**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 4

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

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

**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 потенціалів

**Мета роботи:** Ознайомитись на практиці із основними поняттями транспортних задач, навчитись знаходити початкові опорні плани (за методами північно-західного кута, мінімального елемента та евристичним методом Фойгеля) та оптимальні плани задач за допомогою методу потенціалів.

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

3. Чи можна застосувати для розв’язування транспортних задач симплекс-метод? Відповідь обґрунтуйте.

Так, його можна проінтерпретувати як СМ наступним чином.

1) Заповнені клітинки транспортної таблиці (де є перевезення) відповідають

базовим змінним, незаповнені – небазовим.

2) Знаходження клітинки, що буде заповнюватись, відповідає пошуку змінної, що

вводитиметься до бази. Знаходження клітинки в циклі, відміченої знаком “–“ з

найменшим значенням перевезення, відповідає пошуку змінної, що буде

виводитися з бази.

3) Переміщення перевезення в межах циклу відповідає переходу до нової СТ в

СМ.

4) Незаповнена клітинка з найбільшим додатним значенням потенціалу dij ,

враховуючи, що ТЗ – це задача на знаходження мінімуму, є аналогом

найбільшого за абсолютною величиною від’ємного значення dj.

5) Значення потенціалів незаповнених клітинок (небазових змінних) dij

відповідають коефіцієнтам dj останнього рядка СТ (за умови відповідної

переіндексації змінних).

6) Потенціали рядків та потенціали стовпців відповідають значенням змінних

двоїстої задачі. Знаючи значення двоїстих змінних, на кожній ітерації

визначається dij як різниця між правою та лівою частиною відповідного

обмеження двоїстої задачі.

7) Теорема про потенціали є видозміною теореми про доповнювальну

нежорсткість (другої теореми двоїстості).

6. Що таке відкрита модель транспортної задачі і як вона розв’язується? Транспортна задача є відкритою, якщо загальні запаси і потреби є незбалансовані. Тому спочатку її приводять до задачі закритого типу і тоді розв’язують.

9. У чому суть методу мінімального елемента?

Знаходження клітинки з мінімальною вартістю перевезення і здійснити максимально можливе переміщення вантажу в дану клітинку.

**ЗАВДАННЯ**

1. Отримати індивідуальний варіант завдання.

2. Написати програму розв’язування транспортної задачі за методом

потенціалів (для пошуку опорного початкового плану реалізувати

один з методів: північно-західного кута, метод мінімального елемента, евристичний метод Фойгеля - такий, який не дає зразу оптимального розв'язку) згідно з 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.

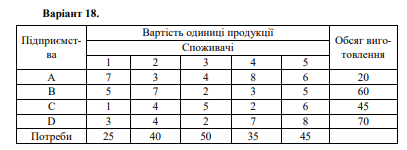


Рис. 1. Індивідуальне завдання

1. Код програми:

#pragma once

#include <string>

#include <vector>

#include "Cell.h"

#include "LoopCellInfo.h"

#include "Potential.h"

class TransportProblem

{

public:

void Start();

private:

bool IsDegenerate();

void CreateProblem(const bool fromConsole = false, std::string fileName = "main.json");

void ShowTable(const std::string& message = "");

void FindInitialFucntion();

void CalculatePotensials();

void CalculateFunction(std::string mes = "");

bool FindLoopAndRecalculate();

std::pair<bool, std::vector<LoopCellInfo>> FindNextElementInLoop(const std::vector<LoopCellInfo>& path, bool horizontal);

std::vector<LoopCellInfo> FindNeigbors(LoopCellInfo loopCellInfo, bool horizontal);

std::vector<Potential> m\_v\_rows;

std::vector<Potential> m\_u\_cols;

std::vector<double> m\_a\_rows;

std::vector<double> m\_b\_cols;

std::vector<std::vector<Cell>> m\_table;

double m\_result;

};

#include "TransportProblem.h"

#include <json.h>

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <fstream>

#include <sstream>

#include "color.hpp"

void TransportProblem::Start() {

CreateProblem();

if (!IsDegenerate()) {

std::cout << "Proble is not degenerate!";

return;

}

FindInitialFucntion();

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

ShowTable();

CalculatePotensials();

ShowTable();

while (FindLoopAndRecalculate()) {

CalculatePotensials();

ShowTable();

}

CalculateFunction("Found optimal:");

}

void TransportProblem::CreateProblem(const bool fromConsole, std::string fileName) {

if (fromConsole) {

std::cout << "Enter number of supply locations:";

int suplyNumber;

std::cin >> suplyNumber;

m\_a\_rows.resize(suplyNumber);

std::cout << "Enter supplies:";

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

std::cout << "A[" << i+1 << "] = ";

std::cin >> m\_a\_rows[i];

}

std::cout << "Enter number of demand locations:";

int demandNumber;

std::cin >> demandNumber;

m\_b\_cols.resize(demandNumber);

std::cout << "Enter demands:";

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

std::cout << "B[" << i +1 << "] = ";

std::cin >> m\_b\_cols[i];

}

m\_table.resize(suplyNumber, std::vector<Cell>(demandNumber, Cell{ 0,0,false }));

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

for (int j = 0; j < demandNumber; ++j) {

std::cout << "a[" << i + 1 << "][" << j + 1 << "] = ";

std::cin >> m\_table[i][j].top;

}

}

}

else {

std::ifstream jsonFile(fileName);

if (jsonFile.fail())

{

throw std::runtime\_error("File was not found.\n");

}

nlohmann::json parsedJson = nlohmann::json::parse(jsonFile);

const auto& parsedJsonSupply = parsedJson.at("Supply");

m\_a\_rows.resize(parsedJsonSupply.at("Count").get<int>());

for (int i = 0; i < m\_a\_rows.size(); ++i) {

m\_a\_rows[i] = parsedJsonSupply.at("values")[i].get<double>();

}

const auto& parsedJsonDemand = parsedJson.at("Demand");

m\_b\_cols.resize(parsedJsonDemand.at("Count").get<int>());

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

m\_b\_cols[i] = parsedJsonDemand.at("values")[i].get<double>();

}

m\_table.resize(m\_a\_rows.size(), std::vector<Cell>(m\_b\_cols.size(), Cell{ 0,0,false }));

const auto& parsedJsontableValues = parsedJson.at("TableValues");

for (int i = 0; i < m\_a\_rows.size(); ++i) {

for (int j = 0; j < m\_b\_cols.size(); ++j) {

m\_table[i][j].top = parsedJsontableValues[i][j].get<double>();

}

}

}

m\_v\_rows.resize(m\_a\_rows.size(), Potential{ 0, false });

m\_u\_cols.resize(m\_b\_cols.size(), Potential{ 0, false });

}

bool TransportProblem::IsDegenerate() {

double sum1 = 0;

double sum2 = 0;

for (auto a : m\_a\_rows) {

sum1 += a;

}

for (auto b : m\_b\_cols) {

sum2 += b;

}

return sum1 == sum2;

}

void TransportProblem::ShowTable(const std::string& message) {

std::cout << "\n" + message;

std::cout << "\n";

std::cout << std::setw(15 + 16\*m\_b\_cols.size()) << std::setfill('-') << "";

//other rows

std::cout << "\n";

std::cout << std::setfill(' ');

//1 row

std::cout << std::setw(15) << std::left << "|";

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

std::cout << "|b[" << std::to\_string(i + 1) << "] = " << std::setw(8) << std::setprecision(3) << std::left << m\_u\_cols[i].Value;

}

std::cout << "|";

std::cout << "\n";

std::cout << std::setw(15 + 16 \* m\_b\_cols.size()) << std::setfill('-') << "";

//other rows

std::cout << "\n";

std::cout << std::setfill(' ');

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

std::cout << "|a[" + std::to\_string(rowIndex + 1) + "] = " << std::setw(7) << std::setprecision(3) << std::left << m\_v\_rows[rowIndex].Value;

for (int colIndex = 0; colIndex < m\_u\_cols.size(); ++colIndex) {

std::cout << "|" << std::setw(15) << std::setprecision(3) << std::left << m\_table[rowIndex][colIndex].top;

}

std::cout << "|";

std::cout << "\n";

std::cout << std::setw(15) << std::left << "|";

for (int colIndex = 0; colIndex < m\_u\_cols.size(); ++colIndex) {

std::ostringstream ss;

ss << std::setw(15) << std::setprecision(3) << std::right << m\_table[rowIndex][colIndex].bottom;

auto value = m\_table[rowIndex][colIndex].used ? dye::red(ss.str()) : dye::green(ss.str());

std::cout << "|" << value;

}

std::cout << "|";

std::cout << "\n";

std::cout << std::setw(15 + 16 \* m\_b\_cols.size()) << std::setfill('-') << "";

std::cout << "\n";

std::cout << std::setfill(' ');

}

std::cout << "\n";

}

void TransportProblem::CalculateFunction(std::string mes) {

std::cout << "\n"+mes+"\n";

m\_result = 0;

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

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

if (m\_table[i][j].used) {

m\_result += m\_table[i][j].top \* m\_table[i][j].bottom;

}

}

}

std::cout << "\nF = " << m\_result << "\n";

}

bool TransportProblem::FindLoopAndRecalculate() {

double max = -1;

int maxRowIndex = -1;

int maxColIndex = -1;

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

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

if (m\_table[i][j].used) {

continue;

}

if (m\_table[i][j].bottom >= 0 && max < m\_table[i][j].bottom) {

max = m\_table[i][j].bottom;

maxRowIndex = i;

maxColIndex = j;

}

}

}

if (max == -1) {

return false;

}

std::vector<LoopCellInfo> path;

path.push\_back(LoopCellInfo{ m\_table[maxRowIndex][maxColIndex], true, maxRowIndex, maxColIndex });

m\_table[maxRowIndex][maxColIndex].used = true;

auto resultLoop = FindNextElementInLoop(path, true);

if (resultLoop.first) {

path = resultLoop.second;

}

else {

return false;

}

std::cout << "\nPath:\n";

for (auto& cell : path)

{

std::cout << "a[" << cell.RowIndex + 1 << "][" << cell.ColIndex + 1 << "] =>";

}

std::cout << "\n";

double min = 10000;

double minRowIndex = -1;

double minColIndex = -1;

for (const auto& cell : path) {

if (!cell.Plus) {

if (min > cell.Cell.bottom) {

min = cell.Cell.bottom;

minRowIndex = cell.RowIndex;

minColIndex = cell.ColIndex;

}

}

}

for (const auto& cell : path) {

m\_table[minRowIndex][minColIndex].used = false;

cell.Cell.bottom = cell.Plus ? cell.Cell.bottom + min : cell.Cell.bottom - min;

}

m\_table[maxRowIndex][maxColIndex].bottom = min;

return true;

}

std::pair<bool, std::vector<LoopCellInfo>> TransportProblem::FindNextElementInLoop(const std::vector<LoopCellInfo>& path, bool horizontal) {

std::vector<LoopCellInfo> neighbors;

neighbors = FindNeigbors(path.back(), horizontal);

if (path.size() == 1 && neighbors.empty()) {

horizontal = !horizontal;

neighbors = FindNeigbors(path.back(), horizontal);

}

else if (neighbors.empty()) {

return std::pair<bool, std::vector<LoopCellInfo>>(false, path);

}

horizontal = !horizontal;

const auto& start = path.front();

for (auto a : neighbors) {

if ((a.RowIndex == start.RowIndex) && (a.ColIndex == start.ColIndex) && (a.Cell.top == start.Cell.top)

&& (a.Cell.bottom == start.Cell.bottom) && path.size() > 2) {

return std::pair<bool, std::vector<LoopCellInfo>>(true, path);

}

}

for (auto& loopCellInfo : neighbors) {

auto newPath = path;

newPath.push\_back(loopCellInfo);

auto result = FindNextElementInLoop(newPath, horizontal);

if (result.first) {

return result;

}

}

return std::pair<bool, std::vector<LoopCellInfo>>(false, path);

}

std::vector<LoopCellInfo> TransportProblem::FindNeigbors(LoopCellInfo loopCellInfo, bool horizontal) {

std::vector<LoopCellInfo> neighbors;

if (horizontal) {

for (int colIndex = 0; colIndex < m\_u\_cols.size(); ++colIndex) {

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

if (loopCellInfo.ColIndex == colIndex) {

continue;

}

neighbors.push\_back(LoopCellInfo{ m\_table[loopCellInfo.RowIndex][colIndex], !loopCellInfo.Plus, loopCellInfo.RowIndex,colIndex });

}

}

}

else {

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

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

if (loopCellInfo.RowIndex == rowIndex) {

continue;

}

neighbors.push\_back(LoopCellInfo{ m\_table[rowIndex][loopCellInfo.ColIndex], !loopCellInfo.Plus, rowIndex,loopCellInfo.ColIndex });

}

}

}

return neighbors;

}

void TransportProblem::FindInitialFucntion() {

std::vector<int> usedRows;

std::vector<int> usedColumns;

int count = 0;

bool foundInitial = false;

while (!foundInitial) {

double min = 10000000000.f;

int min\_i = -1;

int min\_j = -1;

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

if (std::find(usedRows.begin(), usedRows.end(), i) != usedRows.end()) {

continue;

}

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

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

continue;

}

if (min > m\_table[i][j].top) {

min = m\_table[i][j].top;

min\_i = i;

min\_j = j;

}

}

}

if (m\_a\_rows[min\_i] < m\_b\_cols[min\_j]) {

m\_table[min\_i][min\_j].bottom = m\_a\_rows[min\_i];

m\_table[min\_i][min\_j].used = true;

usedRows.push\_back(min\_i);

m\_b\_cols[min\_j] -= m\_a\_rows[min\_i];

m\_a\_rows[min\_i] -= m\_a\_rows[min\_i];

count++;

}

else if (m\_a\_rows[min\_i] == m\_b\_cols[min\_j]) {

m\_table[min\_i][min\_j].bottom = m\_b\_cols[min\_j];

m\_table[min\_i][min\_j].used = true;

usedColumns.push\_back(min\_j);

usedRows.push\_back(min\_i);

m\_a\_rows[min\_i] -= m\_b\_cols[min\_j];

m\_b\_cols[min\_j] -= m\_b\_cols[min\_j];

count++;

}

else {

m\_table[min\_i][min\_j].bottom = m\_b\_cols[min\_j];

m\_table[min\_i][min\_j].used = true;

usedColumns.push\_back(min\_j);

m\_a\_rows[min\_i] -= m\_b\_cols[min\_j];

m\_b\_cols[min\_j] -= m\_b\_cols[min\_j];

count++;

}

if (usedRows.size() == m\_table.size() || usedColumns.size() == m\_table[0].size()) {

foundInitial = true;

}

}

if (count < m\_a\_rows.size() + m\_b\_cols.size() - 1) {

double min = 100000.f;

int min\_i = -1;

int min\_j = -1;

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

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

if (m\_table[i][j].used) {

continue;

}

if (min > m\_table[i][j].top) {

min = m\_table[i][j].top;

min\_i = i;

min\_j = j;

}

}

}

m\_table[min\_i][min\_j].bottom = 0;

m\_table[min\_i][min\_j].used = true;

}

}

void TransportProblem::CalculatePotensials() {

auto size\_v = m\_v\_rows.size();

m\_v\_rows.clear();

m\_v\_rows.resize(size\_v, Potential{ 0, false });

auto size\_u = m\_u\_cols.size();

m\_u\_cols.clear();

m\_u\_cols.resize(size\_u, Potential{ 0, false });

m\_v\_rows.back().Value = 0;

m\_v\_rows.back().Found = true;

std::vector<int> usedRows;

for (int colIndex = 0; colIndex < m\_u\_cols.size(); ++colIndex) {

if (m\_table[m\_v\_rows.size() - 1][colIndex].used) {

m\_u\_cols[colIndex].Value = m\_table[m\_v\_rows.size() - 1][colIndex].top - m\_v\_rows.back().Value;

m\_u\_cols[colIndex].Found = true;

}

}

usedRows.push\_back(m\_v\_rows.size() - 1);

bool AllFound = false;

while (usedRows.size() != size\_v && !AllFound) {

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

if (std::find(usedRows.begin(), usedRows.end(), i) != usedRows.end()) {

continue;

}

int usedCells = 0;

int usedForPotentials = 0;

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

if ((m\_v\_rows[i].Found && m\_u\_cols[j].Found) || !m\_table[i][j].used) {

continue;

}

usedCells++;

if (m\_v\_rows[i].Found && m\_u\_cols[j].Found) {

usedForPotentials++;

}

if (m\_v\_rows[i].Found && !m\_u\_cols[j].Found) {

m\_u\_cols[j].Value = m\_table[i][j].top - m\_v\_rows[i].Value;

m\_u\_cols[j].Found = true;

usedForPotentials++;

}

if (!m\_v\_rows[i].Found && m\_u\_cols[j].Found) {

m\_v\_rows[i].Value = m\_table[i][j].top - m\_u\_cols[j].Value;

m\_v\_rows[i].Found = true;

usedForPotentials++;

}

}

if (usedCells == usedForPotentials) {

usedRows.push\_back(i);

}

}

int AllFoundCount = 0;

for (auto v : m\_v\_rows) {

if (v.Found) {

AllFoundCount++;

}

}

for (auto u : m\_u\_cols) {

if (u.Found) {

AllFoundCount++;

}

}

if (AllFoundCount == m\_v\_rows.size() + m\_u\_cols.size()) {

AllFound = true;

}

}

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

for (int colIndex = 0; colIndex < m\_u\_cols.size(); ++colIndex) {

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

continue;

}

m\_table[rowIndex][colIndex].bottom = m\_v\_rows[rowIndex].Value + m\_u\_cols[colIndex].Value - m\_table[rowIndex][colIndex].top;

}

}

}

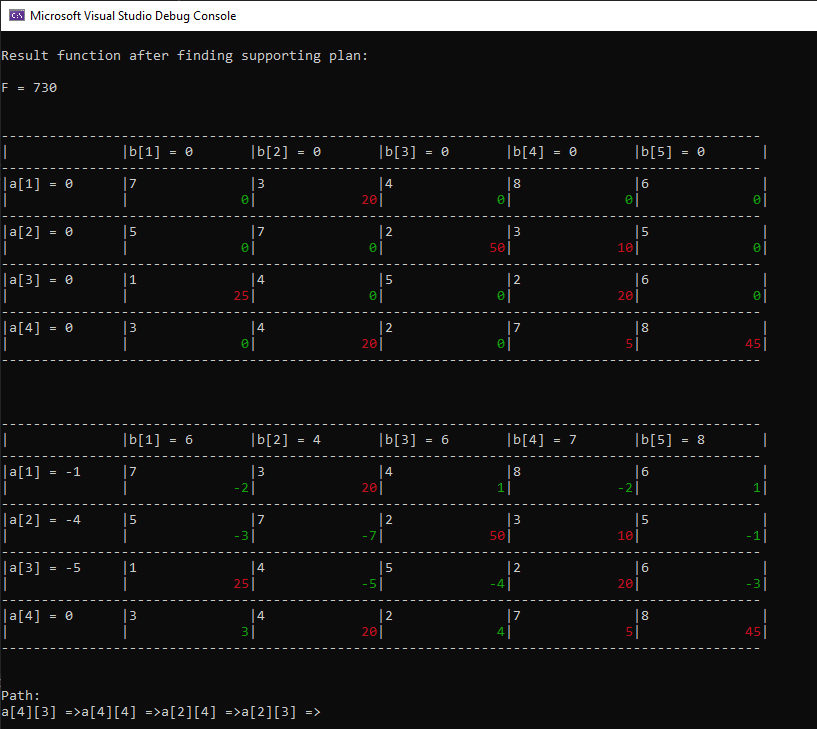


Рис. 2. Результати програми

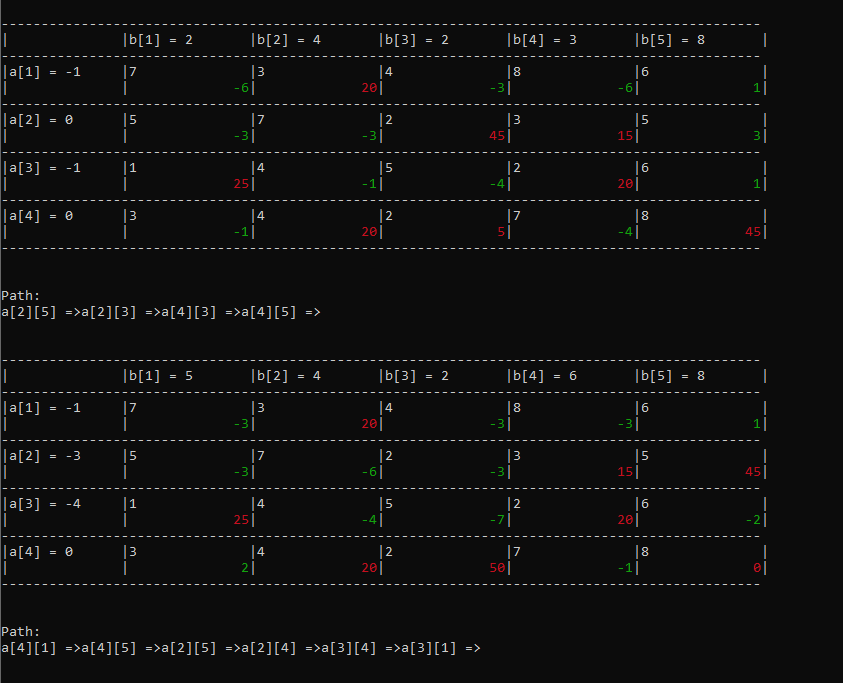


Рис. 3. Результати програми

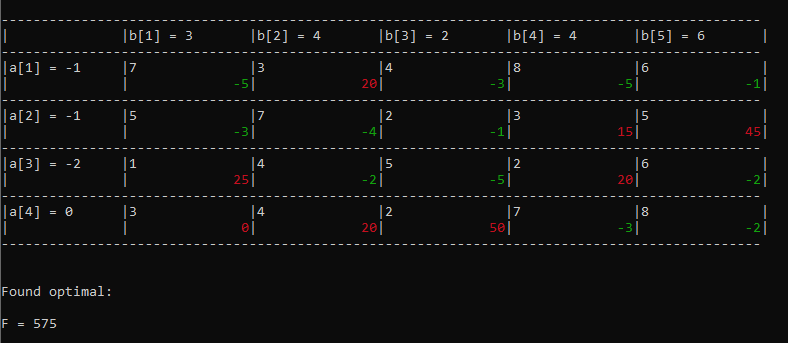


Рис. 4. Результати програми

**ВИСНОВКИ**

На цій лабораторній роботі я ознайомився на практиці із основними поняттями транспортних задач, навчився знаходити початкові опорні плани (за методами північно-західного кута, мінімального елемента та евристичним методом Фойгеля) та оптимальні плани задач за допомогою методу потенціалів.