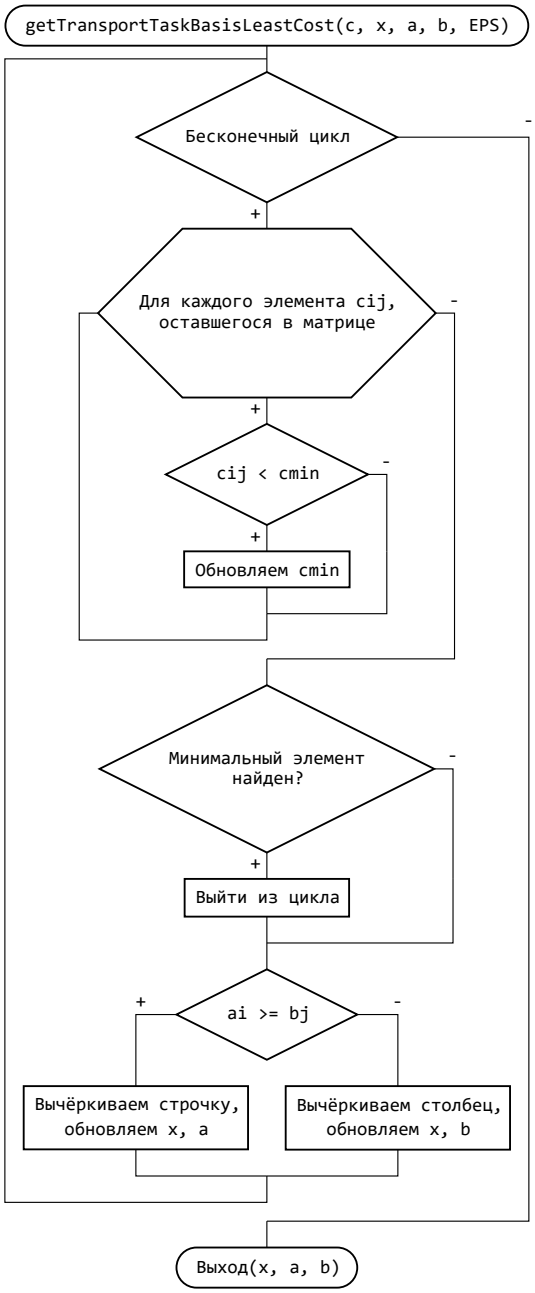
Задание: составить и отладить программы решения транспортной задачи распределительным методом и методом потенциалов. В рамках подготовки тестовых данных решить задачу вручную.

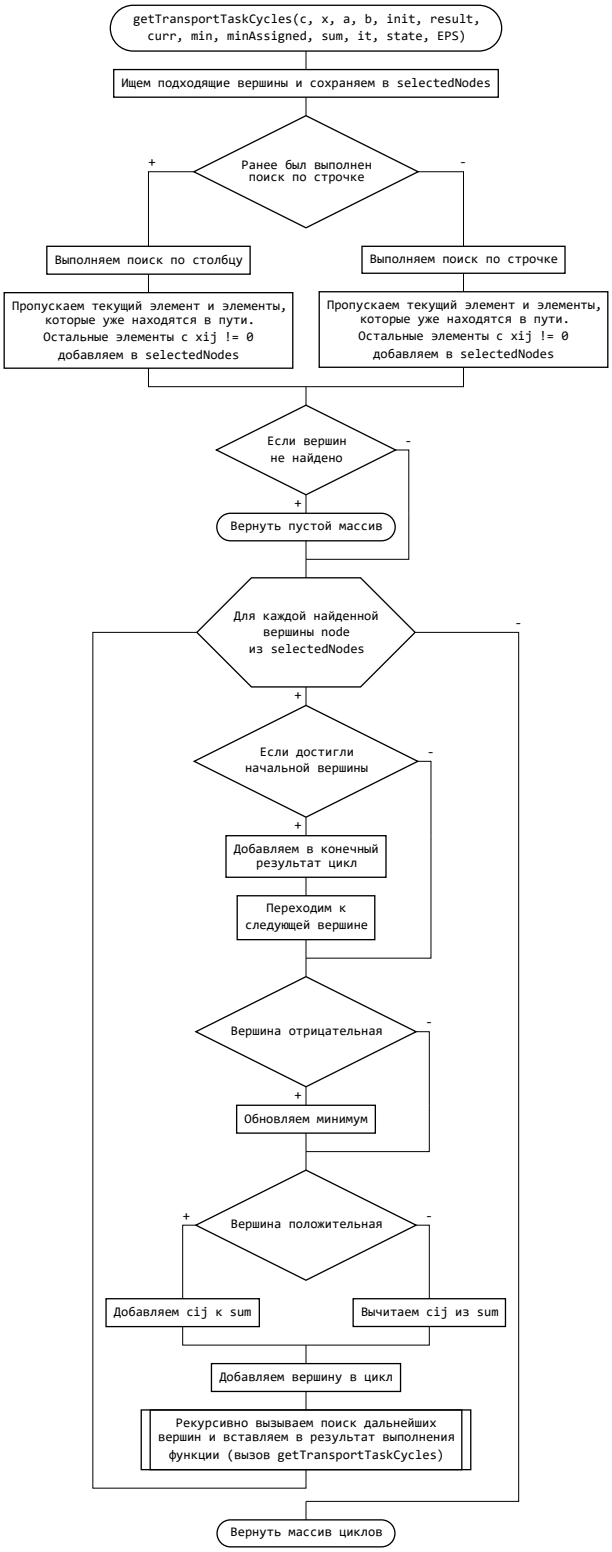
Вариант 10

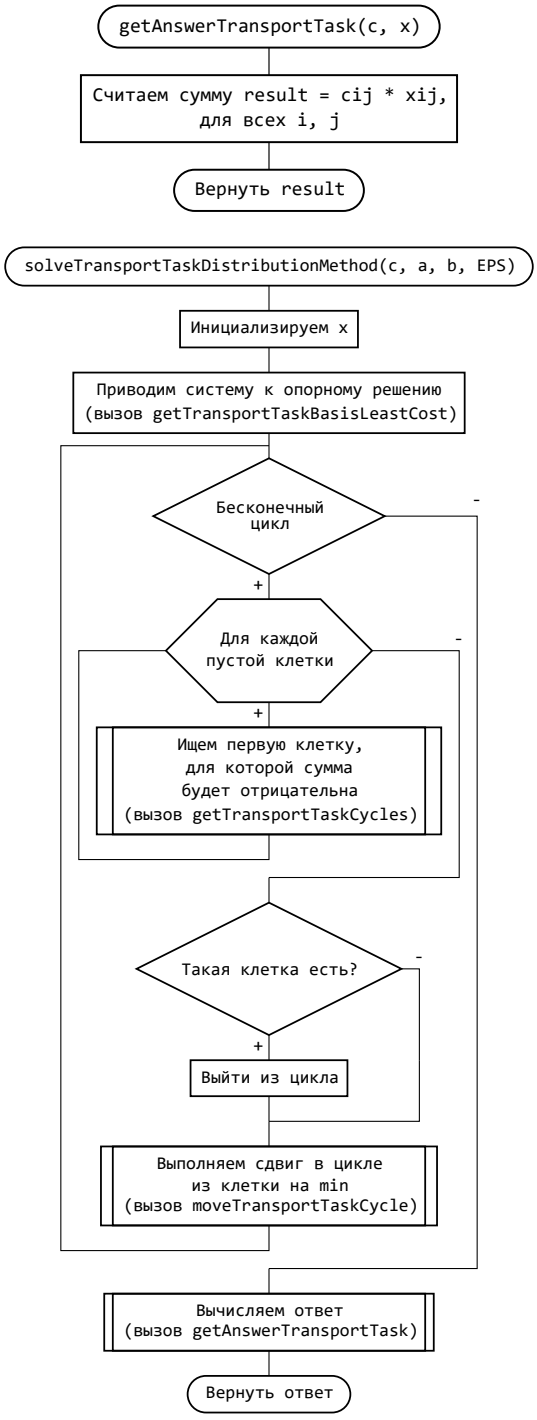
$$\vec{a} = (14, 14, 14, 14);$$
$$\vec{b} = (13, 5, 13, 12, 13);$$
$$C = \begin{pmatrix} 16 & 26 & 12 & 24 & 3 \\ 5 & 2 & 19 & 27 & 2 \\ 29 & 23 & 25 & 16 & 8 \\ 2 & 25 & 14 & 15 & 21 \end{pmatrix}$$

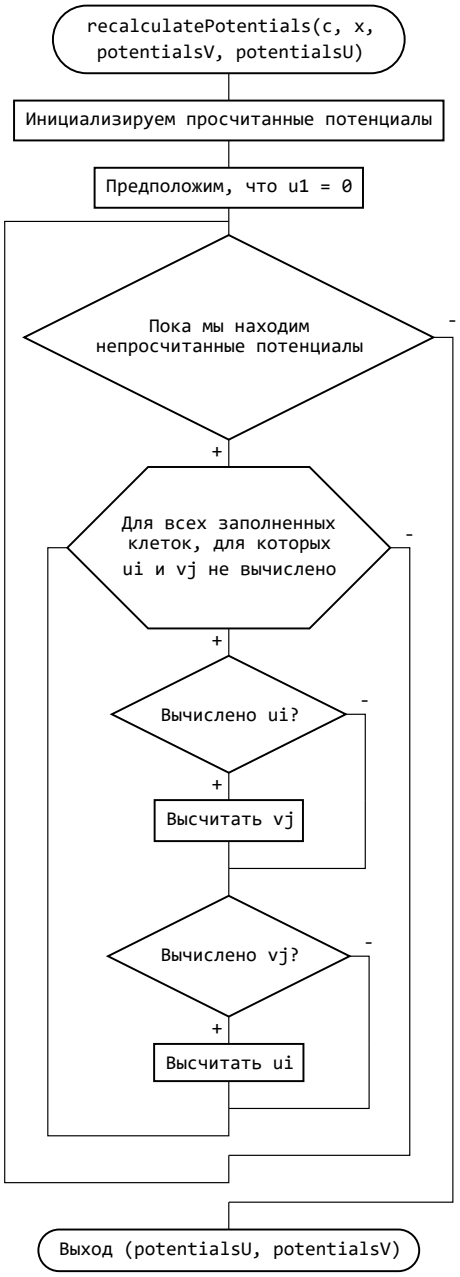
Блок-схемы:

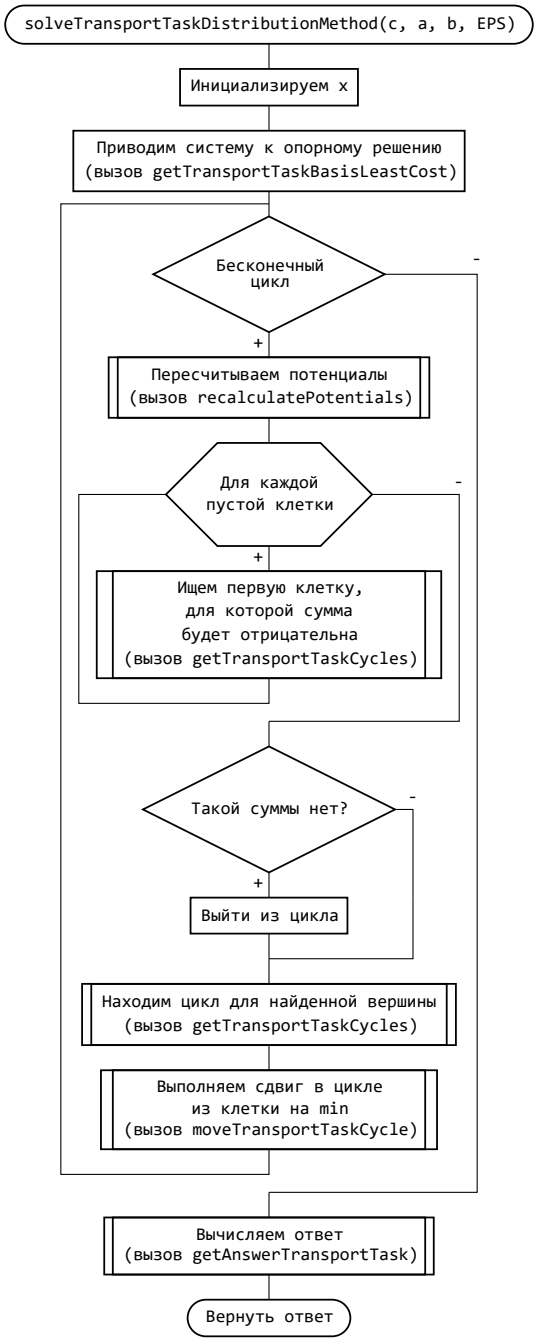



getTransportTaskBasisLeastCost(c, x, a, b, EPS)
Бесконечный цикл
-
+
Для каждого элемента cij, оставшегося в матрице
-
+
cij < cmin
-
+
Обновляем cmin
Минимальный элемент найден?
-
+
Выйти из цикла
+
ai >= bj
-
Вычёркиваем строчку, обновляем x, a
Вычёркиваем столбец, обновляем x, b
Выход(x, a, b)


getTransportTaskCycles(c, x, a, b, init, result, curr, min, minAssigned, sum, it, state, EPS)
Ищем подходящие вершины и сохраняем в selectedNodes
Ранее был выполнен поиск по строчке
+
-
Выполняем поиск по столбцу
Выполняем поиск по строчке
Пропускаем текущий элемент и элементы, которые уже находятся в пути. Остальные элементы с xij != 0 добавляем в selectedNodes
Пропускаем текущий элемент и элементы, которые уже находятся в пути. Остальные элементы с xij != 0 добавляем в selectedNodes
Если вершин не найдено
Вернуть пустой массив
Для каждой найденной вершины node из selectedNodes
Если достигли начальной вершины
Добавляем в конечный результат цикл
Переходим к следующей вершине
Вершина отрицательная
Обновляем минимум
Вершина положительная
Добавляем cij к sum
Вычитаем cij из sum
Добавляем вершину в цикл
Рекурсивно вызываем поиск дальнейших вершин и вставляем в результат выполнения функции (вызов getTransportTaskCycles)
Вернуть массив циклов
moveTransportTaskCycle(x, path, min, EPS)
Для всех вершин в пути
Вершина положительная
Увеличиваем вершину на min
Уменьшаем вершину на min
Выход (x)


getAnswerTransportTask(c, x)
Считаем сумму result = cij * xij,
для всех i, j
Вернуть result
solveTransportTaskDistributionMethod(c, a, b, EPS)
Инициализируем x
Приводим систему к опорному решению
(вызов getTransportTaskBasisLeastCost)
Бесконечный
цикл
-
+
Для каждой
пустой клетки
-
+
Ищем первую клетку,
для которой сумма
будет отрицательна
(вызов getTransportTaskCycles)
Такая клетка есть?
-
+
Выйти из цикла
Выполняем сдвиг в цикле
из клетки на min
(вызов moveTransportTaskCycle)
Вычисляем ответ
(вызов getAnswerTransportTask)
Вернуть ответ


recalculatePotentials(c, x, potentialsV, potentialsU)
Инициализируем просчитанные потенциалы
Предположим, что u1 = 0
Пока мы находим непросчитанные потенциалы
-
+
Для всех заполненных клеток, для которых ui и vj не вычислено
-
+
Вычислено ui?
-
+
Высчитать vj
Вычислено vj?
-
+
Высчитать ui
Выход (potentialsU, potentialsV)



Листинг программы:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <limits>
#include <algorithm>
#include <array>

#include "../libs/alg/alg.h"

int main() {
    // Подготовить входные данные
    std::vector<std::array<double, 5>> c = {
    {{16}, {26}, {12}, {24}, {3}}},
    {{5}, {2}, {19}, {27}, {2}}},
    {{29}, {23}, {25}, {16}, {8}}},
    {{2}, {25}, {14}, {15}, {21}}}};
```

```
    std::array<double, 4> a{{14}, {14 }, {14}, {14}};
    std::array<double, 5> b{{13}, {5}, {13}, {12}, {13}};

    std::cout << solveTransportTaskPotentials(c, a, b, 0.00000001);
}
```

Ссылка на репозиторий

```
#pragma once

#include <vector>
#include <tuple>

#include "../fraction.hpp"

template <std::size_t T, std::size_t MatrixLines, typename CountType>
void getTransportTaskBasisNorthEast(std::vector<std::array<CountType, T>> &c, std::vector<std::array<CountType, T>> &x,
↪  std::array<CountType, MatrixLines> &a, std::array<CountType, T> &b, CountType EPS) {
    int k = 0;
    int r = 0;

    // Пока не дошли до края матрицы
    while (k < MatrixLines && r < T) {
        // Если ak == br
        if (abs(a[k] - b[r]) < EPS) {
            if (r == T - 1) {
                // Вычёркиваем строчку
                x[k][r] = b[r];
                a[k] -= b[r];
                r++;
            } else {
                // Вычёркиваем столбец
                x[k][r] = a[k];
                b[r] -= a[k];
                k++;
            }
        // Если ak > br
        } else if (a[k] > b[r]) {
            // Вычёркиваем строчку
            x[k][r] = b[r];
            a[k] -= b[r];
            r++;
        // Если ak < br
        } else {
            // Вычёркиваем столбец
            x[k][r] = a[k];
            b[r] -= a[k];
            k++;
        }
    }
}
```

```cpp
template <std::size_t T, std::size_t MatrixLines, typename CountType>
void getTransportTaskBasisLeastCost(std::vector<std::array<CountType, T>> &c, std::vector<std::array<CountType, T>> &x,
↪  std::array<CountType, MatrixLines> &a, std::array<CountType, T> &b, CountType EPS) {
    std::array<bool, T> usedCols = {};
    std::array<bool, MatrixLines> usedRows = {};

    // Бесконечный цикл
    while(true) {
        bool foundAny = false;
        int iMin, jMin;

        // Для каждого элемента cij, оставшегося в матрице
        for (int i = 0; i < MatrixLines; i++) {
            if (usedRows[i]) continue;

            for (int j = 0; j < T; j++) {
                if (usedCols[j]) continue;

                // Если cij < cmin
                if (!foundAny || c[iMin][jMin] > c[i][j]) {
                    // Обновлям данные
                    iMin = i;
                    jMin = j;

                    foundAny = true;
                }
            }
        }

        // Если элемент не найден, выходим из цикла
        if (!foundAny) break;

        // Если ai >= bj
        if (abs(a[iMin] - b[jMin]) < EPS || a[iMin] > b[jMin]) {
            // Вычёркиваем строчку, обновляем x, a
            usedCols[jMin] = true;
            x[iMin][jMin] = b[jMin];
            a[iMin] -= b[jMin];
        // Иначе
        } else {
            // Вычёркиваем столбец, обновляем x, b
            usedRows[iMin] = true;
            x[iMin][jMin] = a[iMin];
            b[jMin] -= a[iMin];
        }
    }
}

template <std::size_t T, std::size_t MatrixLines, typename CountType>
std::vector<std::tuple<CountType, CountType, std::vector<std::pair<int, int>>>>
↪  getTransportTaskCycles(std::vector<std::array<CountType, T>> c, std::vector<std::array<CountType, T>> x,
std::array<CountType, MatrixLines> a, std::array<CountType, T> b, std::pair<int, int> init, std::vector<std::pair<int,
↪  int>> result, std::pair<int, int> curr,
```

```
CountType min, bool minAssigned, CountType sum, int it, int state, CountType EPS) {
    std::vector<std::tuple<CountType, CountType, std::vector<std::pair<int, int>>>> recResult;
    std::vector<std::pair<int, int>> selectedNodes;

    // Ищем подходящие вершины и сохраняем в selectedNodes
    if (state == -1 || state == 0) {
        // Выполняем поиск по столбцу, если ранее был выполнен поиск по строчке
        // или мы находимся в начальной вершине
        state = 1;

        // Пропускаем текущий элемент и элементы, которые уже находятся в пути.
        // Остальные элементы с xij != 0 добавляем в selectedNodes
        for (int k = 0; k < MatrixLines; k++) {
            if (k == curr.first) continue;
            if ((abs(x[k][curr.second]) > EPS && std::find(result.begin(), result.end(), std::pair<int, int>(k,
            ↪  curr.second)) == result.end()) ||
            (it > 1 && std::pair<int, int>(k, curr.second) == init)) {
                selectedNodes.push_back({k, curr.second});
            }
        }
    } else {
        // Иначе по строке
        state = 0;

        // Пропускаем текущий элемент и элементы, которые уже находятся в пути.
        // Остальные элементы с xij != 0 добавляем в selectedNodes
        for (int k = 0; k < T; k++) {
            if (k == curr.second) continue;
            if ((abs(x[curr.first][k]) > EPS && std::find(result.begin(), result.end(), std::pair<int, int>(curr.first,
            ↪  k)) == result.end()) ||
            (it > 1 && std::pair<int, int>(curr.first, k) == init)) {
                selectedNodes.push_back({curr.first, k});
            }
        }
    }

    // Если вершин не найдено, возвращаем пустой массив
    if (selectedNodes.empty()) {
        return {};
    }

    it++;
    // Для каждой найденной вершины из selectedNodes
    for (auto& node : selectedNodes) {
        // Если достигли начальной вершины, добавляем в конечный результат цикл
        // и переходим к следующей вершине
        if (node == init) {
            recResult.push_back({sum, min, result});
            continue;
        }

        // Обновляем минимум, если вершина отрицательная
        auto newMin = min;
        auto newMinAssigned = minAssigned;
```

```
        if (it % 2 && (x[node.first][node.second] < newMin || !newMinAssigned)) {
            newMin = x[node.first][node.second];

            newMinAssigned = true;
        }

        // Добавляем cij к sum, если вершина со знаком плюс, иначе - вычитаем
        auto newSum = sum;
        if (it % 2 == 0) {
            newSum += c[node.first][node.second];
        } else {
            newSum -= c[node.first][node.second];
        }

        // Добавляем вершину в цикл
        auto newResult = result;
        newResult.push_back(node);

        // Рекурсивно вызываем поиск дальнейших вершин и вставляем
        // в результат выоплнения функции
        auto recNextResult = getTransportTaskCycles(c, x, a, b, init, newResult, node, newMin, newMinAssigned, newSum,
        ↪  it, state, EPS);
        recResult.insert(recResult.end(), recNextResult.begin(), recNextResult.end());
    }

    // Возвращаем массив циклов
    return recResult;
}

template <std::size_t T, std::size_t MatrixLines, typename CountType>
std::vector<std::tuple<CountType, CountType, std::vector<std::pair<int, int>>>>
↪  getTransportTaskCycles(std::vector<std::array<CountType, T>> c, std::vector<std::array<CountType, T>> x,
std::array<CountType, MatrixLines> a, std::array<CountType, T> b, int i, int j, CountType EPS) {
    // Инициализируем путь result, сумму sum и min
    std::pair<int, int> init = {i, j};
    std::vector<std::pair<int, int>> result = {init};
    std::pair<int, int> curr = init;
    CountType sum = c[i][j];
    bool minAssigned = false;
    CountType min;
    int it = 0;
    int state = -1;

    // Рекурсивно вызываем метод поиска циклов
    return getTransportTaskCycles(c, x, a, b, init, result, curr, min, minAssigned, sum, it, state, EPS);
}

template <std::size_t T, typename CountType>
void moveTransportTaskCycle(std::vector<std::array<CountType, T>> &x, std::vector<std::pair<int, int>> path, CountType
↪  min, CountType EPS) {
    // Для всех вершин в пути
    for (int i = 0; i < path.size(); i++) {
        // Если вершина с плюсом, добавляем сдвиг min, иначе - вычитаем
        if (i % 2 == 0) {
```

```
            x[path[i].first][path[i].second] += min;
        } else {
            x[path[i].first][path[i].second] -= min;
        }
    }
}

template <std::size_t T, std::size_t MatrixLines, typename CountType>
CountType getAnswerTransportTask(std::vector<std::array<CountType, T>> &c, std::vector<std::array<CountType, T>> &x) {
    // Считаем сумму result = cij * xij, для всех i, j
    CountType result = c[0][0] * x[0][0];
    for (int i = 0; i < MatrixLines; i++) {
        for (int j = 0; j < T; j++) {
            if (i == 0 && j == 0) continue;

            result += c[i][j] * x[i][j];
        }
    }

    // Вернуть result
    return result;
}

template <std::size_t T, std::size_t MatrixLines, typename CountType>
CountType solveTransportTaskDistributionMethod(std::vector<std::array<CountType, T>> c, std::array<CountType,
↪  MatrixLines> a, std::array<CountType, T> b, CountType EPS) {
    // Инициализируем x
    std::vector<std::array<CountType, T>> x;
    for (int i = 0; i < MatrixLines; i++) {
        x.push_back({});
    }

    // Приводим систему к опорному решению
    getTransportTaskBasisLeastCost(c, x, a, b, EPS);

    // В бесконечном цикле
    while (true) {
        bool foundAny = false;
        std::tuple<CountType, CountType, std::vector<std::pair<int, int>>> search;

        // Ищем незаполненную клетку, для которой сумма будет отрицательна
        for (int i = 0; i < MatrixLines && !foundAny; i++) {
            for (int j = 0; j < T; j++) {
                if (abs(x[i][j]) > EPS) continue;

                auto resArray = getTransportTaskCycles(c, x, a, b, i, j, EPS);
                if (resArray.empty()) continue;
                auto res = resArray[0];

                if (std::get<0>(res) < -EPS) {
                    search = res;
                    foundAny = true;
                    break;
                }
```

```
            }
        }

        if (!foundAny) break;

        std::vector<std::pair<int, int>> path = std::get<2>(search);
        CountType minv = std::get<1>(search);

        // Если такая сумма есть, то выполняем сдвиг на min, иначе - выходим из цикла
        moveTransportTaskCycle(x, path, minv, EPS);
    }

    // Возвращаем ответ
    return getAnswerTransportTask<T, MatrixLines, CountType>(c, x);
}

template <std::size_t T, std::size_t MatrixLines, typename CountType>
void recalculatePotentials(std::vector<std::array<CountType, T>> &c, std::vector<std::array<CountType, T>> &x,
↪  std::array<CountType, T> &potentialsV, std::array<CountType, MatrixLines> &potentialsU, CountType EPS) {
    // Инициализируем просчитанные потенциалы
    std::array<bool, T> potentialsVFound = {};
    std::array<bool, MatrixLines> potentialsUFound = {};

    // Предположим, что u1 = 0
    potentialsUFound[0] = true;
    potentialsU = {0};

    // Пока мы находим непросчитанные потенциалы
    bool foundAny = true;
    while (foundAny) {
        foundAny = false;

        // Для всех заполненных клеток, для которых ui и vj не вычислено
        for (int i = 0; i < MatrixLines; i++) {
            for (int j = 0; j < T; j++) {
                if (abs(x[i][j]) < EPS) continue;
                if (potentialsUFound[i] && potentialsVFound[j]) continue;

                // Если вычислен ui, вычисляем vj
                if (potentialsUFound[i]) {
                    potentialsVFound[j] = true;
                    potentialsV[j] = c[i][j] - potentialsU[i];

                    foundAny = true;
                }

                // Если вычислен vj, вычисляем ui
                if (potentialsVFound[j]) {
                    potentialsUFound[i] = true;
                    potentialsU[i] = c[i][j] - potentialsV[j];

                    foundAny = true;
                }
            }
```

```
        }
    }

}

template <std::size_t T, std::size_t MatrixLines, typename CountType>
CountType solveTransportTaskPotentials(std::vector<std::array<CountType, T>> c, std::array<CountType, MatrixLines> a,
↪  std::array<CountType, T> b, CountType EPS) {
    // Инициализируем x
    std::vector<std::array<CountType, T>> x;
    for (int i = 0; i < MatrixLines; i++) {
        x.push_back({});
    }

    // Приводим систему к опорному решению
    getTransportTaskBasisLeastCost(c, x, a, b, EPS);

    std::array<CountType, T> potentialsV;
    std::array<CountType, MatrixLines> potentialsU;

    // В бесконечном цикле
    while (true) {
        // Пересчитываем потенциалы
        recalculatePotentials(c, x, potentialsV, potentialsU, EPS);
        int i, j;
        bool foundAny = false;
        // Для каждой пустой клетки
        for (i = 0; i < MatrixLines; i++) {
            for (j = 0; j < T; j++) {
                if (x[i][j] > EPS) continue;

                CountType t = c[i][j] - (potentialsU[i] + potentialsV[j]);

                // Ищем первую клетку, для которой сумма будет отрицательна
                if (t < -EPS) {
                    foundAny = true;
                    break;
                }
            }
            if (foundAny) break;
        }

        // Если такой суммы нет, выходим из цикла
        if (!foundAny) break;

        // Иначе находим цикл для найденной вершины
        auto search = getTransportTaskCycles(c, x, a, b, i, j, EPS)[0];
        // И выполняем сдвиг на min
        moveTransportTaskCycle(x, std::get<2>(search), std::get<1>(search), EPS);
    }

    // Возвращаем ответ
    return getAnswerTransportTask<T, MatrixLines, CountType>(c, x);
```

```
}
```

Ссылка на репозиторий

Результат выполнения программы:

```
426
```

Результаты вычислений:

$$\vec{a} = (14, 14, 14, 14);$$
$$\vec{b} = (13, 5, 13, 12, 13);$$
$$C = \begin{pmatrix} 16 & 26 & 12 & 24 & 3 \\ 5 & 2 & 19 & 27 & 2 \\ 29 & 23 & 25 & 16 & 8 \\ 2 & 25 & 14 & 15 & 21 \end{pmatrix}$$

a \ b	13	5	13	12	13
14	16	26	12	24	3
14	5	2	27	27	2
14	29	23	25	16	8
14	2	25	14	15	21

Для приведения к опорному виду используем метод наименьшей стоимости

a \ b	13	5	13	12	13
14	16	26	12 10	24	3 4
14	5	2 5	27	27	2 9
14	29	23	25 2	16 12	8
14	2 13	25	14 1	15	21

Введём потенциалы, где $u_1 = 0$.

		v_1	v_2	v_3	v_4	v_5
	b / a	13	5	13	12	13
u_1	14	16	26	12 / 10	24	3 / 4
u_2	14	5	2 / 5	27	27	2 / 9
u_3	14	29	23	25 / 2	16 / 12	8
u_4	14	2 / 13	25	14 / 1	15	21

$$\begin{cases} u_1 + v_3 = 12 \\ u_1 + v_5 = 3 \\ u_2 + v_2 = 2 \\ u_2 + v_5 = 2 \\ u_3 + v_3 = 25 \\ u_3 + v_4 = 16 \\ u_4 + v_1 = 2 \\ u_4 + v_3 = 14 \end{cases}$$

$$\begin{cases} u_1 = 0 \\ u_2 = -1 \\ u_3 = 13 \\ u_4 = 2 \end{cases} \quad \begin{cases} v_1 = 0 \\ v_2 = 3 \\ v_3 = 12 \\ v_4 = 3 \\ v_5 = 3 \end{cases}$$

Просчитаем γ:

$\gamma_{11} = 16$
$\gamma_{12} = 23$
$\gamma_{14} = 21$
$\gamma_{21} = 6$
$\gamma_{23} = 8$
$\gamma_{24} = 25$
$\gamma_{31} = 16$
$\gamma_{32} = 7$
$\gamma_{35} = -8$
$\gamma_{42} = 20$
$\gamma_{44} = 10$
$\gamma_{45} = 16$

$\gamma_{35} < 0$, цикл в этой точке:

$3,4 \to 1,5 \to 1,3 \to 3,3$. Минимум: 2. Выполним сдвиг, получим новое опорное решение:

		v_1	v_2	v_3	v_4	v_5
	b / a	13	5	13	12	13
u_1	14	16	26	12 / 12	24	3 / 2
u_2	14	5	2 / 5	27	27	2 / 9
u_3	14	29	23	25	16 / 12	8 / 2
u_4	14	2 / 13	25	14 / 1	15	21

$$\begin{cases} u_1 + v_3 = 12 \\ u_1 + v_5 = 3 \\ u_2 + v_2 = 2 \\ u_2 + v_5 = 2 \\ u_3 + v_4 = 16 \\ u_3 + v_5 = 8 \\ u_4 + v_1 = 2 \\ u_4 + v_3 = 14 \end{cases}$$

$$\begin{cases} u_1 = 0 \\ u_2 = -1 \\ u_3 = 5 \\ u_4 = 2 \end{cases} \quad \begin{cases} v_1 = 0 \\ v_2 = 3 \\ v_3 = 12 \\ v_4 = 11 \\ v_5 = 3 \end{cases}$$

Просчитаем γ:

$\gamma_{11} = 16$
$\gamma_{12} = 23$
$\gamma_{14} = 13$
$\gamma_{21} = 6$
$\gamma_{23} = 8$
$\gamma_{24} = 17$
$\gamma_{31} = 24$
$\gamma_{32} = 15$
$\gamma_{33} = 8$
$\gamma_{42} = 20$
$\gamma_{44} = 2$
$\gamma_{45} = 16$

Так как все $\gamma > 0$, мы достигли оптимального плана.

$Z_{min} = 12 \cdot 12 + 2 \cdot 3 + 5 \cdot 2 + 9 \cdot 2 + 12 \cdot 16 + 2 \cdot 8 + 13 \cdot 2 + 1 \cdot 14 = 426.$

Вывод: в ходе лабораторной работы изучили математическую модель транспортной задачи, овладели методами решения этой задачи.