**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 5

**З дисципліни:** *“Дослідження операцій”*

**На тему:** *“Розв’язування транспортної задачі ЛП методом*

*диференціальних рент”*

**Лектор:**

доц. каф. ПЗ

Журавчак Л. М.

**Виконав:**

ст. гр. ПЗ-23

Михалевич П.-І.В.

**Прийняла:**

асист. каф. ПЗ

Івасько Н. М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑= \_\_\_\_\_ .

Львів – 2022

**Тема роботи:** Розв’язування транспортної задачі ЛП методом диференціальних рент.

**Мета роботи:** Ознайомитись на практиці із розв’язуванням транспортних задач за допомогою побудови умовно оптимального розподілу та покращування його за методом диференціальних рент.

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

3.

6.

9.

**ЗАВДАННЯ**

1. Отримати індивідуальний варіант завдання.

2. Написати програму розв’язування транспортної задачі методом диференціальних рент згідно із варіантом із Додатка до лабораторних робіт № 4 та № 5.

3. Оформити звіт про виконану роботу.

4. Продемонструвати викладачеві результати, відповісти на запитання

стосовно виконання роботи.

**ХІД ВИКОНАННЯ**

1. Індивідуальне завдання.

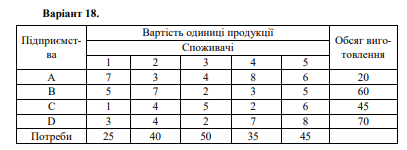


Рис. 1. Індивідуальне завдання

1. Код програми:

void TransportProblem::StartRent() {

CreateProblem(false, "main.json");

if (!IsClosed()) {

std::cout << "Proble is not closed type!";

}

FindInitialFucntion();

CalculateFunction("Result function after finding supporting plan:");

ShowTableRents();

CalculateDiference();

ShowTableRents();

while (HasNegative()) {

GoToNextRentTable();

CalculateDiference();

ShowTableRents();

}

CalculateFunction("Found optimal:");

}

double TransportProblem::CalculateGived(std::vector<double> a\_rows, std::vector<double> b\_cols, std::vector<std::vector<Cell>> table) {

std::vector<double> minValues(table[0].size(), 1000000);

std::vector<int> minValuesCount(table[0].size(), 0);

std::vector<int> minValuesRowCount(table.size(), 0);

for (int colIndex = 0; colIndex < table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < table.size(); rowIndex++) {

table[rowIndex][colIndex].bottom = 0;

table[rowIndex][colIndex].used = false;

if (minValues[colIndex] > table[rowIndex][colIndex].top) {

minValues[colIndex] = table[rowIndex][colIndex].top;

minValuesCount[colIndex] = 0;

}

if (minValues[colIndex] == table[rowIndex][colIndex].top) {

minValuesCount[colIndex] += 1;

}

}

}

for (int colIndex = 0; colIndex < table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < table.size(); rowIndex++) {

if (minValues[colIndex] == table[rowIndex][colIndex].top && minValuesCount[colIndex] == 1 && !table[rowIndex][colIndex].used) {

table[rowIndex][colIndex].used = true;

if (b\_cols[colIndex] != 0) {

if (a\_rows[rowIndex] < b\_cols[colIndex]) {

table[rowIndex][colIndex].bottom = (a\_rows[rowIndex]) < 0 ? 0 : a\_rows[rowIndex];

b\_cols[colIndex] -= a\_rows[rowIndex];

a\_rows[rowIndex] -= a\_rows[rowIndex];

}

else {

table[rowIndex][colIndex].bottom = b\_cols[colIndex];

a\_rows[rowIndex] -= b\_cols[colIndex];

b\_cols[colIndex] -= b\_cols[colIndex];

}

}

else {

table[rowIndex][colIndex].bottom = 0;

}

}

}

}

for (int rowIndex = 0; rowIndex < table.size(); rowIndex++) {

for (int colIndex = 0; colIndex < table[0].size(); colIndex++) {

if (minValues[colIndex] == table[rowIndex][colIndex].top) {

minValuesRowCount[rowIndex] += 1;

}

}

}

for (int colIndex = 0; colIndex < table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < table.size(); rowIndex++) {

if (minValues[colIndex] == table[rowIndex][colIndex].top && minValuesRowCount[rowIndex] == 1 && !table[rowIndex][colIndex].used) {

table[rowIndex][colIndex].used = true;

if (b\_cols[colIndex] != 0) {

if (a\_rows[rowIndex] < b\_cols[colIndex]) {

table[rowIndex][colIndex].bottom = (a\_rows[rowIndex]) < 0 ? 0 : a\_rows[rowIndex];

b\_cols[colIndex] -= a\_rows[rowIndex];

a\_rows[rowIndex] -= a\_rows[rowIndex];

}

else {

table[rowIndex][colIndex].bottom = b\_cols[colIndex];

a\_rows[rowIndex] -= b\_cols[colIndex];

b\_cols[colIndex] -= b\_cols[colIndex];

}

}

else {

table[rowIndex][colIndex].bottom = 0;

}

}

}

}

for (int colIndex = 0; colIndex < table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < table.size(); rowIndex++) {

if (minValues[colIndex] == table[rowIndex][colIndex].top && !table[rowIndex][colIndex].used) {

table[rowIndex][colIndex].used = true;

if (b\_cols[colIndex] != 0) {

if (a\_rows[rowIndex] < b\_cols[colIndex]) {

table[rowIndex][colIndex].bottom = (a\_rows[rowIndex]) < 0 ? 0 : a\_rows[rowIndex];

b\_cols[colIndex] -= a\_rows[rowIndex];

a\_rows[rowIndex] -= a\_rows[rowIndex];

}

else if (a\_rows[rowIndex] == b\_cols[colIndex]) {

table[rowIndex][colIndex].bottom = b\_cols[colIndex];

a\_rows[rowIndex] -= b\_cols[colIndex];

b\_cols[colIndex] -= b\_cols[colIndex];;

}

else {

table[rowIndex][colIndex].bottom = b\_cols[colIndex];

a\_rows[rowIndex] -= b\_cols[colIndex];

b\_cols[colIndex] -= b\_cols[colIndex];

}

}

else {

table[rowIndex][colIndex].bottom = 0;

}

}

}

}

double sum = 0.f;

for (int colIndex = 0; colIndex < table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < table.size(); rowIndex++) {

if (table[rowIndex][colIndex].used) {

sum += table[rowIndex][colIndex].bottom;

}

}

}

return sum;

}

void TransportProblem::CalculateDiference() {

auto a\_rows = m\_a\_rows;

auto b\_cols = m\_b\_cols;

m\_dif\_cols.clear();

m\_dif\_rows.clear();

m\_dif\_cols.resize(m\_b\_cols.size(), 1000000);

m\_dif\_rows.resize(m\_a\_rows.size(), { -1, false });

std::vector<double> minValues(m\_table[0].size(), 1000000);

std::vector<int> minValuesCount(m\_table[0].size(), 0);

std::vector<int> minValuesRowCount(m\_table.size(), 0);

std::vector<int> usedRows;

std::vector<int> usedColumns;

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

m\_table[rowIndex][colIndex].bottom = 0;

m\_table[rowIndex][colIndex].used = false;

if (minValues[colIndex] > m\_table[rowIndex][colIndex].top) {

minValues[colIndex] = m\_table[rowIndex][colIndex].top;

minValuesCount[colIndex] = 0;

}

if (minValues[colIndex] == m\_table[rowIndex][colIndex].top) {

minValuesCount[colIndex] += 1;

}

}

}

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

if (minValues[colIndex] == m\_table[rowIndex][colIndex].top && minValuesCount[colIndex] == 1 && !m\_table[rowIndex][colIndex].used) {

m\_table[rowIndex][colIndex].used = true;

if (m\_b\_cols[colIndex] != 0) {

if (m\_a\_rows[rowIndex] < m\_b\_cols[colIndex]) {

m\_table[rowIndex][colIndex].bottom = (m\_a\_rows[rowIndex]) < 0 ? 0 : m\_a\_rows[rowIndex];

m\_b\_cols[colIndex] -= m\_a\_rows[rowIndex];

m\_a\_rows[rowIndex] -= m\_a\_rows[rowIndex];

}

else {

m\_table[rowIndex][colIndex].bottom = m\_b\_cols[colIndex];

m\_a\_rows[rowIndex] -= m\_b\_cols[colIndex];

m\_b\_cols[colIndex] -= m\_b\_cols[colIndex];

}

}

else {

m\_table[rowIndex][colIndex].bottom = 0;

}

usedColumns.push\_back(colIndex);

}

}

}

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

if (std::find(usedColumns.begin(), usedColumns.end(), colIndex) != usedColumns.end()) {

continue;

}

if (minValues[colIndex] == m\_table[rowIndex][colIndex].top) {

minValuesRowCount[rowIndex] += 1;

}

}

}

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

if (minValues[colIndex] == m\_table[rowIndex][colIndex].top && minValuesRowCount[rowIndex] == 1 && !m\_table[rowIndex][colIndex].used) {

m\_table[rowIndex][colIndex].used = true;

if (m\_b\_cols[colIndex] != 0) {

if (m\_a\_rows[rowIndex] < m\_b\_cols[colIndex]) {

m\_table[rowIndex][colIndex].bottom = (m\_a\_rows[rowIndex]) < 0 ? 0 : m\_a\_rows[rowIndex];

m\_b\_cols[colIndex] -= m\_a\_rows[rowIndex];

m\_a\_rows[rowIndex] -= m\_a\_rows[rowIndex];

}

else {

m\_table[rowIndex][colIndex].bottom = m\_b\_cols[colIndex];

m\_a\_rows[rowIndex] -= m\_b\_cols[colIndex];

m\_b\_cols[colIndex] -= m\_b\_cols[colIndex];

}

}

else {

m\_table[rowIndex][colIndex].bottom = 0;

}

}

}

}

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

if (minValues[colIndex] == m\_table[rowIndex][colIndex].top && !m\_table[rowIndex][colIndex].used) {

m\_table[rowIndex][colIndex].used = true;

if (m\_b\_cols[colIndex] != 0) {

if (m\_a\_rows[rowIndex] < m\_b\_cols[colIndex]) {

m\_table[rowIndex][colIndex].bottom = (m\_a\_rows[rowIndex]) < 0 ? 0 : m\_a\_rows[rowIndex];

m\_b\_cols[colIndex] -= m\_a\_rows[rowIndex];

m\_a\_rows[rowIndex] -= m\_a\_rows[rowIndex];

}

else if (m\_a\_rows[rowIndex] == m\_b\_cols[colIndex]) {

m\_table[rowIndex][colIndex].bottom = m\_b\_cols[colIndex];

m\_a\_rows[rowIndex] -= m\_b\_cols[colIndex];

m\_b\_cols[colIndex] -= m\_b\_cols[colIndex];;

}

else {

m\_table[rowIndex][colIndex].bottom = m\_b\_cols[colIndex];

m\_a\_rows[rowIndex] -= m\_b\_cols[colIndex];

m\_b\_cols[colIndex] -= m\_b\_cols[colIndex];

}

}

else {

m\_table[rowIndex][colIndex].bottom = 0;

}

}

}

}

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

if (m\_b\_cols[colIndex] == 0) {

continue;

}

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

if (m\_table[rowIndex][colIndex].used) {

m\_a\_rows[rowIndex] -= m\_b\_cols[colIndex];

m\_b\_cols[colIndex] -= m\_b\_cols[colIndex];

break;

}

}

}

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

m\_dif\_rows[rowIndex] = { m\_a\_rows[rowIndex], (m\_a\_rows[rowIndex] > 0)};

}

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

bool findPositive = false;

if (m\_dif\_rows[rowIndex].first == 0) {

auto table = m\_table;

auto a\_rows\_copy = a\_rows;

auto b\_cols\_copy = b\_cols;

a\_rows\_copy[rowIndex] += 1;

auto checkSum = CalculateGived(a\_rows\_copy, b\_cols\_copy, table);

double sum = 0.f;

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

if (m\_table[rowIndex][colIndex].used) {

sum += m\_table[rowIndex][colIndex].bottom;

}

}

}

if (sum == checkSum) {

m\_dif\_rows[rowIndex] = { 0, true };

}

}

}

m\_a\_rows = a\_rows;

m\_b\_cols = b\_cols;

for (int colIndex = 0; colIndex < m\_table[0].size(); colIndex++) {

bool positive = false;

for (int rowIndex = 0; rowIndex < m\_table.size(); rowIndex++) {

if (m\_table[rowIndex][colIndex].used && m\_dif\_rows[rowIndex].second) {

continue;

}

else if (m\_table[rowIndex][colIndex].used && !m\_dif\_rows[rowIndex].second) {

bool positive = false;

for (int rowIndex2 = 0; rowIndex2 < m\_table.size(); rowIndex2++) {

if (m\_table[rowIndex2][colIndex].used && m\_dif\_rows[rowIndex2].second) {

positive = true;

break;

}

}

if (positive) {

break;

}

for (int rowIndex2 = 0; rowIndex2 < m\_table.size(); rowIndex2++) {

if (rowIndex2 == rowIndex || !m\_dif\_rows[rowIndex2].second) {

continue;

}

if (m\_dif\_cols[colIndex] > std::abs(m\_table[rowIndex][colIndex].top - m\_table[rowIndex2][colIndex].top)) {

m\_dif\_cols[colIndex] = std::abs(m\_table[rowIndex][colIndex].top - m\_table[rowIndex2][colIndex].top);

}

}

}

}

if (positive) {

break;

}

}

}

bool TransportProblem::HasNegative() {

bool allZero = true;

for (const auto& [value, positive] : m\_dif\_rows) {

if (value != 0) {

allZero = false;

break;

}

}

return !allZero;

}

void TransportProblem::GoToNextRentTable() {

auto minel = 1000000;

for (int i = 0; i < m\_dif\_cols.size(); ++i) {

if (minel > m\_dif\_cols[i]) {

minel = m\_dif\_cols[i];

}

}

for (int rowIndex = 0; rowIndex < m\_table.size(); ++rowIndex) {

if (m\_dif\_rows[rowIndex].second) {

continue;

}

for (int colIndex = 0; colIndex < m\_table[0].size(); ++colIndex) {

m\_table[rowIndex][colIndex].top += minel;

}

}

}

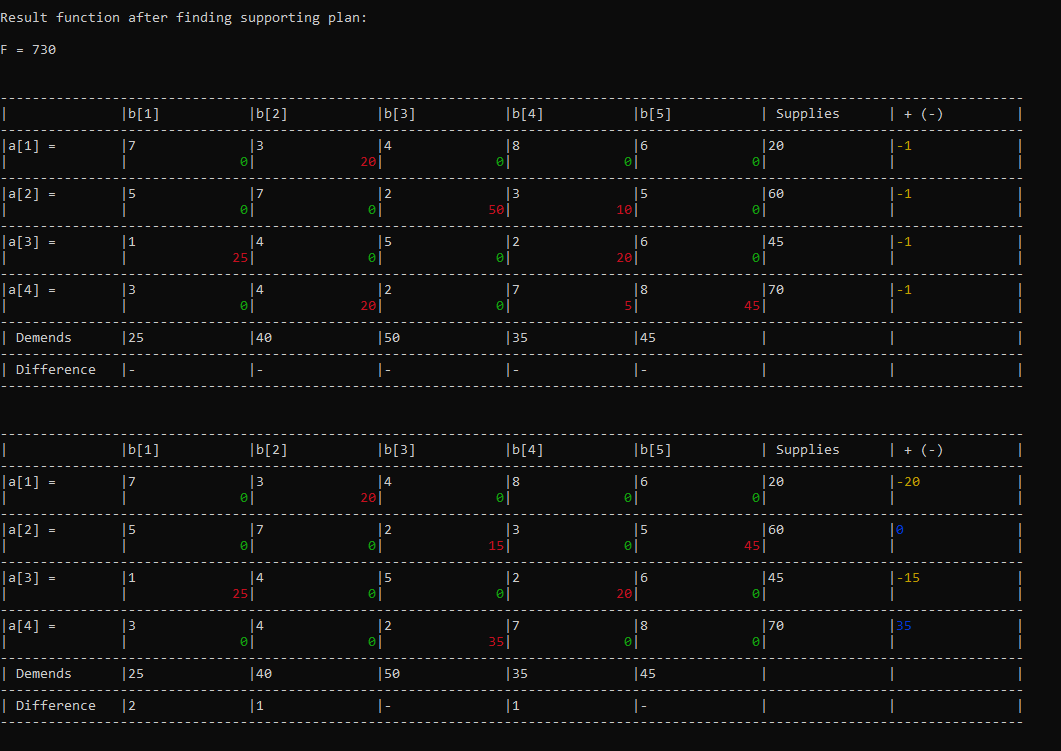


Рис. 2. Результати програми

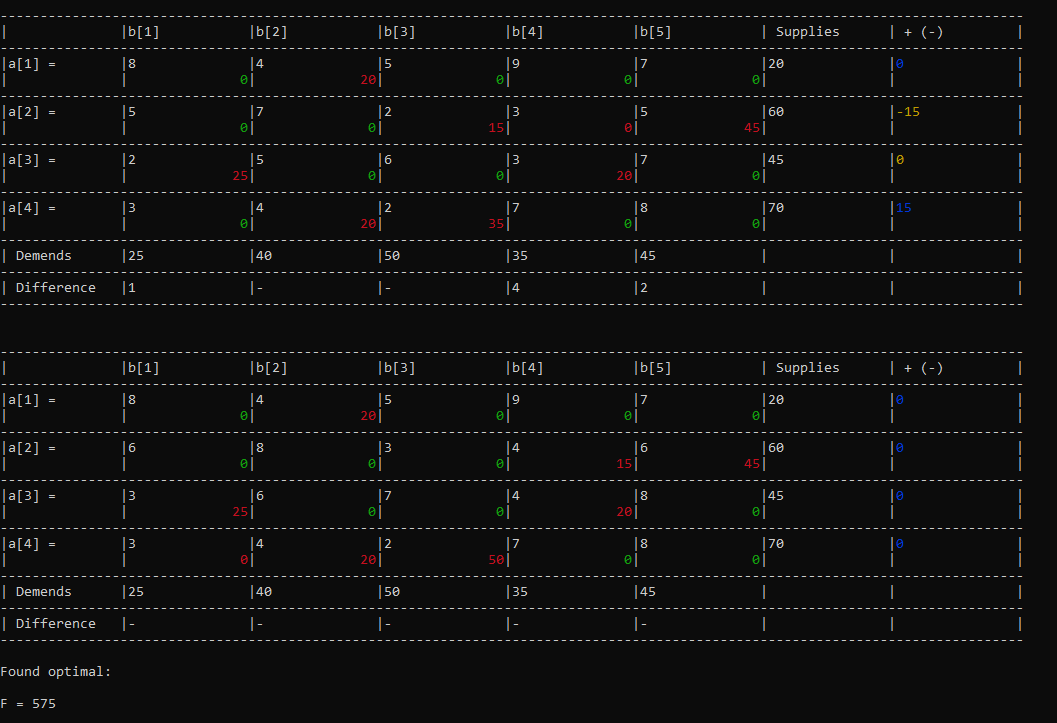


Рис. 3. Результати програми

**ВИСНОВКИ**

На цій лабораторній роботі я ознайомився на практиці із розв’язуванням транспортних задач за допомогою побудови умовно оптимального розподілу та покращування його за методом диференціальних рент.