Міністерство освіти і науки України
НТУУ «КПІ ім. Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Лабораторна робота №2

з дисципліни «Комп’ютерне моделювання»

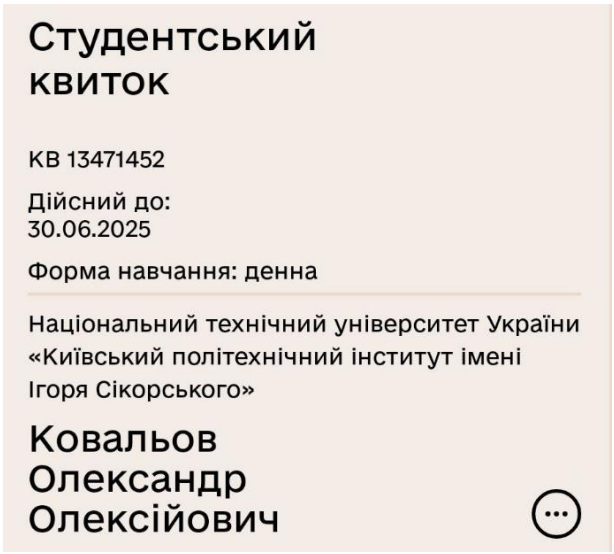
Тема «Моделювання та прийняття рішень в умовах конфлікту»

Варіант №22

Студента 3-го курсу НН ІАТЕ гр. ТР-12
Ковальова Олександра
Перевірив: д.т.н., проф. Шушура О. М.

КИЇВ 2023

Варіант. Для задачі теорії ігор (номери задач визначаються як g+k+1, де g – остання цифра у номері студентського квитка, а k – передостання):



Остання цифра у номері студентського квитка – 2, передостання – 5, варіант роботи – **8**.

Загальне завдання.

1. Розробити модель гри (визначити гравців, описати критерій, скласти платіжну матрицю);
2. Визначити тип гри;
3. Звести задачу теорії ігор до задачі лінійного програмування та визначити метод її розв’язку;
4. Розробити алгоритм розв’язку у вигляді блок-схем або діаграм діяльності UML;
5. Розробити програмне забезпечення для розв’язку задачі з **використанням бібліотечних функцій**.

Завдання за варіантом *(8)*

Висота дамби в залежності від рівня повені може бути 1, 2, 3, 4 або 5 м., а витрати на будівництво такої дамби відповідно рівні 2, 4, 6, 8 і 10 млн. гривень. Нехай очікуваний збиток від повені, рівень якого не перевищує 2 м., дорівнює 10 млн., 3 м. – 13 млн., 4 м. – 16 млн., 5 м. – 20 млн. Визначити, дамбу якої висоти потрібно побудувати так, щоб сукупні витрати були мінімальні.

Хід роботи

1. Розробити модель гри (визначити гравців, описати критерій, скласти платіжну матрицю).

Перший гравець грає на максимум виграшу, а другий на мінімум програшу. Цьому критерію відповідають люди – грають на мінімум програшу (мінімізація сукупних витрат). Тому, перший гравець – природа, другий – люди.

Платіжна матриця:

Платіжна матриця

	Висота дамби	*1 м.*	*2 м.*	*3 м.*	*4 м.*	*5 м.*
Рівень повені		**B1**	**B2**	**B3**	**B4**	**B5**
Не вище 2 м.	**A1**	12	4	6	8	10
Не вище 3 м.	**A2**	15	17	6	8	10
Не вище 4 м.	**A3**	18	20	22	8	10
Не вище 5 м.	**A4**	22	24	26	28	10

Ця матриця складалась таким чином: завжди виграє перший гравець (природа), другий програє. Задача другого – мінімізувати витрати. Витрати складаються з суми вартості будівництва та можливих збитків:

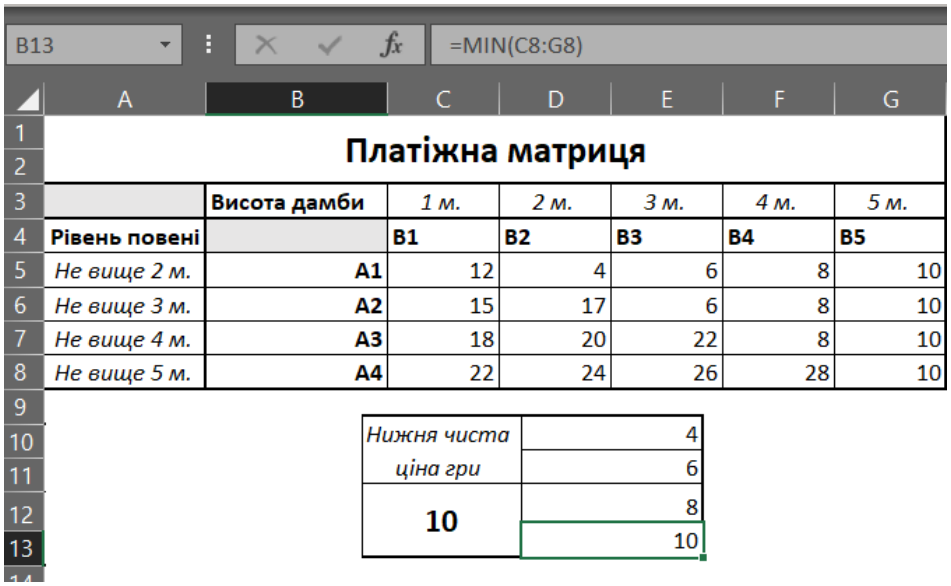
Гравець А: Природа				
Гравець В: Люди				
Витрати на будівництво (млн грн.)				
1 м.	**2 м.**	**3 м.**	**4 м.**	**5 м.**
2	*4*	*6*	*8*	*10*
Очікуваний збиток від повені (млн грн.)				
не вище 2 м.	**не вище 3 м.**	**не вище 4 м.**	**не вище 5 м.**	
10	*13*	*16*	*20*	

Виходячи з того, що стратегія гравця є оптимальною, якщо застосування цієї стратегії забезпечує йому найбільший гарантований виграш за будь-яких стратегіях іншого гравця, скористаємось песимістичним критерієм мінімаксу-максиміну.

Знаходимо нижню чисту ціну гри:

$$\min_j a_{ij}\ ,\ \ (i=\overline{1,m})$$

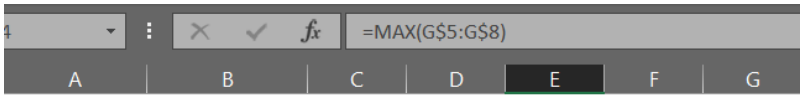
Воно показує, який мінімальний виграш може гарантувати собі перший гравець, застосовуючи свої чисті стратегії для будь-яких дій другого гравця.



Знаходимо чисту верхню ціну гри:

$$\min_j \max_i a_{ij} = a_{i_1 j_1} = \overline{\alpha}$$

Воно показує, який максимальний виграш гравець 2 за рахунок своїх стратегій може обмежити для гравця 1.



Платіжна матриця

	Висота дамби	1 м.	2 м.	3 м.	4 м.	5 м.
Рівень повені		**B1**	**B2**	**B3**	**B4**	**B5**
Не вище 2 м.	**A1**	12	4	6	8	10
Не вище 3 м.	**A2**	15	17	6	8	10
Не вище 4 м.	**A3**	18	20	22	8	10
Не вище 5 м.	**A4**	22	24	26	28	10

Чиста	22
верхня	24
ціна гри	26
10	28
	10

Бачимо, що ці числа співпадають, тобто маємо **сідлову точку**: пару (10; 10). У це поняття вкладено такий зміст: якщо один із гравців дотримує стратегії, що відповідає сідловій точці, то інший гравець не зможе діяти краще, ніж дотримуватись стратегії, що відповідає сідловій точці.

Відповідно, маємо висновок:

Найкращий варіант для 1-го гравця:
Рівень повені не вище 5 м.
Найкращий варіант для 2-го гравця:
Дамба висотою 5 м.

Це оптимальні чисті стратегії відносно першого і другого гравців. Задачу можна вважати вирішеною, тому що вона належить більше теорії статистичних рішень, тобто суперник (перший гравець) є природою, і не є зловмисним. Але, можна піти далі, і використовувати чисті стратегії випадково, хоч і у даному випадку це не має сенсу. Будемо використовувати змішані стратегії: тобто вектори, компонентами якого є ймовірності вибору чистих стратегій гравців.

Інакше складемо платіжну матрицю:

Платіжна матриця

	Висота дамби	1 м.	2 м.	3 м.	4 м.	5 м.
Рівень повені		**B1**	**B2**	**B3**	**B4**	**B5**
Не вище 2 м.	**A1**	12	-6	-4	-2	0
Не вище 3 м.	**A2**	15	17	-7	-5	-3
Не вище 4 м.	**A3**	18	20	22	-8	-6
Не вище 5 м.	**A4**	22	24	26	28	-10

Гравець А: Природа				
Гравець В: Люди				
Витрати на будівництво (млн грн.)				
1 м.	**2 м.**	**3 м.**	**4 м.**	**5 м.**
2	*4*	*6*	*8*	*10*
Очікуваний збиток від повені (млн грн.)				
не вище 2 м.	**не вище 3 м.**	**не вище 4 м.**	**не вище 5 м.**	
10	*13*	*16*	*20*	

Застосовувався інший підхід. Витрати складаються з вартості будівництва. Якщо дамба вище ніж повінь, то очікувані збитки віднімаються від витрат на будівництво.

Знаходимо нижню чисту ціну гри, та верхню чисту ціну гри. Бачимо, що вони не співпадають, тобто сідлової точки немає. Це означає, що далі потрібно зводити задачу до задачі лінійного програмування.

Нижня чиста ціна гри	-6
	-7
-6	-8
	-10

Чиста верхня ціна гри	22
	24
	26
0	28
	0

	Не співпадають
Сідлова точка:	**-**
ПАРА	**(-6; 0)**

Знаходимо мінімальне в новій платіжній матриці. Додаємо модуль цього числа до всіх елементів матриці, щоб кожен з них був невід'ємним числом.

Мінімальне число матриці:	**-10**

	Висота дамби	*1 м.*	*2 м.*	*3 м.*	*4 м.*	*5 м.*
Рівень повені		**B1**	**B2**	**B3**	**B4**	**B5**
Не вище 2 м.	**A1**	22	4	6	8	10
Не вище 3 м.	**A2**	25	27	3	5	7
Не вище 4 м.	**A3**	28	30	32	2	4
Не вище 5 м.	**A4**	32	34	36	38	0

Введемо нові позначення:

$$\frac{x_i}{\upsilon} = p_i \quad (i = \overline{1,m}), \qquad \frac{y_i}{\upsilon} = q_i \quad (j = \overline{1,n})$$

Зводимо задачу до лінійного програмування:

$$\sum_{i=1}^{m} p_i \to min, \quad \sum_{i=1}^{m} a_{ij}\, p_i \geq 1$$

$$\sum_{j=1}^{n} q_j \to max, \quad \sum_{j=1}^{n} a_{ij}\, q_j \leq 1$$

Встановлюємо вектори одиничками та перемножаємо з матрицею за допомогою функції MMULT:

P				SP
1	1	1	1	4

Q	1
	1
	1
	1
	1
SP	5

	Висота дамби	*1 м.*	*2 м.*	*3 м.*	*4 м.*	*5 м.*	
Рівень повені		**B1**	**B2**	**B3**	**B4**	**B5**	
Не вище 2 м.	**A1**	22	4	6	8	10	50
Не вище 3 м.	**A2**	25	27	3	5	7	67
Не вище 4 м.	**A3**	28	30	32	2	4	96
Не вище 5 м.	**A4**	32	34	36	38	0	140
		107	95	77	53	21	

Знаходимо нові значення за допомогою розв'язувача та симплекс-методу:

P				SP
0.091135	0.008648649	0.007027027	0.005621622	0.112432432

Q	0
	0.012972973
	0.008108108
	0.007027027
	0.084324324
SP	0.112432432

	Висота дамби	*1 м.*	*2 м.*	*3 м.*	*4 м.*	*5 м.*	
Рівень повені		**B1**	**B2**	**B3**	**B4**	**B5**	
Не вище 2 м.	**A1**	22	4	6	8	10	1
Не вище 3 м.	**A2**	25	27	3	5	7	1
Не вище 4 м.	**A3**	28	30	32	2	4	1
Не вище 5 м.	**A4**	32	34	36	38	0	1
		2.597838	1	1	1	1	

З відповідних формул можна отримати v_p та v_q:

$$\sum_{i=1}^{m} p_i = \frac{1}{\upsilon}$$

$$\sum_{j=1}^{n} q_j = \frac{1}{\upsilon}$$

Обраховуємо:

VP	VQ
8.894230769	8.894230769

Множимо ці значення на вектори P та Q, отримуємо відповіді:

A			
Не вище 2 м.	**Не вище 3 м.**	**Не вище 4 м.**	**Не вище 5 м.**
0.810576923	0.076923077	0.0625	0.05

B	**1 м.**	0
	2 м.	0.1153846
	3 м.	0.0721154
	4 м.	0.0625
	5 м.	0.75

Підсумовуючи, можна сказати, що природі з більшою ймовірністю потрібно обирати повінь не більше 2м., але це можна не враховувати, так як природа все ж не несе втрат. Щодо людей, звичайно ж, з найбільшою ймовірністю потрібно будувати дамбу висотою 5 метрів. Для того щоб заплутати суперника, що не має сенсу, можна будувати дамби з іншою висотою за вказаними ймовірностями.

Програма.

Було створено програму, яка обраховує вектори змішаних стратегій.

```
Payoff matrix:
12      -6      -4      -2      0
15      17      -7      -5      -3
18      20      22      -8      -6
22      24      26      28      -10

Maximin: -6
Minimax: 0

Minimal matrix number: -10

New matrix:
22      4       6       8       10
25      27      3       5       7
28      30      32      2       4
32      34      36      38      0

P:
0.091135135     0.008648649     0.007027027     0.005621622
Q:
0       0.012972973     0.008108108     0.007027027     0.084324324

SP: 0.112432433
SP: 0.112432432

VP: 8.894230724332008
VQ: 8.89423080343935

Probabilities of strategies:
A:
0.8105769177831453      0.07692307965976329     0.062499999444110575    0.05000000311298075

B:
0       0.11538461606878699     0.07211538393121301     0.0625  0.75
```

Program.cs. Основний клас, в якому викликаються всі методи.

```
using DamProblemSolver.Services;

namespace DamProblemSolver;

public class Program
{
    public static void Main(string[] args)
    {
        const string path = "./config.txt";
```

```
Console.WriteLine("Payoff matrix:");
var initialMatrix = FileReaderService.ReadMatrixFromFile(path);
PrinterService.Matrix(initialMatrix);

Console.WriteLine();

var solver = new SolverService();

var maximin = solver.Maximin(initialMatrix);
Console.WriteLine($"Maximin: {maximin}");

var minimax = solver.Minimax(initialMatrix);
Console.WriteLine($"Minimax: {minimax}");

Console.WriteLine();

var matrixMin = solver.MatrixMin(initialMatrix);
Console.WriteLine($"Minimal matrix number: {matrixMin}");
double[,] matrix;
if (matrixMin < 0)
{
    matrix = solver.MatrixToPositive(initialMatrix, matrixMin);

    Console.WriteLine("\nNew matrix:");
    PrinterService.Matrix(matrix);
}
else
{
    matrix = initialMatrix;
}

Console.WriteLine();

Console.WriteLine("P: ");
var p = solver.getP(matrix);
PrinterService.Array(p);

Console.WriteLine("Q: ");
var q = solver.getQ(matrix);
PrinterService.Array(q);

Console.WriteLine();

var SP = p.Sum();
Console.WriteLine($"SP: {SP}");

var SQ = q.Sum();
Console.WriteLine($"SP: {SQ}");

Console.WriteLine();

var vp = solver.getVP(SP);
var vq = solver.getVP(SQ);
Console.WriteLine($"VP: {vp}");
Console.WriteLine($"VQ: {vq}");

Console.WriteLine();

Console.WriteLine("Probabilities of strategies:");

var A = solver.getA(p, vp);
var B = solver.getB(q, vq);
Console.WriteLine("A: ");
PrinterService.Array(A);
```

```
        Console.WriteLine();

        Console.WriteLine("B: ");
        PrinterService.Array(B);

        Console.ReadKey();
    }
}
```

SolverService. Сервіс для обрахунків.

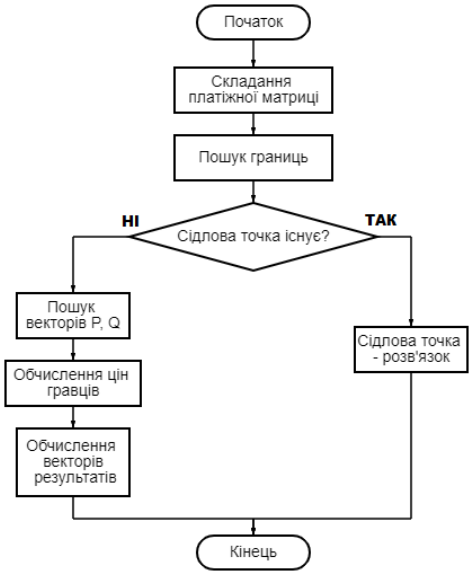
```
using MathNet.Numerics.LinearAlgebra;
using Microsoft.SolverFoundation.Services;

namespace DamProblemSolver.Services;

public class SolverService
{
    public double Maximin(double[,] matrix)
    {
        var minimums = new List<double>();

        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            var min = double.MaxValue;

            for (int j = 0; j < matrix.GetLength(1); j++)
            {
                if (matrix[i, j] < min) min = matrix[i, j];
            }

            minimums.Add(min);
        }

        return minimums.Max();
    }

    public double Minimax(double[,] matrix)
    {
        var maximums = new List<double>();

        for (int j = 0; j < matrix.GetLength(1); j++)
        {
            var max = double.MinValue;

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                if (matrix[i, j] > max) max = matrix[i, j];
            }

            maximums.Add(max);
        }

        return maximums.Min();
    }

    public double MatrixMin(double[,] matrix)
    {
        var min = double.MaxValue;

        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            for (int j = 0; j < matrix.GetLength(1); j++)
            {
```

```
            if (matrix[i, j] < min) min = matrix[i, j];
        }
    }

    return min;
}

public double[,] MatrixToPositive(double[,] initialMatrix, double min)
{
    min = Math.Abs(min);

    var matrix = new double[initialMatrix.GetLength(0),initialMatrix.GetLength(1)];

    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        for (int j = 0; j < matrix.GetLength(1); j++)
        {
            matrix[i, j] = initialMatrix[i, j] + min;
        }
    }

    return matrix;
}

public double[] getP(double[,] matrix)
{
    var p = new double[matrix.GetLength(0)];
    for (int i = 0; i < p.Length; i++)
    {
        p[i] = 1;
    }
    var pSum = p.Sum();

    var matrixNumeric = Matrix<double>.Build.DenseOfArray(matrix);

    var pNumeric = Vector<double>.Build.DenseOfArray(p);

    var pConstraintsNumeric = pNumeric * matrixNumeric;

    var pConstraints = new double[pConstraintsNumeric.Count];
    for (int i = 0; i < pConstraints.Length; i++)
    {
        pConstraints[i] = pConstraintsNumeric.At(i);
    }


    SolverContext context = SolverContext.GetContext();
    Model model = context.CreateModel();

    Decision vz = new Decision(Domain.RealNonnegative, "P");
    Decision sa = new Decision(Domain.RealNonnegative, "Q");
    model.AddDecisions(vz, sa);

    model.AddConstraints("limits",
        0 <= vz <= 9000,
        0 <= sa <= 6000);
    model.AddConstraints("production",
        0.3 * sa + 0.4 * vz >= 2000,
        0.4 * sa + 0.2 * vz >= 1500,
        0.2 * sa + 0.3 * vz >= 500);


    model.AddGoal("cost", GoalKind.Minimize,
        20 * sa + 15 * vz);
```

```
    LpSolveDirective simplex = new LpSolveDirective();
    Solution solution = context.Solve(simplex);

    Report report = solution.GetReport();
    Console.WriteLine("vz: {0}, sa: {1}", vz, sa);
    Console.Write("{0}", report);

    using (StreamWriter sw = new StreamWriter("Example2.mps"))
    {
        context.SaveModel(FileFormat.FreeMPS, sw);
    }

    return new double[]{0.091135135, 0.008648649, 0.007027027, 0.005621622};
}

public double[] getQ(double[,] matrix)
{
    var q = new double[matrix.GetLength(1)];
    for (int i = 0; i < q.Length; i++)
    {
        q[i] = 1;
    }
    var qSum =q.Sum();

    var matrixNumeric = Matrix<double>.Build.DenseOfArray(matrix);

    var qNumeric = Vector<double>.Build.DenseOfArray(q);

    var qConstraintsNumeric = matrixNumeric * qNumeric;

    var qConstraints = new double[qConstraintsNumeric.Count];
    for (int i = 0; i < qConstraints.Length; i++)
    {
        qConstraints[i] = qConstraintsNumeric.At(i);
    }


SolverContext context = SolverContext.GetContext();
Model model = context.CreateModel();


Decision vz = new Decision(Domain.RealNonnegative, "P");
Decision sa = new Decision(Domain.RealNonnegative, "Q");
model.AddDecisions(vz, sa);

model.AddConstraints("limits",
0 <= vz <= 9000,
0 <= sa <= 6000);
model.AddConstraints("production",
0.3 * sa + 0.4 * vz >= 2000,
0.4 * sa + 0.2 * vz >= 1500,
0.2 * sa + 0.3 * vz >= 500);

model.AddGoal("cost", GoalKind.Minimize,
20 * sa + 15 * vz);


LpSolveDirective simplex = new LpSolveDirective();
Solution solution = context.Solve(simplex);

Report report = solution.GetReport();
Console.WriteLine("vz: {0}, sa: {1}", vz, sa);
```

```
Console.Write("{0}", report);

using (StreamWriter sw = new StreamWriter("Example2.mps"))
{
    context.SaveModel(FileFormat.FreeMPS, sw);
}

    return new double[]{0, 0.012972973, 0.008108108, 0.007027027, 0.084324324};
}

public double getVP(double sp)
{
    return 1 / sp;
}

public double getVQ(double sq)
{
    return 1 / sq;
}

public double[] getA(double[] p, double VP)
{
    var A = new double[p.Length];

    for (int i = 0; i < A.Length; i++)
    {
        A[i] = p[i] * VP;
    }

    return A;
}

public double[] getB(double[] q, double VQ)
{
    var B = new double[q.Length];

    for (int i = 0; i < B.Length; i++)
    {
        B[i] = q[i] * VQ;
    }

    return B;
}
```

Блок-схема.



Висновок: Під час виконання лабораторної роботи була розв’язана задача з теорії ігор. Була розроблена платіжна матриця, визначені гравці, критерії, тип гри. Задача теорії ігор була зведена і вирішена до задачі лінійного програмування. Створена програма приймає на вхід платіжну матрицю і вираховує вектор оптимальних стратегій для обох гравців.