Ліхачова В. КН-19-1

**Лабораторна робота №3**

**Програмування засноване на тестуванні**

**Мета роботи**: навчитися розробляти й реалізовувати модульні тести з використанням систем автоматизованого тестування серії Unit.

**Хід виконання роботи**

1. Вивчити теоретичні відомості.

2. Ознайомитися із принципами роботи системи автоматизованого тестування Unit.

3. Реалізувати модульні тести, що забезпечують перевірку основної функціональності кожного класу відповідно до розробленого проекту програмного продукту.

4. Домогтися успішного виконання всіх модульних тестів.

5. Зробити висновки з проекту програмного продукту.

**Хід роботи*:***

***Лістинг коду класів:***

abstract public class general\_information

{

public string name;

public string address;

public string removing\_extra\_spaces(string str)

{

var newstr = string.Join(" ", str.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries).Select(item => item.Trim()));

return newstr;

}

public string fix\_the\_registry(string str)

{

string[] newstr = str.Split();

for (int i = 0; i < newstr.Length; i++)

{

newstr[i] = char.ToUpper(newstr[i][0]) + newstr[i].Substring(1).ToLower();

}

str = string.Join(" ", newstr);

Console.WriteLine(str);

return str;

}

}

public class Provider : general\_information

{

public Provider(String name\_provider, String address\_provider, uint supply, float cost) //Постачальник

{

name\_provider = removing\_extra\_spaces(name\_provider);

address\_provider = removing\_extra\_spaces(address\_provider);

name\_provider = fix\_the\_registry(name\_provider);

address\_provider = fix\_the\_registry(address\_provider);

this.name = name\_provider;

this.address = address\_provider;

this.supply = supply;

this.cost = cost;

}

public uint supply; //Запаси

public float cost; //Вартість товару

~Provider()

{

}

}

public class Client : general\_information //Клієнт

{

public Client(String name\_client, string address\_client, uint order)

{

name\_client = removing\_extra\_spaces(name\_client);

address\_client = removing\_extra\_spaces(address\_client);

name\_client = fix\_the\_registry(name\_client);

address\_client = fix\_the\_registry(address\_client);

this.name = name\_client;

this.order = order;

this.address = address\_client;

}

public uint order; //потреби споживача

~Client()

{

}

}

public class Expenses //витрати

{

public float[,] fuel\_spending; //витрати на топливо

public float[] product\_costs; //витрати на товар

private float[,] general\_expenses; //Загальна сума

public float[,] Return\_GeneralExpenses()

{

general\_expenses = new float[fuel\_spending.GetLength(0), fuel\_spending.GetLength(1)];

for (int i = 0; i < fuel\_spending.GetLength(0); i++)

for (int j = 0; j < general\_expenses.GetLength(1); j++)

{

general\_expenses[i, j] = fuel\_spending[i, j] + product\_costs[i];

}

return general\_expenses;

}

~Expenses()

{

}

}

public class OptimalPlan //опорний план

{

private float[,] cost; //масив з витратами

private float[,] plan; //опорний план

private uint[] stock; //масив запасів

private uint[] order; //масив попиту

private int non\_zero; //кількість не-нульових елементів масиву plan

private int[] U; //Масив потенціалів u

private int[] V; //Масив потенціалів v

int index\_i\_zero;

int index\_j\_zero;

public bool plan\_optimality;

public OptimalPlan(float[,] \_cost, uint[] \_stock, uint[] \_needs)

{

cost = \_cost;

stock = \_stock;

order = \_needs;

Initialization();

}

private void Initialization()

{

plan = new float[cost.GetLength(0), cost.GetLength(1)];

Array.Copy(cost, plan, cost.Length);

U = new int[stock.Length]; //Масив потенціалів u

V = new int[order.Length]; //Масив потенціалів v

}

public void Start\_calculating()

{

Metod\_MinimalElement();

FindingPotentials();

}

private void Metod\_MinimalElement()

{

uint[] stock\_remnant; //залишки запасів

stock\_remnant = new uint[stock.Length];

uint[] needs\_remnant; //залишки попиту

needs\_remnant = new uint[order.Length];

Array.Copy(stock, stock\_remnant, stock.Length);

Array.Copy(order, needs\_remnant, order.Length);

float[,] auxiliary; //допоміжний масив

auxiliary = new float[cost.GetLength(0), cost.GetLength(1)];

Array.Copy(cost, auxiliary, cost.Length);

int condition; //умова невиродженності

non\_zero = 0; //кількість базисних клітинок

float min = 9999999; //мінімальний елемент

int index1 = 0, index2 = 0; //змінні для збереження індексу

for (int k = 0; k < cost.Length; k++)

{

for (int i = 0; i < stock.Length; i++)

{

for (int j = 0; j < order.Length; j++)

{

if (auxiliary[i, j] < min)

{

min = auxiliary[i, j];

index1 = i;

index2 = j;

}

}

}

if (stock\_remnant[index1] >= needs\_remnant[index2]) //якщо запасів більше ніж потрібно замовнику, то

{

plan[index1, index2] = needs\_remnant[index2]; //в план записується попит

if (plan[index1, index2] != 0)

{

non\_zero++;

}

stock\_remnant[index1] = stock\_remnant[index1] - needs\_remnant[index2]; //розрахунок залишку запасів на складі

needs\_remnant[index2] = 0;

}

else if (stock\_remnant[index1] <= needs\_remnant[index2])

{

plan[index1, index2] = stock\_remnant[index1];

if (plan[index1, index2] != 0)

{

non\_zero++;

}

needs\_remnant[index2] = needs\_remnant[index2] - stock\_remnant[index1]; //розрахунок залишку потреб замовника

stock\_remnant[index1] = 0;

}

auxiliary[index1, index2] = 88888;

min = 9999999;

}

condition = stock.Length + order.Length - 1;

if (condition == non\_zero)//перевірка на невиродженість опорного плану

{

Console.WriteLine("GOOD");

}

else

{

float minimal = 999999;

index\_i\_zero = 0;

index\_j\_zero = 0;

for (int i = 0; i < stock.Length; i++)

{

for (int j = 0; j < order.Length; j++)

{

if (plan[i, j] == 0)

{

if (cost[i, j] < minimal)

{

minimal = cost[i, j];

index\_i\_zero = i;

index\_j\_zero = j;

}

}

}

}

plan[index\_i\_zero, index\_j\_zero] = -1;

non\_zero = non\_zero + 1;

}

}

static double[] gauss(double[,] a, double[] y, int n)

{

double[] x;

double max;

int k, index;

const double eps = 0.00001; // точность

x = new double[n];

k = 0;

while (k < n)

{

// Пошук рядку з максимальним a[i][k]

max = Math.Abs(a[k, k]);

index = k;

for (int i = k + 1; i < n; i++)

{

if (Math.Abs(a[i, k]) > max)

{

max = Math.Abs(a[i, k]);

index = i;

}

}

// Перестановка рядків

if (max < eps)

{

double[] error = new double[1] { 0 };

// нет ненулевых диагональных элементов

return error;

}

double temp;

for (int j = 0; j < n; j++)

{

temp = a[k, j];

a[k, j] = a[index, j];

a[index, j] = temp;

}

temp = y[k];

y[k] = y[index];

y[index] = temp;

// Нормализация уравнений

for (int i = k; i < n; i++)

{

temp = a[i, k];

if (Math.Abs(temp) < eps) continue; // для нулевого коэффициента пропустить

for (int j = 0; j < n; j++)

a[i, j] = a[i, j] / temp;

y[i] = y[i] / temp;

if (i == k) continue; // уравнение не вычитать само из себя

for (int j = 0; j < n; j++)

a[i, j] = a[i, j] - a[k, j];

y[i] = y[i] - y[k];

}

k++;

}

// обратная подстановка

for (k = n - 1; k >= 0; k--)

{

x[k] = y[k];

for (int i = 0; i < k; i++)

y[i] = y[i] - a[i, k] \* x[k];

}

return x;

}

private void FindingPotentials()

{

int[] u\_index = new int[non\_zero]; //Масив для збереження індексів на перетині не-нульового елементу масива (u - А)

int[] v\_index = new int[non\_zero]; //Масив для збереження індексів на перетині не-нульового елементу масива (v - В)

int[] v\_index\_correction = new int[non\_zero];

recalc:

int k = 0;

for (int i = 0; i < stock.Length; i++)

{

for (int j = 0; j < order.Length; j++)

{

if (plan[i, j] != 0)

{

u\_index[k] = i;

v\_index[k] = j;

k++;

}

}

}

int coincidence = 0;

for (int p = 1; p < non\_zero; p++)

{

if (u\_index[0] == u\_index[p] || v\_index[0] == v\_index[p])

{

coincidence++;

}

}

if (coincidence > 0)

{

int[,] index = new int[non\_zero + 1, 2];

double[,] a = new double[non\_zero + 1, non\_zero + 1];

double[] y = new double[non\_zero + 1];

y[non\_zero] = 0;

int count = 0;

for (int i = 0; i < plan.GetLength(0); i++)

{

for (int j = 0; j < plan.GetLength(1); j++)

{

if (plan[i, j] != 0)

{

y[count] = cost[i, j];

count++;

}

}

}

for (int i = 0; i < non\_zero; i++)

{

v\_index\_correction[i] = v\_index[i] + stock.Length;

}

for (int i = 0; i < non\_zero; i++)

{

index[i, 0] = u\_index[i];

index[i, 1] = v\_index\_correction[i];

}

for (int i = 0; i < non\_zero; i++)

{

a[i, index[i, 0]] = 1;

a[i, index[i, 1]] = 1;

}

a[non\_zero, 0] = 1;

for (int i = 1; i < non\_zero + 1; i++)

{

a[non\_zero, i] = 0;

}

double[] potentials = new double[non\_zero + 1];

potentials = gauss(a, y, non\_zero + 1);

for (int i = 0; i < stock.Length; i++)

{

U[i] = (int)potentials[i];

}

count = 0;

for (int i = stock.Length; i < potentials.Length; i++)

{

V[count] = (int)potentials[i];

count++;

}

}

else

{

plan[index\_i\_zero, index\_j\_zero] = 0; //прибираємо псевдоноль

float minimal = 999999;

for (int j = 0; j < order.Length; j++)

{

if (plan[0, j] == 0)

{

if (cost[0, j] < minimal)

{

minimal = cost[0, j];

index\_i\_zero = 0;

index\_j\_zero = j;

}

}

}

plan[index\_i\_zero, index\_j\_zero] = -1; //розміщаємо "нульовy" підставку

goto recalc;

}

for (int i = 0; i < V.Length; i++)

{

Console.WriteLine(" V[" + i + "]=" + V[i]);

}

for (int i = 0; i < U.Length; i++)

{

Console.WriteLine(" U[" + i + "]=" + U[i]);

}

CalculateGrades();

}

private void CalculateGrades()

{

int[] grades;

int stop = 0;

int index\_i = 0, index\_j = 0;

int count = 0;

int null\_cells = 0;

for (int i = 0; i < stock.Length; i++)

{

for (int j = 0; j < order.Length; j++)

{

if (plan[i, j] == 0)

{

null\_cells++;

}

}

}

grades = new int[null\_cells];

for (int i = 0; i < stock.Length; i++)

{

for (int j = 0; j < order.Length; j++)

{

if (plan[i, j] == 0)

{

grades[count] = (int)cost[i, j] - (U[i] + V[j]);

Console.WriteLine((int)cost[i, j] + "- (" + U[i] + "+" + V[j] + ") = " + grades[count]);

if (grades[count] < 0)

{

stop = 1;

index\_i = i;

index\_j = j;

}

count++;

}

}

}

if (stop != 1)

{

Console.WriteLine("The plan is optimal");

plan\_optimality = true;

}

else

{

Console.WriteLine("The plan is not optimal");

plan\_optimality = false;

Recalculation(index\_i, index\_j);

}

}

private void Recalculation(int i, int j)

{

float[,] recalculation = new float[cost.GetLength(0), cost.GetLength(1)];

int counter = 1;

int[] current\_index = new int[2];

int[] old\_index = new int[2];

int[] deadlock = new int[2] { 666, 666 };

recalculation[i, j] = 1;

current\_index[0] = i;

current\_index[1] = j;

int stop\_minus;

int stop\_plus;

int finish = 0;

while (true)

{

if (counter % 2 != 0)

{

stop\_minus = 0;

for (int horizon = 0; horizon < order.Length; horizon++)

{

if (horizon != current\_index[1] && plan[current\_index[0], horizon] != 0 && plan[current\_index[0], horizon] != -1 && horizon != deadlock[1])

{

recalculation[current\_index[0], horizon] = -1;

old\_index[1] = current\_index[1];

current\_index[1] = horizon;

break;

}

else

{

stop\_minus++;

if (stop\_minus > order.Length - 1)

{

recalculation[current\_index[0], current\_index[1]] = 0;

deadlock[0] = current\_index[0];

current\_index[0] = old\_index[0];

stop\_minus = 0;

break;

}

}

}

}

else if (counter % 2 == 0)

{

stop\_plus = 0;

for (int vertical = 0; vertical < stock.Length; vertical++)

{

if (current\_index[1] == j && vertical == i && counter > 1)

{

finish = 1;

break;

}

if (vertical != current\_index[0] && plan[vertical, current\_index[1]] != 0 && plan[vertical, current\_index[1]] != -1 && vertical != deadlock[0])

{

recalculation[vertical, current\_index[1]] = 1;

old\_index[0] = current\_index[0];

current\_index[0] = vertical;

break;

}

else

{

stop\_plus++;

if (stop\_plus > stock.Length - 1)

{

recalculation[current\_index[0], current\_index[1]] = 0;

deadlock[1] = current\_index[1];

current\_index[1] = old\_index[1];

stop\_plus = 0;

break;

}

}

}

}

counter++;

if (finish == 1) break;

}

float min = 99999999;

for (int k = 0; k < stock.Length; k++)

{

for (int p = 0; p < order.Length; p++)

{

if (recalculation[k, p] == -1 && recalculation[k, p] < min)

{

min = plan[k, p];

}

}

}

for (int k = 0; k < plan.GetLength(0); k++)

{

for (int p = 0; p < plan.GetLength(1); p++)

{

if (recalculation[k, p] == -1)

{

plan[k, p] = plan[k, p] - min;

}

if (recalculation[k, p] == 1)

{

plan[k, p] = plan[k, p] + min;

}

}

}

FindingPotentials();

}

}

class Report

{

String report\_text;

String FileName;

private float[,] optimal\_plan;

string[] provider\_addresses;

string[] clien\_addresses;

public void get\_data\_for\_report(float[,] plan, string[] provider\_addresses\_, string[] clien\_addresses\_)

{

optimal\_plan = new float[plan.GetLength(0), plan.GetLength(1)];

provider\_addresses = new string[provider\_addresses\_.Length];

clien\_addresses = new string[clien\_addresses\_.Length];

Array.Copy(plan, optimal\_plan, plan.Length);

Array.Copy(provider\_addresses\_, provider\_addresses, provider\_addresses.Length);

Array.Copy(clien\_addresses\_, clien\_addresses, clien\_addresses.Length);

Fill\_out\_a\_report();

}

private void Fill\_out\_a\_report()

{

for (int i = 0; i < optimal\_plan.GetLength(0); i++)

{

for (int j = 0; j < optimal\_plan.GetLength(1); j++)

{

if (optimal\_plan[i, j] != 0)

{

report\_text = report\_text + "З адреси " + provider\_addresses[i] + " треба направити " + optimal\_plan[i, j] + " одиниць вантажу до споживача за адресою " + clien\_addresses[j] + ". \n";

}

}

}

Console.WriteLine(report\_text);

}

private void Filename\_confirmation(String name)

{

FileName = name + ".txt";

}

public bool Save\_report(string name)

{

int error = 0;

Filename\_confirmation(name);

StreamWriter SW = new StreamWriter(new FileStream(FileName, FileMode.Create, FileAccess.Write));

try

{

SW.Write(report\_text);

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

error = 1;

}

finally

{

SW.Close();

}

if (error == 1)

{

return false;

}

else

{

return true;

}

}

}

***Лістинг коду модульних тестів:***

using Xunit;

using ConsoleApp2;

namespace ConsoleApp2.Tests

{

public class UnitTest1

{

[Fact]

public void Test\_class\_Provider()

{

string name = " ковальов мирослав михайлович ";

string address = "Корольова, 29";

uint supply = 20;

float cost = 210;

string check\_string = "Ковальов Мирослав Михайлович";

var provider1 = new Provider(name, address, supply, cost);

Assert.Equal(check\_string, provider1.name);

}

[Fact]

public void Test\_class\_Client()

{

string name = "Виноградов Марк Михайлович";

string address = " поштова, 64 ";

uint order = 20;

string check\_string = "Поштова, 64";

var client1 = new Client(name, address, order);

Assert.Equal(check\_string, client1.address);

}

[Fact]

public void Test\_class\_Expenses()

{

var expenses = new Expenses();

expenses.product\_costs = new float [3]{210,240,220};

expenses.fuel\_spending = new float[3, 4] { { 50,63,72,80 }, { 42,64,55,73},{ 22,46,52,30 } };

float[,] check\_arr = new float[3, 4] { { 260, 273, 282, 290 }, { 282, 304, 295, 313 }, { 242, 266, 272, 250 } };

float[,] general\_expenses = expenses.Return\_GeneralExpenses();

Assert.Equal(check\_arr, general\_expenses);

}

[Fact]

public void Test\_class\_OptimalPlan\_Metod\_MinimalElement\_1()

{

uint[] stock = { 20, 35, 30 };

uint[] needs = { 10, 15, 25, 30 };

float[,] general\_expenses = new float[3, 4] { { 11, 4, 3, 1 }, { 6, 8, 9, 7 }, { 4, 8, 4, 2 } };

float[,] check = new float [3,4]{ {0,0,0,20 },{0,15,15,0 },{10,0,10,10}};

var OP = new OptimalPlan(general\_expenses, stock, needs);

float[,] plan = new float[3,4];

plan = OP.Metod\_MinimalElement();

Assert.Equal(check, plan);

}

[Fact]

public void Test\_class\_OptimalPlan\_Metod\_MinimalElement\_2()

{

uint[] stock = { 10,25,10,15};

uint[] needs = { 15, 10, 15,20};

float[,] general\_expenses = new float[4, 4] { { 60, 70, 63, 82 }, { 75, 70, 68, 60 }, { 78, 90, 86, 67 },{60,55,79,85 } };

float[,] check = new float[4, 4] {{ 10, 0, -1, 0 }, { 0, 0, 5, 20 }, {0, 0, 10, 0 },{ 5,10,0,0}};

var OP = new OptimalPlan(general\_expenses, stock, needs);

float[,] plan = OP.Metod\_MinimalElement();

Assert.Equal(check, plan);

}

[Fact]

public void Test\_class\_OptimalPlan\_FindingPotentials\_1()

{

uint[] stock = { 20, 35, 30 };

uint[] needs = { 10, 15, 25, 30 };

float[,] general\_expenses = new float[3, 4] { { 11, 4, 3, 1 }, { 6, 8, 9, 7 }, { 4, 8, 4, 2 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

int[] U\_check = new int[3] { 0, 6, 1 };

int[] V\_check = new int[4] { 3, 2, 3, 1 };

Assert.Equal(U\_check, OP.U);

Assert.Equal(V\_check, OP.V);

}

[Fact]

public void Test\_class\_OptimalPlan\_FindingPotentials\_2()

{

uint[] stock = { 10, 25, 10, 15 };

uint[] needs = { 15, 10, 15, 20 };

float[,] general\_expenses = new float[4, 4] { { 60, 70, 63, 82 }, { 75, 70, 68, 60 }, { 78, 90, 86, 67 }, { 60, 55, 79, 85 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

int[] U\_check = new int[4] { 0, 5, 23, 0 };

int[] V\_check = new int[4] { 60, 55, 63, 55 };

Assert.Equal(U\_check, OP.U);

Assert.Equal(V\_check, OP.V);

}

[Fact]

public void Test\_class\_OptimalPlan\_CalculateGrades\_1()

{

uint[] stock = { 20, 35, 30 };

uint[] needs = { 10, 15, 25, 30 };

float[,] general\_expenses = new float[3, 4] { { 11, 4, 3, 1 }, { 6, 8, 9, 7 }, { 4, 8, 4, 2 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

OP.CalculateGrades();

Assert.False(OP.plan\_optimality);

}

[Fact]

public void Test\_class\_OptimalPlan\_CalculateGrades\_2()

{

uint[] stock = { 10, 25, 10, 15 };

uint[] needs = { 15, 10, 15, 20 };

float[,] general\_expenses = new float[4, 4] { { 60, 70, 63, 82 }, { 75, 70, 68, 60 }, { 78, 90, 86, 67 }, { 60, 55, 79, 85 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

OP.CalculateGrades();

Assert.False(OP.plan\_optimality);

}

[Fact]

public void Test\_class\_OptimalPlan\_Recalculations\_1()

{

uint[] stock = { 20, 35, 30 };

uint[] needs = { 10, 15, 25, 30 };

float[,] general\_expenses = new float[3, 4] { { 11, 4, 3, 1 }, { 6, 8, 9, 7 }, { 4, 8, 4, 2 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

OP.CalculateGrades();

float[,] recalc\_plan\_chekc = new float[3, 4] { {0,0,0,20 }, { 10,15,5,0}, { 0,0,20,10 } };

Assert.Equal(recalc\_plan\_chekc, OP.recalc\_plan);

}

[Fact]

public void Test\_class\_OptimalPlan\_Recalculations\_2()

{

uint[] stock = { 10, 25, 10, 15 };

uint[] needs = { 15, 10, 15, 20 };

float[,] general\_expenses = new float[4, 4] { { 60, 70, 63, 82 }, { 75, 70, 68, 60 }, { 78, 90, 86, 67 }, { 60, 55, 79, 85 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

OP.CalculateGrades();

float[,] recalc\_plan\_chekc = new float[4, 4] { { 10, 0, -1, 0 } ,{0,0,15, 10}, {0,0,0,10 },{ 5,10,0,0} };

Assert.Equal(recalc\_plan\_chekc, OP.recalc\_plan);

}

[Fact]

public void Test\_class\_OptimalPlan\_final\_check\_1()

{

uint[] stock = { 20, 35, 30 };

uint[] needs = { 10, 15, 25, 30 };

float[,] general\_expenses = new float[3, 4] { { 11, 4, 3, 1 }, { 6, 8, 9, 7 }, { 4, 8, 4, 2 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

OP.CalculateGrades();

OP.FindingPotentials(OP.recalc\_plan);

OP.CalculateGrades();

Assert.True(OP.plan\_optimality);

}

[Fact]

public void Test\_class\_OptimalPlan\_final\_check\_2()

{

uint[] stock = { 10, 25, 10, 15 };

uint[] needs = { 15, 10, 15, 20 };

float[,] general\_expenses = new float[4, 4] { { 60, 70, 63, 82 }, { 75, 70, 68, 60 }, { 78, 90, 86, 67 }, { 60, 55, 79, 85 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials(OP.plan);

OP.CalculateGrades();

OP.FindingPotentials(OP.recalc\_plan);

OP.CalculateGrades();

Assert.True(OP.plan\_optimality);

}

[Fact]

public void Test\_class\_Report\_1()

{

string[] provider\_addresses = new string[] { "Солом’янська, 76", "Копиленка, 11", "Паторжинського, 54" };

string[] clien\_addresses = new string[] { " Володимирська, 20", "Михайла Грушевського, 32", "Арсенальна, 3", "М. Коцюбинського, 05" };

uint[] stock = { 20, 35, 30 };

uint[] needs = { 10, 15, 25, 30 };

float[,] general\_expenses = new float[3, 4] { { 11, 4, 3, 1 }, { 6, 8, 9, 7 }, { 4, 8, 4, 2 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials();

OP.CalculateGrades();

OP.FindingPotentials();

OP.CalculateGrades();

string text\_check = "З адреси Солом’янська, 76 треба направити 20 одиниць вантажу до споживача за адресою М. Коцюбинського, 05"+".\n"+

"З адреси Копиленка, 11 треба направити 10 одиниць вантажу до споживача за адресою Володимирська, 20" + ".\n" +

"З адреси Копиленка, 11 треба направити 15 одиниць вантажу до споживача за адресою Михайла Грушевського, 32" + ".\n" +

"З адреси Копиленка, 11 треба направити 5 одиниць вантажу до споживача за адресою Арсенальна, 3"+".\n" +

"З адреси Паторжинського, 54 треба направити 20 одиниць вантажу до споживача за адресою Арсенальна, 3" + ".\n" +

"З адреси Паторжинського, 54 треба направити 10 одиниць вантажу до споживача за адресою М. Коцюбинського, 05" + ".\n";

var report = new Report();

report.get\_data\_for\_report(OP.return\_plan(), provider\_addresses, clien\_addresses);

Assert.Equal(text\_check, report.report\_text);

}

[Fact]

public void Test\_class\_Report\_2()

{

string[] provider\_addresses = new string[] { "Солом’янська, 76", "Копиленка, 11", "Паторжинського, 54" };

string[] clien\_addresses = new string[] { "Львівська, 86", "Б. Грінченка, 43", "Арсенальна, 3", "Різницька, 51" };

uint[] stock = { 20, 35, 30 };

uint[] needs = { 10, 15, 25, 30 };

float[,] general\_expenses = new float[3, 4] { { 11, 4, 3, 1 }, { 6, 8, 9, 7 }, { 4, 8, 4, 2 } };

var OP = new OptimalPlan(general\_expenses, stock, needs);

OP.Metod\_MinimalElement();

OP.FindingPotentials();

OP.CalculateGrades();

OP.FindingPotentials();

OP.CalculateGrades();

var report = new Report();

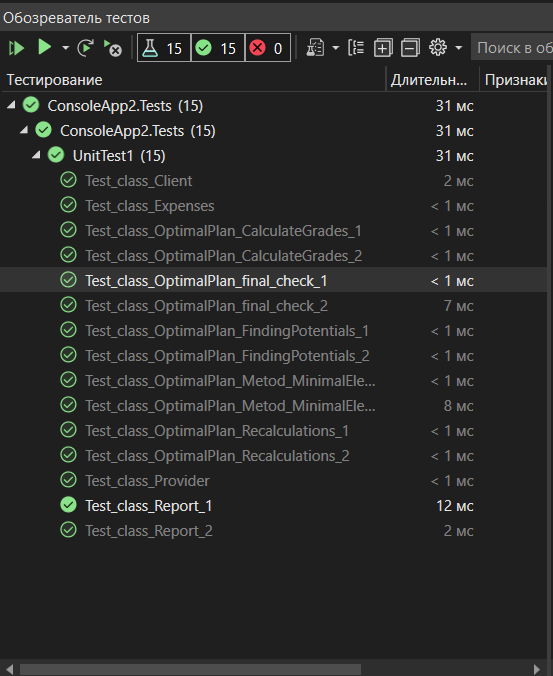
report.get\_data\_for\_report(OP.return\_plan(), provider\_addresses, clien\_addresses);

Assert.True(report.Save\_report("REPORT"));

}

}

}



*Рисунок 1 – Результати виконання тестів*

**Висновок**:в процесі виконання лабораторної роботи я ознайомилася із принципами роботи системи автоматизованого тестування Unit.

Було реалізовано модульні тести, що забезпечують перевірку основної функціональності кожного класу відповідно до розробленого проекту програмного продукту.