Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Омский государственный технический университет»

Кафедра «Автоматизированные системы обработки информации и

управления»

**РАСЧЁТНО-ГРАФИЧЕСКАЯ РАБОТА**

На тему: «Транспортная задача»

По дисциплине: «Оптимизация и принятие решений в технических системах»

Выполнили:

студенты группы ИВТ-143

Граков М.С.

Щербина К.А.

Проверил:

Рафалович С.А.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Омск-2015

Реферат

Пояснительная записка: 29 с., 20 рис., 0 табл., 0 прил., 1 источник

ТРАНСПОРТНАЯ ЗАДАЧА, МЕТОД СЕВЕРО-ЗАПАДНОГО УГЛА, МИНИМАЛЬНЫЕ ЗАТРАТЫ

Цель работы: самостоятельное, практическое исследование, приобретение на практике практических знаний и закрепление теоретического материала.

Степень внедрения: для учебных целей.

Содержание

[Введение 4](#_Toc432940170)

[Схемы алгоритмов 5](#_Toc432940171)

[Тестирование 18](#_Toc432940172)

[Листинг 20](#_Toc432940173)

[Заключение 28](#_Toc432940174)

[Список используемых источников 29](#_Toc432940175)

# Введение

Целью работы стало самостоятельное, практическое исследование, приобретение на практике практических знаний и закрепление теоретического материала по предмету «Оптимизация и принятие решений в технических системах» на тему «Транспортная задача».

Транспортная задача используется для моделирования оптимизации экономических проблем, связанных с формированием оптимального плана перевозок грузов и оптимального распределения индивидуальных контрактов на транспортировки заказов.

Цель задачи состоит в минимизации суммарной стоимости перевозок. Эта цель может быть достигнута с помощью оптимальной организации перевозок грузов. Следовательно, за неизвестные можно принять количество грузов, перевозимого от каждого поставщика каждому потребителю.

**Пусть – количество груза, перевозимого от i-го поставщика j-му потребителю. Параметры задачи – число поставщиков и потребителей, количество груза на каждом складе, необходимое количество товара для каждого потребителя, тарифы на перевозки.**

**Критерием эффективности (целевой функцией) являются суммарные затраты S на перевозку, равные сумме произведений тарифов на перевозку на количество перевозимого сырья от каждого поставщика каждому потребителю.**

**Целевая функция и ограничения линейны, т.е. данная задача относится к задачам линейного программирования, однако, благодаря особой структуре, эта задача получила специальное название: транспортная задача или транспортная модель.**

# Схемы алгоритмов

Основная функция solveTask, реализующая весь алгоритм задачи, представлена на рисунках 1 и 2.

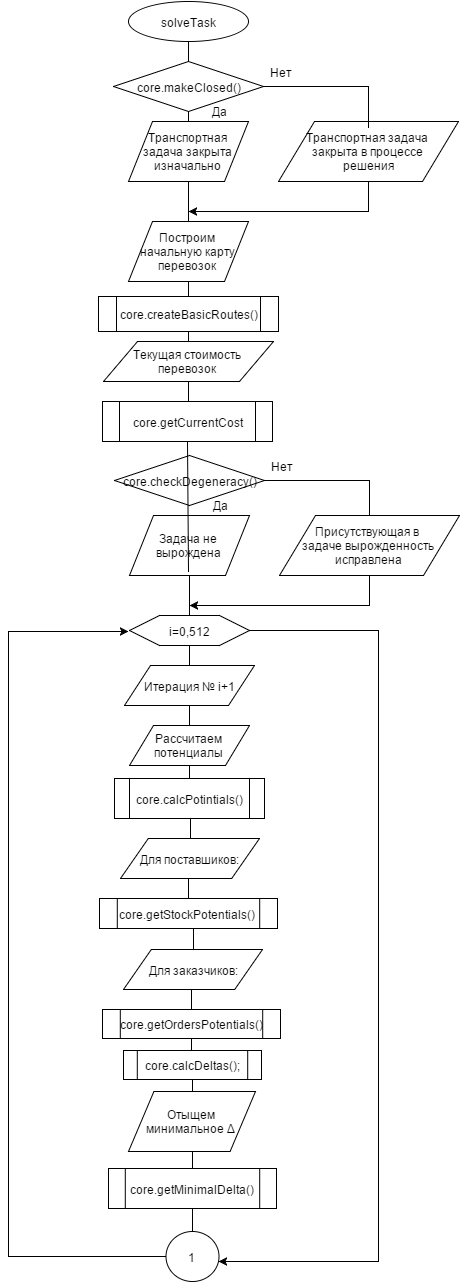


Рисунок 1 – функция SolveTask

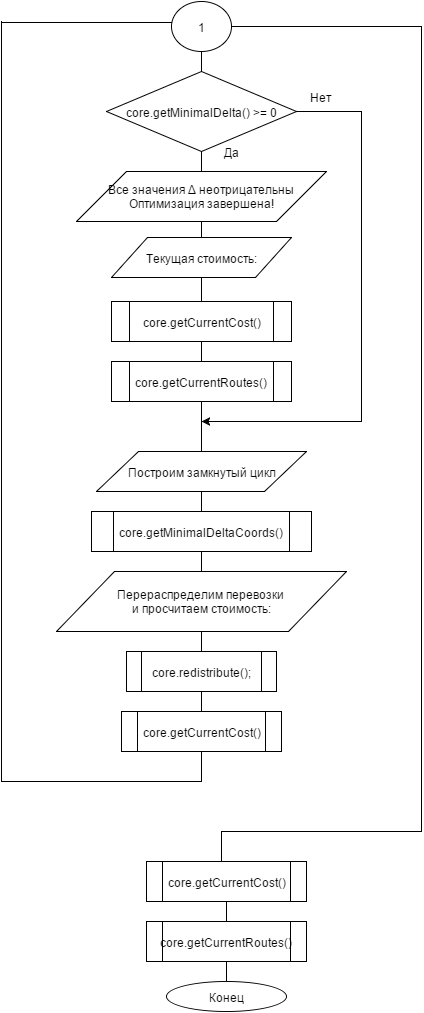


Рисунок 2 – функция SolveTask

Функция makeClosed, реализующая «закрытие» транспортной задачи, если она была «открыта» (количество груза на складах и требуемое количество груза не совпадало), представлена на рисунке 3.

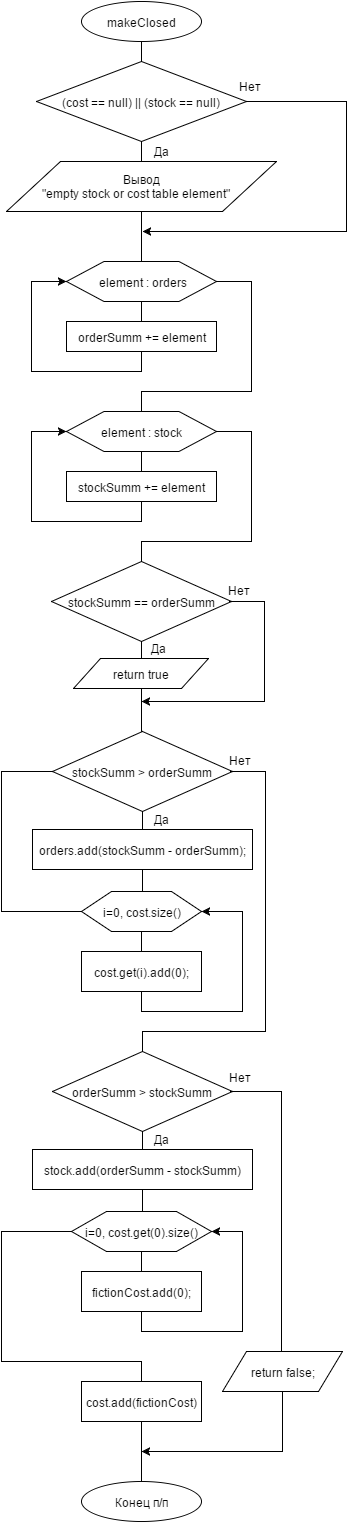


Рисунок 4 – функция makeClosed

Функция createBasicRoutes, реализующая построение начальной карты перевозок, представлена на рисунке 5.

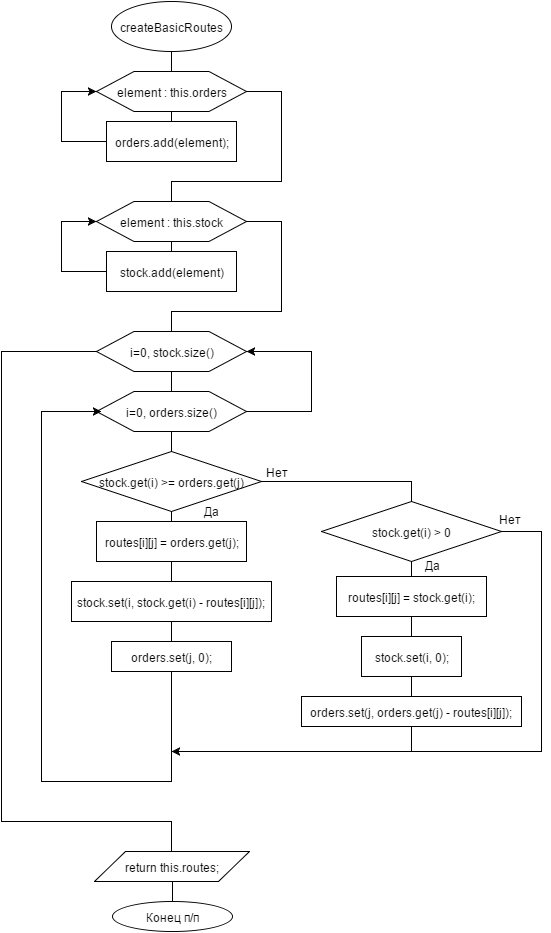


Рисунок 5 – функция createBasicRoutes

Функция getCurrentCost, реализующая определение текущей стоимости перевозок, представлена на рисунке 6.

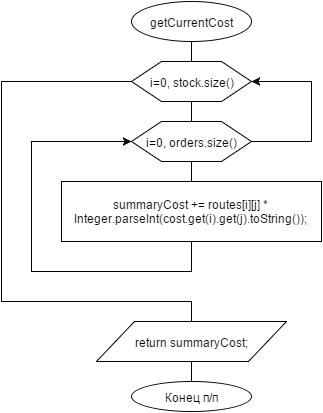


Рисунок 6 – функция getCurrentCost

Функция checkDegeneracy, реализующая проверку транспортной задачи на вырожденность и ее исправление, представлена на рисунке 7.

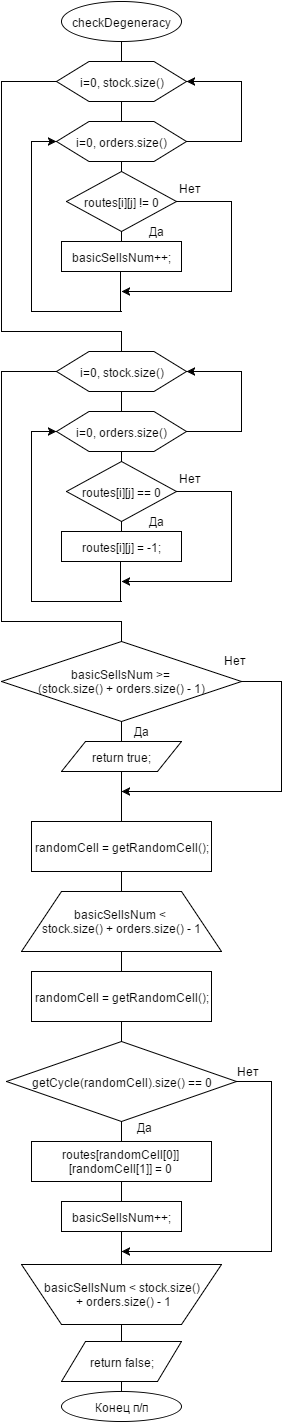


Рисунок 7 – функция checkDegeneracy

Функция calcPotintials, реализующая расчёт потенциалов, представлена на рисунке 8.

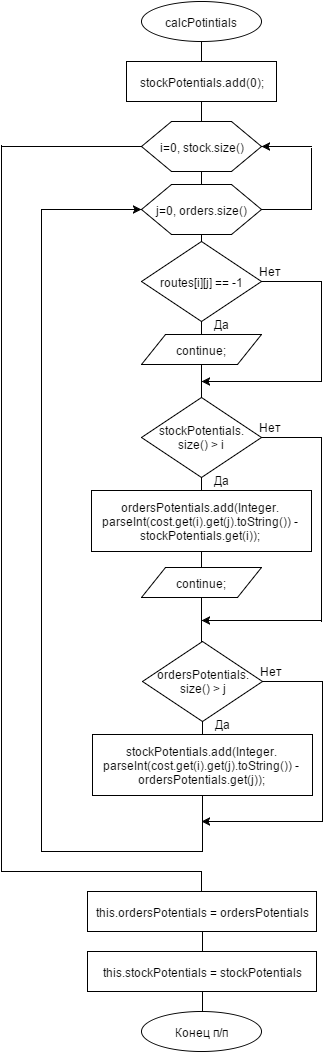


Рисунок 8 – функция calcPotintials

Функция calcDeltas, выполняющая расчёт дельты для каждой ячейки, представлена на рисунке 9.

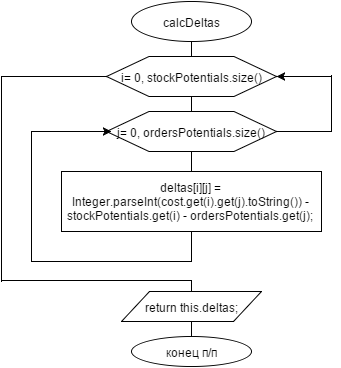


Рисунок 9 – функция calcDeltas

Функция getMinimalDeltaCoords, выполняющая определение минимальной дельты среди всех ячеек, представлена на рисунке 10.

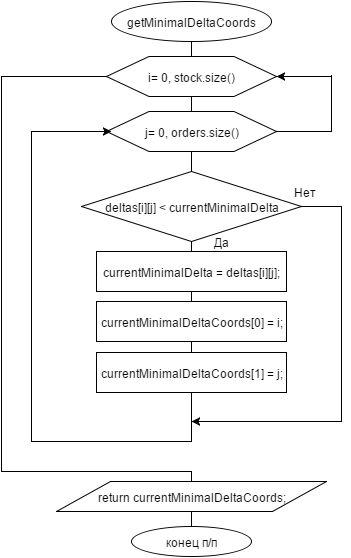


Рисунок 10 - getMinimalDeltaCoords

Функция getRandomCell, выполняющая определение случайной ячейки на таблице тарифов, представлена на рисунке 11.

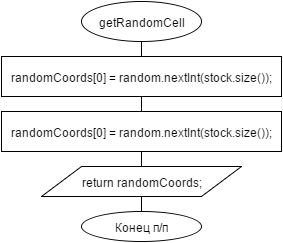


Рисунок 11 – функция getRandomCell

Функция getCycle, реализующая поиск замкнутого цикла для перераспределения товара в текущей итерации, представлена на рисунке 12.

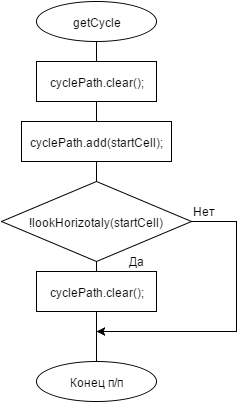


Рисунок 12 – функция getCycle

Функция lookHorizotaly, реализующая поиск следующей в цикле горизонтальной ячейки в тарифной таблице представлена на рисунке 13.

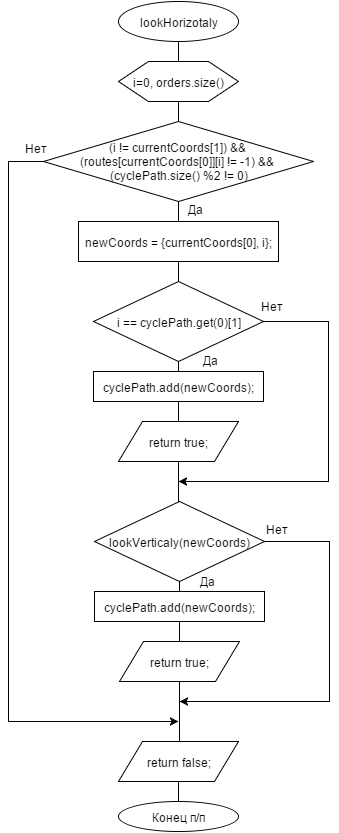


Рисунок 13 – функция lookHorizotaly

Функция lookVerticaly, реализующая поиск следующей в цикле вертикальной ячейки в тарифной таблице представлена на рисунке 14.

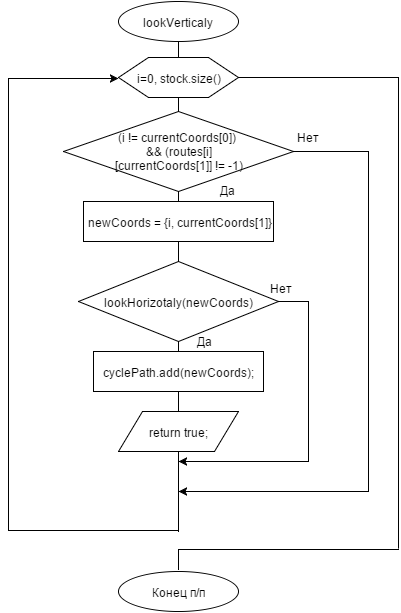


Рисунок 14 - функция lookVerticaly

Функция redistribute, выполняющая перераспределение поставок, представлена на рисунке 15.

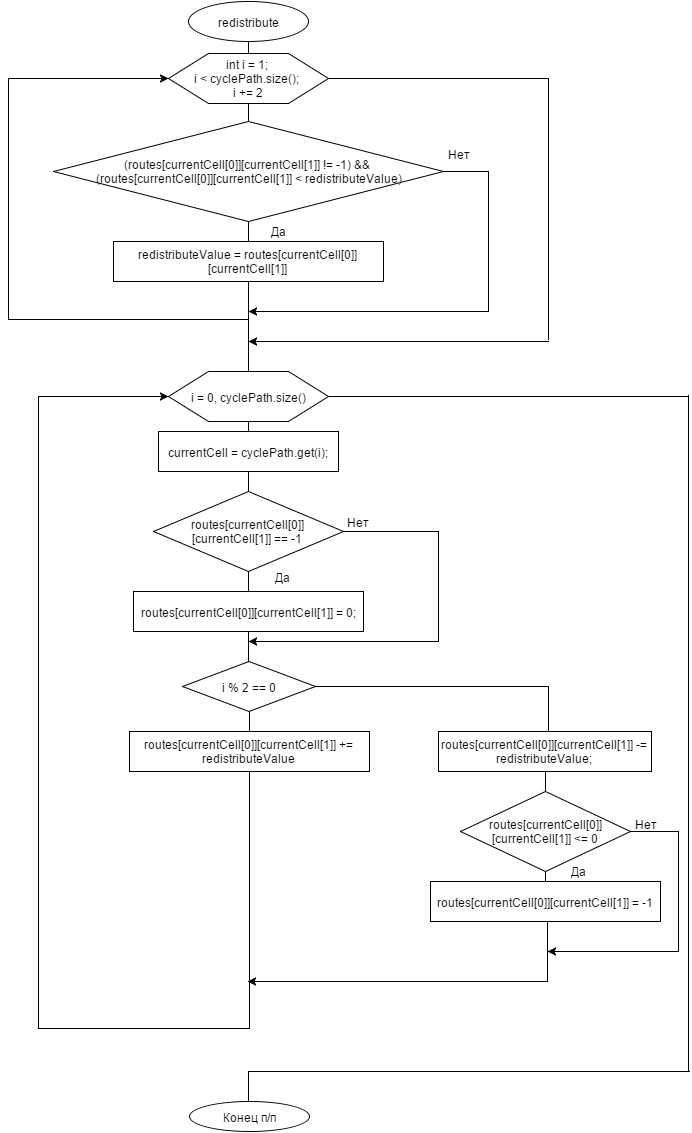


Рисунок 15 – функция redistribute

Функция getMinimalDelta, выполняющая получение минимальной дельты в текущей итерации, представлена на рисунке 16.

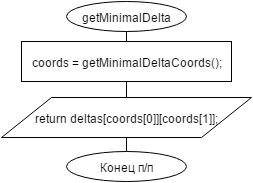


Рисунок 16 – функция getMinimalDelta

# Тестирование

1. Изначально открытая задача.

В качестве исходных данных возьмём значения из пособия.

Скриншот работы программы представлен на рисунке 17.

Исходные данные и ответ из пособия представлены на рисунке 18.

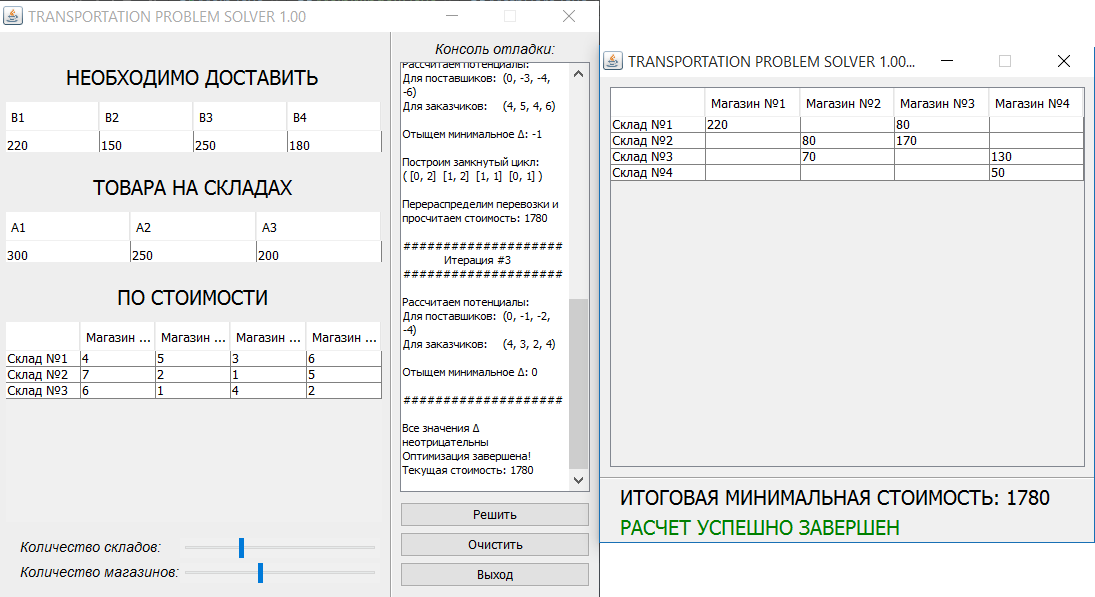


Рисунок 17 – скриншот программы

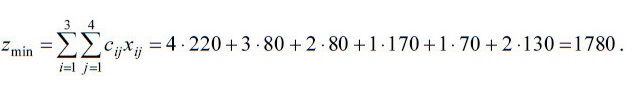
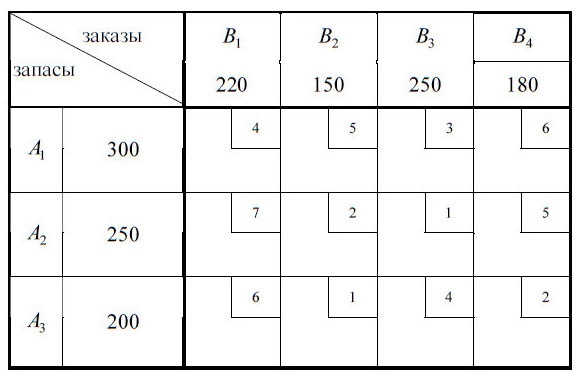


Рисунок 18 – Исходные данные и ответ из пособия

1. Транспортная задача изначально закрыта.

В качестве исходных данных возьмём свои придуманные значения.

Скриншот работы программы представлен на рисунке 19.

Проверка правильности работы программы произведём с помощью сайта решения транспортной работы, представлено на рисунке 20.

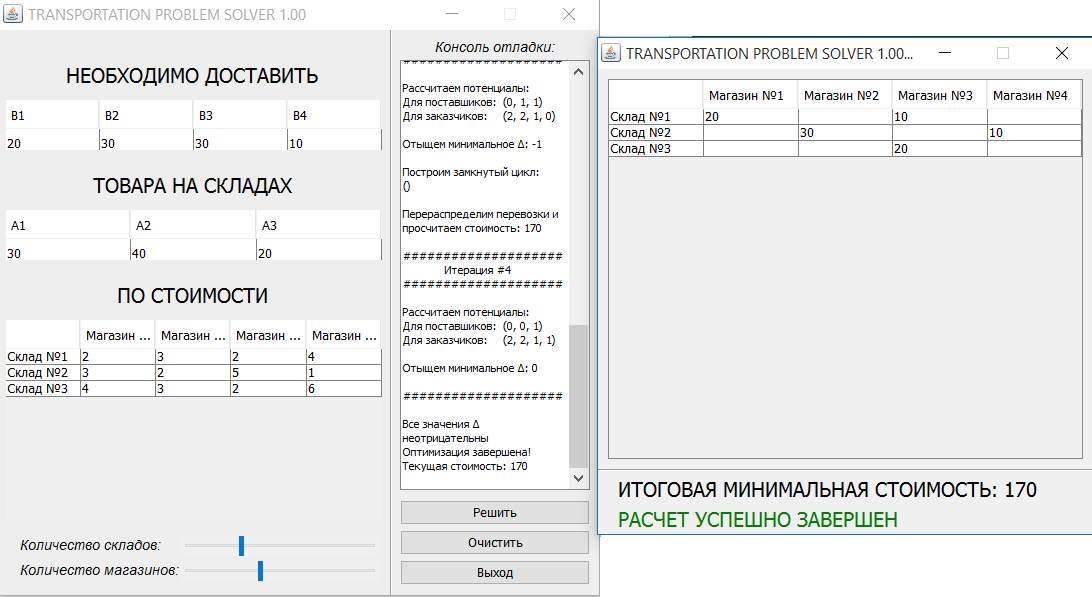


Рисунок 19 – скриншот программы

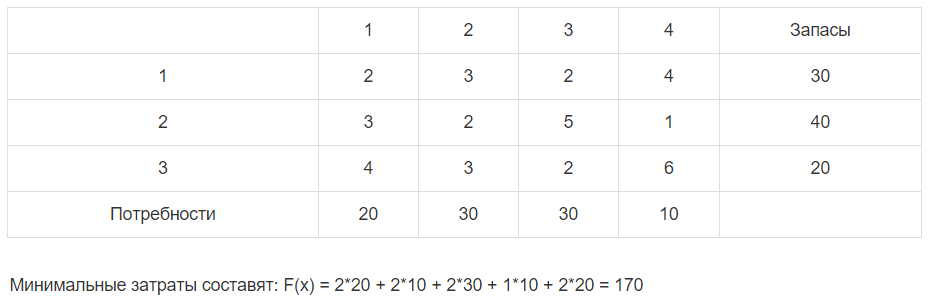


Рисунок 20 – Исходные данные и ответ, полученные с помощью онлайн-решения стороннего сервиса

# Листинг исходного текста программы

MathCore:

public class MathCore {

private List<Integer> orders = new ArrayList<>();

private List<Integer> stock = new ArrayList<>();

private List<Integer> ordersPotentials = new ArrayList<>();

private List<Integer> stockPotentials = new ArrayList<>();

private List<List> cost = new ArrayList<>();

private List<int[]> cyclePath = new ArrayList<>();

private int[][] routes;

public int[][] deltas;

MathCore(List orders, List stock, List cost) {

if (orders != null) {

this.orders = orders;

}

if (cost != null) {

this.cost = cost;

}

if (stock != null) {

this.stock = stock;

}

}

/\*\*

\* Will make our task conditions math-closed

\* Will finalize all Lists to arrays

\*

\* @return true, if we have closed problem; false, if problem was closed in method

\*/

public boolean makeClosed() {

if ((cost == null) || (stock == null)) {

throw new RuntimeException("empty stock or cost table element");

}

int orderSumm = 0, stockSumm = 0;

for (int element : orders) {

orderSumm += element;

}

for (int element : stock) {

stockSumm += element;

}

if (stockSumm == orderSumm) {

return true; // we have closed problem

}

if (stockSumm > orderSumm) {

orders.add(stockSumm - orderSumm);

for (int i = 0; i < cost.size(); i++) {

cost.get(i).add(0);

}

}

if (orderSumm > stockSumm) {

stock.add(orderSumm - stockSumm);

List<Integer> fictionCost = new ArrayList<>();

for (int i = 0; i < cost.get(0).size(); i++) {

fictionCost.add(0);

}

cost.add(fictionCost);

}

return false; // problem was closed just now

}

/\*\*

\* will create basis routing table by NW-method

\*

\* @return int[][] routing table

\*/

public int[][] createBasicRoutes() {

routes = new int[stock.size()][orders.size()];

List<Integer> orders = new ArrayList<>();

for (int element : this.orders) {

orders.add(element);

}

List<Integer> stock = new ArrayList<>(this.stock.size());

for (int element : this.stock) {

stock.add(element);

}

for (int i = 0; i < stock.size(); i++) {

for (int j = 0; j < orders.size(); j++) {

if (stock.get(i) >= orders.get(j)) {

routes[i][j] = orders.get(j);

stock.set(i, stock.get(i) - routes[i][j]);

orders.set(j, 0);

} else if (stock.get(i) > 0) {

routes[i][j] = stock.get(i);

stock.set(i, 0);

orders.set(j, orders.get(j) - routes[i][j]);

}

}

}

return this.routes;

}

/\*\*

\* will calculate current summary cost for thr transportation

\* Also will write this value to class-member variable

\*

\* @return Current cost value (int)

\*/

public int getCurrentCost() {

int summaryCost = 0;

for (int i = 0; i < stock.size(); i++) {

for (int j = 0; j < orders.size(); j++) {

if (routes[i][j] > 0) {

summaryCost += routes[i][j] \* Integer.parseInt(cost.get(i).get(j).toString());

}

}

}

return summaryCost;

}

/\*\*

\* Will check problem for degeneracy and fix it if need

\*

\* @return true, if probles wasn't degeneracy and false, if it's

\*/

public boolean checkDegeneracy() {

int basicSellsNum = 0;

for (int i = 0; i < stock.size(); i++) {

for (int j = 0; j < orders.size(); j++) {

if (routes[i][j] > 0) {

basicSellsNum++;

}

}

}

for (int i = 0; i < stock.size(); i++) {

for (int j = 0; j < orders.size(); j++) {

if (routes[i][j] == 0) {

routes[i][j] = -1;

}

}

}

if (basicSellsNum >= (stock.size() + orders.size() - 1)) {

return true;

}

while (basicSellsNum < stock.size() + orders.size() - 1) {

int[] randomCell = getRandomCell();

while (routes[randomCell[0]][randomCell[1]] != -1) {

randomCell = getRandomCell();

}

if (getCycle(randomCell).size() == 0) {

routes[randomCell[0]][randomCell[1]] = 0;

basicSellsNum++;

}

}

return false;

}

/\*\*

\* getter for read-only access to routes array

\*

\* @return int[][] of routes

\*/

public int[][] getCurrentRoutes() {

return this.routes;

}

/\*\*

\* Will calculate potentials for orders and stock when cell is part of basis

\*/

public void calcPotintials() {

List<Integer> ordersPotentials = new ArrayList<>();

List<Integer> stockPotentials = new ArrayList<>();

stockPotentials.add(0);

for (int i = 0; i < stock.size(); i++) {

for (int j = 0; j < orders.size(); j++) {

if (routes[i][j] == -1) {

continue;

}

if (stockPotentials.size() > i) {

ordersPotentials.add(Integer.parseInt(cost.get(i).get(j).toString()) - stockPotentials.get(i));

continue;

}

if (ordersPotentials.size() > j) {

stockPotentials.add(Integer.parseInt(cost.get(i).get(j).toString()) - ordersPotentials.get(j));

}

}

}

this.ordersPotentials = ordersPotentials;

this.stockPotentials = stockPotentials;

}

/\*\*

\* getter for orderPotentials

\*

\* @return String with orders potentials

\*/

public String getOrdersPotentials() {

return ordersPotentials.toString().substring(1, ordersPotentials.toString().length() - 1);

}

/\*\*

\* getter for stockPotentials

\*

\* @return String with stocks potentials

\*/

public String getStockPotentials() {

return stockPotentials.toString().substring(1, stockPotentials.toString().length() - 1);

}

/\*\*

\* Will calculate deltas for every route cell. For basic cell it's must be zero always

\*

\* @return int[][] deltas array of int

\*/

public int[][] calcDeltas() {

deltas = new int[stock.size()][orders.size()];

for (int i = 0; i < stockPotentials.size(); i++) {

for (int j = 0; j < ordersPotentials.size(); j++) {

deltas[i][j] = Integer.parseInt(cost.get(i).get(j).toString()) - stockPotentials.get(i) - ordersPotentials.get(j);

}

}

return this.deltas;

}

/\*\*

\* Will find minimal delta in delta table

\*

\* @return int[2], coords of minimal deltas

\*/

public int[] getMinimalDeltaCoords() {

int[] currentMinimalDeltaCoords = new int[2];

int currentMinimalDelta = Integer.MAX\_VALUE;

for (int i = 0; i < stock.size(); i++) {

for (int j = 0; j < orders.size(); j++) {

if (deltas[i][j] < currentMinimalDelta) {

currentMinimalDelta = deltas[i][j];

currentMinimalDeltaCoords[0] = i;

currentMinimalDeltaCoords[1] = j;

}

}

}

return currentMinimalDeltaCoords;

}

/\*\*

\* Will get random (really) cell in routes area

\*

\* @return int[2] with coords

\*/

private int[] getRandomCell() {

int[] randomCoords = new int[2];

Random random = new Random();

randomCoords[0] = random.nextInt(stock.size());

randomCoords[1] = random.nextInt(orders.size());

return randomCoords;

}

/\*\*

\* Return List with coordinates of basis array's cells when it can be or empty list when cycle isn't possible

\* List.size() == 0 when cycle is't possible

\*

\* @param startCell

\* @return List with coords of cell

\*/

public List<int[]> getCycle(int[] startCell) {

cyclePath.clear();

cyclePath.add(startCell);

if (!lookHorizotaly(startCell)) {

cyclePath.clear();

}

return cyclePath;

}

/\*\*

\* Will search for basis cell onto this line of routing table

\*

\* @param currentCoords

\* @return true, if we can go to start cell and false, if not

\*/

private boolean lookHorizotaly(int[] currentCoords) {

for (int i = 0; i < orders.size(); i++) {

if ((i != currentCoords[1]) && (routes[currentCoords[0]][i] != -1) && (cyclePath.size() %2 != 0)) {

int[] newCoords = {currentCoords[0], i};

if (i == cyclePath.get(0)[1]) {

cyclePath.add(newCoords);

return true;

}

if (lookVerticaly(newCoords)) {

cyclePath.add(newCoords);

return true;

}

}

}

return false;

}

/\*\*

\* Will search for basis cell onto this column of routing table

\*

\* @param currentCoords

\* @return true, if we can go to start cell and false, if not

\*/

private boolean lookVerticaly(int[] currentCoords) {

for (int i = 0; i < stock.size(); i++) {

if ((i != currentCoords[0]) && (routes[i][currentCoords[1]] != -1)) {

int[] newCoords = {i, currentCoords[1]};

if (lookHorizotaly(newCoords)) {

cyclePath.add(newCoords);

return true;

}

}

}

return false;

}

/\*\*

\* Will redistribute routing plane by cyclePath

\*/

public void redistribute() {

int redistributeValue = Integer.MAX\_VALUE;

for (int i = 1; i < cyclePath.size(); i += 2) {

int[] currentCell = cyclePath.get(i);

if ((routes[currentCell[0]][currentCell[1]] != -1) && (routes[currentCell[0]][currentCell[1]] < redistributeValue)) {

redistributeValue = routes[currentCell[0]][currentCell[1]];

}

}

for (int i = 0; i < cyclePath.size(); i++) {

int[] currentCell = cyclePath.get(i);

if (routes[currentCell[0]][currentCell[1]] == -1) {

routes[currentCell[0]][currentCell[1]] = 0;

}

if (i % 2 == 0) {

routes[currentCell[0]][currentCell[1]] += redistributeValue;

} else {

routes[currentCell[0]][currentCell[1]] -= redistributeValue;

if (routes[currentCell[0]][currentCell[1]] <= 0) {

routes[currentCell[0]][currentCell[1]] = -1;

}

}

}

}

/\*\*

\* Will return current minimal delta in routing table

\*

\* @return int, minimal delta

\*/

public int getMinimalDelta() {

int[] coords = getMinimalDeltaCoords();

return deltas[coords[0]][coords[1]];

}

}

Solver:

import java.util.ArrayList;

import java.util.List;

public class Solver {

MainWindow mainWindow;

DebugLogger logger;

Solver(MainWindow mainWindow, DebugLogger logger) {

if (mainWindow != null) {

this.mainWindow = mainWindow;

}

if (logger != null) {

this.logger = logger;

}

}

public List solveTask(int maxIterations) {

List result = new ArrayList();

MathCore core = new MathCore(mainWindow.getShopNeeds(), mainWindow.getStorageStock(), mainWindow.getCostTable());

/\*Make our task closed\*/

if (core.makeClosed()) {

logger.writeLine("Транспортная задача закрыта изначально");

} else {

logger.writeLine("Транспортная задача закрыта в процессе решения");

}

logger.separate();

/\*create first routing map\*/

logger.writeLine("Построим начальную карту перевозок:");

logger.writeDoubleArrayInt(core.createBasicRoutes());

/\*calculate current cost\*/

logger.writeLine("Текущая стоимость перевозок: " + core.getCurrentCost());

logger.separate();

/\*check for degeneracy and fix it\*/

if (core.checkDegeneracy()) {

logger.writeLine("Задача не вырождена");

} else {

logger.writeLine("Присутствующая в задаче вырожденность исправлена");

}

logger.separate();

/\*start iterations. maxIterations is need for avoid looping\*/

logger.writeLine("Начинаем итерационный процесс:");

for (int i = 0; i < maxIterations; i++) {

/\*write some decorations\*/

logger.separate();

logger.writeLine("####################");

logger.writeLine((" Итерация #" + (i + 1)));

logger.writeLine("####################");

logger.separate();

/\*calculate potentials for orders and storages\*/

logger.writeLine("Рассчитаем потенциалы:");

core.checkDegeneracy();

core.calcPotintials();

logger.writeLine("Для поставшиков: (" + core.getStockPotentials() + ")");

logger.writeLine("Для заказчиков: (" + core.getOrdersPotentials() + ")");

logger.separate();

/\*calculate deltas for every cell\*/

core.calcDeltas();

/\*find minimal delta\*/

logger.write("Отыщем минимальное Δ: ");

logger.writeLine(String.valueOf(core.getMinimalDelta()));

logger.separate();

/\*logger.writeDoubleArrayInt(core.deltas);

logger.writeDoubleArrayInt(core.getCurrentRoutes());\*/

/\*check the iterative process to perfection\*/

if (core.getMinimalDelta() >= 0) {

logger.writeLine("####################");

logger.separate();

logger.writeLine("Все значения Δ неотрицательны\nОптимизация завершена!");

logger.writeLine("Текущая стоимость: " + core.getCurrentCost());

result.add(core.getCurrentCost());

result.add(core.getCurrentRoutes());

result.add(true);

return result;

}

/\*make a cyclical route on basic cells\*/

logger.writeLine("Построим замкнутый цикл:");

List<int[]> cycle = core.getCycle(core.getMinimalDeltaCoords());

logger.writePath(cycle);

logger.separate();

/\*redistribute routing\*/

logger.write("Перераспределим перевозки и просчитаем стоимость: ");

core.redistribute();

/\*calculate current cost\*/

logger.writeLine("" + core.getCurrentCost());

}

/\*return our data as-is. Maybe something is wrong?\*/

result.add(core.getCurrentCost());

result.add(core.getCurrentRoutes());

result.add(false);

return result;

}

}

# Заключение

Результатом работы является программа, решающая транспортную задачу северо-западным углом.

Программа реализована на языке программирования Java SE 1.8.

Все вводимые данные выводятся в виде таблицы. Все пошаговые действия выводятся в окне консоли отладки.

В программе удобный и понятный пользовательский интерфейс. Для ввода данных используется клавиатура и мышь. Данные, выводимые программой, соответствуют тем, что получены при расчетах без использования компьютера. Таким образом, поставленная задача была выполнена.

# Список используемых источников

1. Источник 1