**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №4

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

**з дисципліни** *“Дослідження операцій”*

**Лектор:**

проф. каф. ПЗ

Журавчак, Л. М.

**Виконав:**

ст. гр. ПЗ-33

Чернявський М.Ю.

**Прийняв:**

ст. викл. каф. ПЗ

Івасько Н. М.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

**Тема:** Розв’язування транспортної задачі ЛП методом потенціалів.

**Мета:** 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.

**Порядок виконання лабораторної роботи № 4:**

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.

1. Оформити звіт про виконану роботу.
2. Продемонструвати викладачеві результати, відповісти на запитання

стосовно виконання роботи.

# Хід Роботи

Варіант №23

1. Написати програму розв’язування транспортної задачі за методом потенціалів.

Зображення, що містить текст, число, знімок екрана, Шрифт

Автоматично згенерований опис

# Результат виконання програми

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

*Рис. 1. Оптимальна ціна*

*Зображення, що містить знімок екрана, текст, Прямокутник, квадрат

Автоматично згенерований опис*

*Рис. 2. Перша матриця (результат методу мінімального елемента)*

*Зображення, що містить знімок екрана, текст, Прямокутник, квадрат

Автоматично згенерований опис*

*Рис. 3. Друга матриця методу потенціалів*

*Зображення, що містить знімок екрана, Прямокутник, квадрат, Паралель

Автоматично згенерований опис*

*Рис. 4. Третя матриця методу потенціалів*

*Зображення, що містить знімок екрана, текст, Прямокутник, квадрат

Автоматично згенерований опис*

*Рис. 5. Четверта матриця методу потенціалів*

# Код програми

MainWindow.xaml.cs:

using System.Globalization;  
using System.Text.RegularExpressions;  
using System.Windows;  
using System.Windows.Controls;  
using Lab04\_OR.Methods;  
using System.Linq;  
using System.Windows.Media;  
  
namespace Lab04\_OR;  
  
*/// <summary>  
/// Interaction logic for MainWindow.xaml  
/// </summary>*public partial class MainWindow : Window  
{  
 private int \_numSuppliers;  
 private int \_numDemands;  
  
 public MainWindow()  
 {  
 \_numSuppliers = 3;  
 \_numDemands = 3;  
  
 InitializeComponent();  
 GenerateTransportGrid();  
 PreDefine();  
 }  
  
 private void SupplierComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)  
 {  
 \_numSuppliers = int.Parse((SupplierComboBox.SelectedItem as ComboBoxItem)!.Content.ToString()!);  
 GenerateTransportGrid();  
 }  
  
 private void ConsumerComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)  
 {  
 \_numDemands = int.Parse((ConsumerComboBox.SelectedItem as ComboBoxItem)!.Content!.ToString()!);  
 GenerateTransportGrid();  
 }  
  
 private void GenerateTransportGrid()  
 {  
 TransportGrid ??= new Grid();  
 TransportGrid.RowDefinitions.Clear();  
 TransportGrid.ColumnDefinitions.Clear();  
 TransportGrid.Children.Clear();  
  
 *// Create Row Definitions (numSuppliers + 1 for header, +1 for supplies input)* for (int i = 0; i <= \_numSuppliers + 1; i++)  
 {  
 TransportGrid.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1, GridUnitType.Star) });  
 }  
  
 *// Create Column Definitions (numDemands + 1 for header, +1 for demands input)* for (int j = 0; j <= \_numDemands + 1; j++)  
 {  
 TransportGrid.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(1, GridUnitType.Star) });  
 }  
  
 *// Create headers (top row and left column)* for (int i = 1; i <= \_numSuppliers; i++)  
 {  
 TextBlock supplierHeader = new TextBlock  
 {  
 Text = "A" + i,  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center  
 };  
 Grid.SetRow(supplierHeader, i);  
 Grid.SetColumn(supplierHeader, 0);  
 TransportGrid.Children.Add(supplierHeader);  
 }  
 TextBlock demand = new TextBlock  
 {  
 Text = "D",  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center  
 };  
 Grid.SetRow(demand, \_numSuppliers + 1);  
 Grid.SetColumn(demand, 0);  
 TransportGrid.Children.Add(demand);  
  
 for (int j = 1; j <= \_numDemands; j++)  
 {  
 TextBlock consumerHeader = new TextBlock  
 {  
 Text = "B" + j,  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center  
 };  
 Grid.SetRow(consumerHeader, 0);  
 Grid.SetColumn(consumerHeader, j);  
 TransportGrid.Children.Add(consumerHeader);  
 }  
 TextBlock supplies = new TextBlock  
 {  
 Text = "S",  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center  
 };  
 Grid.SetRow(supplies, 0);  
 Grid.SetColumn(supplies, \_numDemands + 1);  
 TransportGrid.Children.Add(supplies);  
  
 *// Fill grid cells with TextBoxes for input (except the first row/column for headers)* for (int i = 1; i <= \_numSuppliers; i++)  
 {  
 for (int j = 1; j <= \_numDemands; j++)  
 {  
 TextBox cell = new TextBox  
 {  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center,  
 MinWidth = 30,  
 MinHeight = 30,  
 };  
  
 *// Attach the validation event handler to each TextBox* cell.PreviewTextInput += Cell\_PreviewTextInput;  
  
 Grid.SetRow(cell, i);  
 Grid.SetColumn(cell, j);  
 TransportGrid.Children.Add(cell);  
 }  
 }  
  
 *// Add TextBoxes for supplies (last column)* for (int i = 1; i <= \_numSuppliers; i++)  
 {  
 TextBox supplyBox = new TextBox  
 {  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center,  
 MinWidth = 30,  
 MinHeight = 30,  
 };  
  
 supplyBox.PreviewTextInput += Cell\_PreviewTextInput;  
  
 Grid.SetRow(supplyBox, i);  
 Grid.SetColumn(supplyBox, \_numDemands + 1);  
 TransportGrid.Children.Add(supplyBox);  
 }  
  
 *// Add TextBoxes for demands (last row)* for (int j = 1; j <= \_numDemands; j++)  
 {  
 TextBox demandBox = new TextBox  
 {  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center,  
 MinWidth = 30,  
 MinHeight = 30,  
 };  
  
 demandBox.PreviewTextInput += Cell\_PreviewTextInput;  
  
 Grid.SetRow(demandBox, \_numSuppliers + 1);  
 Grid.SetColumn(demandBox, j);  
 TransportGrid.Children.Add(demandBox);  
 }  
 TextBox sum = new TextBox  
 {  
 VerticalAlignment = VerticalAlignment.Center,  
 HorizontalAlignment = HorizontalAlignment.Center,  
 MinWidth = 30,  
 MinHeight = 30,  
 };  
  
 sum.PreviewTextInput += Cell\_PreviewTextInput;  
  
 Grid.SetRow(sum, \_numSuppliers + 1);  
 Grid.SetColumn(sum, \_numDemands + 1);  
 TransportGrid.Children.Add(sum);  
 }  
  
 *// Event handler for TextBox validation: allows only numeric input* private void Cell\_PreviewTextInput(object sender, System.Windows.Input.TextCompositionEventArgs e)  
 {  
 Regex regex = new Regex("[^0-9]+"); *// Only digits are allowed* e.Handled = regex.IsMatch(e.Text);  
 }  
 private void SolveButton\_OnClick(object sender, RoutedEventArgs e)  
 {  
 *// Read cost matrix from the grid* decimal[,] costMatrix = new decimal[\_numSuppliers, \_numDemands];  
 for (int i = 1; i <= \_numSuppliers; i++)  
 {  
 for (int j = 1; j <= \_numDemands; j++)  
 {  
 TextBox cell = (TextBox)TransportGrid.Children  
 .Cast<UIElement>()  
 .FirstOrDefault(e => Grid.GetRow(e) == i && Grid.GetColumn(e) == j);  
 costMatrix[i - 1, j - 1] = decimal.Parse(cell.Text);  
 }  
 }  
  
 *// Read supplies and demands* int[] supplies = new int[\_numSuppliers];  
 for (int i = 1; i <= \_numSuppliers; i++)  
 {  
 TextBox supplyBox = (TextBox)TransportGrid.Children  
 .Cast<UIElement>()  
 .FirstOrDefault(e => Grid.GetRow(e) == i && Grid.GetColumn(e) == \_numDemands + 1);  
 supplies[i - 1] = int.Parse(supplyBox.Text);  
 }  
  
 int[] demands = new int[\_numDemands];  
 for (int j = 1; j <= \_numDemands; j++)  
 {  
 TextBox demandBox = (TextBox)TransportGrid.Children  
 .Cast<UIElement>()  
 .FirstOrDefault(e => Grid.GetRow(e) == \_numSuppliers + 1 && Grid.GetColumn(e) == j);  
 demands[j - 1] = int.Parse(demandBox.Text);  
 }  
  
 *// Solve using UV method* var uvMethod = new UVMethod(this, costMatrix, supplies, demands);  
 decimal result = uvMethod.Solve();  
  
 *// Display result (total cost)* MessageBox.Show($"Total Transportation Cost: {result}", "Result");  
 }  
  
 public void ShowMatrixUV(int[,] allocationMatrix, decimal[,] deltaMatrix, decimal[] u, decimal[] v)  
 {  
 Grid dynamicGrid = new Grid  
 {  
 Margin = new Thickness(10),  
 ShowGridLines = true *// Optional: Show grid lines* };  
  
 int rows = allocationMatrix.GetLength(0) + 1;  
 int cols = allocationMatrix.GetLength(1) + 1;  
  
 for (int i = 0; i < rows; i++)  
 {  
 dynamicGrid.RowDefinitions.Add(new RowDefinition());  
 }  
  
 for (int i = 0; i < cols; i++)  
 {  
 dynamicGrid.ColumnDefinitions.Add(new ColumnDefinition());  
 }  
   
 CreateCell(dynamicGrid, 0, 0, @"U\V");  
  
 for (int i = 0; i < rows - 1; i++)  
 {  
 CreateCell(dynamicGrid, i + 1, 0, u[i].ToString());  
 }  
  
 for (int i = 0; i < cols - 1; i++)  
 {  
 CreateCell(dynamicGrid, 0, i + 1, v[i].ToString());  
 }  
  
 for (int i = 0; i < rows - 1; i++)  
 {  
 for (int j = 0; j < cols - 1; j++)  
 {  
 if (allocationMatrix[i, j] == -1)  
 {  
 CreateCell(dynamicGrid, i + 1, j + 1, deltaMatrix[i, j].ToString(), Brushes.Teal);  
 }  
 else  
 {  
 CreateCell(dynamicGrid, i + 1, j + 1, allocationMatrix[i, j].ToString(), Brushes.SeaGreen);  
 }  
 }  
 }  
   
 DynamicGridContainer.Children.Add(dynamicGrid);  
 }  
  
 private void CreateCell(Grid dynamicGrid, int x, int y, string text, Brush? color = null)  
 {  
 TextBox textBox = new TextBox()  
 {  
 Text = text,  
 Width = 60,  
 Height = 30,  
 Margin = new Thickness(5),  
 Background = color ?? Brushes.White,  
 IsReadOnly = true *// Make the TextBox read-only to prevent editing* };  
  
 dynamicGrid.Children.Add(textBox);  
  
 Grid.SetRow(textBox, x);  
 Grid.SetColumn(textBox, y);  
 }  
  
 private void PreDefine()  
 {  
 decimal[,] costs = new [,]  
 {  
 {4.2M, 10, 5, 9},  
 {5, 8, 5, 9},  
 {6, 4, 4, 7.3M},  
 {7, 5, 11, 4},  
 {3, 11, 8, 5}  
 };  
 int[] demands = new[] { 35, 22, 30, 15};  
 int[] suppliers = new[] { 17, 33, 20, 12, 20 };  
   
 SupplierComboBox.SelectedValue = "5";  
 ConsumerComboBox.SelectedValue = "4";  
   
   
 *// decimal[,] costs = new [,]  
 // {  
 // {4.2M, 10, 5, 9},  
 // {5, 8, 5, 9},  
 // {6, 4, 4, 7.3M},  
 // {7, 5, 11, 4},  
 // {3, 11, 8, 5}  
 // };  
 // int[] demands = new[] { 35, 22, 30, 15};  
 // int[] suppliers = new[] { 17, 33, 20, 12, 20 };  
 //   
 // SupplierComboBox.SelectedValue = "5";  
 // ConsumerComboBox.SelectedValue = "4";* for (int i = 1; i <= \_numSuppliers; i++)  
 {  
 for (int j = 1; j <= \_numDemands; j++)  
 {  
 ((TextBox)GetGridElement(TransportGrid, i, j)).Text = costs[i - 1, j - 1].ToString();  
 }  
 }  
 for(int i = 1; i <= \_numSuppliers; i++)  
 {  
 ((TextBox)GetGridElement(TransportGrid, i, \_numDemands + 1)).Text = suppliers[i - 1].ToString();  
 }  
 for(int i = 1; i <= \_numDemands; i++)  
 {  
 ((TextBox)GetGridElement(TransportGrid, \_numSuppliers + 1, i)).Text = demands[i - 1].ToString();  
 }  
  
 ((TextBox)GetGridElement(TransportGrid, \_numSuppliers + 1, \_numDemands + 1)).Text = suppliers.Sum().ToString();  
  
 }  
  
 private UIElement GetGridElement(Grid grid, int row, int column)  
 {  
 foreach (UIElement element in grid.Children)  
 {  
 if (Grid.GetRow(element) == row && Grid.GetColumn(element) == column)  
 {  
 return element;  
 }  
 }  
 return null;  
 }  
}

**LeastCostCellMethod.cs:**

using System;  
using System.Linq;  
using System.Windows;  
  
namespace Lab04\_OR.Methods  
{  
 public class LeastCostCellMethod  
 {  
 private readonly decimal[,] \_costMatrix;  
 private readonly int[,] \_allocation;  
 private readonly int[] \_supplies;  
 private readonly int[] \_demands;  
  
 public LeastCostCellMethod(decimal[,] costMatrix, int[] supplies, int[] demands)  
 {  
 \_costMatrix = costMatrix;  
 \_supplies = supplies;  
 \_demands = demands;  
  
 *// Create an allocation matrix initialized to zero* \_allocation = new int[\_supplies.Length, \_demands.Length];  
 }  
  
 *// Method to solve the transportation problem using Least Cost Cell method* public decimal Solve()  
 {  
 decimal totalCost = 0;  
  
 *// Copy of supplies and demands to keep track of remaining* int[] suppliesRemaining = (int[])\_supplies.Clone();  
 int[] demandsRemaining = (int[])\_demands.Clone();  
  
 while (suppliesRemaining.Any(s => s > 0) && demandsRemaining.Any(d => d > 0))  
 {  
 *// Find the least cost cell* (int minRow, int minCol) = FindLeastCostCell(suppliesRemaining, demandsRemaining);  
  
 *// Find the minimum of supply and demand for the selected cell* int allocationAmount = Math.Min(suppliesRemaining[minRow], demandsRemaining[minCol]);  
  
 *// Allocate this amount* \_allocation[minRow, minCol] = allocationAmount;  
  
 *// Update total cost* totalCost += allocationAmount \* \_costMatrix[minRow, minCol];  
  
 *// Update remaining supply and demand* suppliesRemaining[minRow] -= allocationAmount;  
 demandsRemaining[minCol] -= allocationAmount;  
 }  
  
 return totalCost;  
 }  
  
 *// Method to return the cost matrix (useful for inspection or testing)* public decimal[,] GetCostMatrix()  
 {  
 return \_costMatrix;  
 }  
 public int[,] GetAllocationMatrix()  
 {  
 return \_allocation;  
 }  
  
 *// Helper method to find the least cost cell from the remaining supplies and demands* private (int, int) FindLeastCostCell(int[] suppliesRemaining, int[] demandsRemaining)  
 {  
 decimal minCost = decimal.**MaxValue**;  
 int minRow = -1;  
 int minCol = -1;  
  
 for (int i = 0; i < \_supplies.Length; i++)  
 {  
 if (suppliesRemaining[i] > 0) *// Only consider rows with remaining supplies* {  
 for (int j = 0; j < \_demands.Length; j++)  
 {  
 if (demandsRemaining[j] > 0) *// Only consider columns with remaining demands* {  
 if (\_costMatrix[i, j] <= minCost)  
 {  
 minCost = \_costMatrix[i, j];  
 minRow = i;  
 minCol = j;  
 }  
 }  
 }  
 }  
 }  
  
 return (minRow, minCol);  
 }  
 }  
}

**UVMethod.cs:***// UVMethod.cs*namespace Lab04\_OR.Methods  
{  
 public class UVMethod  
 {  
 private MainWindow \_mainWindow;  
 private decimal[,] \_costMatrix;  
 private decimal[,] \_deltaMatrix;  
 private int[,] \_allocation;  
 private int[] \_supplies;  
 private int[] \_demands;  
 private decimal[] \_u;  
 private decimal[] \_v;  
  
 private const int **EMPTY\_CELL** = -1; *// Sentinel value for empty cells* public UVMethod(MainWindow mainWindow, decimal[,] costMatrix, int[] supplies, int[] demands)  
 {  
 \_mainWindow = mainWindow;  
 \_costMatrix = costMatrix;  
 \_supplies = supplies;  
 \_demands = demands;  
 \_deltaMatrix = new decimal[\_costMatrix.GetLength(0), \_costMatrix.GetLength(1)];  
  
 \_u = new decimal[supplies.Length];  
 \_v = new decimal[demands.Length];  
  
 *// Initialize the allocation matrix with the EMPTY\_CELL value* \_allocation = new int[supplies.Length, demands.Length];  
 }  
  
 *// Solve using UV Method* public decimal Solve()  
 {  
 var leastCostSolver = new LeastCostCellMethod(\_costMatrix, \_supplies, \_demands);  
 leastCostSolver.Solve();  
 \_allocation = leastCostSolver.GetAllocationMatrix();  
 for (int i = 0; i < \_allocation.GetLength(0); i++)  
 {  
 for (int j = 0; j < \_allocation.GetLength(1); j++)  
 {  
 if(\_allocation[i, j] == 0)  
 \_allocation[i, j] = **EMPTY\_CELL**; *// Indicate unallocated cells* }  
 }  
  
 if (!IsBalanced())  
 {  
 BalanceSystem();  
 }  
  
 CheckAndResolveDegeneracy();  
  
 bool optimalSolutionFound = false;  
  
 while (!optimalSolutionFound)  
 {  
 CalculateUV();  
  
 var (row, col, maxOpCost) = FindEnteringVariable();  
  
 \_mainWindow.ShowMatrixUV(\_allocation, \_deltaMatrix, \_u, \_v);  
 if (maxOpCost >= 0)  
 {  
 optimalSolutionFound = true;  
 }  
 else  
 {  
 PerformPivot(row, col);  
 }  
 }  
  
 return CalculateTotalCost();  
 }  
  
 private void CalculateUV()  
 {  
 \_u = new decimal[\_allocation.GetLength(0)];  
 \_v = new decimal[\_allocation.GetLength(1)];  
 bool[] \_uIsSet = new bool [\_u.Length];  
 bool[] \_vIsSet = new bool [\_v.Length];  
  
 int maxValues = 0;  
 int rowWithMaxValues = 0;  
 for (int i = 0; i < \_allocation.GetLength(0); i++)  
 {  
 int numOfValuesInRow = 0;  
 for (int j = 0; j < \_allocation.GetLength(1); j++)  
 {  
 numOfValuesInRow += \_allocation[i, j] != **EMPTY\_CELL** ? 1 : 0;  
 }  
  
 if (numOfValuesInRow > maxValues)  
 {  
 maxValues = numOfValuesInRow;  
 rowWithMaxValues = i;  
 }  
 }  
  
 \_u[rowWithMaxValues] = 0;  
 \_uIsSet[rowWithMaxValues] = true;  
  
 while(!(\_uIsSet.All(x => x) && \_vIsSet.All(x => x)))  
 {  
 for (int i = 0; i < \_u.Length; i++)  
 {  
 for (int j = 0; j < \_v.Length; j++)  
 {  
 if (\_allocation[i, j] != **EMPTY\_CELL**)  
 {  
 if (\_uIsSet[i] && !\_vIsSet[j])  
 {  
 \_v[j] = \_costMatrix[i, j] - \_u[i];  
 \_vIsSet[j] = true;  
 }  
 else if (\_vIsSet[j] && !\_uIsSet[i])  
 {  
 \_u[i] = \_costMatrix[i, j] - \_v[j];  
 \_uIsSet[i] = true;  
 }  
 }  
 }  
 }  
 }  
 }  
  
 private (int, int, decimal) FindEnteringVariable()  
 {  
 decimal minOpCost = 0;  
 int enteringRow = -1, enteringCol = -1;  
  
 for (int i = 0; i < \_u.Length; i++)  
 {  
 for (int j = 0; j < \_v.Length; j++)  
 {  
 if (\_allocation[i, j] == **EMPTY\_CELL**) *// Non-basic variable* {  
 decimal opCost = \_costMatrix[i, j] - (\_u[i] + \_v[j]);  
 \_deltaMatrix[i, j] = opCost;  
 if (opCost < minOpCost)  
 {  
 minOpCost = opCost;  
 enteringRow = i;  
 enteringCol = j;  
 }  
 }  
 }  
 }  
  
 return (enteringRow, enteringCol, minOpCost);  
 }  
  
 private void PerformPivot(int row, int col)  
 {  
 List<(int, int)> loop = FindLoop(row, col);  
  
 if (loop == null || loop.Count == 0)  
 {  
 throw new InvalidOperationException("No valid loop found for reallocation.");  
 }  
  
 int minAllocation = int.**MaxValue**;  
  
 for (int i = 1; i < loop.Count; i += 2)  
 {  
 (int r, int c) = loop[i];  
 if (\_allocation[r, c] < minAllocation)  
 {  
 minAllocation = \_allocation[r, c];  
 }  
 }  
  
 for (int i = 0; i < loop.Count; i++)  
 {  
 (int r, int c) = loop[i];  
  
  
 if (\_allocation[r, c] == 0 && !(r == row && c == col) && minAllocation == 0)  
 {  
 \_allocation[r, c] = **EMPTY\_CELL**; *// Reset to empty* continue;  
 }  
   
 if (i % 2 == 0)  
 {  
 \_allocation[r, c] += minAllocation;  
 }  
 else  
 {  
 \_allocation[r, c] -= minAllocation;  
 }  
 }  
 }  
  
 *// Find the loop formed by the new entering cell at (row, col)* private List<(int, int)> FindLoop(int row, int col)  
 {  
 *// We'll find the loop by tracing through the rows and columns, connecting allocated cells.* List<(int, int)> loop = new List<(int, int)>();  
  
 *// Add the starting cell (the entering variable)* loop.Add((row, col));  
 \_allocation[row, col] = 0;  
  
 *// We now need to find a path that alternates between rows and columns  
 // and returns to the starting point.  
 // For simplicity, we'll perform a search to find the loop.* bool loopFound = TraceLoop(loop, row, col, true); *// Start with row trace* if (!loopFound)  
 {  
 return null; *// No loop found* }  
  
 return loop;  
 }  
  
*// Recursive helper function to trace the loop  
// - "isRow" indicates whether we're tracing rows (if true) or columns (if false)* private bool TraceLoop(List<(int, int)> loop, int currentRow, int currentCol, bool isRow)  
 {  
 *// Base case: If loop has returned to the starting point and has more than 3 elements, it's complete* (int row, int col) = loop[0];  
 if (loop.Count > 3 && (row == currentRow || col == currentCol))  
 {  
 return true; *// Loop completed* }  
  
 if (isRow)  
 {  
 *// Search the current row for other allocations in the same row* for (int j = 0; j < \_allocation.GetLength(1); j++)  
 {  
 *// Ignore empty cells marked with -1 and skip the current column* if (j != currentCol && \_allocation[currentRow, j] != **EMPTY\_CELL**)  
 {  
 *// Add the next step in the loop* loop.Add((currentRow, j));  
  
 *// Recur to trace the column now* if (TraceLoop(loop, currentRow, j, false))  
 {  
 return true;  
 }  
  
 *// Backtrack if no loop found* loop.RemoveAt(loop.Count - 1);  
 }  
 }  
 }  
 else  
 {  
 *// Search the current column for other allocations in the same column* for (int i = 0; i < \_allocation.GetLength(0); i++)  
 {  
 *// Ignore empty cells marked with -1 and skip the current row* if (i != currentRow && \_allocation[i, currentCol] != **EMPTY\_CELL**)  
 {  
 *// Add the next step in the loop* loop.Add((i, currentCol));  
  
 *// Recur to trace the row now* if (TraceLoop(loop, i, currentCol, true))  
 {  
 return true;  
 }  
  
 *// Backtrack if no loop found* loop.RemoveAt(loop.Count - 1);  
 }  
 }  
 }  
  
 return false; *// No valid loop found* }  
  
  
 private void CheckAndResolveDegeneracy()  
 {  
 int m = \_allocation.GetLength(0);  
 int n = \_allocation.GetLength(1);  
 int requiredAllocations = m + n - 1;  
 int currentAllocations = 0;  
 for (int i = 0; i < m; i++)  
 {  
 for (int j = 0; j < n; j++)  
 {  
 if (\_allocation[i, j] != **EMPTY\_CELL** && \_allocation[i, j] > 0)  
 {  
 currentAllocations++;  
 }  
 }  
 }  
 if (currentAllocations < requiredAllocations)  
 {  
 ResolveDegeneracy(currentAllocations, requiredAllocations);  
 }  
 }  
 private void ResolveDegeneracy(int currentAllocations, int requiredAllocations)  
 {  
 int m = \_allocation.GetLength(0);  
 int n = \_allocation.GetLength(1);  
 Dictionary<int, List<int>> keyCells = new Dictionary<int, List<int>>();  
 for (int i = 0; i < n; i++)  
 {  
 keyCells.Add(i, new List<int>());  
 for (int j = 0; j < m; j++)  
 {  
 if (\_allocation[j, i] != **EMPTY\_CELL**)  
 {  
 keyCells[i].Add(j);  
 }  
 }  
 }  
 var fulfilledRows = keyCells  
 .Where(x => x.Value.Count >= 2)  
 .SelectMany(x => x.Value)  
 .Distinct()  
 .ToList();  
 var lonelyCells = keyCells  
 .Where(x => x.Value.Count < 2 && !fulfilledRows  
 .Contains(x.Value.FirstOrDefault()))  
 .Select(x => x.Key)  
 .ToList();  
 foreach (var lonelyCell in lonelyCells)  
 {  
 \_allocation[fulfilledRows.FirstOrDefault(), lonelyCell] = 0;  
 currentAllocations++;  
 if (currentAllocations == requiredAllocations)  
 break;  
 }  
 }  
  
 private bool IsForbidden(int row, int col)  
 {  
 return false;  
 }  
  
 private decimal CalculateTotalCost()  
 {  
 decimal totalCost = 0;  
  
 for (int i = 0; i < \_u.Length; i++)  
 {  
 for (int j = 0; j < \_v.Length; j++)  
 {  
 if (\_allocation[i, j] != **EMPTY\_CELL** && \_allocation[i, j] > 0)  
 {  
 totalCost += \_allocation[i, j] \* \_costMatrix[i, j];  
 }  
 }  
 }  
  
 return totalCost;  
 }  
  
 private bool IsBalanced()  
 {  
 return \_supplies.Sum() == \_demands.Sum();  
 }  
  
 private void BalanceSystem()  
 {  
 int supplySum = \_supplies.Sum();  
 int demandSum = \_demands.Sum();  
  
 if (supplySum > demandSum)  
 {  
 int[] newDemands = new int[\_demands.Length + 1];  
 Array.Copy(\_demands, newDemands, \_demands.Length);  
 newDemands[^1] = supplySum - demandSum;  
 \_demands = newDemands;  
  
 decimal[,] newCostMatrix = new decimal[\_supplies.Length, \_demands.Length];  
 Array.Copy(\_costMatrix, newCostMatrix, \_costMatrix.Length);  
 \_costMatrix = newCostMatrix;  
 }  
 else if (demandSum > supplySum)  
 {  
 int[] newSupplies = new int[\_supplies.Length + 1];  
 Array.Copy(\_supplies, newSupplies, \_supplies.Length);  
 newSupplies[^1] = demandSum - supplySum;  
 \_supplies = newSupplies;  
  
 decimal[,] newCostMatrix = new decimal[\_supplies.Length, \_demands.Length];  
 Array.Copy(\_costMatrix, newCostMatrix, \_costMatrix.Length);  
 \_costMatrix = newCostMatrix;  
 }  
 }  
  
 public int[,] GetAllocation()  
 {  
 return \_allocation;  
 }  
 }  
}

**Висновки:** протягом виконання лабораторної роботи 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.