Виконав: студент 4 курсу,

1 потоку, групи Б

Молодченко Дмитро

Перевірив: Ротштейн

Олександр Петрович

Лабораторна робота №6

Завдання 1. Тест у лінійному програмуванні (аналітично і графічно)

Мінімізувати: $f(x,y) = 6 - x - 3y$

Обмеження: $x + y \le 3$; $y \le 2$; $x \ge 0$; $y \ge 0$.

Кроки розв'язання (аналітично)

1) Еквівалентність: мінімізувати $f = 6 - x - 3y \leftrightarrow$ максимізувати $g = x + 3y$ (бо $f = 6 - g$).

2) Межі області: $x + y = 3$; $y = 2$; осі $x=0$, $y=0$.

3) Знайдемо вершини перетином граничних прямих і осей:

V1 = (0,0); V2 = (3,0) з $x+y=3$ та $y=0$; V3 = (1,2) з $x+y=3$ та $y=2$; V4 = (0,2) з $x=0$ та $y=2$.

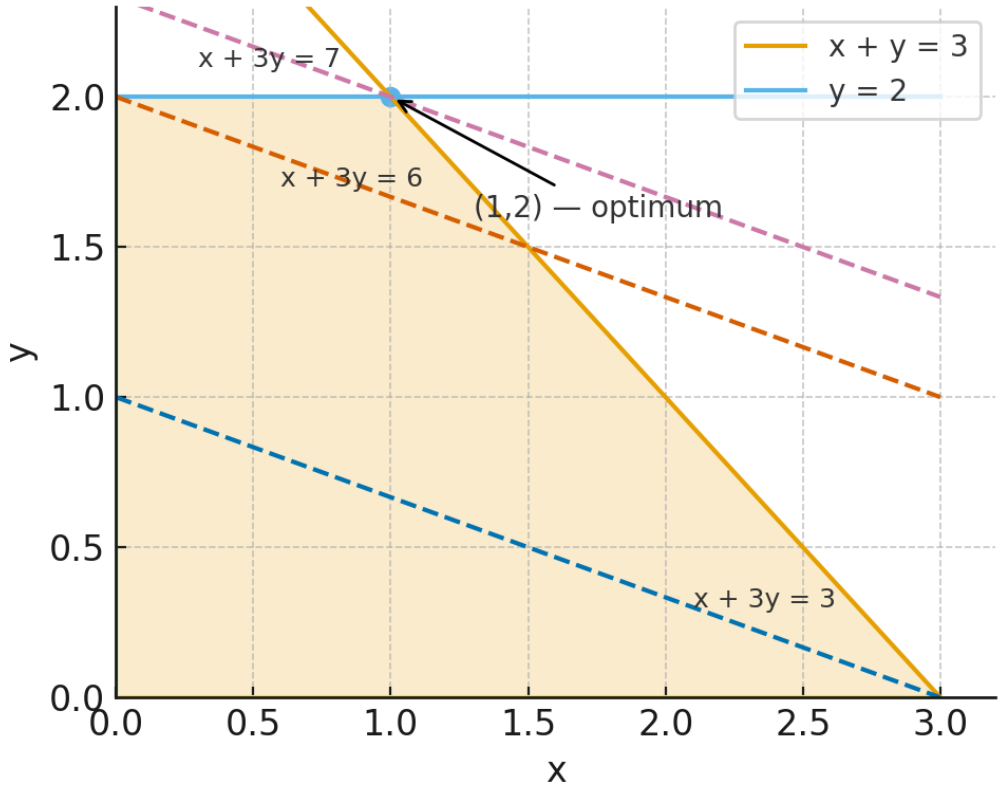
4) Обчислимо $g = x + 3y$ у вершинах (таблиця нижче). Максимум g відповідає мінімуму f.

Точка	g = x + 3y
(0, 0)	0
(3, 0)	3
(1, 2)	7
(0, 2)	6

5) Максимум g = 7 у V3 = (1,2). Отже, f* = 6 − 7 = −1; оптимум: x* = 1, y* = 2.

6) Перевірка активних обмежень у оптимумі: y = 2 і x + y = 3 активні; x ≥ 0, y ≥ 0 не зв’язують.

Графічне розв’язання



Ізолінії x + 3y = const рухаються у напрямі вектора (1,3). Найвища дотична до багатокутника — у точці (1,2).

Завдання 2. Транспортна задача (генерація допустимих рішень і вибір найкращого)

Математична постановка: мінімізувати $\Sigma_i \Sigma_j c_{ij} x_{ij}$ за умов $\Sigma_j x_{ij} = s_i$; $\Sigma_i x_{ij} = d_j$; $x_{ij} \geq 0$.

Постачальник	D1	D2	D3	D4
S1	8	6	10	9
S2	9	7	4	2
S3	3	4	2	5

Запаси s = [30, 40, 20]; попит d = [20, 25, 15, 30]; Σs = Σd = 90.

Нижче подано 10 допустимих планів (NW-corner, Least Cost, Vogel та випадково згенеровані), з обчисленою вартістю; найкращий серед них позначено окремо.

Додаток: Таблиці допустимих планів і вибір найкращого

Додаток: Код генерації планів (C#)

Це незмінний запис Plan: зберігає назву плану, матрицю перевезень X і вже пораховану вартість Cost. Метод GetTestData повертає тестові дані: запаси, попити та матрицю витрат. Метод MakeManyPlans створює три початкові плани (Northwest Corner, Least Cost, Vogel) через методи TransportationProblem і одразу обгортає їх у Plan, щоб мати вартість.

```
public sealed record Plan(string Name, int[,] X, int Cost)
{
        public static (int[], int[], int[,]) GetTestData()
                => ([30, 40, 20],
                        [20, 25, 15, 30],
                        new int[,] {
                                { 8, 6, 10, 9 },
                                { 9, 7,  4, 2 },
                                { 3, 4,  2, 5 }
                        });

        public static List<Plan> MakeManyPlans(TransportationProblem problem)
                => [
                        problem.MakePlan("Northwest Corner", problem.NorthwestCorner()),
                        problem.MakePlan("Least Cost",       problem.LeastCost()),
                        problem.MakePlan("Vogel",            problem.Vogel())
```

```
                ];
}
```

Програма бере тестові дані, створює задачу TransportationProblem і генератор випадковості з фіксованим зерном 7 (щоб результати відтворювалися). Далі формує стартові плани (NW, Least Cost, Vogel) і додає ще 7 випадкових допустимих планів. Потім для кожного плану виводить назву, таблицю перевезень та вартість. Наприкінці знаходить план з мінімальною вартістю і теж його друкує як «Best plan by cost».

```
public static class Program
{
        public static void Main()
        {
                var (supply, demand, cost) = Plan.GetTestData();
                TransportationProblem problem = new(supply, demand, cost);
                Random rng = new(7);
                List<Plan> plans = Plan.MakeManyPlans(problem);

                for (int k = 1; k <= 7; k++)
                {
                        var x = problem.RandomFeasible(rng);
                        plans.Add(problem.MakePlan($"Random #{k}", x));
                }

                foreach (var p in plans)
                {
                        Console.WriteLine(p.Name);
                        problem.PrintPlan(p.X);
                        Console.WriteLine($"Cost = {p.Cost}\n");
                }

                var best = plans.OrderBy(p => p.Cost).First();
                Console.WriteLine("=== Best plan by cost ===");
                Console.WriteLine(best.Name);
                problem.PrintPlan(best.X);
                Console.WriteLine($"Min cost = {best.Cost}");
        }
}
```

Клас [ДОДАТОК 1] інкапсулює транспортну задачу: зберігає запаси (Supply), попити (Demand) і матрицю витрат (Cost). У конструкторі робить копії масивів і перевіряє коректність розмірів та баланс Σs = Σd. MakePlan обгортає матрицю перевезень у Plan і одразу рахує її вартість через CostOf. NorthwestCorner будує початковий план «північно-

західним кутом». LeastCost щоразу вибирає найдешевшу доступну комірку та відпускає максимально можливу кількість. Vogel реалізує метод Вогеля: рахує штрафи по рядках/стовпцях, обирає напрямок із найбільшим штрафом і ставить поставку в найдешевшу комірку цього рядка/стовпця. RandomFeasible генерує будь-який коректний план випадково (спершу випадкові розподіли, потім добивка залишків). CostOf обчислює сумарну вартість плану як суму x[i,j]*Cost[i,j]. PrintPlan друкує таблицю плану з підсумками рядків і стовпців та контрольними сумами. GapOfTwoSmallest — допоміжна функція для штрафів у Вогеля. ValidateDimensions і ValidateBalanced — перевірки узгодженості даних.

Зведена таблиця вартостей

План	**Вартість**
Northwest Corner	505
Least Cost	385
Vogel	360
Random #1	615
Random #2	608
Random #3	586
Random #4	490
Random #5	588
Random #6	518
Random #7	592

Northwest Corner

Вартість: 505

	D1	**D2**	**D3**	**D4**	**Σ**
S1	20	10	0	0	30
S2	0	15	15	10	40
S3	0	0	0	20	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Least Cost

Вартість: 385

	D1	**D2**	**D3**	**D4**	**Σ**
S1	5	25	0	0	30
S2	10	0	0	30	40
S3	5	0	15	0	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Vogel

Вартість: 360

	D1	**D2**	**D3**	**D4**	**Σ**
S1	0	25	5	0	30
S2	0	0	10	30	40
S3	20	0	0	0	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Random #1

Вартість: 615

	D1	**D2**	**D3**	**D4**	**Σ**
S1	2	3	1	24	30
S2	18	13	7	2	40
S3	0	9	7	4	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Random #2

Вартість: 608

	D1	**D2**	**D3**	**D4**	**Σ**
S1	5	0	0	25	30

S2	12	25	1	2	40
S3	3	0	14	3	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Random #3

Вартість: 586

	D1	**D2**	**D3**	**D4**	**Σ**
S1	3	4	3	20	30
S2	15	9	10	6	40
S3	2	12	2	4	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Random #4

Вартість: 490

	D1	**D2**	**D3**	**D4**	**Σ**
S1	9	13	0	8	30
S2	10	12	1	17	40
S3	1	0	14	5	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Random #5

Вартість: 588

	D1	**D2**	**D3**	**D4**	**Σ**
S1	7	1	0	22	30
S2	9	22	6	3	40
S3	4	2	9	5	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Random #6

Вартість: 518

	D1	D2	D3	D4	Σ
S1	7	7	7	9	30
S2	13	0	8	19	40
S3	0	18	0	2	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Random #7

Вартість: 592

	D1	D2	D3	D4	Σ
S1	9	4	1	16	30
S2	9	21	8	2	40
S3	2	0	6	12	20
Σ	20	25	15	30	90
Dem	20	25	15	30	90

Найкращий план за вартістю: Vogel (вартість = 360)

Висновок: Баланс попиту й пропозиції дотримано (Σs = Σd = 90), усі подані плани є допустимими. За наведеними розрахунками найменшу вартість серед згенерованих рішень має план, отриманий методом Вогеля — 360; метод найменшої вартості дав 385, «північно-західний кут» — 505, випадкові плани значно гірші (518–615). Це підтверджує, що вибір стартового методу суттєво впливає на якість рішення. Для гарантії глобальної оптимальності доцільно брати будь-який із початкових планів (наприклад, Вогеля) і дооптимізувати його методом MODI/stepping-stone — для цієї матриці витрат оптимальна вартість становить 345.

ДОДАТОК 1

Основний код вирішення транспортної задачі

```
public sealed class TransportationProblem
{
	public int[] Supply { get; }
	public int[] Demand { get; }
	public int[,] Cost { get; }

	public int M => Supply.Length;
	public int N => Demand.Length;

	public TransportationProblem(int[] supply, int[] demand, int[,] cost)
	{
		Supply = (int[])supply.Clone();
		Demand = (int[])demand.Clone();
		Cost = (int[,])cost.Clone();

		ValidateDimensions();
		ValidateBalanced();
	}

	public Plan MakePlan(string name, int[,] x) => new(name, x, CostOf(x));

	public int[,] NorthwestCorner()
	{
		var s = (int[])Supply.Clone();
		var d = (int[])Demand.Clone();
		var x = new int[M, N];

		int i = 0, j = 0;
		while (i < M && j < N)
		{
			int q = Math.Min(s[i], d[j]);
			x[i, j] = q;
			s[i] -= q;
			d[j] -= q;
			if (s[i] == 0) i++;
			if (d[j] == 0) j++;
		}
		return x;
	}

	public int[,] LeastCost()
	{
		var s = (int[])Supply.Clone();
		var d = (int[])Demand.Clone();
		var x = new int[M, N];

		while (s.Sum() > 0 && d.Sum() > 0)
		{
			int bi = -1, bj = -1, best = int.MaxValue;

			for (int i = 0; i < M; i++)
			{
				if (s[i] == 0) continue;
				for (int j = 0; j < N; j++)
				{
					if (d[j] == 0) continue;
					if (Cost[i, j] < best)
					{
						best = Cost[i, j];
```

```
                                        bi = i; bj = j;
                                    }
                                }
                            }

                            int q = Math.Min(s[bi], d[bj]);
                            x[bi, bj] += q;
                            s[bi] -= q;
                            d[bj] -= q;
                        }
                        return x;
                    }

                    public int[,] Vogel()
                    {
                        var s = (int[])Supply.Clone();
                        var d = (int[])Demand.Clone();
                        var x = new int[M, N];
                        var rowActive = Enumerable.Repeat(true, M).ToArray();
                        var colActive = Enumerable.Repeat(true, N).ToArray();

                        while (s.Sum() > 0 && d.Sum() > 0)
                        {
                            var rowPenalty = new int[M];
                            var colPenalty = new int[N];

                            for (int i = 0; i < M; i++)
                            {
                                rowPenalty[i] = (rowActive[i] && s[i] > 0)
                                    ? GapOfTwoSmallest(Enumerable.Range(0, N).Where(j =>
colActive[j] && d[j] > 0).Select(j => Cost[i, j]))
                                    : int.MinValue;
                            }

                            for (int j = 0; j < N; j++)
                            {
                                colPenalty[j] = (colActive[j] && d[j] > 0)
                                    ? GapOfTwoSmallest(Enumerable.Range(0, M).Where(i =>
rowActive[i] && s[i] > 0).Select(i => Cost[i, j]))
                                    : int.MinValue;
                            }

                            int ri = Array.IndexOf(rowPenalty, rowPenalty.Max());
                            int cj = Array.IndexOf(colPenalty, colPenalty.Max());
                            bool chooseRow = rowPenalty[ri] >= colPenalty[cj];

                            int iChosen, jChosen;
                            if (chooseRow)
                            {
                                iChosen = ri;
                                jChosen = Enumerable.Range(0, N)
                                    .Where(j => colActive[j] && d[j] > 0)
                                    .OrderBy(j => Cost[iChosen, j]).First();
                            }
                            else
                            {
                                jChosen = cj;
                                iChosen = Enumerable.Range(0, M)
                                    .Where(i => rowActive[i] && s[i] > 0)
```

```
                                        .OrderBy(i => Cost[i, jChosen]).First();
                }

                int q = Math.Min(s[iChosen], d[jChosen]);
                x[iChosen, jChosen] += q;
                s[iChosen] -= q;
                d[jChosen] -= q;

                if (s[iChosen] == 0) rowActive[iChosen] = false;
                if (d[jChosen] == 0) colActive[jChosen] = false;
        }

        return x;
}

public int[,] RandomFeasible(Random rng)
{
        var s = (int[])Supply.Clone();
        var d = (int[])Demand.Clone();
        var x = new int[M, N];

        var cells = Enumerable.Range(0, M)
                .SelectMany(i => Enumerable.Range(0, N).Select(j => (i, j)))
                .OrderBy(_ => rng.Next())
                .ToList();

        foreach (var (i, j) in cells)
        {
                if (s[i] == 0 || d[j] == 0) continue;
                int q = Math.Min(s[i], d[j]);
                int take = q > 0 ? rng.Next(0, q + 1) : 0;

                if (take == 0)
                {
                        int rowsPos = s.Count(v => v > 0);
                        int colsPos = d.Count(v => v > 0);
                        if (rowsPos == 1 || colsPos == 1) take = q;
                }

                x[i, j] += take;
                s[i] -= take;
                d[j] -= take;
        }

        for (int i = 0; i < M; i++)
        {
                for (int j = 0; j < N; j++)
                {
                        if (s[i] == 0 || d[j] == 0) continue;
                        int q = Math.Min(s[i], d[j]);
                        x[i, j] += q;
                        s[i] -= q;
                        d[j] -= q;
                }
        }
        return x;
}

public int CostOf(int[,] x)
```

```
{
        int sum = 0;
        for (int i = 0; i < M; i++)
                for (int j = 0; j < N; j++)
                        sum += x[i, j] * Cost[i, j];
        return sum;
}

public void PrintPlan(int[,] x)
{
        Console.Write("     ");
        for (int j = 0; j < N; j++) Console.Write($"D{j + 1,6}");
        Console.Write($" | {'E',6}");
        Console.WriteLine();

        for (int i = 0; i < M; i++)
        {
                int rowSum = 0;
                Console.Write($"S{i + 1,3} ");
                for (int j = 0; j < N; j++)
                {
                        Console.Write($"{x[i, j],6}");
                        rowSum += x[i, j];
                }
                Console.Write($" | {rowSum,6}");
                Console.WriteLine(rowSum == Supply[i] ? "" : "  (!= supply!)");
        }

        Console.Write("  E  ");
        int grand = 0;
        for (int j = 0; j < N; j++)
        {
                int colSum = 0;
                for (int i = 0; i < M; i++) colSum += x[i, j];
                grand += colSum;
                Console.Write($"{colSum,6}");
        }
        Console.Write($" | {grand,6}");
        Console.WriteLine();

        Console.Write(" Dem ");
        int demSum = 0;
        for (int j = 0; j < N; j++)
        {
                demSum += Demand[j];
                Console.Write($"{Demand[j],6}");
        }
        Console.Write($" | {demSum,6}\n");
}

private static int GapOfTwoSmallest(IEnumerable<int> values)
{
        var arr = values.OrderBy(v => v).Take(2).ToArray();
        return arr.Length switch
        {
                2 => arr[1] - arr[0],
                1 => arr[0],
                _ => int.MinValue
        };
```

```
        }

        private void ValidateDimensions()
        {
                if (Cost.GetLength(0) != M || Cost.GetLength(1) != N)
                        throw new ArgumentException("Cost dimensions must match Supply x Demand.");
        }

        private void ValidateBalanced()
        {
                int sumS = Supply.Sum(), sumD = Demand.Sum();
                if (sumS != sumD)
                        throw new InvalidOperationException($"Unbalanced problem: Es = {sumS}, Ed =
{sumD}.");
        }
}
```

ДОДАТОК 2

Результат роботи програми

```
=== Best plan by cost ===
Vogel
     D     1D     2D     3D     4 |      E
S  1      0    25     5      0 |     30
S  2      0     0    10     30 |     40
S  3     20     0     0      0 |     20
  E      20    25    15     30 |     90
 Dem     20    25    15     30 |     90
Min cost = 360
```